Федеральное агентство по образованию
АМУРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ГОУВПО "АмГУ"
Факультет математики и информатики

УТВЕРЖДАЮ
Зав. кафедрой МАиМ

_____ Т.В. Труфанова

«___»_____ 2007 г.

СПЕЦКУРС
ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

УЧЕБНО-МЕТОДИЧЕСКИЙ КОМПЛЕКС ПО ДИСЦИПЛИНЕ

для специальности 010501 – "Прикладная математика и информатика"

Составитель: А.В. Рыженко

Благовещенск
2007 г.

ББК

Печатается по решению редакционно-издательского совета факультета математики и информатики Амурского государственного университета

Рыженко А.В.

Объектно-ориентированное программирование: Учебно-методический комплекс по дисциплине для студентов АмГУ очной формы обучения специальности 010501 "Прикладная математика и информатика". – Благовещенск: Амурский гос. ун-т, 2007. – 97 с.

Учебно-методический комплекс по дисциплине "Объектно-ориентированное программирование" предназначен для студентов специальности 010501 – "Прикладная математика и информатика" очной формы обучения, призван помочь ведущим преподавателям и студентам в организации процесса изучения дисциплины. Комплекс содержит рабочую программу дисциплины, план-конспект лекций, материалы для проведения лабораторных работ, контролирующие материалы для осуществления промежуточного и итогового контроля, справочный материал и библиографический список.

I. РАБОЧАЯ ПРОГРАММА ДИСЦИПЛИНЫ

Рабочая программа по дисциплине "Объектно-ориентированное программирование" для специальности 010501 – "Прикладная математика и информатика".

Курс 4. Семестр 7, 8. Лекции 48 (16+32) час. Экзамен 8 семестр. Практические (семинарские) занятия (нет). Зачет 7 семестр. Лабораторные занятия 32 (16+16) час. Самостоятельная работа 30 час. Всего часов 110 час.

Составитель А.В. Рыженко, ст. преподаватель. Факультет математики и информатики. Кафедра математического анализа и моделирования. Благовещенск, 2006 г.

1. ЦЕЛИ И ЗАДАЧИ ДИСЦИПЛИНЫ, ЕЕ МЕСТО В УЧЕБНОМ ПРОЦЕССЕ

1.1. Цель преподавания дисциплины.

Целью изучения данной дисциплины является ознакомление студентов с языком программирования и платформой Java 2, в том числе средствами объектно-ориентированного программирования, а также освоение методикой построения объектно-ориентированных программ.

1.2. Задачи изучения дисциплины.

В результате изучения дисциплины студенты должны знать: 1) основные концепции объектно-ориентированного программирования (инкапсуляция, наследования и полиморфизм); 2) основные принципы организации сложных объектно-ориентированных систем; 3) основные конструкции языка программирования Java 2; 4) средства объектно-ориентированного программирования на Java 2; 5) методику объектно-ориентированного анализа и проектирования.

В результате изучения дисциплины студенты должны: 1) иметь представление об основных тенденциях развития современных информационных технологий; 2) представление об объектно-ориентированных библиотеках.

В результате изучения дисциплины студенты должны уметь: 1) писать программы, на языке Java 2 с использованием объектно-ориентированного подхода; 2) применять приемы и методы объектно-ориентированного программирования в своей практической деятельности.

1.3. Перечень дисциплин с указанием разделов (тем), усвоение которых студентами необходимо при изучении данной дисциплины.

Дисциплина является спецкурсом в рамках специальности 010501 "Прикладная математика и информатика" и базируется на общем курсе "Практикум на ЭВМ", который является составной частью цикла специальных дисциплин, определяющих подготовку студентов в области современных информационных технологий и предусмотрен государственным стандартом высшего профессионального образования для данной специальности.

2. СОДЕРЖАНИЕ ДИСЦИПЛИНЫ

2.1. Федеральный компонент.

Основные понятия и модели: объект, класс, данные, методы, доступ, наследование свойств; системы объектов и классов; проектирование объектно-ориентированных программ: методы и алгоритмы; объектно-ориентированные языки; классификация, архитектура, выразительные средства, технология применения; интерфейс: правила организации, методы и средства программирования; объектно-ориентированные системы: методы, языки и способы программирования.

2.2. Наименование тем, их содержание, объем в лекционных часах.

Тема 1. Введение. Языки программирования. Интерфейс прикладных программ – 2 час: 1) алгоритм, универсальные алгоритмические модели;

2) языки программирования и их классификация; 3) интерфейс прикладных программ, программирование под Windows.

Тема 2. История создания JAVA. Основные особенности платформы и её эволюция – 2 час: 1) история создания Java; 2) история развития; 3) платформа Java; 4) основные версии и продукты; 5) апплеты; 6) свойства языка Java; 7) процесс создания программ на Java.

Тема 3. Основы объектно-ориентированного программирования – 4 час: 1) методология процедурно-ориентированного программирования; 2) методология объектно-ориентированного программирования; 3) объекты: состояние, поведение, уникальность; 4) классы: инкапсуляция, наследование, полиморфизм; 5) типы отношений между классами; 6) метаклассы; 7) достоинства и недостатки объектно-ориентированного программирования.

Тема 4. Лексика языка – 2 часов: 1) кодировка; 2) анализ программы: пробелы, комментарии, лексемы; 3) виды лексем: идентификаторы, ключевые слова, литералы; 4) операторы языка.

Тема 5. Типы данных – 2 часа: 1) переменные; 2) примитивные типы данных; 3) ссылочные типы данных; 4) классы Object, String, Class.

Тема 6. Операторы и структура кода – 2 час: 1) нормальное и прерванное выполнение операторов; 2) блоки и локальные переменные; 3) метки; 4) оператор if; 5) оператор switch; 6) управление циклами; 7) операторы break и continue; 8) именованные блоки; 9) оператор return.

Тема 7. Ошибки при работе программы. Исключения – 2 час: 1) причины возникновения ошибок; 2) обработка исключительных ситуаций; 3) конструкция try-catch; 4) конструкция try-catch-finally; 5) использование оператора throw; 6) проверяемые и непроверяемые исключения; 7) создание пользовательских классов исключений; 8) переопределение методов и исключения.

Тема 8. Преобразование типов – 2 часов: 1) тождественное преобразование; 2) преобразование примитивных типов; 3) преобразование

ссылочных типов; 4) запрещенные преобразования; 5) применение приведений.

Тема 9. Массивы – 2 часов: 1) объявление и инициализация массивов; 2) многомерные массивы; 3) преобразование типов для массивов; 4) клонирование массивов.

Тема 10. Имена, пакеты – 4 час: 1) простые и составные имена; 2) имена и идентификаторы; 3) элементы пакетов; 4) платформенная поддержка пакетов; 5) модуль компиляции; 6) уникальность имен пакетов; 7) область видимости имен.

Тема 11. Объявление классов – 4 час: 1) модификаторы доступа; 2) объявление классов; 3) дополнительные свойства классов: метод main, параметры методов, перегруженные методы.

Тема 12. Объектная модель в Java – 4 час: 1) статические элементы; 2) ключевые слова this, super; 3) абстрактные классы; 4) интерфейсы; 5) полиморфизм.

Тема 13. Пакет JAVA.AWT – 4 час: 1) апплеты; 2) базовые классы; 3) основные компоненты; 4) менеджеры компоновки; 5) окна; 6) меню; 7) обработка событий.

Тема 14. Потоки выполнения. Синхронизация – 4 час: 1) многопоточная архитектура; 2) класс Thread; 3) интерфейс Runnable; 4) работа с приоритетами; 5) демон-потоки; 6) модификатор volatile; 7) блокировки; 8) методы wait, notify, notifyAll.

Тема 15. Пакет JAVA.IO – 8 часов: 1) система ввода-вывода, потоки данных; 2) классы InputStream и OutputStream и реализация в классах наследниках; 3) классы FileInputStream и FileOutputStream и их наследники; 4) сериализация объектов; 5) классы Reader и Writer; 6) работа с файловой системой.

Тема 16. Введение в сетевые протоколы – 8 часов: 1) основные понятия модели OSI; 2) утилиты для работы с сетью; 3) классы для работы с TCP-протоколом; 4) классы для работы с UDP.

2.3. Практические и семинарские занятия, их содержание и объем в часах.

Практические и семинарские занятия не предусмотрены.

2.4. Лабораторные занятия, их наименование и объем в часах.

1. Введение в Java приложения: консоль и простые окна – 3 часа.
2. Введение в Java2: апплеты, простые понятия объектно-ориентированного программирования – 2 часа.
3. Введение в Java2: простейшие управляющие структуры – 2 часа.
4. Введение в Java2: простейшие управляющие структуры. Продолжение – 3 часа.
5. Введение в Java2: Методы – 2 часа.
6. Введение в Java2: Массивы – 3 часа.
7. Введение в Java2: Классы – 3 часа.
8. Введение в Java2: Классы (наследование) – 2 часа.
9. Введение в Java2: Классы (полиморфизм) – 2 часа.
10. Введение в Java2: Графика и Java 2D – 2 часа.
11. Введение в Java2: ГИП (графический интерфейс пользователя) – 2 часа.
12. Введение в Java2: Обработка исключений – 3 часа.
13. Введение в Java2: Потоки ввода-вывода – 3 часа.

Выбор темы самостоятельной работы и практического задания оговариваются с каждым студентом отдельно.

2.5. Самостоятельная работа студентов.

1. Апплеты.
2. Пакет JAVA.LANG.
3. Пакет JAVA.UTIL.
4. Основные компоненты JAVA.AWT.
5. Основные понятия модели OSI, утилиты для работы с сетью.

2.6. Вопросы к экзамену.

1. Понятие алгоритма. Языки программирования и их классификация.
2. Интерфейс прикладных программ (API).
3. Язык программирования Java. Свойства языка. Процесс создания программы на Java.
4. Что такое Java? История создания и развития. Платформа Java, JVM: преимущества и недостатки.
5. Методология процедурно-ориентированного программирования. Методология объектно-ориентированного программирования. Объекты и классы.
6. Типы отношений между классами. Достоинства и недостатки ООП.
7. Лексика языка. Кодировка, анализ программы. Виды лексем. Операторы и операции.
8. Типы данных. Переменные. Примитивные типы данных.
9. Ссылочные типы данных. Объекты и правила работы с ними. Класс Object, String, Class.
10. Простые и составные имена. Различие между именами и идентификаторами.
11. Пакеты java как аналог библиотек в других языках программирования.
12. Модуль компиляции.
13. Область видимости имен.
14. Модификаторы доступа.
15. Объявления классов: заголовок, тело класса, объявление полей и методов.
16. Объявление классов: объявление конструкторов, инициализаторы.
17. Дополнительные свойства классов: метод main, параметры методов, перегруженные методы.
18. Преобразование типов. Виды приведений.
19. Преобразование типов. Применение приведений.

20. Объектная модель в Java. Статические элементы. Ключевые слова super, this, abstract.
21. Интерфейсы: объявление, реализация, применение.
22. Объектная модель в Java. Полиморфизм: поля, методы.
23. Массивы – тип данных в Java.
24. Преобразование типов для массивов. Клонирование массивов.
25. Операторы Java: блоки и локальные переменные, метки, оператор if.
26. Операторы Java: switch, управление циклами.
27. Операторы и структура кода: break и continue, именованные блоки, return.
28. Ошибки при работе программы.
29. Обработка исключительных ситуаций (exception).
30. Обработка исключительных ситуаций: оператор throw, обрабатываемые и необрабатываемые исключительные ситуации.
31. Создание пользовательских классов исключений. Переопределение методов и исключения.
32. Пакет java.awt: Апплеты.
33. Пакет java.awt: Базовые классы основные компоненты.
34. Пакет java.awt: Менеджеры компоновки, окна, меню, обработка событий.
35. Базовые классы для работы с потоками.
36. Синхронизация. Блокировки.
37. Потоки: методы wait, notify, notifyAll.
38. Пакет java.lang. Основные классы пакета.
39. Обзор пакет java.util.
40. Потоки данных: пакет java.io. Классы Input/OutputStream и их наследники.
41. Сериализация объектов. Классы Reader, Writer и их наследники. Класс File.
42. Работа в сети. Пакет java.net.

3. УЧЕБНО-МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ ПО ДИСЦИПЛИНЕ

3.1. Перечень обязательной (основной) литературы.

1. *Эккель Брюс*. Философия Java. Библиотека программиста. 3-е изд. – СПб.: Питер, 2003. – 971 с.
2. *Буч Г.* Объектно-ориентированный анализ и проектирование с примерами приложений на C++: Пер. с англ. – 2-е изд. – М.: Бином, СПб.: Невский диалект, 1998. – 560 с.
3. *Иванова Е.Б.*, *Вершинин М.М.* Java 2. Enterprise Edition. Технологии проектирования и разработка. – СПб.: БХВ-Петербург, 2003. – 1088 с.
4. *Хабибуллин И.Ш.* Самоучитель Java 2. – СПб.: БХВ-Петербург, 2005. – 720 с.
5. *Бишоп Д.* Эффективная работа: Java 2. – СПб.: Питер; К.: BHV, 2002. – 592 с.

3.2. Перечень дополнительной литературы.

1. *Флэнаган Д.* Java. Справочник.: Пер. с англ. – СПб.: Символ-Плюс, 2004. – 1040 с.
2. *Бадд Т.* Объектно-ориентированное программирование в действии: Пер. с англ. – СПб.: Питер, 1997. – 464 с.
3. *Х.М. Дейтел*, *П.Дж. Дейтел* и др. Технология программирования на Java 2. В трех книгах. Пер. с англ. – М.: ООО “Бином-Пресс”, 2003.
4. *Блох Дж.* Java. Эффективное программирование: Пер. с англ. – М.: Лори, 2002. – 224 с.

3.3. Перечень наглядных и иных пособий.

1. Карточки с заданиями к лабораторным работам / *А.В. Рыженко*.
2. Документация на языке *HTML* Java2 SDK API / *Sun Microsystems*.

3.4. Средства обеспечения освоения дисциплины.

1. Пакет разработчика для Java 2 фирмы Sun Microsystems.
2. Удобный текстовый редактор типа TextPad или JPadPro.

4. МАТЕРИАЛЬНО-ТЕХНИЧЕСКОЕ ОБЕСПЕЧЕНИЕ ДИСЦИПЛИНЫ

Компьютерный класс кафедры МАиМ.

5. КРИТЕРИИ ОЦЕНКИ ЗНАНИЙ

Зачет проводится в письменной или устной формах в виде тестового задания по лекционным материалам 7 семестра, которое содержит 20 вопросов. Для сдачи зачета не менее чем на половину вопросов должен быть дан верный ответ. Необходимым условием допуска студента к зачету является выполнение и сдача всех лабораторных работ.

Экзамен проводится в письменной или устной формах по билетам, каждый из которых включает два вопроса и практическое задание. В качестве практического задания студенту предлагается тест, содержаний 20 вопросов. Необходимое условие допуска студента на экзамен – выполнение и сдача всех лабораторных работ.

Оценка **"*отлично*"** ставится за полные ответы на вопросы. Студент может допустить единичные несущественные ошибки, самостоятельно им исправляемые. При изложении ответа студент должен самостоятельно выделять существенные признаки изученного, формулировать выводы и обобщения, свободно оперировать фактами, использовать сведения из дополнительных источников. Допускается три неверных ответа в практическом задании.

Оценка **"*хорошо*"** ставится за полные ответы на вопросы. Студент может допустить отдельные несущественные ошибки, исправляемые им после указания на них преподавателем. При изложении ответа студент должен выделять существенные признаки изученного, выявлять причинно-следственные связи, формулировать выводы и обобщения, в которых могут быть отдельные несущественные ошибки. В практической части допускается шесть неверных ответов.

Оценка **"*удовлетворительно*"** ставится за неполные ответы на вопросы. Студент допускает отдельные существенные ошибки, исправляемые им с помощью преподавателя. При изложении ответа студент проявляет затруднения при выделении существенных признаков изученного, при выявлении причинно-следственных связей и формулировке выводов. При выполнении практической части допускается не более половины ошибок от общего количества вопросов.

Оценка **"*неудовлетворительно*"** ставится при неполных бессистемных ответах на вопросы. При этом студентом допускает существенные ошибки, неисправляемые им даже с помощью преподавателя. При изложении ответа студент проявляет полное незнание и непонимание материала. При выполнении практической части допущены более половины ошибок от общего количества вопросов.

II. ГРАФИК САМОСТОЯТЕЛЬНОЙ УЧЕБНОЙ РАБОТЫ СТУДЕНТОВ ПО ДИСЦИПЛИНЕ НА КАЖДЫЙ СЕМЕСТР С УКАЗАНИЕМ ЕЕ СОДЕРЖАНИЯ, ОБЪЕМА В ЧАСАХ, СРОКОВ И ФОРМ КОНТРОЛЯ

График самостоятельной учебной работы студентов по дисциплине на каждый семестр (с указанием ее содержания, объема в часах, сроков и форм контроля) приведен в рабочей программе дисциплины.

III. МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ПО ПРОВЕДЕНИЮ ЛАБОРАТОРНЫХ ЗАНЯТИЙ, ДЕЛОВЫХ ИГР, РАЗБОРУ СИТУАЦИЙ И Т. П. СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ (ОСНОВНОЙ И ДОПОЛНИТЕЛЬНОЙ)

Проведение деловых игр, разбор ситуаций и т. п. рабочей программой дисциплины не предусмотрены.

Форма проведения лабораторного занятия: а) приветствие студентов, 1 мин.; б) определение личного состава студенческой группы, 4 мин.;

в) объявление тематики и вопросов лабораторного занятия, 1 мин.; г) выполнение заданий, 70 мин.; д) подведение итогов лабораторного занятия, 13 мин.; е) прощание со студентами, 1 мин.

Список рекомендуемой литературы (основной и дополнительной) приведен в рабочей программе.

IV. КРАТКИЙ КОНСПЕКТ ЛЕКЦИЙ (ПО КАЖДОЙ ТЕМЕ) ИЛИ ПЛАН-КОНСПЕКТ

7 семестр

1. Введение

1.1 Алгоритм

С момента проявления математики существовало явление, которое начали четко осознавать только в начале XX века. Это явление называется – **алгоритм**.

Понятие. *Алгоритм – это предписание, однозначно задающее процесс преобразования исходной информации в виде последовательности элементарных дискретных шагов, приводящих за конечное число их применений к результату.*

С точки зрения математики это некорректное определение, так как это не строгое формальное определение алгоритма, а интуитивное, ибо в нем используются неточные понятия – предписание, элементарные шаги и т.п. До тех пор, пока не существует точного определения алгоритма, невозможно точно доказать ничего, относящегося к свойствам алгоритмов. Например, у математиков стала возникать мысль, что не для всяких математических задач вообще можно найти процедуру решения, которая бы являлась алгоритмом. А как доказать, что чего-то нет, если это «чего-то» не определено? Так возникла необходимость в определении понятия «**любой алгоритм**», то есть в максимально общем понятии алгоритма. Новое направление исследований получило название «**теория алгоритмов**».

Для решения этой задачи в теории алгоритмов используется идея построения ***универсальных алгоритмических моделей.*** То есть описываются какие-то формальные модели (наборы объектов и правил оперирования этими объектами), позволяющие строить любые алгоритмы в рамках этих моделей. Яркими **примерами** этих моделей являются абстрактные машины Тьюринга, Поста, нормальные алгоритмы Маркова.

Например, машина Тьюринга оперирует над множеством нулей и единиц, используя небольшое количество команд (команды управления пишущей каретки над бесконечной лентой).

Однако предложенные алгоритмические модели формально ничего общего с приведенным выше понятием не имеют с точки зрения строго математического соответствия. Например, нельзя математически доказать, что для любой вычислимой функции (для которой существует алгоритм, который ее вычисляет) существует машина Тьюринга (алгоритм вычисления этой функции, реализованный в рамках алгоритмической модели Тьюринга), хотя прецедентов, отрицающих этот факт не известно. Поэтому тезис «для всякой вычислимой функции существует машина Тьюринга, которая ее вычисляет» декларируется без доказательства и считается справедливым.

В теории алгоритмов в 30-х годах XX века было установлено несколько важных фактов:

- в универсальной алгоритмической модели всегда существует универсальный алгоритм, то есть алгоритм, который способен моделировать работу любого другого алгоритма, описанного в этой модели. Универсальный алгоритм ***U*** устроен следующим образом: имеется метод кодирования ***S*** для любого алгоритма ***A*** в данной модели. Если на вход универсального алгоритма ***U*** подать код ***S (A)*** алгоритма ***A*** и исходные данные ***x***, то результат работы ***U*** будет равен результату работы ***A*** над ***x***: $\forall A\ \exists S(A)\ U(S(A),x) = A(x)$.
  Другими словами, метод кодирования ***S*** это ***язык программирования***, а код ***S (A)*** есть ***программа алгоритма A*** на языке ***S***.
- все универсальные алгоритмические модели можно свести к простейшим алгоритмическим моделям, имеющим минимальное число объектов и операций над ними.

Эти факты в основном определяют устройство современных вычислительных систем.

1.2 Языки программирования

Определение. В самом общем случае **языком программирования** называют фиксированную систему обозначений для описания *алгоритмов* и *структур данных*.

Характерной особенностью языков программирования является посредничество в диалоге между человеком и машиной.

Свойства языка программирования:

1. *Небольшой ограниченный набор команд*. Язык большинства современных ЭВМ крайне беден (см. второй факт из теории алгоритмов!) и состоит из команд типа «выделить память определенного размера», «запомнить информацию в определенном месте памяти», «считать информацию из определенного места памяти» и т.п. Простота и малочисленность команд позволяет легко реализовать этот язык с помощью аппаратных средств.

2. *Функциональная полнота*. Используя команды из набора языка можно описать любой алгоритм.

Запись сложных алгоритмов на языке, имеющем небольшой набор команд, приводит к громоздким программам, в которых с высокой вероятностью могут быть ошибки. Существует два пути решения этой проблемы:

- Усложнение структуры ЭВМ с аппаратной поддержкой мощных языков программирования (Например, 3D ускорители в видеоадаптерах, сетевые компьютеры фирмы Sun).
- Программное моделирование более мощного входного языка. Другими словами создание виртуальной вычислительной машины. Данные виртуальные машины могут располагаться в несколько уровней, самым нижним уровнем является аппаратная часть ЭВМ. Для перевода языка программирования виртуальной машины одного уровня на язык другого уровня создаются специальные программы – переводчики (**трансляторы**). Трансляция может быть многошаговой (зависит от количества уровней вложенности виртуальных машин).

1.3 Классификация языков программирования

Приведем классификацию языков с точки зрения виртуальных машин:

Языки низкого уровня

- Язык **микрокоманд**. Микрокоманды обычно задают простые передачи данных между оперативной памятью и быстрыми регистрами, между самими регистрами и обрабатывающими элементами (например, сумматорами). На основе простейшего набора микрокоманд пишутся специальные микропрограммы, определяющие элементарные операции ЭВМ.
- **Машинный** язык. Язык, команды которого построены из последовательностей микрокоманд. Синтаксически данный язык представляет собой последовательность нулей и единиц, реализуется аппаратурой ЭВМ.
- Язык **ассемблера**. Язык символического кодирования, его операторы те же команды машинного языка, но имеют мнемонические названия, а в качестве операндов команд имеют не конкретные адреса в оперативной памяти, а их символические имена.
- **Макроязыки.** Замена часто встречающихся последовательностей команд на более крупные единицы – макрокоманды. Макропроцессор – программа-переводчик с языка макрокоманд на более низкий язык, то есть замена макрокоманд на последовательности команд более низкого уровня.

Все языки низкого уровня ориентированы на конкретный тип ЭВМ, для которых они созданы.

Языки высокого уровня

Отличие языков высокого уровня от предыдущей группы языков заключается в том, что они ориентированны не на систему команд той или иной ЭВМ, а на систему операторов, характерных для записи определенного класса алгоритмов. Например, операторы присваивания (формулы, значения которых присваиваются определенным переменным); перехода (изменяют естественный порядок выполнения операторов); цикла (обеспечивают многократное повторение группы операторов); условные операторы (поддерживают управление изменением порядка выполнения операторов); операторы ввода-вывода (диалог программы с «внешней средой»), а также операторы описания данных, используемых в программе.

Примерами языков программирования данного уровня являются: Fortran, Algol, Pascal, C++, Java.

Языки сверхвысокого уровня

К языкам сверхвысокого уровня можно отнести лишь АЛГОЛ-68 и APL.

Язык APL (Applied Programming Language) – повышение уровня за счет введения в язык сверхмощных операций и операторов. Например, операция ‘+’ может означать как сложение чисел, так и матриц. Программа очень компактна, но плохо читаема.

Другие классификации языков программирования

Также существует классификация, согласно которой языки программирования делятся на вычислительные и языки символьной обработки.

Предыдущие языки относятся к первому типу. Языками символьной обработки являются LISP (List Processing Language), Prolog (Programming in Logic), РЕФАЛ (Алгоритмический язык рекурсивных функций). Эти языки программирования получили широкое применение в решении задач построения систем искусственного интеллекта.

В заключение, необходимо отметить тенденцию перехода от языков описания алгоритмов решения задач к языкам постановки задач. В языках большую часть начинают занимать средства описания того, не «как» получается решение, а «что» необходимо получить.

2. Интерфейс прикладных программ (API)

В сложной операционной системе типа Windows, допускающей одновременное выполнение нескольких программ в одно и то же время, параллельные процессы организуются посредством виртуальных машин. Вообще, операционная система является сама виртуальной машиной, базирующейся на машине более низкого уровня. В основе каждой виртуальной машины лежит библиотека функций, построенных на командах более низкого уровня. Иерархия библиотек функций выглядит следующим образом: функции BIOS – базовой операционной системы, расположенной в ПЗУ; функции операционной системы; функции прикладной или системной

программы. Часто используемые функции прикладных или системных программ объединяют в библиотеки функций (файлы DLL).

При написании программ (будем называть программы приложениями – **application**) для Microsoft Windows или для Macintosh на языках программирования, подобных языку C++, открытие окна программы и размещение в нем различных объектов требует значительных усилий. Вводный пример темы программирования для Windows в издании “Visual C++ 5.0” содержит более 100 строк. Подобная программа на Java или Windows.net содержит около 30 строк.

Для каждой операционной системы или среды программирования существует документация, содержащая концептуальную и справочную информацию, посвященную специфике программирования под данную систему. Особое место в данной документации занимает описание Application Programming Interface (интерфейса прикладных программ) – совокупности функций, сообщений, структур данных и других объектов, которые открывают доступ к возможностям той или иной системы программирования или операционной системы. При описании, например, функций из API, приводят название функции, список и описания входных параметров, тип возвращаемого функцией значения, специальные атрибуты функции (определение доступа к функции и т.п.).

Пример. Описание функции **insert** объекта **JChoice** в **API Java 2**

public void insert(String item, int index)

Inserts the item into this choice at the specified position.

Вставляет элемент в этот компонент выбора в указанную позицию.

Parameters (Параметры**):**

item - the item to be inserted (вставляемый элемент**)**

index - the position at which the item should be inserted (позиция, в которую вставляется элемент**)**

Throws (Исключительные ситуации**):**

IllegalArgumentException - if index is less than 0 (Если индекс меньше 0**).**

В данном примере описана функция **insert.** Строка «**public void insert(String item, int index)**» называется заголовком функции. Название функции всегда находится непосредственно слева от открывающей круглой скобки. В круглых скобках перечисляются входные параметры функции через запятую**.** Для каждого параметра всегда указывается его тип. Например, параметр **item** имеет тип **String**. Если функция не имеет параметров, то круглые скобки все равно указываются. Например, функция **init()** не имеет параметров. Непосредственно слева от названия функции всегда (кроме специального случая) указывается тип возвращаемого ею

значения. В данном примере функция не возвращает никакого значение, об этом говорит тип void («пусто»). Слева от типа возвращаемого значения следует список модификаторов функции, обозначающих ее специальные свойства. В данном примере – это один модификатор public («общедоступный»), что значит – данную функцию можно вызвать из любого места программы.

При описании функции из API расшифровывается смысл самой функции, её параметров, иногда приводятся примеры использования. Если во время исполнения функции могут возникнуть исключительные ситуации, они также перечисляются.

2.1 Программирование под Windows

Для написания программ, работающих с консолью, то есть выводящих последовательный текст на экран и воспринимающих нажатия клавиш с клавиатуры, достаточно использовать стандартные функции ввода и вывода операционной системы. На языке Pascal функции **write** и **read** вызывают простейшие стандартные операции ввода-вывода операционной системы.

При использовании графической операционной системы простейших команд ввода-вывода не достаточно. Необходимо создать окно приложения, в котором будет организован диалог с пользователем, обеспечить это окно необходимыми стандартными объектами управления (закрыть окно, меню пользователя и т.п.), организовать вывод графических изображений на поверхности окна и отследить сообщения операционной системы о нажатых клавишах, относящихся к данному приложению.

Рассмотрим **Win32 API –**совокупность функций и структур данных 32-разрядной версии операционной системы Windows. Win32 API позволяет разрабатывать приложения, выполняемые на всех 32 разрядных платформах фирмы Microsoft: Windows 3.x, Windows NT, Windows 9x, Windows Me и т.п.

Win32 API можно разбить на следующие категории:

— Graphics Device Interface (GDI)– интерфейс графических устройств;

— Подсистема управления окнами;

— Системные сервисы;

— Мультимедиа;

— Remote procedure call (RPC) – вызовы удаленных процедур.

Интерфейс графических устройств

GDI – предоставляет функции и соответствующие структуры, используемые приложениями для вывода графики на дисплеи, принтеры и другие устройства.

С помощью этих функций рисуются линии, кривые, текст, растровые изображения. Цвет и стиль этих элементов зависят от предварительно создаваемых объектов – перьев (pens), кистей (brushes) и шрифтов (fonts). Соответственно – перья для рисования линий, кисти – для закраски замкнутых фигур, шрифты для вывода текста.

Программы направляют вывод на заданное устройство, создавая для него **контекст**. Контекст устройства (device context) – это структура, управляемая GDI и содержащая информацию об устройстве (его рабочих режимах и подключенных к нему объектах GDI).

Программы могут направлять вывод на **физическое устройство** (дисплей, принтер и т.д.) или на **«логическое»** (память или метафайл). Логические устройства позволяют сохранять выводимые данные в той форме, в какой их легко впоследствии передать на физическое устройство. Сохраненные в метафайле данные можно выводить на физические устройства сколь угодно раз.

Подсистема управления окнами

Эта подсистема позволяет создавать пользовательский интерфейс и управлять им. С помощью ее функций создаются окна, и в них отображается информация. Почти все приложения создают как минимум одно окно.

Приложения определяют общее поведение и внешний вид своих окон, создавая **оконные классы** и **оконные процедуры**. Оконные классы описывают характеристики окон по умолчанию (например, способно ли окно обрабатывать двойной щелчок мышью, есть ли в нем меню и т.д.). Оконная процедура – программа, описывающая взаимодействие с пользователем.

Программы формируют **вывод в окно** с помощью функций **GDI**. Поскольку окна размещаются на одном экране, то они не получают доступ ко всему экрану. Система следит за тем, чтобы данные попадали именно в те окна, для которых они предназначены. Приложения рисуют в окнах в ответ на **системные запросы** или при обработке поступающих **сообщений**. Например, когда размеры или координаты окна изменяются, система обычно передает программе сообщение, требуя дорисовать ранее невидимые участки окна.

Программа принимает ввод с клавиатуры или от мыши в виде сообщений. Система преобразует перемещение мыши или щелчки мыши и нажатие на клавиши на клавиатуре в сообщение и размещает их в **очереди сообщений** приложения. Такая очередь автоматически создается для каждого приложения; оно извлекает сообщения из очереди и пересылает их соответствующим оконным процедурам для обработки.

Приложения часто реагируют на командные сообщения, открывая диалоговые окна и предлагая пользователю ввести дополнительную информацию. **Диалоговое окно (dialog box)** – это временное окно, отображающее или предлагающее ввести какую-либо информацию. Обычно в нем присутствуют элементы управления (controls) – небольшие, специализированные окна, представляющие кнопки и поля, через которые пользователь вводит и выбирает данные. Существуют элементы для ввода и прокрутки текста, выбора из списков и др. Диалоговые окна обрабатывают ввод в эти элементы и передают данные в приложение для выполнения запрошенной команды.

Системный сервис

Системный сервис – это набор функций, предоставляющий программам доступ к ресурсам компьютера и операционной системы (памяти, файловой системе, процессам и т.п.). Например, выделение и освобождение памяти или запуск нескольких потоков команд в рамках одного приложения.

Функции системного сервиса обеспечивают доступ к файлам, каталогам и устройствам ввода-вывода.

Функции системного сервиса открывают доступ к информации о системе и установленных в ней приложениях. Они позволяют определить специфические характеристики компьютера: подключена ли мышь, каково разрешение экрана и т.д.

Приложения могут копировать информацию из одного процесса в другой (Inter-Process Communication, IPC). А в операционных системах, поддерживающих средства защиты данных, соответствующие функции позволяют защищать данные от несанкционированного доступа.

Функции системного сервиса дают программам средства обработки особых ситуаций, возникающих при исполнении (отслеживать ошибки и исключения), и регистрации событий в системе.

Мультимедиа

Мультимедийные функции обеспечивают программам доступ к аудио- и видеоинформации. Они расширяют возможности приложений, позволяя комбинировать эти формы представления информации с традиционными.

Вызовы удаленных процедур

Механизм RPC обеспечивает поддержку распределенных вычислений, позволяя задействовать ресурсы и вычислительную мощь компьютеров в сети. Используя RPC можно создать распределенное приложение: клиентская часть представляет информацию пользователю, серверная часть сохраняет, считывает и обрабатывает данные, а также берет на себя основной объем вычислительных задач клиента. Например, совместно используемые базы данных, удаленные файл-серверы и удаленные серверы печати.

Распределенное приложение, выполняемое как процесс в одном адресном пространстве, обращается к процедурам, исполняемым в адресном пространстве на другом компьютере. Самой программе такие вызовы представляются обычными вызовами локальных процедур, но на самом деле они активизируют процедуры RPC, которые взаимодействуют с адресным пространством на другом компьютере.

Примеры базовых библиотек

Для разработчиков программ под Windows в среде Visual C++ существует библиотека классов Microsoft Foundation Classes, в которой, например содержаться все прототипы окон Windows, элементы управления и т.п.

В Java также предусмотрен пакет программ обеспечивающий базовый набор классов для создания пользовательского интерфейса: Abstract Windowing Toolkit (AWT). В Java 2 появился аналогичный пакет классов Swing, поглотивший AWT – Java Foundation Classes. Одной из его

особенностей является предоставление возможности изменения графического представления пользовательского окна: так называемый интерфейс “look and feel” – представление как в обычном Windows, как в OS фирмы Sun – Solaris, или стиль Metal – платформо-независимое представление окна и элементов управления. Для реализации других специфических функций служат соответствующие пакеты: Util, Net, Events, IO, System и другие.

3. Язык программирования Java

Можно выделить три ключевых элемента, которые объединились в технологии языка Java и сделали ее в корне отличной от всего, существующего на сегодняшний день.

- Java предоставляет для широкого использования свои **апплеты** (applets) – небольшие, надежные, динамичные, не зависящие от платформы активные сетевые приложения, встраиваемые в страницы Web. Апплеты Java могут настраиваться и распространяться потребителям с такой же легкостью, как любые документы HTML.
- Java является объектно-ориентированным языком, сочетая простой и знакомый синтаксис с надежной и удобной в работе средой разработки. Это позволяет широкому кругу программистов быстро создавать новые программы и новые апплеты.
- Java предоставляет программисту богатый набор классов объектов для ясного абстрагирования многих системных функций, используемых при работе с окнами, сетью и для ввода-вывода. Ключевая черта этих классов заключается в том, что они обеспечивают создание независимых от используемой платформы абстракций для широкого спектра системных интерфейсов.

3.1 История создания

Язык Java зародился как часть проекта создания передового программного обеспечения для различных бытовых приборов. Реализация проекта была начата на языке C++, но вскоре возник ряд проблем, наилучшим средством борьбы с которыми было изменение самого инструмента – языка программирования. Стало очевидным, что необходим платформо-независимый язык программирования, позволяющий создавать программы, которые не приходилось бы компилировать отдельно для каждой архитектуры, и можно было бы использовать на различных процессорах под различными операционными системами.

В 1990 году в компании ***Sun Microsystems*** создана команда ведущих разработчиков из шести человек, которая приступила к разработке нового объектно-ориентированного языка программирования, названного ***Oak*** (дуб), в честь дерева, росшего под окном одного из начальников фирмы. Вскоре компания Sun Microsystems преобразовала команду в компанию First Person. Новая компания обладала интереснейшей концепцией, но не могла найти ей подходящего применения. С появлением браузера Mosaic, с которого

началось бурное развитие Internet, было решено использовать Oak в создании Internet-приложений. Так Oak стал самостоятельным продуктом, вскоре был написан Oak-компилятор и Oak-браузер ***WebRunner***. В 1995 году компания Sun Microsystems приняла решение объявить о новом продукте, переименовав его в ***Java*** (вероятно по названию сорта кофе). Когда Java оказалась в руках Internet, стало необходимым запускать Java-аплеты – небольшие программы, загружаемые через Internet. WebRunner был переименован в ***HotJava*** и компания Netscape, одна из ведущих в сфере Internet-технологий, встала на поддержку Java-продуктов.

3.2 Апплеты и приложения

С помощью средств разработки на языке Java фирмы Sun Microsystems можно создавать два типа программ: апплеты и приложения.

Java-апплет – это небольшая программа, динамически загружаемая по сети – точно так же, как картинка, звуковой файл или элемент мультипликации, встроенные в web-страницу. Апплеты исполняются внутри web-браузеров, которые по умолчанию имеют возможность интерпретировать откомпилированные коды апплетов. В распоряжении апплетов имеются все ресурсы и возможности web-браузера. В тоже время апплет по умолчанию не имеет полного доступа к ресурсам локального компьютера, на котором исполняется.

Java-приложение – это полноценная программа, исполняемая на локальном компьютере и имеющая доступ ко всем его системным ресурсам.

3.2 Свойства языка Java

Существует ряд требований к современному языку программирования, которые учитываются при выборе средства создания программ: простота и мощь, объектная ориентированность, надежность, безопасность, архитектурная независимость, возможность работы с параллельными потоками команд, возможность интерпретации, высокая производительность и легкость в изучении.

Объектная ориентированность. Поскольку при разработке языка Java отсутствовала тяжелая наследственность, как это было со многими известными языками при переходе к объектно-ориентированной подходу (например, переход от C к C++, или от Pascal к Pascal with Objects), то для реализации объектов был избран удобный прагматичный подход. Практически всё в Java является объектами, в то же время, ради повышения производительности, числа и другие простые типы данных Java не являются объектами. Таким образом, достаточно познакомиться с основными понятиями объектно-ориентированного программирования, для того, чтобы начать программировать на Java.

Архитектурная независимость. Среда Java – это нечто гораздо большее, чем просто язык программирования. В нее встроен набор ключевых объектов, содержащих основные абстракции реального мира, с которым придется иметь дело программам. Основой популярности Java являются встроенные классы-абстракции, сделавшие его языком, действительно

независимым от платформы, в отличие от библиотек, подобных MFC/COM, OWL, VCL, NeXTStep, Motif и OpenDoc, привязанных к конкретным платформам.

Вопрос о долговечности и переносимости кода решается с помощью виртуальной Java-машины и нескольких жестких требований, которые позволяют, однажды написав, всегда запускать программу в любом месте и в любое время.

Надежность. Среда Java ограничивает свободу в нескольких ключевых областях и таким образом способствует обнаружению ошибок на ранних стадиях разработки программы. В то же время в ней отсутствуют многие источники ошибок, свойственных другим языкам программирования (строгая типизация, например). Большинство используемых сегодня программ «отказывают» в одной из двух ситуаций: при выделении памяти, либо при возникновении исключительных ситуаций. В традиционных средах программирования распределение памяти является задачей программиста, который должен следить за всей используемой в программе памятью, не забывая освобождать ее по мере того, как потребность в ней отпадает. Ошибки происходят, когда захваченная память систематически не освобождается, либо освобождается память, которая все еще используется какой-либо частью программы. Исключительные ситуации в традиционных средах программирования часто возникают в таких, например, случаях, как деление на нуль или попытка открыть несуществующий файл. Среда Java снимает обе эти проблемы, благодаря автоматическому сборщику мусора для освобождения незанятой памяти и встроенным объектно-ориентированным средствам для обработки исключительных ситуаций.

Безопасность. Один из ключевых принципов разработки языка Java заключался в обеспечении защиты от несанкционированного доступа. Программы на Java не могут вызывать глобальные функции и получать доступ к произвольным системным ресурсам, что обеспечивает в Java уровень безопасности, недоступный для других языков. Существует возможность настраивать уровень безопасности при работе с Java приложениями и апплетами. В качестве базовых объектов поставляются средства шифрования высокого класса.

Интерпретация и высокая производительность. Способность Java исполнять свой код на любой из поддерживаемых платформ достигается тем, что ее программы транслируются в некое промежуточное представление, называемое байт-кодом (**byte-code**). Байт-код, в свою очередь, может интерпретироваться в любой системе, для которой существует реализация виртуальной Java-машины. Одним из недостатков интерпретаторов (например, Basic, Perl) обеспечивающих независимость от платформы, является потеря производительности. Для решения этой проблемы в Java существует несколько вариантов оптимизации выполнения программ. Например, использование JIT-компиляторов (**Just In Time compilers**), когда байт-код переводится непосредственно в машинные коды платформы, на

которой исполняется. При этом достигается очень высокая производительность.

3.3 Процесс создания программы на Java

Сравним два способа создания программ: в случае конкретной платформы и с помощью средств Java.

Создавая программы на большинстве языков программирования, приходиться задавать вопрос, в какой операционной системе и на каком процессоре будет работать данная программа? Например, если вы разрабатываете приложение для Windows на языке C++, то можете воспользоваться библиотекой функций Microsoft Foundation Classes, если на платформе Macintosh – функциями из Mac OS Toolbox.

Например, при создании программы под Windows на языке C++ происходит следующее. В каком либо текстовом редакторе создается исходный код программы на языке C++, сохраняется в файл (*.cpp), затем компилируется в объектную программу (*.obj). Объектная программа с помощью редактора связей, объединяясь с необходимыми системными и прикладными библиотеками, превращается в исполняемый файл, который можно запускать только на платформе Windows или совместимой с ней платформе. Кстати выбор платформы возможен на этапе компиляции, но это не гарантирует того, что при написании программы в исходных кодах не были задействованы какие-либо платформо-зависимые функции системы.

Создавая приложения на Java, вы можете не задумываться, в какой операционной системе и с каким процессором будет работать ваша программа. Java включает собственный набор машинно-независимых библиотек, которые называют **пакетами**.

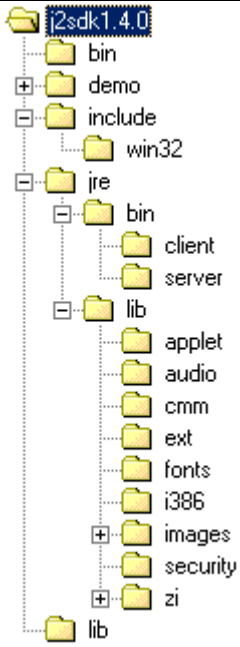
Причина независимости от платформы в том, что компилятор не генерирует непосредственные инструкции процессору. Он создает промежуточный код, или байт-код (bytecodes), для некоторой абстрактной виртуальной машины Java Virtual Machine – JVM. Файл с байт-кодами

называется файлом класса (*.class). Таким образом, такой машины физически не существует, но она программно реализована для большинства известных платформ. Результат работы компилятора Java интерпретируется JVM на каждой конкретной платформе. Следовательно, создав один раз исполняемый код программы, можно запускать эту программу на разных платформах без предварительной перекомпиляции исходных кодов и настройки программы под особенности конкретных платформ.

Виртуальная машина Java является частью набора программ, работающего на компьютере конечного пользователя – **среды выполнения Java (Java Runtime Environment).** Пакет **JRE** включен в состав популярных Internet-браузеров (Microsoft Internet Explorer, Netscape Navigator), что обеспечивает запуск апплетов. Для того чтобы конечный пользователь смог запустить Java-приложение необходимо наличие пакета JRE, который можно получить в составе Java™ 2 Software Development Kit (J2 SDK набор разработчика) или загрузить его отдельно с сервера www.sun.com.

3.4 Структура пакета J2 SDK

Пакеты Java 2 SDK являются бесплатными. Текущая версия пакета может быть получена по адресу http://java.sun.com/j2se/. При инсталляции в каталоге J2 SDK будет построена структура подкаталогов пакета. Кратко рассмотрим эту структуру и выясним содержимое и назначение всех подкаталогов при развертывании J2 SDK на платформе Windows (все аспекты работы с J2 SDK в дальнейшем будем рассматривать на примере J2 SDK для Windows).



bin - каталог инструментария разработчика, в частности содержит компилятор **javac.exe.**
demo - каталог с примерами.
include - используется при совместном использовании Java и C++.
jre - каталог инструментария пользователя (то, что поставляется конечному пользователю при установке готового приложения).
jre\bin - Java-машина(ы) (JVM), в частности файл **java.exe**.
jre\lib - библиотеки Java для конечных пользователей и файлы настройки.
lib - библиотеки Java для разработчиков.

3.5 Компиляция и запуск программ

Для создания программ на Java можно использовать как простейшие средства, предоставляемые пакетом J2 SDK, так и мощные визуальные средства, такие как, например, Sun JavaOne Studio, IBM Visual Age, Borland JBuilder и т.п. Будем рассматривать элементарные средства, то есть пакет J2 SDK и какой-либо простейший текстовый реактор, существенным свойством которого является только наличие в нем поддержки длинных имен файлов.

После инсталляции пакета J2 SDK можно транслировать программу на Java, вызывая компилятор **javac.exe**, или запускать приложения на выполнение виртуальной машиной **java.exe** из командной строки с передачей параметров.

Как и в любой операционной системе, в Windows можно автоматизировать запуск программ с параметрами из командной строки. Это можно осуществлять с помощью **bat**-файлов или **cmd**-файлов. Опишем отдельные bat-файлы для компиляции и выполнения Java-приложений.

Файл для компиляции java программ (**c.bat**)

```
set     JDKBASE=C:\j2sdk1.4.0
set     WORKSPACE=C:\work
%JDKBASE%\bin\javac -classpath %WORKSPACE%  %1 %2 %3 %4 %5
```

Файл для выполнения java программ (**r.bat**)

```
set     JDKBASE=C:\j2sdk1.4.0
set     WORKSPACE=C:\work
%JDKBASE%\bin\java -classpath %WORKSPACE%  %1 %2 %3 %4 %5
```

В данных файлах присваиваются значения двум переменным окружения работы программы. Переменная **JDKBASE** хранит путь к директории с установленным пакетом J2 SDK, переменная **WORKSPACE** путь к рабочей директории, в которой находятся исходные коды программ, и будут располагаться откомпилированные файлы классов. Данные bat-файлы предполагают запуск с параметрами.

Файл с кодом простейшей программы на Java (**Hello.java**)

```
public class Hello
{
        public static void main(String[ ] args)
        {
                System.out.println("Hello World");
        }
}
```

При компиляции данного файла, если не было допущено синтаксических ошибок, на консоль не будет ничего выведено. В противном случае будут выведены сообщения об ошибках.

Приведем два способа компиляции программы из командной строки – с использованием bat-файла и без него. В качестве параметра компилятору всегда передается имя файла с исходным текстом программы **обязательно с указанием расширения .java**.

```
Rem С помощью bat-файла
c Hello.java

Rem Без использования bat-файла
javac.exe Hello.java
```

Также можно привести два способа запуска откомпилированной программы. В качестве параметра виртуальной машине передается имя файла с байт-кодом программы всегда **без указания расширения .class,** то есть только имя файла. В результате работы программы на экран будет выведено предложение «**Hello World**».

```
Rem С помощью bat-файла
r Hello

Rem Без использования bat-файла
java.exe Hello
```

Проанализируем текст программы Hello.java. На данном примере можно продемонстрировать несколько основных принципов программирования на Java.

- Весь программный код в Java заключен внутри описаний **классов** (типов). Вне описания класса (или описания **интерфейса** – особого вида класса) не может быть никакого программного текста (за исключением нескольких специальных директив). Описание класса начинается с заголовка – в данном примере это строка «**public class Hello**», где справа от ключевого слова **class** следует название этого класса, а справа его атрибуты. Ключевое слово **public** имеет тот же смысл, что и в описании функции – данный класс является общедоступным.
- Имя файла без расширения java должно с точностью до регистра символов совпадать с названием класса, описанного в данном файле. То есть каждый файл с именем ***.java** должен содержать класс с именем *. Причем, если у класса есть атрибут **public**, то его описание должно находится в одноименном файле. Файл ***.java** может содержать описания многих классов, но атрибут **public** может иметь только один, имя которого * совпадает с именем файла.
- Внутри класса может находиться конструкция

```
public static void main(String[ ] args)
{
  // тело функции
}
```

Это описание функции **main**, которая является точкой входа в программу. Эта функция вызывается первой при запуске программы. С нее начинаются все процессы в программе. Если мы запускаем на выполнение файл Hello.class, то первым делом в нем ищется код функции main. Если таковой отсутствует, то виртуальная машина выдает сообщение об ошибке и завершает работу. Описание функции main является статическим, и не должно изменятся при описании различных классов.

- Как и любая другая, программа на Java допускает передачу в нее параметров. Это возможно путем передачи параметров в функцию main. Как видно из примера, это массив строк – **String args[]**.

Если запустить программу **Hello.class** с параметрами **abc, def, 123, lab**

r Hello abc def 123 lab

то в функцию **main** будет передан массив **args[4]**, где

args[0]=”abc”;
args[1]=”def”;
args[2]=”123”;
args[3]=”lab”;

4. JAVA – объектно-ориентированный язык программирования

4.1 Принципы объектно-ориентированного программирования

Java является объектно-ориентированным языком программирования. Его основное отличие от распространенных объектно-ориентированных языков в том, что на этих языках возможна реализация нескольких стилей программирования. Например, на языке C++ можно писать программы, не зная принципов объектно-ориентированного подхода. В Java нет средств, позволяющих писать не объектно-ориентированные программы.

Приведем основные аспекты объектно-ориентированного подхода.

1. Все является объектом. Программа содержит описания объектов (классы), но это только описания принципа функционирования объектов, описание их принципиальных свойств. Для того чтобы программа начала работать, необходимо создать в памяти машины экземпляры этих объектов, которые существуют и изменяются в ходе работы программы и уничтожаются при завершении ее выполнения.
2. У каждого объекта есть свойства – другие объекты или элементарные данные (числа, символы и т.п.). Таким образом, экземпляр объекта фактически есть область памяти, хранящая конкретные значения свойств данного объекта.
3. Программа есть совокупность объектов, взаимодействующих друг с другом. Каждый объект обладает рядом характерных для него функций, называемых **методами**. Доступ к свойствам объекта (изменение, присвоение новых значений и т.д.) осуществляется посредством вызова методов данного объекта. Взаимодействие между объектами реализуется посредством вызова их методов друг у друга.
4. Все объекты строго типизированы. То есть на момент создания экземпляра объекта строго известно, какого типа (класса) этот объект, какими принципиальными свойствами и методами он обладает.
5. Возможно наследование объектов, которое позволяет описывать новые классы на базе существующих, добавляя в них новые возможности или переопределяя существующие.

Существует три понятия из теории объектно-ориентированного подхода:

Инкапсуляция – объединение в рамках объекта присущих ему свойств и методов, манипулирующих этими свойствами. Таким образом, объект защищен от неправильного использования его свойств извне.

Наследование – объект может наследовать свойства и методы другого объекта и добавлять к ним черты, характерные только для него.

Полиморфизм – обозначение одинаковыми именами сходных по смыслу методов и свойств, что совсем не обозначает одинаковость их устройства. То есть

унифицированный интерфейс обращения к различным объектам и их методам.

Одной из характерных черт подхода является программирование, управляемое событиями.

4.2 Синтаксис программ на Java.

Все переменные в Java являются **ссылками** на объекты, причем ссылки хранят адреса расположения объектов в памяти машины. На один объект может быть сколько угодно ссылок. Если на объект нет ни одной ссылки, то память, занимаемая им, автоматически освобождается. Ссылка, которая не ссылается ни на один объект, принимает значение **null**. В Java невозможно сделать ссылку на произвольный адрес в памяти и невозможно производить операции со ссылками, равными null.

Будем называть **блоком** текст программы, заключенный между операторными скобками **{** и **}.**

Если ссылка встречается в блоке первый раз, то она должна быть описана. То есть должно быть указано, какого **типа** эта ссылка, на какой **класс** объектов она указывает. При попытке сделать ссылку одного типа на объект другого типа вызовет исключительную ситуацию, которая может быть отслежена компилятором или виртуальной машиной.

```
// Пример описания ссылки str на объект типа String
String str;
```

При описании ссылки можно сразу указать объект, на который она ссылается. Для этого объект должен либо уже существовать, либо его нужно создать. Все объекты создаются явно, с помощью оператора **new**.

```
// Пример описания ссылки str на объект типа String и создание этого объекта
String str=new String(”строка символов”);
// То же самое, если ссылка уже описана
String str=null;
…
str=new String(“строка символов”);
```

Базовой особенностью любого языка является способность описывать новые типы. В Java это классы, то есть класс это способ описания типа объекта.

```
// Пример описания класса Name
class Name
{
  // тело класса или блок описания класса
}
```

В нарушение чистоты объективно-ориентированного подхода в Java есть элементарные или примитивные типы данных это int (целое число), char (символ), float (число с плавающей точкой), double (число с плавающей точкой), byte (один байт) и т.д., которые ссылками на объекты не являются.

Тип	Ключевое слово	Размер в байтах	Описание
Логический	boolean	1	-
Символьный	char	2	Код символа в таблице Unicode
Байтовый	byte	1	Знаковый целый тип (-128 … 127)
Короткий целый	short	2	Знаковый тип (-2^{15} - 2^{15}-1)
Целый	int	4	Знаковый тип (-2^{31} - 2^{31}-1)
Длинный целый	long	8	Знаковый тип (-2^{63} - 2^{63}-1)
Вещественный	float	4	Знаковый тип
Вещественный двойной точности	double	8	Знаковый тип
Пустой	void	-	Используется при описании методов

Внутри тела класса в произвольном порядке описываются свойства и методы класса. Для большей понятности программы желательно придерживаться следующих правил:

- Имена классов принято начинать с большой буквы, а имена свойств и методов с маленькой. Если имя состоит из нескольких слов, то каждое слово начинается с большой буквы, например, NullPointerException.
- Все описания свойств класса размещаются в начале описания класса, затем описание методов.
- Два типа комментариев: все, что начинается с двух символов '/', является комментарием и этот комментарий продолжается до конца данной строки; все, что начинается с символов "/*" является комментарием, который должен быть закрыт символами "*/".
- Для автоматического создания документации к программе перед описанием классов, свойств или методов следует заключать комментарий в следующую конструкцию:

```
/**
 * Прибавление нового экземпляра вложенного тега к содержанию данного тега.
 * @param TagHashCode код вложенного тега, экземпляр которого нужно создать.
 * @return вновь созданный экземпляр вложенного тега.
 */
   public Tag addTag(Tag t)
  {  // тело метода или блок описания метода
  }
```

Свойства класса также называют **полями** класса (**Fields**). Практически это переменные, описанные в блоке описания класса, когда как другие переменные описываются только в блоках описания методов.

```
// Пример описания переменных внутри класса и внутри функций.
// Области видимости переменных.
class Name // заголовок класса
{       int i=8;                    // Переменная i простейшего типа.
        SomeClass s;                // Ссылка на объект класса SomeClass.

        void  function (float  x)   // Заголовок описание метода function.
        {       int y=i;            // Переменная y простейшего типа,
                                    // которой присваивается значение свойства i данного
                                            // класса (y=8).
                AnyClass z;         // Ссылка z на объект класса AnyClass.
                …
        }

        void  other_function ( )    // Заголовок описание метода other_function.
        {       int i=2;            // Локальная для данного метода переменная i,
                                    // параллельно сосуществует со свойством i данного
                                            // класса.
                int y=this.i;       // Переменной y присваивается значение свойства i
                                            // данного класса (y=8).
                y=i;                // Переменной y присваивается значение переменной i
                                    // данного метода (y=2) .
                …
        }

}
```

Время жизни переменной в Java определяется следующим образом. ***Переменная создается в точке ее описания и существует до момента окончания того блока, в котором находится данное описание.***

В отличие от понятия время жизни переменных объект существует до тех пор, пока существует хотя бы одна ссылка на этот объект. Область видимости объекта определяется областью видимости ссылок на этот объект.

```
// Пример времени жизни объекта.
class Name // заголовок класса
{       SomeClass globalReference=null;
        void  function ( )                  // Заголовок описание метода function.
        {  // Локальной переменной присваивается ссылка на новый объект
           //  класса SomeClass
           SomeClass localReference=new SomeClass();
           // Глобальной переменной присваивается ссылка на объект, на который
           //  ссылается локальная переменная
           globalReference= localReference;
        } // при выходе из функции локальная переменная localReference уничтожается, но
          // объект класса SomeClass, созданный в этой функции жив, так как на него
          // все еще ссылается глобальная переменная globalReference
}
```

Все методы класса являются функциями. Правила описания метода рассмотрена в пункте «**Интерфейс прикладных программ**».

```
// Описание метода класса.
<атрибуты> <тип возвращаемого значения><имя
функции>(<аргументы>)
```

```
{    // тело функции
}
```

4.3 Доступ к свойствам класса и вызов методов

Так как объект начинает свое существование с момента создание экземпляра его класса в памяти машины, то доступ к его полям (свойствам) возможен только через него. То есть при обращении к полям класса необходимо всегда указывать имя конкретного его экземпляра.

```
// Доступ к полям класса
class SomeClass  //какой-то класс
{          Object    obj=new Object(); // поле класса
           void function()
           { Object   o=obj;     //доступ к полю класса внутри описания класса
           }
           public static void main(String args[ ])
           { SomeClass  c=new SomeClass(); // создание экземпляра класса
             Object o=c.obj;     // доступ к полю экземпляра SomeClass  класса вне описания
                                                 // класса.
           }
}
class OtherClass     //другой класс
{          void other_function()
           {  SomeClass  c=new SomeClass(); // создание экземпляра класса
             Object o=c.obj;     // доступ к полю экземпляра SomeClass  класса вне описания
                                                 // класса.
           }
}
```

При вызове обычного (не статического) метода класса обязательно должен быть указан объект этого класса и метод вызывается для этого объекта. Т.е. вызов метода – это вызов метода объекта.

```
// Доступ к полям класса
class SomeClass  //какой-то класс
{          void function(int a)
           {          …
           }
           int get()
           {          …
                      int x=5;
                      function(x);  // доступ к методу экземпляра SomeClass  класса внутри
                                                 // описания класса.
           }
           public static void main(String args[ ])
           { SomeClass  c=new SomeClass(); // создание экземпляра класса
             c.function(5);       // доступ к методу экземпляра SomeClass  класса вне описания
                      int y=c.get();         // класса.
           }
}
class OtherClass     //другой класс
{          void other_function()
           {  SomeClass  a=new SomeClass(); // создание экземпляра класса
             a.function(5);       // доступ к методу экземпляра SomeClass  класса вне описания
                      int y=a.get();         // класса.
           }
```

```
}
```

Существует два способа передачи параметров в функцию: по значению или по ссылке. В Java существует всего один тип передачи – передача по значению. Это означает, что при вызове метода ему передается текущее значение параметра. Например, функция ***function(Object ob)***. Внутри ***function*** можно изменить параметр ***ob*** (т.е. присвоить ему ссылку на другой объект, например, ***ob=new Object()***), но это никак не повлияет на объект, изначально переданный в функцию. С другой стороны можно внутри ***function*** менять данные того объекта, на который ссылается ***ob***, и это реально отразится на этом объекте после завершения работы функции.

4.4 Операции и операторы Java

Выше были рассмотрены основные правила оформления программ на языке Java (что характерно любому объектно-ориентированному языку), а именно каждый файл содержит описание класса, в котором определяются его свойства и методы, причем один из методов – main позволяет запускать на исполнение виртуальной машиной данный класс. Это особенность программы на объектно-ориентированном языке. Но любая программа в первую очередь всегда запись алгоритма, для чего в Java предусмотрены традиционные средства.

Простейшие арифметические операции Java (например, +, -, *, /) просты и интуитивно понятны.

```
// Пример арифметического выражения
((a+b)*c)/5;
```

Операция деления одного целого на другое выдает целое, причем не округляет, а отбрасывает дробную часть. В Java имеется операция %, которая обозначает остаток от деления.

Операция присваивания обозначается символом '='. Значение выражения справа от знака '=' присваивается левому операнду. Это значение может быть использовано другими операциями. Последовательность из нескольких операций присваивания выполняется справа налево.

```
// Пример записи операции присваивания
a=b+c;
a=b=c+1;
```

Операции сравнения: > больше, < меньше, >= больше или равно, <= меньше или равно, **!**= не равно, = = равно ли? Следует обратить внимание, что сравнение на равенство обозначается двумя знаками '='. Операндами этих операций могут быть арифметические данные, результат – типа boolean.

Операции инкремента ++, декремента --.

```
// операции инкремента
a++;            // эквивалентна a=a+1;
a--;            // эквивалентна a=a-1;
a=3;            // Пусть a=3;
b=(++a)*2;      // В результате вычисления выражения a=4; b=8;
b=(a++)*2;      // В результате вычисления выражения a=4; b=6;
b=(--a)*2;              // В результате вычисления выражения a=2; b=4;
b=(a--)*2;              // В результате вычисления выражения a=2; b=6;
```

Расширенные операции присваивания

```
// разъяснение расширенных операций присваивания
a+=b;           // эквивалентна a=a+b;
a- =b;          // эквивалентна a=a-b;
a / =b;         // эквивалентна a=a / b;
a * =b;         // эквивалентна a=a * b;
```

Логические операции. Операнды этих операций должны быть типа boolean, результат – boolean, значения этого типа могут быть true (правда) и false (ложь).

```
!       - отрицание
&&      - логическое 'и'
||      - логическое 'или'
// пример логического выражения
boolean         a, b, c, d;             // описание логических
переменных.
d =  ! ( ( a && b) || c );   // логическое выражение.
```

Операции **&&** и || имеют одну особенность – их правый операнд может и не вычисляться, если результат уже известен по левому операнду. Так, если левый операнд операции && – ложь, то правый операнд вычисляться не будет, т.к. результат все равно – ложь.

Условная операция с тремя операндами

```
// синтаксис
//      <переменная>=(<условие>)? <выражение1> :< выражение2>;
// переменная = выражение1, если условие истинно
// переменная = выражение2, если условие ложно
// пример тернарной условной операции
a=(b-c>1)? 0 : 1 ;
```

Оператор – выражение. Такой оператор состоит из одного выражения, в конце стоит символ «;». Его действие состоит в вычислении выражения, значение, вычисленное данным выражением, теряется.

```
// синтаксис
//        <выражение>;
// пример использования
        a=5;
        c++;
        a=(a>c)?a:c;
```

Условный оператор.

```
// синтаксис
//        if ( <условие> )
//                <выражение 1>
//        [else     <выражение 2>]
// пример использования
        if(a==5)
                c++;
        else
                c--;
```

<Условие> – это логическое выражение, т.е. выражение, возвращающее true или false. Как видно из синтаксиса, часть else является необязательной. После if и после else стоит по одному оператору. Если нужно поместить туда несколько операторов, то нужно поставить блок. В Java принято блок ставить всегда, даже если после if или else стоит один оператор.

Оператор цикла по предусловию (while)

```
// синтаксис
//        while ( <условие> )
//                <оператор >
// пример использования
        int i=0;
        while(i<5)
        {       System.out.println( i );
                i++;
        }
```

Оператор цикла по постусловию (do while). Оператор цикла по постусловию отличается от оператора цикла по предусловию только тем, что в нем виток цикла выполняется всегда как минимум один раз, в то время как в операторе по предусловию может не быть ни одного витка цикла (если условие сразу ложно).

```
// синтаксис
//        do
//                <оператор >
//        while ( <условие> );
// пример использования
        int i=0;
       do
        {       System.out.println( i );
                i++;
        }  while(i<5)
```

Оператор цикла For. Заголовок такого цикла содержит три выражения (в простейшем случае). Обычно он служит для организации цикла со счетчиком. Поэтому выражение <инициализация> выполняется один раз перед первым витком цикла. После каждого витка цикла выполняется выражение <оператор 1>, а потом выражение <условие>. Последнее выражение должно быть логическим и служит для задания условия продолжения цикла – выполнения выражения <оператор 2>. Пока <условие> истинно, витки цикла будут продолжаться.

```
// синтаксис
//        for ([<инициализация>], [<условие>], [ <оператор 1 >] )
//                                                [<оператор 2>]
// пример использования
        for (int i=0; i<5;i++)
        {       System.out.println( i );
        }
// пример бесконечного цикла
        for (;;);
```

Для данного оператора цикла существует множество расширений:

- <инициализация> может быть не выражением, а описанием с инициализацией типа «int i = 0».
- <инициализация> может быть списком выражений через запятую, например, «i = 0, r = 1».
- <оператор 1> тоже может быть списком выражений, например, «i++, r+=2».
- Все составляющие (<инициализация>, <условие> и <оператор 1>) являются необязательными. Для выражения <условие> это означает, что условие считается всегда истинным (т.е. выход из цикла должен быть организован какими-то средствами внутри самого цикла).

Оператор выбора. Служит для организации выбора по некоторому значению одной из нескольких ветвей выполнения.

```
// синтаксис
//        switch ( <выражение> )
//        {       case <константа 1>:  <операторы 1>
//                case <константа 2>:  <операторы 2>
//                . . .
//                [default: <операторы default >]
//        }
// пример использования
char c;
switch (c)
{       case ‘a’: System.out.println(“Символ а”);
        case ‘b’: System.out.println(“Символ бэ”);
        case ‘c’: System.out.println(“Символ цэ”);
        default : System.out.println(“Какой-то ещё символ ”);
}
```

В данном операторе <выражение> должно выдавать целочисленное или символьное значение, константы должны быть того же типа, что и значение этого выражения. Элементы « case <константа>: » являются метками перехода, если значение выражения совпадает с константой, то будет осуществлен переход на эту метку. Если значение выражения не совпадает ни с одной из констант, то все зависит от наличия фрагмента <default>. Если он есть, то переход происходит на метку « case <default>: », если его нет, то весь оператор switch пропускается.

В операторе switch фрагменты case не образуют какие-либо блоки. Если после последнего оператора данного case-фрагмента стоит следующий case, то выполнение будет продолжено, начиная с первого оператора следующего case-фрагмента. В силу этого в операторе switch обычно применяется оператор **break**. Он ставится в конце каждого case-фрагмента.

Операторы break, continue и return. В Java нет операторов **goto**, но есть несколько операторов передачи управления. Операторы break и continue могут применяться в циклах, а break еще и в операторе выбора (switch). Выполнение оператора break приводит к немедленному завершению цикла (или оператора switch). Оператор continue вызывает окончание текущего витка цикла и начало нового.

При вложенных циклах операторы break и continue могут относиться не только к тому циклу, в котором они расположены, но и к охватывающему циклу. Для этого охватывающий оператор цикла должен быть помечен меткой, которая должна быть указана в операторе break или continue.

```
// пример вложенных циклов
label: while ( ... )
{
        for(…;…;…)
        {
```

```
                    if( …)  break label; // прерывание не текущего цикла for, а
цикла while
          }
}
```

Для возврата значения из метода класса используется оператор return. Тип возвращаемого значения указывается в заголовке метода перед его именем, тип выражения в операторе return должен соответствовать типу возвращаемого значения.

```
// пример возврата значения из метода
void function(int a)
{         int x=5;
          return a+x;       // возвращается сумма входного параметра и
переменной x
}
```

Оператор return является более мощным средством выхода из операторов цикла и выбора, так как при вызове этого оператора прекращает работать метод, в котором они описаны.

8 семестр

4.5 Структуры данных Java

Массивы. В Java есть как одномерные, так и многомерные массивы. Но реализация массивов в Java имеет свои особенности. Во-первых, массив в Java это объект, одним из свойств которого является размера массива (поле **length**).

```
// пример описания массивов базовых типов
int       array [ ];          // создание ссылки на объект массив типа int
int [ ]   array;   // создание ссылки на объект массив типа int
```

В приведенном выше примере создаются только ссылки на объекты типа массив, сами массивы не создаются, так как не задан их размер. Так как массивы объекты, то создаются они как и любые объекты при помощи оператора **new**.

```
// пример создания массивов базовых типов
int       array [ ];                    // Создание ссылки на объект
массив типа int.
array = new int[25];                    // Создание объекта массив типа int
из 25 элементов.
```

```
//        или
int       array [ ] = new int[25];  // Создание объекта массив типа int из 25
элементов.
//        или
int       array [ ] = {1,2,3,4,5};  // Создание объекта массив типа int
                                    // и инициализация списком элементов.
```

Элементы массива нумеруются с нуля. Java жестко контролирует выход за пределы массива. При попытке обратиться к несуществующему элементу массива возникает исключительная ситуация **IndexOutOfBoundException**. При создании массивов простейших типов все элементы инициализируются автоматически, например если это целочисленный массив – все элементы равны 0. С массивами объектов другая ситуация. Элементы объекты необходимо создавать самостоятельно.

```
// пример обращения к элементам массивов
int       array [ ] = {1,2,3,4,5};  // Создание объекта массив типа int
                                    // и инициализация списком элементов.
array [ 0 ] = 6;                    // Присваивание значения элементу
массива.
int  x = array [ 2 ] * 6;           // Использование элемента массива в
выражении.
// пример обращения к свойству массива length
int       array [ ]          = {1,2,3,4,5};
int       sum      =0;
for ( int i=0; i< array.length ; sum+= array [ i++] ); // Вычисление суммы
всех элементов массива.
System.out.println("Сумма всех элементов массива равна " + sum );
```

Массивы объектов. Массив объектов (уже не массив элементов простейших типов) – это массив ссылок на объекты. Соответственно, нужно создать как массив, так и сами объекты.

```
// пример создания массива объектов
SomeClass       array [ ];              // Создание ссылки на массив
объектов типа SomeClass.
                                        // В данном случае самого массива еще нет.
array = new SomeClass [25];             // Создание массива объектов типа
SomeClass из 25                                 // элементов. Массив
уже есть, но все элементы этого                                 //
массива равны null, то есть не ссылаются ни на один
                                // объект.
for (int j = 0; j < array.length; j++ )
     array [ j ] = new SomeClass ( );   // Присвоение каждому элементу
массива ссылки на                               // новый объект
```

```
типа SomeClass. Таким образом имеем
        // полноценный массив.
// Альтернативный способ создания массива объектов через список инициализации
SomeClass       array [ ] = {       new SomeClass();
                                    new SomeClass();
                                    new SomeClass();
                                    new SomeClass();                };
```

Многомерные массивы. Многомерные массивы в Java строятся по принципу "массив массивов". Одномерный массив является объектом, двумерный массив – это массив ссылок на объекты-массивы, трехмерный массив – это массив ссылок на массивы, которые, в свою очередь, являются массивами ссылок на массивы.

```
// пример создания многомерного массива
int ary [ ] [ ]     = new int[3][3]; // Создание двумерного массива целых чисел.
int ary [ ] [ ]     = new int[][] =   {                     // Создание двумерного
                                      { 1 , 2 , 3 }             // массива целых чисел
                                      { 1 , 2 , 3 , 5}      // со списком инициализации
                                      { 1 , 2 }             // с разным количеством
                                  };                        // элементов.
AnyClass  array[ ] [ ]= new  AnyClass[2][3] ;     // Создание двумерного массива объектов типа                                   //AnyClass.
for (int i = 0; i < array.length; i++ )
   for (int j = 0; j < array[i].length; j++ )
     array [ i ] [ j ] = new AnyClass ( );    // Присвоение каждому элементу массива ссылки на                                   // новый объект
типа AnyClass. Таким образом имеем
        // полноценный массив.
```

В приведенных выше примерах имя массива – это переменная-ссылка, содержащая адрес объекта массива. Поэтому присваивание таких переменных друг другу – это не копирование массивов, а копирование ссылок на объекты. В состав стандартной библиотеки Java входят разнообразные средства работы с массивами. В пакете **java.util** имеется класс **Arrays**, который обеспечивает множество полезных операций над массивами.

4.6 Конструкторы классов

Класс является основной структурой данных, он описывает принципиальные свойства и методы объекта. Объект начинает существовать, когда в памяти компьютера создается экземпляр класса этого объекта. Это происходит обычно при вызове оператора **new**.

Конструктор класса – это специальный метод класса, который вызывается при создании объекта класса. Конструкторы (в своем описании) отличаются от других методов класса тем, что их имя совпадает с именем класса. Кроме того, при описании любого метода класса, кроме конструктора, обязательно указывается тип возвращаемого значения, а если метод не возвращает никакого значение, то явно указывается тип **void**. При описании конструктора тип возвращаемого значения вообще не указывается.

При создании объекта Java машина осуществляет следующие действия: организуется поиск class-файла в памяти машины и в доступных каталогах и библиотеках; выделяется память для содержания экземпляра класса; выполняется инициализация полей экземпляра класса; запускается на выполнение конструктор класса. Кстати в формате вызова оператора new присутствует вызов конструктора класса.

```
TheClass  exampler= new  TheClass();   // Создание экземпляра объекта,
причем  TheClass(); -                  // есть вызов
конструктора класса.
```

Конструктор, как и любой другой метод, может иметь параметры. Конструктор без параметров называется конструктором по умолчанию. В классе может быть несколько конструкторов. В этом случае они должны иметь разные наборы параметров. Если в классе нет ни одного конструктора, то генерируется пустой конструктор по умолчанию. Если в классе есть хотя бы один конструктор, то конструктор по умолчанию не генерируется.

```
public class TheClass
{    public TheClass()          // Конструктор без параметров
     {       this(0,"");        // Вызов одного конструктора из другого.
             …
     }
    public TheClass(int a, String str)  // Ещё один конструктор с параметрами
     {        …
     }
    public static void main(String args[]) // Статический метод – точка вхда в программу.
     { TheClass  exampler1= new  TheClass();          // Создание экземпляра объекта
                                                      // с конструктором без
параметров.
      TheClass          exampler2= new  TheClass(5,"строка");  // Создание экземпляра объекта
                                                               // с конструктором  с
параметрами.
     }
}
```

4.7 Работа со строками

Рассмотрим пример класса, занимающего важное место в языке Java. Это класс **String**, он определен в стандартной библиотеке Java и используется для работы со строками.

4.8 Библиотеки и пакеты

Пакет Java – это совокупность классов, объединенных по какому-либо принципу. Множество пакетов образуют библиотеку классов Java – основу Java API. Если при написании программы используется какой-то класс, то нужно подключить пакет, в котором этот класс находится.

Пакеты различаются по составным именам, где слова разделены точками. Это связано с общепринятым в Java принципом построения имен пакетов: в имени пакета присутствует Internet-адрес разработчика пакета в обратном порядке. Например, пакет ***org.w3c.dom*** связан с адресом разработчика ***www.dom.w3c.org***. Кроме того, с именем пакета связана структура каталогов, в которых должны размещаться классы при создании пакета.

Некоторые важные пакеты Java представлены в таблице

java.lang	Содержит фундаментальные классы языка Java.
java.io	Содержит классы для ввода и вывода данных через потоки, файловую систему и сериализацию.
java.util	Содержит набор классов коллекций, модели событий, классов работы календарем и временем, средства интернационализации программ (разборщик строк, генератор случайных чисел, битовые массивы).
java.awt	Содержит все классы для создания пользовательского интерфейса и отображения графики.
java.awt.event	Содержит интерфейсы и классы для работы с различного рода событиями, происходящими с объектами из пакета java.awt.
javax.swing	Содержит набор «легковесных» компонентов (то есть классов, написанных чисто на языке Java), которые в максимально возможной мере работают одинаково на всех платформах. Частично базируется на пакете java.awt.
java.applet	Содержит классы, необходимые для создания апплетов и классы, которые апплет использует для связи со средой браузера.

Для того чтобы класс из библиотеки мог быть использован в программе, можно подключить пакет этого класса, используя директиву **import**. Пакет **java.lang** подключается автоматически.

```
// Подключение пакета java.util
```

```
import          java.util.*;
class SomeClass        //какой-то класс
{    void function() // какая-то функция
     {  Vector v= new Vector();        // Использование класса Vector из подключенного
                                       // пакета        java.util.
        java.awt.Frame frame=new java.awt.Frame(); // Использование класса Frame
                                                   // путем прямого указания пакета java.awt,
     }                                             // в котором этот класс находится.
}
```

Для создания собственных пакетов необходимо в файлах с классами указывать их принадлежность к пакету с помощью директивы **package**, причем имя пакета должно повторять путь к файлу с классом.

Структура каталогов


```
// класс TableEditor описан в файле IskLabs/awt/TableEditor.java
package  IskLabs.awt;

import    IskLabs.structures.*;
import    java.awt.*;

public class TableEditor
{         …
}
```

4.9 Наследование классов

Наследование классов является одним из атрибутов объектно-ориентированного подхода и позволяет строить новые классы на базе существующих, добавляя в них новые возможности или переопределяя существующие. Смысл наследования классов заключается в следующем. Пусть имеется класс X, в котором определены свойства $x_1, x_2, ..., x_n$ и методы $func_1(), func_2(), ..., func_m()$. Тогда можно создать класс Y, который является наследником класса X. Это значит, что все свойства $x_1, x_2, ..., x_n$ и методы $func_1(), func_2(), ..., func_m()$ класса X (кроме конструкторов) являются свойствами и методами класса Y. Кроме того, класс Y может иметь дополнительные свойства и методы, а также возможно переопределить любые из методов $func_1(), func_2(), ..., func_m()$, унаследованные у класса X.

Класс X называют базовым классом, суперклассом или родительским классом, предком. Класс Y называют порожденным, дочерним, подклассом, классом-потомком. От одного класса может быть порождено произвольное

количество новых классов. В результате получается иерархия классов, порожденных один от другого.

```
// Синтаксис наследования классов
class Y extends X       // класс Y наследуется от класса X
{
        // здесь записываются описания дополнительных свойств и методов класса Y
        // или переопределения методов класса X
}
// Пример наследования классов
class First
{       int a,b,c;
        void func()
        {       …
        }
        int function()
        {
        }
}
class Second extends First
{       int x,y,z;                      // новые свойства класса Second
        void procedure()                // новый метод класса Second
        {       …
        }
        int function()          // переопределение метода класса First
        {
        }
        // также в этом классе доступны свойства класса First      a,b,c.
        // также в этом классе доступен метод класса First     func().
        // метод класса First function() переопределен и доступен в измененном виде.
}
```

Переопределение методов возможно только при полном совпадении заголовков методов в обоих классах, участвующих в наследовании.

В Java все классы строятся на базе наследования. По умолчанию все классы неявно наследуются от класса **Object**. То есть этот класс является базовым классом для всех классов Java. Класс **Object** имеет ряд методов, при наследовании методы базового класса наследуются его потомками. Следовательно, все классы Java имеют как минимум те методы, которые есть в классе **Object.**

Некоторые методы класса Object	Описание методов

public **Object()**	Конструктор класса.
public boolean **equals(**Object obj**)**	Определяет факт равенства объектов.
public String **toString()**	Формирование некоторого текстового представления объекта.

Рассмотрим метод **equals**. Его назначение – сравнивать объекты на равенство. Наличие его в классе **Object** (базовом для всех остальных классов) позволяет нам применять этот метод для любых объектов, что очень удобно. Но в классе **Object** он реализован как сравнение адресов объектов. Т.е. при сравнении двух объектов мы получим равенство только в том случае, если на самом деле это один и тот же объект. Естественно, что чаще всего требуется другая реализация данного метода, основанная на сравнении полей объектов.

Как происходит инициализация полей (присваивание начальных значений) при наследовании классов. Все действия по инициализации выполняются этап за этапом в порядке наследования классов. Вот порядок этих действий.

- При первом обращении к классу выделяется память под статические поля класса и выполняется их инициализация.
- Выполняется распределение памяти под создаваемый объект.
- Выполняются все инициализаторы нестатических полей класса.
- Выполняется вызов конструктора класса.

Ключевое слово **super** может использоваться и для явного вызова методов базового класса. Это необходимо, если некоторый метод базового класса был переопределен в порожденном классе.

```
// пример описания абстрактного класса
abstract class SomeClass        // класс абстрактный
{       public abstract function();       //абстрактный метод

        public void func()                // обычный метод
        {       …
        }
}
```

Модификаторы доступа к элементам (классам, методам и свойствам) при наследовании имеют следующий смысл.

- **public** означает, что данный элемент доступен без каких-либо ограничений;
- **private** означает, что доступ разрешен только из данного класса;
- **protected** означает, что доступ разрешен из данного класса и из всех классов-потомков;
- при отсутствии модификатора доступ разрешен из всех классов данного пакета.

В Java есть ключевое слово **final**, используемое как атрибут полей, переменных, параметров и методов. В применении к полям, переменным и параметрам оно означает, что их значение не может быть изменено. Поле или переменная с атрибутом **final** должны получить значение при описании, параметр просто не может быть изменен внутри тела метода. Атрибут **final** в сочетании с атрибутом **static** позволяют создать константы, т.е. поля, неизменные во всей программе. Если нужно запретить переопределение метода во всех порожденных классах, то этот метод можно описать как **final.** Данное ключевое слово может применяться и к классам. Это означает, что данный класс не может быть унаследован другим классом.

4.10 Абстрактные классы и интерфейсы

Класс называется абстрактным, если у него отсутствует описание хотя бы одного метода при наличии его заголовка. В этом случае в описании класса перед словом **class** должен стоять атрибут **abstact** и при описании нереализованных методов тоже должен использоваться этот атрибут.

```
// пример описания абстрактного класса
abstract class SomeClass        // класс абстрактный
{       public abstract function();        //абстрактный метод

        public void func()                 // обычный метод
        {       …
        }
}
```

Как видно из примера, тело абстрактного метода отсутствует, сразу после заголовка метода стоит точка с запятой. Абстрактный класс не может использоваться непосредственно для порождения объектов. Для этого необходимо, используя этот класс как базовый, наследовать другой класс, в котором нужно определить все абстрактные методы.

Интерфейс – это полностью абстрактный класс, не содержащий никаких полей, кроме констант – полей, имеющих атрибуты **static final**. Говорят **класс** ***наследует*** **другой класс**, и **класс** ***удовлетворяет*** **интерфейсу**, **класс** ***реализует, выполняет*** **интерфейс.**

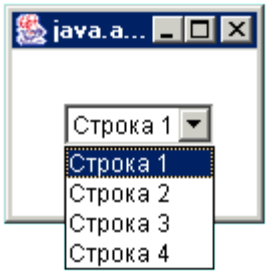
```
// пример описания интерфейса
interface SomeInterface         // заголовок интерфейса
{       public static final int field=100; // константа
        public void function(int x, String y);      //абстрактный метод
        public void func();                         //другой абстрактный метод
}
// пример описания класса, реализующего интерфейс SomeInterface
class anyclass implements SomeInterface
{
```

```
        public void function(int x, String y)       //переопределение
абстрактного метода
        {       …
        }
        public void func()                  // переопределение другого
абстрактного метода
        {       …
        }
}
```

Если класс реализует интерфейс, то внутри описания этого класса, должны быть реализованы все методы (то есть, описаны тела этих методов), описанные в этом интерфейсе.

Интерфейсы широко используются при написании различных стандартов. Стандарт определяет, в частности, набор каких-то интерфейсов. И, кроме того, он содержит описание семантики этих интерфейсов. После этого, прикладные разработчики могут писать программы, используя интерфейсы стандарта. А фирмы-разработчики могут разрабатывать конкретные реализации этих стандартов. При внедрении прикладного программного обеспечения можно взять продукт любой фирмы-разработчика, реализующий данный стандарт.

Рассмотрим пример использования интерфейсов при взаимодействии объектов из пакета java.awt.



Пусть имеются объект ***frame*** – окно программы и стандартный объект ***choice***, в котором происходит выбор элемента из списка путем манипулирования мышью или кнопками «вверх», «вниз» и «ввод». Эти объекты являются частью графического пользовательского интерфейса программы. Объект ***choice*** можно располагать на поверхности окна программы, специальным образом прикрепив его к окну ***frame.add(choice)***. Все вопросы, связанные с выбором элемента из списка являются внутренним делом объекта ***choice***, в то же время этот объект потенциально может быть прикреплен для отображения к любому другому графическому объекту – окну, панели, фрейму и т.д. Каким образом можно получить информацию о выбранном элементе из списка? Для этого в классе ***Choice*** предусмотрена специальная функция, отвечающая за подключение объектов, заинтересованных в получении информации о результатах выборов в объекте ***choice,*** которая называется ***addItemListener(ItemListener l)***. В качестве

аргумента в этой функции используется интерфейс слушателя событий от объекта выбора.

```
// пример описания интерфейса
public interface ItemListener
{ public void itemStateChanged(ItemEvent e)
}
```

Если существует объект (например, ***frame***), заинтересованный в получении информации от ***choice***, то этот объект должен поддерживать интерфейс ***ItemListener***, то есть он должен обязательно иметь и **реализовывать** функцию ***itemStateChanged (ItemEvent e),*** а также необходимо вызывать функцию объекта ***choice*** – ***choice.addItemListener(frame)***.

```
// пример описания интерфейса
class frame extends Frame implements ItemListener
{       Choice choice=new Choice();
            …
            public frame()
            {       …
                    add(choice);
                    choice.addItemListener(this);
            }
            …
            public void itemStateChanged(ItemEvent e)
            {
                    //реакция на выбор объекта в объекте choice
            }
            …
}
```

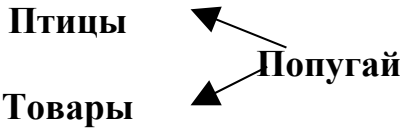
Таким образом, нет необходимости классу ***Choice*** знать о всех возможностях своих слушателей, ему необходимо только, чтобы у них была одна единственная функция ***itemStateChanged,*** которую он будет вызывать при каждом новом выборе элемента из списка. Поэтому в качестве аргумента функции ***addItemListener*** выступает интерфейс, описывающий формат вызова этой функции.

4.11 Множественное наследование

Объектно-ориентированное программирование является одним из ***«способов представления объективной реальности в виде некоторого набора классов»***. Между классами может существовать отношения различных типов (использует, имеет, создает). Особо следует отметить важнейший тип отношений – **«является»**. Обычно для представления этого отношения достаточно единственной иерархической системы наследования классов:

Животные ← Птицы ← Попугай

Однако в некоторых случаях могут встречаться классы, которые логически принадлежат сразу нескольким различным иерархическим структурам:



В некоторых языках (например, C++) существует механизмы реализации такого множественного наследования. В Java этот механизм исключен, ибо порождает множество скрытых ошибок. Решение проблемы состоит в использовании **интерфейсов**.

В некоторых объектно-ориентированных языках для класса разрешается указывать определенный набор соглашений о поведении. Эти соглашения называют **протоколами**. Это понятие протокола эквивалентно понятию интерфейса в Java.

```
// пример класса Попугай, выступающего и как объект Птица, и как
объект Товар
public class Птица
   {  Крыло   левое, правое;

      public лететь();
      public приземлиться(Насест какой_то_насест);
      public чирикать();
}
public interface Продукт
   {
      public void установитьЦену(float ЦенаТовара);     //определение
цены на товар
      public void продать();                  //продажа одной единицы
товара
      public void купить(short Количество,float ЦенаЗаВсё); //покупка
некторого количества
      //товара с общей ценой
   }
public class Попугай extends Птица implements  Продукт
   {  private final float ЦенаКормаВдень=0.60;
      static short Цена;
      static float  СчетВбанке;
      short         количствоНаСкладе;

      public void установитьЦену(float ЦенаТовара)
      { Цена= ЦенаТовара;
      };
      public void продать()
      {       количствоНаСкладе --;
              СчетВбанке += Цена;
```

```
        };
        public void купить(short Количество,float ЦенаЗаВсё)
        {       количствоНаСкладе += Количество;
                СчетВбанке - = ЦенаЗаВсё;
        };
        public void кормить()
        { СчетВбанке - = количствоНаСкладе * ЦенаКормаВдень;
        //покупка ей еды
          super.есть(); // собственно птица ест!
        };
}
```

4.12 Исключительные ситуации

Недостатки прежних методов обработки ошибок

Высшая школа систематически отстает от быстро развивающейся индустрии разработки программного обеспечения и, следовательно, обучает студентов устаревшим методам и приемам работы. Одним из недостатков преподавания методов программирования – исключение из практики контроля и обработки ошибочных ситуаций в программах.

Утверждается, что создание **готового программного продукта** требует в три раза больших затрат времени, чем простое написание программ. Под программой здесь понимается некоторая законченная последовательность команд, которая используется только ее автором. Программный продукт предназначен для посторонних лиц, не являющихся его непосредственными разработчиками.

Пример программы

Запишем программу в псевдокодах, которая предназначена для открытия текстового файла, который находится на жестком диске, и для отображения его содержимого на экране монитора.

```
Получить параметр командной строки, являющейся именем требуемого
файла.
Открыть файл в режиме чтения и получить адрес его блока управления.
Пока(файл содержит еще данные)
{       считать очередную строку из файла
        вывести строку на экран
}
закрыть файл
```

Виды вероятных ошибок, которые могут возникнуть в этой программе:

1. Пользователь не задал значение для параметра командной строки.
2. Файл с указанным пользователем именем не существует.
3. В системе открыто слишком много файлов (системная таблица файлов переполнена).

4. Данный процесс превысил количество файлов, которое операционная система позволяет ему открыть.
5. Указанный файл существует, но недоступен для чтения.
6. При чтении файла оказалось, что он содержит двоичные, а не текстовые данные
7. При чтении файла оказалось, что он пуст.
8. При чтении файла оказалась ошибка ввода-вывода.

Для защиты программы от перечисленных ошибок традиционно объединяют две логические схемы: логику работы программы и логику обработки ошибок.

```
Получить параметр командной строки, являющейся именем требуемого
файла.
        Проверить, не является ли параметр пустым
        Если пуст, вывести сообщение пользователю и выйти
Открыть файл в режиме чтения и получить адрес его блока управления.
        Если таблица файлов переполнена,
        {       ожидать заданное число миллисекунд
                сделать заданное число повторных попыток
                если доступ к файлу осуществить не удалось,
                        вывести сообщение пользователю и выйти
        }
        Если данный процесс открыл слишком много файлов,
                        вывести сообщение пользователю и выйти
        Если файл не существует,
                        вывести сообщение пользователю и выйти
        Если не удается получить доступ к файлу по чтению,
                        вывести сообщение пользователю и выйти
Считать из файла первую строку данных
        Если данные не являются текстом,
                        вывести сообщение пользователю и выйти
        В случае ошибки ввода-вывода
                        вывести сообщение пользователю и выйти
        Если данные в файле отсутствуют,
                        вывести сообщение пользователю и выйти

Пока(файл содержит еще данные)
{       вывести строку на экран
        считать очередную строку из файла
                        Если данные не являются текстом,
                                вывести сообщение пользователю и выйти
                        В случае ошибки ввода-вывода
                                вывести сообщение пользователю и выйти
}
закрыть файл
```

Обработка исключительных ситуаций

Помимо описанного метода обработки ошибок существует еще один, более эффективный метод: метод **обработки исключительных ситуаций.**

Определение. ***Исключительная ситуация*** **–** *это событие, которое происходит в процессе выполнения программы и нарушает нормальное следование потока выполняемых команд.*

Нормальный поток команд – это первый пример программы в псевдокодах из шести строк.

Для перехвата исключений используется синтаксическая конструкция, называемая ***try-catch***-блок.

```
try
        {       //нормальный поток команд
        }
catch (Имя_класса_исключения ex)
        {       // реакция на ситуацию «Имя_класса_исключения»
        }
catch(Exception ex)
        {       // реакция на все остальные ситуации
        }
```

Для организации обработки исключительных ситуаций в программе необходимо выполнить следующие шаги:

1. Поместить нормальный поток команд в блок **try**
2. Отметить, какие ошибки могут возникнуть в процессе выполнения нормальной последовательности команд (в каждом случае должна генерироваться исключительная ситуация).
3. Организация перехвата и обработки всех типов исключительных ситуаций в блоках **catch.**

Существуют два варианта генерации исключительной ситуации – автоматическая генерация (примеры – **NullPointerException**, **ArithmeticException**, **ClassCastException**) и явная программная генерация.

Автоматическая генерация. Если Java-машина обнаруживает некоторую ошибку, например, деление на ноль или ошибку приведения типов, то она сама генерирует соответствующее исключение.

Программная генерация. Сгенерировать исключение можно явно при помощи операции **throw**.

```
throw new IllegalArgumentException("Параметр k должен быть положительным числом");
```

Рассмотрим каким образом обрабатываются ошибки в приведенном выше примере программы чтения файла.

```
// Пример обработки ошибок
try
```

```
{   Получить параметр командной строки, являющейся именем требуемого файла.
        Проверить, не является ли параметр пустым
            Если пуст, генерировать исключительную ситуацию «параметр пуст»
    try
    {
        Открыть файл в режиме чтения и получить адрес его блока управления.
    }
    catch (Исключительная ситуация переполнения таблицы файлов)
    {   ожидать заданное число миллисекунд
        сделать заданное число повторных попыток
        если доступ к файлу осуществить не удалось,
            вывести сообщение пользователю о перегруженности и выйти
    }

    Считать из файла первую строку данных (для определения их наличия)
    Пока(файл содержит еще данные)
    {   вывести строку на экран
        считать очередную строку из файла
    }
    закрыть файл
}
catch (любые другие исключительные ситуации)
{
    вывести пользователю имя возникшей ситуации и выйти
}
```

Пример этой же программы на языке Java.

```
public class FileReader1
{   public static void main(String args[])
    {   try
        {
            if(args.length==0)
            throw (new Exception("введите:FileReader1 имя_файла" ));

            java.io.BufferReader theStream =
                new java.io.BufferReader(
                    new java.io.FileReader(args[0]));
```

```
            String theInputString = new String();
            While((theInputString=theStream.readLine())!=null)
                  {
                        System.out.println(theInpuString);
                  }
            theStream.close();
            }
            catch(Exception e)
            {
                        System.err.println(e.getMessage());
                        e.printStackTrace();
            }
      }
}
```

4.13 Многопоточные вычисления

Поток команд

В современных операционных системах (Unix, Windows NT, 2000) может одновременно выполняться **несколько различных заданий**, хотя у большинства компьютеров установлен только один центральный процессор. Такие системы называются ***многозадачными***. Например, чтобы посмотреть список работающих в данный момент процессов в Windows NT достаточно воспользоваться утилитой *Task Manager*.

Операционная система обеспечивает **защиту** каждого выполняемого процесса от воздействия других процессов. Программа, выполняемая в одном из процессов, **не может получить доступ к данным**, обрабатываемым другой программой. Если только они не используют для взаимодействия механизм коллективного доступа – IPC (Inter Process Communication).

Польза от изоляции процессов друг от друга – минимизация возможности прерывания работы процессов из-за нарушения общей защиты (General Protection Faults) в системе Windows NT, в отличие от Windows 95.

Проблемы технологии многозадачности – издержки системных ресурсов на запуск отдельных процессов. Для того чтобы разделить процесс на два независимых процесса, операционная система создает полную копию памяти процесса, после чего запускает один поток вычислений в первой копии, а другой во второй. На все процедуры копирования и загрузки новых копий программ в память тратится очень много времени.

В конце 80х годов была предложена новая концепция потоков команд, которую называют ***упрощенные процессы*** (lightweight processes). Подобно обычному процессу, поток является независимой последовательностью выполняемых команд процессора. Однако в отличие от процессов, потоки

команд **не защищены** друг от друга средствами операционной системы. Главное их **преимущество**, что они запускаются очень быстро.

При выполнении программ на Java каждая виртуальная машина использует свой собственный процесс. В пределах одной Java программы может быть запущено столько потоков команд, сколько потребуется.

Конкурентное использование процессора

Запущенный на выполнение поток команд вступает в конкуренцию за использование центрального процессора с другими потоками, выполняющимися в этой же виртуальной машине. В последней версии Java для операционных систем Windows NT и Solaris предлагается по два варианта виртуальных машин, отличающихся способом реализации потоков команд. В первом случае организация работы потоков осуществляется с использованием соответствующих механизмов базовой операционной системы (***машинно-зависимые*** потоки). Второй способ реализуется исключительно средствами виртуальной машины (***машинно-независимые*** потоки).

Создание новых потоков

Самый простой и быстрый способ создания программ с несколькими потоками состоит в использовании объекта класса **Thread («нить»).**

```
Tread myThread = new Thread();
```

Любой из потоков имеет приоритет, задаваемый числовым значением в диапазоне от **java.lang.Thread.MIN_PRIORITY** до **java.lang.Thread.MAX_PRIORITY**.

В Java существует два типа объектов Thread. В большинстве случаев организуются потоки, предназначенные для обработки команд интерфейса пользователя. Однако иногда выполнение потоков происходит исключительно в **фоновом** режиме (**setDeamon();**).

В момент старта виртуальной машины обычно создается единственный поток, не являющийся демоном, который инициализируется методом **main()** запускаемого класса. Виртуальная машина продолжает запускать потоки на выполнение, пока не произойдет одно из событий:

- был вызван метод **exit()** и процесс, соответствующий данной виртуальной машине, был остановлен.
- все потоки, которые не являются фоновыми, умерли (благополучно закончили свои миссии или сгенерировали исключительные ситуации).

Есть два пути создания новых потоков. Нужно объявить, что проектируемый класс является подклассом класса **Thread**. Этот подкласс должен переопределить метод **run()** класса **Thread**.

```
class SomeThread extends Thread
{       SomeThread()
                { …
                }
                public void run()
```

```
        {  // то, что должно выполняться в данном потоке
        }
}
```

Экземпляр такого класса может теперь размещаться в памяти и запускаться с помощью метода **start()**.

```
SomeThread p = new SomeThread();
p.start();
```

Другой путь создания новых потоков – объявление класса, который реализует интерфейс **Runnable**. В этом классе должен быть определен метод **run()**.

```
class SomeRun implements Runnable
{       SomeRun ()
        {
        }
        public void run()
        {  // то, что должно выполняться в данном потоке
        }
}
```

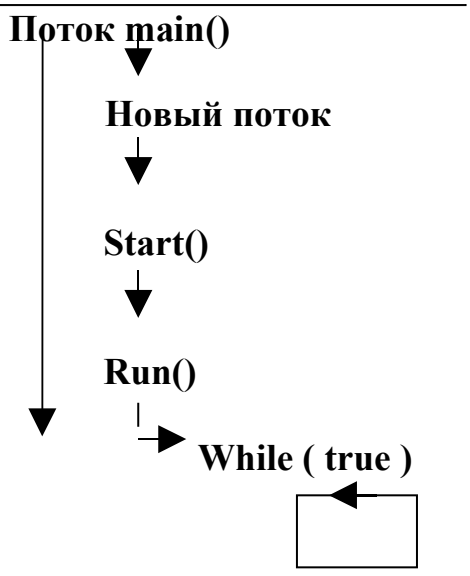
Экземпляр такого класса может теперь размещаться в памяти и запускаться с помощью метода **start()**.

```
SomeRun p = new SomeRun ();
new Thread(p).start();
```

Пример создания нового потока

Рассмотрим пример программы, в которой всего два потока, причем первый поток прекратит работу и остановится в ожидании завершения второго потока, который никогда не остановится.

```
Public class NewThread implements Runnable
{
        public static void  main(String args[])
        {       NewThread demo = new
NewThread();
                Thread newT=new Thread
(demo,"новый поток");
                NewT.start();
                System.out.println("main is over");
        }
        public void run()
        {       while(true)
                {
        System.out.println("выполняется поток
-" +

        Thread.currentThread().toString());
                }
        }
}
```



Для прерывания потока служит метод интерфейса **Runnable – Interrupt()**;

Метод **Sleep()** – служит для приостановки процесса. Зачастую в программах существуют объекты, интересные сразу нескольким потокам. Из-за этого повышается риск появления исключительной ситуации, поэтому является целесообразным обеспечить синхронный доступ к объектам со стороны потоков – обеспечить доступ потока к объекту, только если он не занят в этот момент другим потоком. Для объявления объекта занятым служит ключевое слово **synchronized.**

```
synchronized(object)
{
        object.doSomething();
        …
}
```

Пример синхронизации работы потоков с одним и тем же ресурсом
-ядерный реактор, когда один поток выдвигает графитовые стержни из реактора, а другой поток опускает эти стержни в реактор на один пункт.

Пример взаимной блокировки потоков или процессов – мудрецы за круглым столом с китайскими палочками (между любыми двумя тарелками только одна палочка) – «Проблема обедающих мудрецов».

4.14 Основы межсетевого взаимодействия

За работу по сети отвечает пакет **java.net**, в котором находятся классы, осуществляющие работу со следующими протоколами:

— TCP/IP,
— UDP,
— HTTP.

Работа с протоколом TCP

Для обеспечения работы с TCP основным классом является **java.net.Socket**.

Создание TCP соединения (необходимые шаги):

— получение имени сервера (Хоста - Host) или его IP адреса;
— получение номера порта;
— создание гнезда;
— создание входного или выходного потока, использующего гнезда.

Для создания объекта **java.net.Socket** необходимо наличие следующих аргументов: адреса или имени удаленного Хоста и номера порта.

```
Socket server = new Socket ("sun.com", 13);
```

После создания объекта **Socket** соединение уже установлено. Для того, чтобы читать из данного сетевого соединения необходимо:

— создать входной поток с использованием метода **getInputStream();**
— использовать метод **read()** объекта **InputStream.**

```
// Пример получения с сервера текущего времени
Socket server = new Socket ("sun.com", 13);
 InputStream input = server.getInputStream();
  int byteCount; // служебная переменная, сигнализирующая о том, был ли сигнал конца файла.
  byte inputBuffer[] = new byte[1024]; // массив байтов, куда ведется запись данных из входного потока.
 while ((byteCount = input.read(inputBuffer, 0, 1024)) != -1)
 {
        String    stuff = new String (inputBuffer, 0, byteCount);
        System.out.println ("Received: " + stuff);
 }
```

Данный фрагмент кода осуществляет соединение с удаленным сервером по 13 порту и читает из него данные***. 13 порт возвращает текущую дату и время.***

Работа с DNS-серверами

Адреса машин в глобальной сети представляются в виде последовательности чисел, которые получили название IP адреса, но, как правило, пользователи не знают их, они имеют дело с именами машин.

- **DNS**-служба, обеспечивающая трансляцию между именами машин и их IP адресами.

Java интерфейс к DNS

Класс **InetAddress** предоставляет возможность просмотра адресов.

```
// Пример получения IP адреса сервера sun.com
InetAddress address = InetAddress.getByName("sun.com");
 byte IP[] = address.getAddress();
```

```
System.out.println(address.getHostAddress());
```

Работа с UDP (User Datagram Protocol)

Работа с UDP имеет следующие особенности:

— соединение не устанавливается;

— не существует гарантии доставки сообщения.

При работе с UDP необходимо соблюдать следующую последовательность шагов:

— создание пакета;

— получение или отправка пакета.

При работе с UDP используются следующие основные классы:

— **java.net.DatagramPacket;**

— **java.net.DatagramSocket.**

Работа с классами DatagramPacket и DatagramSocket

При использовании конструктора класса **DatagramPacket** необходимо указывать пакет (массив байт и его длина, а также имя удаленного сервера и его порт).

При работе с **DatagramSocket** можно в конструкторе не указывать адрес удаленного сервера и порт, если эти данные указаны в **DatagramPacket**. Для отправки пакета используется метод **send()**. При получении пакета (метод **recieve()**) необходимо создание дополнительного пакета, в который и будет записана вся поступающая информация. Для того, что бы можно было нормально прочитать содержимое пакета, его необходимо конвертировать при помощи метода **getData()**. При окончании работы с **DatagramSocket** его необходимо закрыть (**close()**).

```
//Пример UDP клиента
byte[]        message = new byte[256];
int           port = 13;
          DatagramSocket socket = new DatagramSocket();
          InetAddress address = InetAdress.getByName("sun.com");

          DatagramPacket packet = new DatagramPacket(message,
message.length,  address, port);
          socket.send(packet);
          packet = new DatagramPacket(message, message.length);
          socket.receive(packet);
          String received = new String(packet.getData(), 0);
          System.out.println("Received: " + received);
          socket.close();
```

Данный фрагмент кода посылает пустой пакет длиной 256 байт на сервер sun.com на 13 порт и читает полученные данные. Результат работы данного фрагмента аналогичен результату работы с TCP.

Создание http-соединений

При создания **URL**-соединения необходимо выполнить следующую последовательность шагов:

- с использованием класса **URL** определить адрес, к которому будет вестись обращение **URL**;
- с использованием классов **URLConnection** и **URL** установить соединение;
- с использованием входного потока прочитать данные.

Рассмотрим пример чтения через создание **http-**соединения и вывод на консоль.

Создание http соединений.

1. Первый шаг:

```
//Создание объекта URL
String ConnAddress = new String(args[0]);
try
        {
                URL url=new URL(ConnAddress);
        }
```

При создании объекта типа URL может произойти исключение, сигнализирующее о том, что строка-аргумент не может быть интерпретирована с точки зрения протокола взаимодействия.

При создании нового объекта можно также использовать и другие конструкторы, например:

```
//Создание объекта URL
URL url=new URL(String protocol, String host, int port, String file);
```

где используются следующие аргументы: название протокола; имя машины; номер порта; путь к файлу.

2. Второй шаг:

```
//Создание объекта URL
URLConnection connection=url.openConnection();
```

Класс **URLConnection** не имеет конструктора, поэтому для создания нового объекта используется объект класса **URL** и метод этого класса **openConnection()**.

3. Третий шаг:

```
//Создание объекта URL
byte ba[]=new byte[1];
int rc=0;
boolean done=false;
        InputStream IS=connection.getInputStream();
        rc=IS.read(ba);
        System.out.write (ba, 0, rc);
        if (rc ==-1) done=true;
```

```
while (!done)
{
        rc=IS.read(ba);
    if (rc != -1) System.out.write (ba, 0, rc);
    if (rc == -1) done=true;
}
```

При помощи объекта **Connection** создается входной поток, в который и происходит запись данных, а в дальнейшем вывод на консоль.

Некоторые вопросы и ответы к разделу

Как открыть **Socket** на клиентской стороне:

```
Socket  aSocket = new Socket(serverName, portNumber);
```

На сервере:

```
ServerSocket  aSocket = null;
 try
                                        {
                                            aSocket = new ServerSocket(
portNumber  );
                                        }
 catch (IOException e)
                                        {
                                        }
```

Как использовать гнезда для создания соединения между сервером и клиентом:

- создание входного/выходного потока;
- чтение выходного потока из гнезда/запись входного потока в гнездо.

Как получить входной/выходной поток из **Socket**:

```
InputStream  in  =  aSocket.getInputStream();
OutputStream  out  =  aSocket.getOutputStream();
```

4.15 Сериализация объектов

Определение. Сериализация объектов – запись состояния объекта целиком (включая все объекты, на который ссылается данный) в поток вывода с последующим воссозданием (десериализацией) этого объекта через какой-то промежуток времени путем чтения его состояния из потока ввода.

Сериализация является технологией, лежащей в основе:

— передачи объектов методом копирования и вставки;
— передачи объектов между клиентом и сервером при удаленном вызове методов;
— работы JavaBeans API.

Простая сериализация

Объекты сериализуются при помощи класса **ObjectOutputStream**, а впоследствии считывается классом **ObjectInputStream**. Они входят в пакет

java.io и обладают дополнительной особенностью – способностью записывать и читать из потока объекты и элементы массива.

Объект сериализуется путем передачи его методу **writeObject()** класса **ObjectOutputStream**. Один вызов этого метода приводит к сериализации всей иерархии объектов. Восстановление объекта предполагает процесс обратный, а именно считывание объекта путем использования метода **readObject()** класса **ObjectInputStream**. Данный метод работает рекурсивно и восстанавливает объект в том же состоянии, в котором они находились в момент сериализации.

```
// сохранение экземпляра объекта в файл
public boolean save(String filename)
{  try
        {       FileOutputStream fileOut = new FileOutputStream(filename);
                ObjectOutput objectOut = new ObjectOutputStream(fileOut);
                objectOut.writeObject(this);
                objectOut.flush();
        }
        catch(IOException e)
        {System.out.println("Error saving..."+e.getMessage());
                return false;
        }
                return true;
};
// восстановление экземпляра объекта из файла
public static MyType load(String filename) throws Exception
{       MyType object;
        FileInputStream fileIn = new FileInputStream(filename);
        ObjectInputStream objectIn = new ObjectInputStream(fileIn);
        object = (MyType) objectIn.readObject();
        return object;
}
```

Пользовательская сериализация

Далеко не каждая переменная состояния программы может и должна быть сериализирована. Это относится к переменным, зависящим от платформы, виртуальной машины или к переменным, связанным с безопасностью.

В поток можно записывать только те классы, для которых определен интерфейс **Serializable** или **Externalizable**. Использование первого говорит о том, что данный объект можно сериализовать. Второй же интерфейс содержит ряд методов, и, как правило, он используется объектами, которые хотят иметь больший контроль над процессами считывания и записи. При сериализации объекта не всегда имеет смысл сохранять все его переменные состояния, для реализации этой функции используется модификатор видимости **transient**.

Сериализация и поддержка версий

Очевидно, что при сериализации объекта необходимо и как-то сохранять некоторую информацию о классе. Эта информация описывается классом

java.io.ObjectStreamClass, в нее входит имя класса и идентификатор версии. Последний параметр важен, так как класс более ранней версии может не восстановить сериализованный объект более поздней версии. Идентификатор класса хранится в переменной типа **long serialVersionUID**. В том случае, если класс не определяет эту переменную, класс **ObjectOutputStream** автоматически вычисляет уникальный идентификатор версии для него с помощью алгоритма **Secure Hash Algorithm** (SHA). При изменении какой-либо переменной класса или какого-нибудь метода не-**private** происходит изменение этого значения. Для вычисления первоначального значения **serialVersionUID** используется утилита **serialver.exe**.

V. МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ (ПРАКТИКУМОВ)

Для выполнения лабораторных заданий студенту необходимо иметь конспект лекций. Студент знакомится с заданием и выполняет его, опираясь на конспект лекций. При выполнении задания связанного с программированием, студент разрабатывает алгоритм решения предложенной задачи, набирает текст программы (реализующей алгоритм), отлаживает и тестирует программу. При написании программы необходимо использовать конструкции языка программирования, изучаемые в рамках лабораторного занятия.

VI. ЛАБОРАТОРНЫЙ ПРАКТИКУМ (ПЕРЕЧЕНЬ ОСНОВНЫХ ТЕМ)

1. Компиляция и запуск программ

Для запуска и компиляции программ на Java необходимо, чтобы на компьютере был установлен пакет **Sun Microsystems Java 2 Software Development Kit** (пакет разработчика для Java 2 фирмы Sun Microsystems) и документация **Java 2 SDK API**. Для компиляции файлов с расширением **.java** и запуска файлов с расширением **.class** можно использовать bat-файлы.

Файл для компиляции java программ (**c.bat**)

```
set         JDKBASE=C:\j2sdk1.4.0
set         WORKSPACE=C:\work
%JDKBASE%\bin\javac -classpath %WORKSPACE%  %1 %2 %3 %4 %5
```

Файл для выполнения java программ (**r.bat**)

```
set         JDKBASE=C:\j2sdk1.4.0
set         WORKSPACE=C:\work
%JDKBASE%\bin\java -classpath %WORKSPACE%  %1 %2 %3 %4 %5
```

В данных файлах присваиваются значения двум переменным окружения работы программы. Переменная **JDKBASE** хранит путь к директории с установленным пакетом J2 SDK, переменная **WORKSPACE** путь к рабочей директории, в которой находятся исходные коды программ, и будут располагаться откомпилированные файлы классов. Данные bat-файлы предполагают запуск с параметрами.

Файл с кодом простейшей программы на Java (**Hello.java**)

```
public class Hello
{
        public static void main(String[ ] args)
        {
                System.out.println("Hello World");
        }
}
```

При компиляции данного файла, если не было допущено синтаксических ошибок, на консоль не будет ничего выведено. В противном случае будут выведены сообщения об ошибках.

Приведем два способа компиляции программы из командной строки – с использованием bat-файла и без него. В качестве параметра компилятору всегда передается имя файла с исходным текстом программы **обязательно с указанием расширения .java**.

```
Rem С помощью bat-файла
c Hello.java

Rem Без использования bat-файла
javac.exe Hello.java
```

Также можно привести два способа запуска откомпилированной программы. В качестве параметра виртуальной машине передается имя файла с байт-кодом программы всегда **без указания расширения .class,** то есть только имя файла. В результате работы программы на экран будет выведено предложение «**Hello World**».

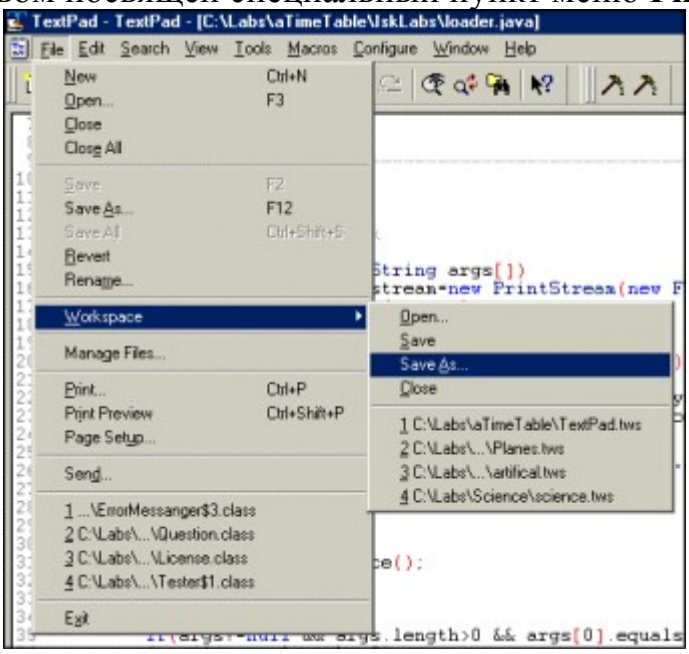
```
Rem С помощью bat-файла
r Hello

Rem Без использования bat-файла
java.exe Hello
```

В качестве альтернативы можно использовать какую-либо среду разработки, например, простой текстовый редактор **TextPad** (Helios Software Solution) с возможностью компиляции и запуска программ на Java.

2.  екстовый редактор TextPad

Текстовый редактор TextPad позволяет компилировать и запускать программы на Java. Для корректной работы с Java программами необходимо следовать следующим правилам:

- Перед началом работы необходимо создать рабочую директорию (папку), в которой будут находиться файлы программ, и сохранить туда настройки TextPad'а в файле с расширение **.tws**– так называемое рабочее пространство (**Workspace**). Операциям с рабочим пространством посвящен специальный пункт меню **File->Workspace.**



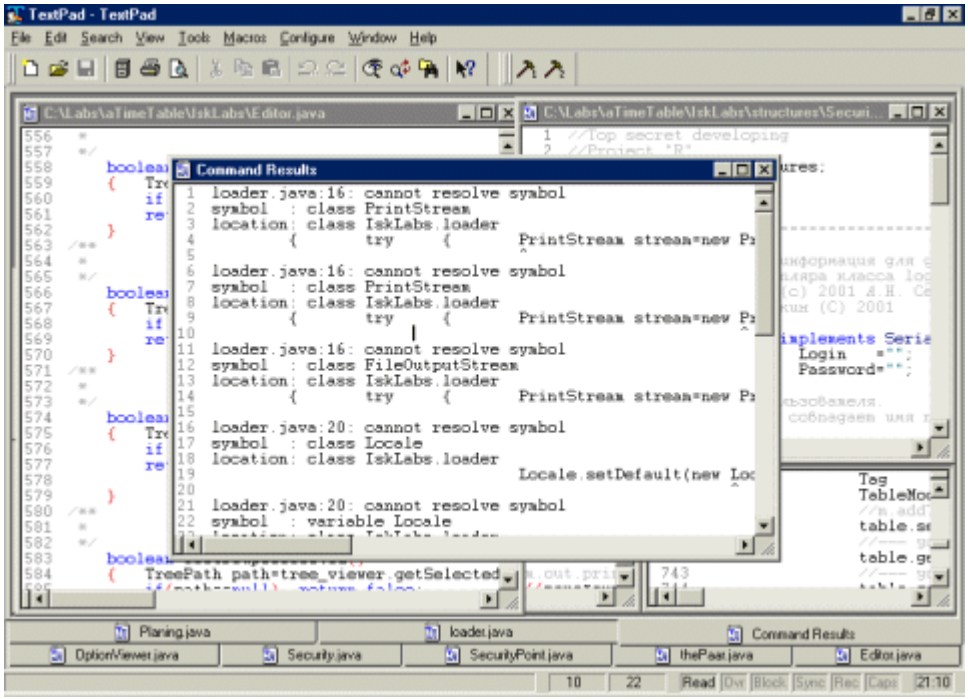
- В последующие сеансы работы при загрузке TextPad не забывать загружать соответствующее рабочее пространство : **File->Workspace->Open…**
- Работа с файлами программ на Java осуществляется стандартно через пункты меню

  File->New
  File->Open
  File->Save
  File->Save As…
  File->Rename…

- Компиляция программ осуществляется через пункт меню

  Tools->Compile Java.
- Запуск откомпилированных программ осуществляется через пункт меню **Tools->Run Java Application.**
- Операции компиляции и запуска Java программ применяются **к активному на данный момент окну** с текстом программы на языке Java.
- Если при компиляции возникли ошибки или при работе программа выводит сообщения на консоль, то эта информация выводится в одно

из дочерних окон редактора TextPad, имеющее название **Command Results**.



3. Лабораторные работы

Ниже приведен список тем лабораторных работ.

Консольные приложения

1. Самая простая программа. Hello.java
2. Печать русского текста на консоли. Example2.java , Example3.java
3. Чтение данных с консоли. Example4.java
4. Работа с числами. Example5.java
5. Работа с массивами. Example6.java
6. Чтение текстового файла с диска. Example7.java
7. Запись текстового файла на диск. Example8.java

Оконные приложения.

1. Простейшее окно. WinExample1.java
2. Стандартное окно с возможностью закрытия. WinExample2.java, WinExample2_a.java
3. Рисование в окне. Изображение текста и линии. WinExample3.java
4. Меню. WinExample4.java
5. Работа с мышкой. WinExample5.java

Основные визуальные компоненты

1. Компонент ***Label*** - метка или ярлык. awtExample1.java
2. Компонент ***Button*** - кнопка. awtExample2.java
3. Компонент ***Button*** в центре экрана. awtExample3.java
4. Компонент ***Checkbox***. awtExample4.java

5. Группа компонентов ***Checkbox***. awtExample5.java
6. Меню из объектов ***CheckBoxMenuItem***. awtExample6.java
7. Компонент ***Choice*** - выбор. awtExample7.java
8. Компонент ***List*** - список. awtExample8.java
9. Компоненты ***PopUpMenu*** и ***TextArea***. awtExample9.java
10. Пример самодельного компонента окна. awtExample10.java

Ниже приводятся примеры лабораторных работ.

```
// Файл Hello.java
// Простейшая программа, выводящая на консоль сообщение "Hello World!".
public class Hello      // Заголовок класса, имя класса должно
                                        // совпадать с названием данного файла.
{
        // Класс, обладающий данной функцией, способен запускаться
        // виртуальной Java машиной на исполнение. Эта функция - точка
        // входа в программу.
        public static void main(String arg[])
        {
                // печать на консоль сообщения.
                System.out.println("Hello World");
        }
}

// Файл Example2.java
// Программа печати русского текста на консоли с использованием потоков.

// Подключение пакета работы с потоками ввода-вывода
import java.io.*;

public class Example2           // Заголовок класса, имя класса должно
                                // совпадать с названием данного файла.
{       // Класс, обладающий данной функцией, способен запускаться
        // виртуальной Java машиной на исполнение. Эта функция - точка
        // входа в программу.
        public static void main(String argv[])
        {       // Попытка выполнить операцию с потоками
                try     { // Открываем поток вывода с указанием кодировки текста.
                          // При запуске программы с консоли с помошью bat-файла кодировака
"CP866",
                          // При запуске из TextPad'а надо изменить кодировку на "Windows-
1251".
                        OutputStreamWriter  stream = new
OutputStreamWriter(System.out,"CP866");
                         // вывод строки символов в поток
                        stream.write("Если вы читаете это сообщение, то кодировка выбрана
верно.\n");
                        // активация передачи данных в поток
                        stream.flush();
                        }
                // Реакция на появление исключительной ситуации в ходе
                // выполнения команд в записанном выше блоке try
                catch (Exception e)
                        {       // вывод информации об исключительной ситуации на консоль
                                System.out.println(e);
                        }
        }
}
```

```
// Файл Example3.java
// Печать русского текста на консоли с помощью метода print() объекта Example3

// Подключение пакета работы с потоками вовода-вывода
import java.io.*;

public class Example3          // Заголовок класса, имя класса должно
                               // совпадать с названием данного файла
{       // Ссылка stream на поток вывода - свойство данного объекта.
        OutputStreamWriter  stream=null;

        // Метод данного объекта, выводящий на поток stream строку символов text.
        public void print(String text)
        {
                // Объяснения в файле Example2
                try     {
                                stream.write(text);
                                stream.flush();
                        }
                catch (Exception e)
                        {
                                System.out.println(e);
                        }
        }
        // Метод данного объекта, аналогичный print, но с переходом на следующую строку.
        public void println(String text)
        { print(text+"\r\n");
        }
        // Метод - конструктор данного объекта,
        // эта функция вызывается при создании экземпляра данного класса.
        public Example3()
        {       // Попытка создания объекта типа OutputStreamWriter и присвоение
                // переменной stream ссылки на этот объект.
                try
                {
                        stream   = new OutputStreamWriter(System.out,"Windows-1251");
                }
                catch (Exception e)
                {
                        System.out.println(e);
                }
                // Использование метода print для вывода сообщения на консоль
                // об успешном создании экземпляра данного класса
                print("Экземпляр класса Example3 создан");
        }

        // Класс, обладающий данной функцией, способен запускаться
        // виртуальной Java машиной на исполнение. Эта функция - точка
        // входа в программу.
        public static void main(String argv[])
        {       // Динамическое создание экземпляра класса Example3,
                // то есть запуск его конструктора.
                new Example3();
        }
}

// Файл Example4.java
// Чтение строки символов с консоли.

// Подключение пакета работы с потоками вовода-вывода
import java.io.*;
```

```
public class Example4          // Заголовок класса, имя класса должно
                               // совпадать с названием данного файла
{       // Ссылка stream на поток вывода - свойство данного объекта.
        OutputStreamWriter  streamOut=null;
        // Ссылка stream на поток ввода - свойство данного объекта.
        BufferedReader              streamIn =null;
        // Объяснения в файле Example3
        public void print(String text)
        {
                try     { streamOut.write(text); streamOut.flush();
                        }
                catch (Exception e)
                        { System.out.println(e);
                        }
        }
        // Объяснения в файле Example2
        public void println(String text)
        {       print(text+"\r\n");
        }
        // Метод - конструктор данного объекта,
        // эта функция вызывается при создании экземпляра данного класса.
        public Example4()
        {
                try     {       // Создания объекта типа OutputStreamWriter и присвоение
                                // переменной stream ссылки на этот объект.
                                streamOut   = new OutputStreamWriter(System.out,"CP866");
                                // Создания объекта типа OutputStreamWriter и присвоение
                                // переменной stream ссылки на этот объект.
                                streamIn  = new BufferedReader(new
InputStreamReader(System.in));
                        }
                catch (Exception e)
                        { System.out.println(e);
                        }

                print("Введите строку символов: ");
                try     {       // Переменная типа строка - String
                                String s;
                                // Чтение строки символов из потока ввода
                                // и присвоение переменной s ссылки на эту строку.
                                s = streamIn.readLine();
                                // Вывод полученной строки символов
                                print("Было введено: "+s);
                        }
                catch (Exception e)
                {       System.out.println(e);
                }

        }
        // Класс, обладающий данной функцией, способен запускаться
        // виртуальной Java машиной на исполнение. Эта функция - точка
        // входа в программу.
        public static void main(String argv[])
        {       // Динамическое создание экземпляра класса Example3,
                // то есть запуск его конструктора.
                new Example4();
        }
}

// Файл Example5.java
// Пример арифметических вычислений
```

```
class Example5
{
	public static void main(String arg[])
	{	// Ссылки на объекты типа Целое число - Integer.
		Integer a1, a2, a3;
		// Создание объектов типа Целое число.
		// Простое присвоение значений сдесь невозможно.
		a2 = new Integer(5);
		a3 = new Integer(6);

		// Простейший тип данных int - Тоже целое число
		// Возможны привычные арифметические операции и
		// прямое присвоение значений
		int i=3;

		// Использование у объектов Integer методов получения
		// значений простейшего типа int.
		i = a2.intValue() + a3.intValue();
		// Создание объекта Integer на основе вычисленного значения
		// переменной простейшщего типа int
		a1 = new Integer(i);
		// вывод результатов вычислений
		System.out.println("i2+i3="+a1);
		// То же самое напрямую:
		System.out.println("5+6="+(5+6));
	}
}

// Файл Example6 .java
// Пример работы с массивами

class Example6
{
	public static void main(String arg[])
	{	// Создание ссылки на массив, состоящий из двух элементов типа int
		int array_int[]= new int[2];

		// Пример присвоения значений элементам массива
			array_int[0] = 5;
			array_int[1] = 6;

		// Вывод элементов массива в цикле
		for (int n=0; n <array_int.length; n++)
					System.out.print(" "+array_int[n]);

		// Задание элементов массива при создании ссылки на массив целых чисел
		int	array[]={5,6,7,8};
		// Задание элементов массива при создании ссылки на массив строк
		String  arrays[] = {"элемент 0","элемент 1","элемент 2"};
		// Сложно устроенный массив объектов
		Object  arrayo[] = {new Integer(13), arrays};

		// Вывод всех элементов массива arrayo[]
		System.out.println("\narrayo[]:");
		for (int n=0; n < arrayo.length; ++n)
		{	Object o = arrayo[n];
			// если элемент o не массив
			if (!o.getClass().isArray())
					System.out.println("\tarrayo["+n+"]="+o);
			// иначе
			else
			{	// объект - массив преобразуется в массив объектов
```

```
                                Object a[] =(Object[]) o;
                                // Вывод всех элементов массива a[]
                                System.out.print("\tarrayo["+n+"]=");
                                for (int n2=0; n2 < a.length; ++n2)
                                        System.out.print(" "+a[n2]);
                        }
                }
        }
}

// Файл Example7 .java
// Чтение текстового файла
// Имя текстового файла передается в программу из командной строки.
// Запускать программу следует с помощью bat-файла.

import java.io.*;

class Example7
{
        public static void main(String args[])
        {
                // Если программа была запущена без параметров
                if(args.length==0)
                {       System.out.println("Запустите программу с параметром");
                        System.exit(1);
                }
                String s=null;  // Ссылка на строку символов
                int l=0;        // Счетчик количества строк в читаемом файле.
                try
                {               // Открытие файла для чтения.
                                BufferedReader in = new BufferedReader(new FileReader(args[0]));
                                // Цикл чтение строк из фалйа
                                while ((s = in.readLine()) != null)
                                {       l++;            // Увеличиние количества прочитанных
строк.
                                        System.out.println(s); // Вывод на консоль прочитанной
строки
                                }
                                in.close(); // Закрытие файла
                }
                catch(IOException e)  // обработка ошибок.
                {       System.out.println(e);
                }
                System.out.println("\nNumber of lines= "+l);
        }
}

//  Файл Example8.java
//  Запись данных в текстовый файл

import java.io.*;

class Example8
{
        public static void main(String arg[])
        {
                try
                {       // Открыть файл "out.txt" для записи.
                        PrintWriter out   = new PrintWriter(new BufferedWriter(new
FileWriter("out.txt")));

                        out.println("Этот тектовый файл");                  // записать строку в
файл "out.txt"
```

```
                        out.println("создан программой Example8");          // записать строку в
файл "out.txt"

                        out.close(); // Закрыть файл

                }
                catch(IOException e) // обработка ошибок.
                {       System.out.println(e);
                }
        }
}

// Файл WinExample1.java
// Простейшее окно, которое невозможно закрыть.

// Подключение пакета классов для работы с окнами
import java.awt.*;

class WinExample1
{
public static void main(String arg[])
        {
                // Создать экземпляр класса стандартное окно
                Frame win = new Frame("Стандартное окно");
                // Установить размеры окна.
                win.setSize(250,100);
                // Показать окно на экране
                win.show();
        }
}

// Файл WinExaple2.java
//  Пример обработки событий окна
// Подключение пакета классов для работы с окнами
import java.awt.*;
// Подключение пакета классов для работы с событиями
import java.awt.event.*;

class WinExample2 extends Frame implements WindowListener
{
        // Реакция на открытие окна.
        public void windowOpened(WindowEvent e)
        {       System.out.println("Window opened.");
        }
        // Реакция на закрытие окна с помощью мыши.
        public void windowClosing(WindowEvent e)
        {       dispose(); // Освобождение ресурсов окна.
                System.out.println("Window closing.");
                //System.exit(0); // Выход из программы.
        }
        // Окно уже закрыто.
        public void windowClosed(WindowEvent e)
        {       System.out.println("Window closed.");
        }
        // Реакция на свертывание окна в иконку.
        public void windowIconified(WindowEvent e)
        {       System.out.println("Window iconified.");
        }
        // Реакция на развертывание окна из иконки.
        public void windowDeiconified(WindowEvent e)
        {       System.out.println("Window Deinconified");
        }
        // Реакция на активацию окна.
```

```
	public void windowActivated(WindowEvent e)
	{	System.out.println("Window activated.");
	}
	// Реакция на деактивацию окна.
	public void windowDeactivated(WindowEvent e)
	{	System.out.println("Window deactivated.");
	}


	// Метод - конструктор данного объекта,
	// эта функция вызывается при создании экземпляра данного класса.
	public WinExample2()
	{	super("Стандартное окно с реакцией на события"); // вызов конструктора
суперкласса

		// Вызов функции установки класса - обработчика событий окна
		// В данном примере это сам класс WinExaple2
		addWindowListener(this);

		setSize(400,200);	// Установить размер
		show();				// Показать окно
	}

	public static void main(String arg[])
	{	// Динамическое создание экземпляра класса WinExaple2,
		// то есть запуск его конструктора.
		new WinExample2();
	}
}

//	Файл WinExaple2.java
//  Пример обработки события закрытия окна

// Подключение пакета классов для работы с окнами
import java.awt.*;
// Подключение пакета классов для работы с событиями
import java.awt.event.*;

class WinExample2_a extends Frame
{	// Метод - конструктор данного объекта,
	// эта функция вызывается при создании экземпляра данного класса.
	public WinExample2_a()
	{	super("Стандартное закрывающееся окно"); // вызов конструктора
суперкласса

		// Вызов функции установки класса - обработчика событий окна
		// В данном примере это сам класс WinExaple2
		addWindowListener(new WindowAdapter()
		{	public void windowClosing(WindowEvent e)
			{
				System.exit(0);
			}
		});

		setSize(400,200);	// Установить размер
		show();				// Показать окно
	}

	public static void main(String arg[])
	{	// Динамическое создание экземпляра класса WinExaple2,
		// то есть запуск его конструктора.
		new WinExample2_a();
```

```
        }
}

//      Файл WinExample3.java
//  Пример рисования текста и линии в окне.

import java.awt.*;
import java.awt.event.*;

class WinExample3 extends Frame    implements WindowListener
{

        // Группа методов реагирующая на события окна.
        public void windowOpened(WindowEvent e) {}
        public void windowClosing(WindowEvent e) { dispose(); System.exit(0); }
        public void windowClosed(WindowEvent e) {}
        public void windowIconified(WindowEvent e) {}
        public void windowDeiconified(WindowEvent e) {}
        public void windowActivated(WindowEvent e) {}
        public void windowDeactivated(WindowEvent e) {}

        // Данный метод вызывается в момент прорисовки окна системой,
        // поэтому в нем программируются все операции рисования в окне.
        public void paint(Graphics g)
        {       // Получить размер данного окна
                Dimension size = getSize();
                int width = size.width, height= size.height;

                g.setColor(Color.red);                          // Установка цвета
                String text = "Java 2 AWT package";             // Текст надписи
                g.setFont(new Font("Arial",Font.PLAIN,20));  // Создание шрифта
                int TextWidth = g.getFontMetrics().stringWidth(text); // Вычисление ширины
текста
                g.drawString(text,(width - TextWidth)/2,height/ 2);     // Вывод текста в центр
экрана

                g.setColor(Color.black);                        // Установка цвета
                g.drawLine(0,0, width,height);                  // Рисование диагональной линии
                g.setColor(Color.green);                        // Установка цвета
                g.drawLine(width,0, 0,height);                  // Рисование диагональной линии
        }

        // Метод - конструктор данного объекта,
        // эта функция вызывается при создании экземпляра данного класса.
        public WinExample3()
        {       super("Графика в окне"); // вызов конструктора суперкласса

                // Вызов функции установки класса - обработчика событий окна
                // В данном примере это сам класс WinExaple2
                addWindowListener(this);

                setSize(400,200);       // Установить размер окна
                show();                         // Показать окно
        }

        public static void main(String arg[])
        {
                // Вывод списка всех доступных шрифтов в данной версии Java.
                Font fonts[]=GraphicsEnvironment.getLocalGraphicsEnvironment().getAllFonts();
                for (int i=0; i < fonts.length; ++i)
                                                        System.out.println(fonts[i]);
                // Динамическое создание экземпляра класса WinExample3,
                // то есть запуск его конструктора.
```

```
                new WinExample3();
        }
}

// Файл WinExample4.java
// Меню
import java.awt.*;
import java.awt.event.*;
class WinExample4 extends Frame implements       ActionListener
{
        // Самодельный метод для создания меню.
        void CreateMenu()
        {
                Menu mColor = new Menu("Цвета");          // Создание первого меню.
                mColor.addActionListener(this);           // Установка обработчика событий
меню
                mColor.add(new MenuItem("Красный"));      // Добавление пункта в меню.
                mColor.add(new MenuItem("Зелёный"));      // Добавление пункта в меню.
                mColor.add(new MenuItem("Чёрный"));       // Добавление пункта в меню.
                mColor.add(new MenuItem("Белый"));        // Добавление пункта в меню.

                MenuBar mBar = new MenuBar();             // Создать полоску меню
                mBar.add(mColor);                         // Добавить в неё созданные меню
                setMenuBar(mBar);                         // Устанавить полоску меню в окно.
        }

        // Метод - обработчик событий меню
        public void actionPerformed(ActionEvent e)
        {
                String cmd = e.getActionCommand();  //команда события. Оно равно команде
имени пункта
                if (cmd.equals("Красный"))          { System.out.println("Выбран красный
цвет");}
                else  if (cmd.equals("Зелёный"))    { System.out.println("Выбран зеленый
цвет");}
                else  if (cmd.equals("Чёрный"))     { System.out.println("Выбран черный
цвет");}
                else if (cmd.equals("Белый"))       { System.out.println("Выбран белый цвет");}
        }

        // Метод - конструктор данного объекта,
        // эта функция вызывается при создании экземпляра данного класса.
        public WinExample4()
        {       super("Меню."); // вызов конструктора суперкласса

                CreateMenu(); // вызвать метод создания меню
                // Вызов функции установки класса - обработчика событий окна
                addWindowListener(new WindowAdapter()
                {       public void windowClosing(WindowEvent e)
                        {
                                System.exit(0);
                        }
                });
                setSize(400,200);                         // Установить размер окна
                show();                                   // Показать окно
        }

        public static void main(String arg[])
        {       // Динамическое создание экземпляра класса WinExample4,
                // то есть запуск его конструктора.
                new WinExample4();
        }
```

```
}

// Файл WinExample5.java
// Графический редактор

import java.awt.*;
import java.awt.event.*;

class WinExample5 extends Frame implements      WindowListener, ActionListener,
                                                MouseMotionListener, MouseListener
{
        // Группа методов реагирующая на события окна.
        public void windowOpened(WindowEvent e) {}
        public void windowClosing(WindowEvent e) { dispose(); System.exit(0); }
        public void windowClosed(WindowEvent e) {}
        public void windowIconified(WindowEvent e) {}
        public void windowDeiconified(WindowEvent e) {}
        public void windowActivated(WindowEvent e) {}
        public void windowDeactivated(WindowEvent e) {}

        Color color =   Color.black;    // Текущий цвет
        Point p1        =       null;                   // Координаты предыдущей точки

        public void mouseClicked(MouseEvent e) {}  // Сделан щелчок мыши.

        public void mousePressed(MouseEvent e)    // Нажата кнопка мыши
        {       p1 = e.getPoint(); // Установка точки начала линии
        }

        public void mouseReleased(MouseEvent e) {}      // Кнопка мыши отжата
        public void mouseEntered(MouseEvent e)    {}    // Мышь над данным окном
        public void mouseExited(MouseEvent e)     {}    // Мышь покинула пределы данного
окна
        public void mouseDragged(MouseEvent e)          // Перемещение при нажатой кнопке
мыши.
        {       Point p2=e.getPoint();                  // Запомнить точку
                Graphics g = getGraphics();             // Получить графический контекст
                g.setColor(color);                      // Установить текущий цвет
                g.drawLine(p1.x,p1.y, p2.x,p2.y);       // Нарисовать линию
                p1 = p2;                                // Последняя точка становится первой.
        }

        public void mouseMoved(MouseEvent e) {} // Простое передвижение мыши.

        // Самодельный метод для создания меню.
        void CreateMenu()
        {
                Menu mColor = new Menu("Цвета");        // Создание первого меню.
                mColor.addActionListener(this);         // Установка обработчика событий
меню
                mColor.add(new MenuItem("Красный"));    // Добавление пункта в меню.
                mColor.add(new MenuItem("Зелёный"));    // Добавление пункта в меню.
                mColor.add(new MenuItem("Чёрный"));     // Добавление пункта в меню.
                mColor.add(new MenuItem("Белый"));      // Добавление пункта в меню.

                MenuBar mBar = new MenuBar();           // Создать полоску меню
                mBar.add(mColor);                       // Добавить в неё созданные меню
                setMenuBar(mBar);                       // Устанавить полоску меню в окно.
        }

        // Метод - обработчик событий меню
        public void actionPerformed(ActionEvent e)
        {
```

```
                //      System.out.println("Menu event: "+e); // для отладки смотрим принятое
событие
                String cmd = e.getActionCommand();  // Выделяем команду события. Оно равно
команде
                                                    // имени пункта
                if (cmd.equals("Красный"))          { color = Color.red ;}          // Установить
цвет.
                else  if (cmd.equals("Зелёный"))    { color = Color.green ;}        // Установить
цвет.
                else  if (cmd.equals("Чёрный"))     { color = Color.black ;}        // Установить
цвет.
                else if (cmd.equals("Белый"))       { color = Color.white ;}        // Установить
цвет.
        }

        // Метод - конструктор данного объекта,
        // эта функция вызывается при создании экземпляра данного класса.
        public WinExample5()
        {       super("Графический редактор."); // вызов конструктора суперкласса

                CreateMenu(); // вызвать метод создания меню

                addWindowListener(this);            // Установить класс обработчика событий
окна
                addMouseMotionListener(this);       // Обрабатывать события движения
мышки.
                addMouseListener(this);             // Обрабатывать события мышки.
                setSize(400,200);                   // Установить размер окна
                show();                             // Показать окно
        }

        public static void main(String arg[])
        {       // Динамическое создание экземпляра класса WinExample5,
                // то есть запуск его конструктора.
                new WinExample5();
        }
}

// Файл awtExample1.java
// Пример с Label

import java.awt.*;
import java.awt.event.*;

class awtExample1 extends Frame implements ActionListener
{
        // Создание экземпляра объекта Label
        Label label = new Label("Это объект Label из пакета java.awt",Label.CENTER);

        // Метод, в котором создается меню.
        void CreateMenu()
        {
                Menu mColor = new Menu("Цвета");                // Создание первого меню.
                        mColor.addActionListener(this);         // Установка обработчика
событий меню
                        mColor.add(new MenuItem("Красный"));    // Добавление пункта в меню.
                        mColor.add(new MenuItem("Зелёный"));    // Добавление пункта в меню.
                        mColor.add(new MenuItem("Чёрный"));     // Добавление пункта в меню.
                        mColor.add(new MenuItem("Белый"));      // Добавление пункта в меню.

                MenuBar mBar = new MenuBar();                   // Создать полоску меню
                        mBar.add(mColor);                       // Добавить в неё созданные
меню
```

```
                    setMenuBar(mBar);                    // Устанавить полоску меню в
окно.
        }
        // Единственный метод интерфейса ActionListener - обработчик событий меню
        public void actionPerformed(ActionEvent e)
        {

        String cmd = e.getActionCommand();          // Получить команду события.
                                                    //Команда равна команде имени пункта
        if (cmd.equals("Красный")) { label.setForeground( Color.red) ;}          // Установить
цвет.
        else    if (cmd.equals("Зелёный")) { label.setForeground(Color.green) ;} // Установить
цвет.
        else    if (cmd.equals("Чёрный")) { label.setForeground(Color.black) ;}  // Установить
цвет.
        else    if (cmd.equals("Белый")) { label.setForeground(Color.white) ;}   // Установить
цвет.
        }

        // Метод - конструктор данного объекта.
        public awtExample1()
        {       super("Пример java.awt.Label"); // вызвать конструктор суперкласса

                CreateMenu(); // вызвать метод создания меню

                // Установить класс обработчика событий закрытия окна
                addWindowListener(new WindowAdapter()
                {       public void windowClosing(WindowEvent e)
                        {
                                                System.exit(0);
                        }
                });
                // Установить размер
                setSize(400,200);
                // Установить раскладчик компонентов в окне, который решает где и как
                // расположить объекты в окне
                setLayout(new BorderLayout());
                // Добавить созданный компонент класса Label в данное окно по центру
                add(label,BorderLayout.CENTER);
                // Показать окно
                show();
        }

        public static void main(String arg[])
        {       // Динамическое создание экземпляра класса awtExample1,
                // то есть запуск его конструктора.
                new awtExample1();
        }
}

// Файл awtExample2.java
// Пример с объектом Button

import java.awt.*;
import java.awt.event.*;

class awtExample2 extends Frame implements ActionListener
{
        // Создание экземпляра объекта Button
        Button button = new Button("Это компонента Button");
        // Метод, в котором создается меню.
        void CreateMenu()
```

```
        {
                Menu mColor = new Menu("Цвета");                    // Создание первого меню.
                        mColor.addActionListener(this);             // Установка обработчика
событий меню
                        mColor.add(new MenuItem("Красный"));        // Добавление пункта в меню.
                        mColor.add(new MenuItem("Зелёный"));        // Добавление пункта в меню.
                        mColor.add(new MenuItem("Чёрный"));         // Добавление пункта в меню.
                        mColor.add(new MenuItem("Белый"));          // Добавление пункта в меню.

                MenuBar mBar = new MenuBar();                       // Создать полоску меню
                        mBar.add(mColor);                           // Добавить в неё созданные
меню
                        setMenuBar(mBar);                           // Устанавить полоску меню в
окно.
        }
        // Единственный метод интерфейса ActionListener - обработчик событий меню
        public void actionPerformed(ActionEvent e) // Обработчик событий меню
        {// Получить команду события. Команда равна команде имени пункта
         String cmd = e.getActionCommand();
         if (cmd.equals("Красный"))             { button.setForeground( Color.red) ;}          //
Установить цвет.
         else    if (cmd.equals("Зелёный"))  { button.setForeground(Color.green) ;}         //
Установить цвет.
         else    if (cmd.equals("Чёрный"))   { button.setForeground(Color.black) ;}         //
Установить цвет.
         else    if (cmd.equals("Белый"))    { button.setForeground(Color.white) ;}  // Установить
цвет.
         // Реакция на нажатие кнопки
         else    if (cmd.equals("press")) { System.out.println("Команда Press - была нажата
кнопка");}
        }

        // Метод - конструктор данного объекта.
        public awtExample2()
        {       // вызвать конструктор суперкласса
                super("Пример java.awt.Button");

                CreateMenu(); // вызвать метод создания меню

                // Установить класс обработчика событий закрытия окна
                addWindowListener(new WindowAdapter()
                {       public void windowClosing(WindowEvent e)
                        {
                                                System.exit(0);
                        }
                });

                setSize(400,200); // Установить размер
                // Установить раскладчик компонентов в окне, который решает где и как
                // расположить объекты в окне
                setLayout(new BorderLayout());
                // Добавить созданный компонент класса Label в данное окно по центру
                add(button,BorderLayout.CENTER);
                // Указать команду, генерируемую при нажатии кнопки
                button.setActionCommand("press");
                // Указать, что данный класс является также обработчиком событий от кнопки
                button.addActionListener(this);
                // Показать окно
                show();
        }

        public static void main(String arg[])
        {       // Динамическое создание экземпляра класса awtExample2,
```

```
                // то есть запуск его конструктора.
                new awtExample2();
        }
}

// Файл awtExample3.java
// Пример с объектом Button в центре экрана

import java.awt.*;
import java.awt.event.*;

class awtExample3 extends Frame implements ActionListener
{
        // Создание экземпляра объекта Button
        Button button = new Button("Это компонента Button");
        // Метод, в котором создается меню.
        void CreateMenu()
        {
                Menu mColor = new Menu("Цвета");                // Создание первого меню.
                        mColor.addActionListener(this);         // Установка обработчика
событий меню
                        mColor.add(new MenuItem("Красный"));    // Добавление пункта в меню.
                        mColor.add(new MenuItem("Зелёный"));    // Добавление пункта в меню.
                        mColor.add(new MenuItem("Чёрный"));     // Добавление пункта в меню.
                        mColor.add(new MenuItem("Белый"));      // Добавление пункта в меню.

                MenuBar mBar = new MenuBar();                   // Создать полоску меню
                        mBar.add(mColor);                       // Добавить в неё созданные
меню
                        setMenuBar(mBar);                       // Устанавить полоску меню в
окно.
        }
        // Единственный метод интерфейса ActionListener - обработчик событий меню
        public void actionPerformed(ActionEvent e) // Обработчик событий меню
        {// Получить команду события. Команда равна команде имени пункта
         String cmd = e.getActionCommand();
         if (cmd.equals("Красный"))             { button.setForeground( Color.red) ;}          //
Установить цвет.
         else   if (cmd.equals("Зелёный"))  { button.setForeground(Color.green) ;}          //
Установить цвет.
         else   if (cmd.equals("Чёрный"))   { button.setForeground(Color.black) ;}          //
Установить цвет.
         else   if (cmd.equals("Белый"))    { button.setForeground(Color.white) ;}  // Установить
цвет.
         // Реакция на нажатие кнопки
         else   if (cmd.equals("press")) { System.out.println("Команда Press - была нажата
кнопка");}
        }

        // Метод - конструктор данного объекта.
        public awtExample3()
        {       // вызвать конструктор суперкласса
                super("Пример java.awt.Button");

                CreateMenu(); // вызвать метод создания меню

                // Установить класс обработчика событий закрытия окна
                addWindowListener(new WindowAdapter()
                {       public void windowClosing(WindowEvent e)
                        {
                                        System.exit(0);
                        }
```

```
                });

                setSize(400,200); // Установить размер
                // Установить раскладчик компонентов в окне, который решает где и как
                // расположить объекты в окне
                GridBagLayout  grid = new GridBagLayout();
                GridBagConstraints c = new GridBagConstraints();
                setLayout(grid);
                grid.setConstraints(button, c); // Указать положение кнопки в сетке.
                // Добавить созданный компонент класса Label в данное окно по центру
                add(button);
                // Указать команду, генерируемую при нажатии кнопки
                button.setActionCommand("press");
                // Указать, что данный класс является также обработчиком событий от кнопки
                button.addActionListener(this);
                // Показать окно
                show();
        }

        public static void main(String arg[])
        {       // Динамическое создание экземпляра класса awtExample3,
                // то есть запуск его конструктора.
                new awtExample3();
        }
}

// Файл awtExample4.java
// Пример с объектом Checkbox
import java.awt.*;
import java.awt.event.*;
class awtExample4 extends Frame implements  ItemListener
{       // Создание экземпляров объекта CheckBox
        Checkbox box1 = new Checkbox("CheckBox 1");
        Checkbox box2 = new Checkbox("CheckBox 2");
        Checkbox box3 = new Checkbox("CheckBox 3");
        Checkbox box4 = new Checkbox("CheckBox 4");
        // Обработчик событий Checkbox.
        public void itemStateChanged(ItemEvent e)
        {
                Object it = e.getItemSelectable(); // Получить объект-источник события
                int state =       e.getStateChange();  // Получить его состояние

                if (it ==  box1)            System.out.print("Выбран CheckBox 1 ");
                else if (it ==  box2)       System.out.print("Выбран CheckBox 2 ");
                else if (it ==  box3)       System.out.print("Выбран CheckBox 3 ");
                else if (it ==  box4)       System.out.print("Выбран CheckBox 4 ");
                if (state == ItemEvent.SELECTED) System.out.println(" состояние = включен");
                else System.out.println(" состояние = выключен");
        }
        // Метод - конструктор данного объекта.
        public awtExample4()
        {       // вызвать конструктор суперкласса
                super("Пример java.awt.CheckBox");
                // Установить класс обработчика событий закрытия окна
                addWindowListener(new WindowAdapter()
                {       public void windowClosing(WindowEvent e)
                        {
                                                System.exit(0);
                        }
                });
                // Установить раскладчик компонентов в окне, который решает где и как
                // расположить объекты в окне, причем в окне объекты будут располагаться
```

```
                // в 4 строки и 1 колонку.
                setLayout(new GridLayout(4, 1));
                add(box1);add(box2);add(box3);add(box4); // Добавить объекты в окно.
                // Устанавить обработчики событий
                box1.addItemListener(this);
                box2.addItemListener(this);
                box3.addItemListener(this);
                box4.addItemListener(this);
                pack();         // Оптимизировать размер окна.
                show();                 // Показать окно
        }
        public static void main(String arg[])
        {       new awtExample4();
        }
}

// Файл awtExample5.java
// Пример с группой объектов Checkbox
import java.awt.*;
import java.awt.event.*;
class awtExample5 extends Frame implements  ItemListener
{       // Создание экземпляр группы объектов CheckBox
        CheckboxGroup group = new CheckboxGroup();
        // Создание экземпляров объекта CheckBox
        Checkbox box1 = new Checkbox("CheckBox 1",true, group);
        Checkbox box2 = new Checkbox("CheckBox 2",true, group);
        Checkbox box3 = new Checkbox("CheckBox 3",true, group);
        Checkbox box4 = new Checkbox("CheckBox 4",true, group);
        // Обработчик событий Checkbox.
        public void itemStateChanged(ItemEvent e)
        {       Object it = e.getItemSelectable(); // Получить объект-источник события
                if (it ==  box1)  System.out.println("Выбран CheckBox 1 ");
                else
                if (it ==  box2)  System.out.println("Выбран CheckBox 2 ");
                else
                if (it ==  box3)  System.out.println("Выбран CheckBox 3 ");
                else
                if (it ==  box4)  System.out.println("Выбран CheckBox 4 ");
        }
        // Метод - конструктор данного объекта.
        public awtExample5()
        {       // вызвать конструктор суперкласса
                super("Пример группы java.awt.CheckBox");
                // Установить класс обработчика событий закрытия окна
                addWindowListener(new WindowAdapter()
                {       public void windowClosing(WindowEvent e)
                        {               System.exit(0);
                        }
                });
                // Установить раскладчик компонентов в окне, который решает где и как
                // расположить объекты в окне, причем в окне объекты будут располагаться
                // в 4 строки и 1 колонку.
                setLayout(new GridLayout(4, 1));
                add(box1);add(box2);add(box3);add(box4); // Добавить объекты в окно.
                // Устанавить обработчики событий
                box1.addItemListener(this);
                box2.addItemListener(this);
                box3.addItemListener(this);
                box4.addItemListener(this);
                pack();         // Оптимизировать размер окна.
                show();                 // Показать окно
        }
```

```
        public static void main(String arg[])
        {       new awtExample5();
        }
}

// Файл awtExample6.java
// Графический редактор с меню из CheckBoxMenuItem.

import java.awt.*;
import java.awt.event.*;

class awtExample6 extends Frame
        implements    ItemListener,MouseMotionListener, MouseListener
{
        Color colour   =       Color.black;    // Текущий цвет
        Point point    =       null;           // Координаты предыдущей точки

        // Пункты меню и их состояние
        CheckboxMenuItem iRed       = new CheckboxMenuItem("Красный",false);
        CheckboxMenuItem iGreen = new CheckboxMenuItem("Зелёный",false);
        CheckboxMenuItem iBlack = new CheckboxMenuItem("Чёрный",true);
        CheckboxMenuItem iWhite = new CheckboxMenuItem("Белый",false);

        // группа методов реакции на события с мышью
        public void mouseClicked(MouseEvent e) {}  // Сделан щелчок мыши.
        public void mousePressed(MouseEvent e)              // Нажата кнопка мыши.
        { point = e.getPoint();                             // Установить точку начала линии.
        }
        public void mouseReleased(MouseEvent e) {}          // Кнопка отжата.
        public void mouseEntered(MouseEvent e){}  // Мышь над окном.
        public void mouseExited(MouseEvent e) {}    // Мышь покинула пределы окна.
        public void mouseDragged(MouseEvent e)   // Перемещении при нажатой кнопке мыши.
        {       Point point2=e.getPoint();                  // Запомнить точку.
                Graphics g = getGraphics();                 // Получить графический контекст.
                g.setColor(colour);                         // Установить текущий цвет.
                g.drawLine(point.x,point.y, point2.x,point2.y); // Рисовать линию.
                point = point2;                             // Последняя точка становится первой.
        }
        public void mouseMoved(MouseEvent e) {} // Ничего не делать при простом движении
мыши.

        void CreateMenu()
        {       Menu mColor = new Menu("Цвета");  // Создаем первое меню.
                mColor.add(iRed);                           // Добавить пункт в меню.
                mColor.add(iGreen);                         // Добавить пункт в меню.
                mColor.add(iBlack);                         // Добавить пункт в меню.
                mColor.add(iWhite);                         // Добавить пункт в меню.
                iRed.addItemListener(this);    // Устанавливаем обработчики событий для iRed.
                iGreen.addItemListener(this); // Устанавливаем обработчики событий для
iGreen.
                iBlack.addItemListener(this);  // Устанавливаем обработчики событий для
iBlack.
                iWhite.addItemListener(this);  // Устанавливаем обработчики событий для
iWhite.
                MenuBar mBar = new MenuBar();       // Создать полоску меню.
                mBar.add(mColor);                           // Добавить в неё созданные меню.
                setMenuBar(mBar);                           // Устанавить полоску меню в окно.
        }

        public void itemStateChanged(ItemEvent e)  // Обработчик событий CheckboxMenuItem
        {       System.out.println("Menu event: "+e);         // печать принятого события
                Object it = e.getItemSelectable();          // Получить объект-источник
```

```
события
                if (it ==  iRed) // Определить выбранный пункт меню и изменить цвет.
                {       colour= Color.red ; iBlack.setState(false);
                        iGreen.setState(false); iWhite.setState(false);
                }
                else if (it ==  iBlack)
                {       colour= Color.black ; iRed.setState(false);
                        iGreen.setState(false); iWhite.setState(false);
                }
                else if (it ==  iGreen)
                {       colour= Color.green ; iRed.setState(false);
                        iBlack.setState(false); iWhite.setState(false);
                }
                else if (it ==  iWhite)
                {       colour= Color.white ; iRed.setState(false);
                        iGreen.setState(false); iBlack.setState(false);
                }
        }

        public awtExample6()
        {       super("java.awt.CheckboxMenuItem");
                CreateMenu();
                // Установить класс обработчика событий закрытия окна
                addWindowListener(new WindowAdapter()
                {       public void windowClosing(WindowEvent e)
                        {               System.exit(0);
                        }
                });
                addMouseMotionListener(this);       // Обрабатчик событий движения мыши.
                addMouseListener(this);             // Обрабатчик событий мыши.
                setSize(400,200);                   // Установить размер окна.
                show();                             // Показать окно.
        }
        public static void main(String arg[])
        {       new awtExample6();
        }
}

// Файл awtExample7.java
// Пример объекта Choice - выбор из списка.
import java.awt.*;
import java.awt.event.*;
class awtExample7 extends Frame implements ItemListener
{
        // Обработчик событий объекта Checkbox
        public void itemStateChanged(ItemEvent e)
        {       System.out.println(e.getItem()); // Вывести выбранную строку
        }
        public awtExample7()
        {       super("java.awt.Choice"); // вызвать конструктор суперкласса
                // Установить класс обработчика событий закрытия окна
                addWindowListener(new WindowAdapter()
                {       public void windowClosing(WindowEvent e)
                        {               System.exit(0);
                        }
                });
                // Создать экземпляр класса Choice
                Choice choice = new Choice();
                choice.add("Строка 1"); // Дбавить строку в объект choice.
                choice.add("Строка 2");
                choice.add("Строка 3");
                choice.add("Строка 4");
```

```
                choice.addItemListener(this);          // Установить обработчик событий.
                setLayout(null);                       // Отменить раскладчик.
                add(choice);                           // Добавить объект в окно.
                choice.setSize(75,20);                 // Установить размер объекта.
                choice.setLocation(30,50);             // Установить положение объекта в окне.
                setSize(130,110);                      // Установить размер окна.
                show();                                // Показать окно
        }
        public static void main(String arg[])
        {       new awtExample7();
        }
}

// Файл awtExample8.java
// Пример объекта List - списка.
import java.awt.*;
import java.awt.event.*;
class awtExample8 extends Frame implements ItemListener
{       // Создать обект типа List
        List list = new List();
        // Обработчик событий списка
        public void itemStateChanged(ItemEvent e)
        {       System.out.println(list.getSelectedItem()); // Вывести выбранную строку
        }
        public awtExample8()
        {       super("java.awt.List"); // вызвать конструктор суперкласса
                // Установить класс обработчика событий закрытия окна
                addWindowListener(new WindowAdapter()
                {       public void windowClosing(WindowEvent e)
                        {               System.exit(0);
                        }
                });
                list.add("Строка 1");                  // Добавить строку в объект list.
                list.add("Строка 2");                  // Добавить строку в объект list.
                list.add("Строка 3");                  // Добавить строку в объект list.
                list.add("Строка 4");                  // Добавить строку в объект list.
                list.addItemListener(this);            // Установить обработчик событий.
                setLayout(null);                       // Отменить раскладчик.
                add(list);                             // Добавить объект в окно.
                list.setSize(75,80);                   // Установить размер объекта.
                list.setLocation(30,50);               // Установить положение объекта в окне.
                setSize(150,170);                      // Установить размер окна.
                show();                                // Показать окно
        }
        public static void main(String arg[])
        {       new awtExample8();
        }
}

// Файл awtExample9.java
// Объекты PopupMenu и TextArea. Пример простейшего текстового редактора.
import java.awt.*;
import java.awt.event.*;

class awtExample9 extends Frame implements ActionListener, MouseListener
{
        public void mouseClicked(MouseEvent e)    {}                    // Сделан щелчок
мышки.
        public void mousePressed(MouseEvent e)                          // Нажата кнопка
        {       if(e.getModifiers() == InputEvent.BUTTON3_MASK) // Реакция на правую кнопку
мыши
                {               Point p = e.getPoint(); // Точка нахождения мыши
```

```
                        pop.show(this,p.x,p.y);
                }
        }
        public void mouseReleased(MouseEvent e) {}      // Кнопка мыши отжата
        public void mouseEntered(MouseEvent e)  {}      // Мышь над окном
        public void mouseExited(MouseEvent e)   {}      // Мышь покинула пределы окна
        PopupMenu pop = new PopupMenu();                // Создание объекта PopupMenu
        TextArea text = new TextArea();                 // Созданиие текстового поля.
        // Функция печати текста в окне
        public void print(String s)
        { text.append(s);
        }
        // Функция печати текста в окне
        public void println(String s)
        { print(s+"\r\n");
        }
        // Отдельный метод для создания меню.
        void CreateMenu()
        {       Menu mWindow = new Menu("Окно");                // Создание первого меню.
                mWindow.addActionListener(this);        // Установка обработчика событий
меню
                mWindow.add(new MenuItem("Очистить"));          // Добавление пункта в меню.
                mWindow.addSeparator();                         // Добавление разделителя в
меню.
                mWindow.add(new MenuItem("Выход"));             // Добавление пункта в меню.

                MenuBar mBar = new MenuBar();                   // Создание полоски меню
                mBar.add(mWindow);                              // добавление в неё
созданного меню
                setMenuBar(mBar);                               // Установка полосы меню в
окно.
                // Формирование PopupMenu
                MenuItem i = new MenuItem("Открыть");   i.addActionListener(this); pop.add(i);
                i = new MenuItem("Сохранить");                  i.addActionListener(this);
pop.add(i);
                i = new MenuItem("Закрыть");                    i.addActionListener(this);
pop.add(i);
        }
        // Метод - обработчик событий меню
        public void actionPerformed(ActionEvent e)
        {       String cmd = e.getActionCommand();
                println(cmd); // Печать полученной от меню команды
        }

        public awtExample9() // Конструктор.
        {       super("Простейший текстовый редактор. java.awt.PopupMenu &
java.awt.TextArea");
                CreateMenu();
                // Установить класс обработчика событий закрытия окна
                addWindowListener(new WindowAdapter()
                {       public void windowClosing(WindowEvent e)
                        {               System.exit(0);
                        }
                });
                add(text);                                      // Установить текстовое поле в окно.
                text.addMouseListener(this);  // Обрабатывать события мыши.
                text.add(pop);                                  // Добавить в текстовое поле
PopupMenu.
                setSize(400,200);                               // Установить размер окна.
                show();                                         // Показать окно
        }

public static void main(String arg[])
```

```
        {       new awtExample9();
        }
}

//      Файл awtExample10.java
//  Создание собственного компонента окна - окружности.
import java.awt.*;
import java.awt.event.*;
// Класс MCircle компонента окна
class MCircle  extends Component
{       protected int diametr; // Диаметр окружности
        Point p0;                               // Координата нажатия мышки на компоненте.
        Point pc;                               // Координата компоненты в момент нажатия
мышки.
        Cursor cursor;                          // Место для хранение курсора.
        Dimension      dim      = null;  // Жестко заданный размер данного компонента

        public MCircle(int diametr)
        {       super();
                this.diametr = diametr; // установить свойству класса значение диаметра
                // задать статический размер данного компонента окна
                dim = new Dimension(diametr, diametr);
                setSize(diametr, diametr);      // задать размер данного компонента окна
                // Разрешить события мыши.
           enableEvents(AWTEvent.MOUSE_EVENT_MASK |
AWTEvent.MOUSE_MOTION_EVENT_MASK );
        }
        public void paint(Graphics g) // Стандартный метод прорисовки окна
        {       g.drawOval(0, 0, diametr-1, diametr-1); // рисование окружности
        }
        //Отменить изменение размера по setSize(), то есть размер жестко задан.
        public void setSize(int w, int h) {super.setSize(diametr, diametr);}
        // Предпочтительный размер - жестко задан.
        public Dimension getPreferredSize() { return dim; }
        // Минимальный размер.
        public Dimension getMinimumSize() { return dim; }
        // Максимальный размер
        public Dimension getMaxmumSize() { return dim; }

        public void setLocation(int x, int y) // Установить положение компоненты.
        {       int r = diametr / 2;
                super.setLocation(x-r, y-r); // Точка (0,0) в центре круга
        }
        public boolean contains(int x, int y) // Проверка принадлежности точки компоненте
( окружности)
        {       int r = diametr/2;                      // радиус
                int rc2 = ((r-x)*(r-x)+(r-y)*(r-y));    // квадрат удаления точки от центра
                int r2 = r*r;                           // Квадрат внешнего радиуса
                int ri2 = (r-4)*(r-4);                  // Квадрат внутреннего радиуса (граница
зацепки)
                return((rc2 <= r2) && (rc2 >= ri2));    // Условие точки в кольце
   }
        public void processMouseMotionEvent(MouseEvent e)  // Обработка перемещения мыши
        {       switch(e.getID())
                {       // Перетаскивание
                  case MouseEvent.MOUSE_DRAGGED:    // Координата нового перемещения
                                                    Point p = e.getPoint();
                                                    // Новое положение
                                                    Point np = new Point(pc.x+p.x-
p0.x,pc.y+ p.y-p0.y);
                                                    setLocation(np);        // Переместить
компонент
```

```
                                                pc = np;        // Запомнить новое
положение
                                                break;
                }
                super.processMouseEvent(e); // Передать обработку события суперклассу.
        }
        public void processMouseEvent(MouseEvent e) // Обработка событий мыши
        {       switch(e.getID())
                {       // Кнопка нажата
                 case MouseEvent.MOUSE_PRESSED: // Запоминаем координату мышки в
компоненте
                                                p0 = e.getPoint();
                                                // Запоминаем положение компоненты.
                                                pc = getLocation();
                                                break;
                        // Мышь в компоненте
                 case MouseEvent.MOUSE_ENTERED:         // Запомнить курсор
                                                cursor = getCursor();
                                                // Изменить курсор
                                                setCursor(new
Cursor(Cursor.HAND_CURSOR));
                                                break;
                        // Мышка вышла из компоненты
                        case MouseEvent.MOUSE_EXITED:   // Восстановить курсор
                                                setCursor(cursor);
                                                break;
                 }
                super.processMouseEvent(e);  // Передать обработку события суперклассу.
        }

}
// Класс окна.
class awtExample10 extends Frame
{       public awtExample10()
        {       super("Самодельный компонент окна - окружность");
                // Установить класс обработчика событий закрытия окна
                addWindowListener(new WindowAdapter()
                {       public void windowClosing(WindowEvent e)
                        {               System.exit(0);
                        }
                });
                setLayout(null);                                // Отключить раскладку

                MCircle cyrcle = new MCircle(100);    // Создаём круг
                add(cyrcle);                                    // Добавляем в окно
                cyrcle.setLocation(50,50);                      // Устанавливаем его положение
                MCircle cyrcle2 = new MCircle(150); // Аналогично со вторым кругом
                add(cyrcle2);
                cyrcle.setLocation(50,50);

                setSize(400,200);                                       // Установить размер окна
                setLocation(200,200);                                   // Положение окна
                show();                                                 // Показать окно
        }

public static void main(String arg[])
        {       new awtExample10();
        }
}
```

VII. ПЕРЕЧЕНЬ ПРОГРАММНЫХ ПРОДУКТОВ, РЕАЛЬНО ИСПОЛЬЗУЕМЫХ В ПРАКТИКЕ ДЕЯТЕЛЬНОСТИ ВЫПУСКНИКОВ И СООТВЕТСТВУЮЩЕЕ УЧЕБНО-МЕТОДИЧЕСКОЕ ПОСОБИЕ, РАСКРЫВАЮЩЕЕ ОСОБЕННОСТИ И ПЕРСПЕКТИВЫ ИСПОЛЬЗОВАНИЯ ДАННЫХ ПРОГРАММНЫХ ПРОДУКТОВ

Студенты могут создавать приложения в области своей профессиональной деятельности с помощью пакета разработчиков Java 2 SDK Software Development Kit фирмы Sun Microsystems версии 1.5.0 (или более поздних). В рабочей программе приведена литература, посвященная данной среде программирования.

VIII. МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ПРИМЕНЕНИЮ СОВРЕМЕННЫХ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ ДЛЯ ПРЕПОДАВАНИЯ УЧЕБНОЙ ДИСЦИПЛИНЫ (В Т. Ч. РАЗРАБОТАННЫЕ ВЕДУЩИМИ ПРЕПОДАВАТЕЛЯМИ ФИЛИАЛА)

При преподавании данной дисциплины можно использовать электронные тестирующие и учебные материалы. Методические указания прилагаются к этим материалам.

IX. МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПРОФЕССОРСКО-ПРЕПОДАВАТЕЛЬСКОМУ СОСТАВУ ПО ОРГАНИЗАЦИИ МЕЖСЕССИОННОГО И ЭКЗАМЕНАЦИОННОГО КОНТРОЛЯ ЗНАНИЙ СТУДЕНТОВ (МАТЕРИАЛЫ ПО КОНТРОЛЮ КАЧЕСТВА ОБРАЗОВАНИЯ)

Преподаватель готовит контролирующие материалы в виде тестов, задач и в другой форме. Во время проведения контроля знаний студентов преподаватель объясняет студентам правила работы с контролирующими материалами и выдаёт эти материалы студентам. После истечения установленного времени контролирующие материалы собираются и обрабатываются.

X. КОМПЛЕКТЫ ЗАДАНИЙ ДЛЯ ЛАБОРАТОРНЫХ РАБОТ, КОНТРОЛЬНЫХ РАБОТ, ДОМАШНИХ ЗАДАНИЙ

Задания для лабораторных, контрольных работ и домашних заданий берутся из книг, реквизиты которых приведены в рабочей программе.

XI. ФОНД ТЕСТОВЫХ И КОНТРОЛЬНЫХ ЗАДАНИЙ ДЛЯ ОЦЕНКИ КАЧЕСТВА ЗНАНИЙ ПО ДИСЦИПЛИНЕ

Фонд тестовых и контрольных заданий для оценки качества знаний по дисциплине приведен в приложении А.

XII. КОМПЛЕКТЫ ЭКЗАМЕНАЦИОННЫХ БИЛЕТОВ ДЛЯ КАЖДОГО ИЗ ПРЕДУСМОТРЕННЫХ ЭКЗАМЕНОВ ПО ДИСЦИПЛИНЕ И КОНТРОЛЬНЫЕ ВОПРОСЫ К ЗАЧЕТУ

Комплекты экзаменационных билетов составляются на основе перечня вопросов, приведенного в рабочей программе, по следующей форме.

ГОУВПО «Амурский государственный университет»

Утверждено на заседании кафедры
«___» _____ 200 г.
Заведующий кафедрой

Утверждаю: _____

Кафедра *математического анализа и моделирования*
Факультет *математики и информатики*
Курс 4
Дисциплина "*Объектно-ориентированное программирование"*

Экзаменационный билет 1

1. Понятие алгоритма. Языки программирования и их классификация.
2. Сериализация объектов. Классы Reader, Writer и их наследники. Класс File.
3. Тестовое задание.

XIII. КАРТА ОБЕСПЕЧЕННОСТИ ДИСЦИПЛИНЫ КАДРАМИ ПРОФЕССОРСКО-ПРЕПОДАВАТЕЛЬСКОГО СОСТАВА

Дисциплину в полном объёме ведёт:

1. Фамилия, имя, отчество: Рыженко Андрей Викторович

2. Учёное звание: –

3. Учёная степень: канд. техн. наук

ПРИЛОЖЕНИЕ А

1. Что выдаст программа?

```
public class test {
 public static void main (String args[]) {
  System.out.println(6 ^ 3);
 }
}
```

1) 243
2) 5
3) 18
4) 0
5) 6

2. Что произойдет при запуске программы?

```
public class My {
 public static void main (String[] args) {
  String s;
  System.out.println("s=" + s);
 }
}
```

1) Программа скомпилирует и напечатает "s="
2) Программа не скомпилируется, потому что строка s не инициализирована
3) Программа скомпилируется, но при вызове метода toString возникнет исключительная ситуация NullPointerException
4) Программа скомпилируется и напечатает "s=null"
5) Программа не скомпилируется, потому что на строку s нельзя ссылаться

3. Для чего среда Java ограничивает приложениям прямой доступ к памяти?

1) это требование объектной ориентированности языка Java
2) для достижения высоких требований безопасности сетевых приложений
3) для достижения архитектурной независимости java программ
4) этим исключается многие ошибки исполнения Java приложений еще на стадии разработки программ

4. Что необходимо поместить на место комментария в строке 4, чтобы класс скомпилировался?

```
1. public class ExceptionTest {
2.  class TestException extends Exception {}
3.  public void runTest () throws TestException {}
4.  public void test () /* Point X*/  {
5.   runTest ();
6.  }
7. }
```

1) catch (Exception e)
2) throws RuntimeException
3) throws Exception
4) Класс скомпилируется и без изменения кода
5) catch (TestException e)

5. Сколько байт оперативной памяти занимает число типа double?

1) 2
2) 4
3) 6
4) 8
5) Объем выделяемой памяти зависит от компилятора

6) Объем выделяемой памяти зависит от интерпретатора

6. Что произойдет при попытке запуска программы?

```
1. class A { public byte getNumber () { return 1; }}
2. class B extends A {
3.  public short getNumber() { return 2; }
4.  public static void main (String args[]) {
5.    B b = new B (); System.out.println(b.getNumber());
6.  }
7. }
```

1) Программа откомпилируется и напечатает "2"
2) Компилятор выдаст ошибку в строке 3
3) Программа откомпилируется, но в процессе выполнения возникнет исключительная ситуация
4) Компилятор выдаст ошибку в строке 5
5) Программа откомпилируется и напечатает "1"

7. Каково максимальное значение для переменной типа byte?

1) 127
2) 128
3) 256
4) зависит от компилятора
5) зависит от интерпретатора

8. Какой метод позволяет вычислить косинус 42 градусов?

1) Double d = Math.cos(Math.toRadians(42));
2) Double d = Math.cosine(Math.toRadians(42));
3) Double d = Math.cosine(42);
4) Double d = Math.cos(42);
5) Double d = Math.cos(Math.toDegrees(42));

9. Какие слова являются зарезервированными словами языка Java?

1) switch
2) throws
3) throw
4) implement
5) super
6) synchronized

10. Что произойдет при попытке запуска программы?

```
1. public class Test {
2.  public static void replaceJ(String text) { text.replace('j', 'l');}
3.  public static void main (String  args[]) {
4.   String text = new String ("java");
5.   replaceJ(text); System.out.println(text);
6.  }
7. }
```

1) Программа напечатает "java"
2) Программа напечатает "lava"
3 Компилятор выдаст ошибку в строке 4
4) Компилятор выдаст ошибку в строке 2
5) При выполнении возникнет исключительная ситуация

11. Сколько байт оперативной памяти занимает число типа long?

1) 2
2) 4
3) 6
4) 8
5) Объем выделяемой памяти зависит от компилятора
6) Объем выделяемой памяти зависит от интерпретатора

12. Какой тип должна иметь переменная i?

```
switch (i) {
 default: System.out.println("Hello");
}
```

1) object
2) byte
3) float
4) double
5) long

13. Какие объявления переменных допустимы?

1) int x;
2) boolean b1=true, b2=false;
3) float y=7.0;
4) short z=1.0;
5) char c='\u042F';
6) char c="A";

14. Что напечатает программа?

```
1. class Test { public int getLength() { return 4; } }
2. public class Sub extends Test {
3.  public long getLength() { return 5; }
4.  public static void main (String[] args) {
5.   Test sooper = new Test();
6.   Sub sub = new Sub();
7.   System.out.println( sooper.getLength() + "," + sub.getLength());
8.  }
9. }
```

1) 4, 4
2) 4, 5
3) 5, 4
4) 5, 5
5) Компилятор выдаст ошибку

15. Какие из перечисленных идентификаторов являются корректными?

1) abc
2) 1ab
3) _bc
4) _1c_$ac
5) $ac
6) for_
7) Int
8) byte

16. Какое значение переменной j напечатает программа?

```
public class My {
 public static void main (String[] args) {
  int i=1, j=0;
  switch (i) {
   case 2: j+=6;
   case 4: j+=1;
   default: j+=2;
   case 0: j+=4;
  }
  System.out.println("j="+j);
 }
}
```

1) 4

2) 2
3) 1
4) 6
5) 0

17. Равны ли следующие числа:

1) 5 и 05
2) 9 и 09
3) 10 и 010
4) 0x5A и 90L

18. Какие три выражения принимают значение true.

```
public class My {
 private int val;
 public My (int v) { val=v; }
 public static void main (String[] args) {
  My a = new My(10);
  My b = new My(10);
  My c = a;
  int d =10;
  double e = 10.0;
 }
}
```

1) d==e
2) b==d
3) a==c
4) d==10.0
5) a==b

19. Чему будет равно следующее выражение и значения переменных x и y после вычислений?

```
int x=0, y=0;
System.out.println((++x==1) || (y++==1));
```

1) ложно, x=1, y=1
2) истинно, x=1, y=1
3) ложно, x=0, y=0
4) истинно, x=0, y=0
5) ложно, x=0, y=1
6) истинно, x=0, y=1
7) ложно, x=1, y=0
8) истинно, x=1, y=0

20. Что напечатает программа?

```
public class My {
 public static void main (String args[]) {
  int i=0;
  while (i) { if (i==4) { break; } ++i; }
  System.out.println("i="+i);
 }
}
```

1) 0
2) Компилятор выдаст ошибку
3) 5
4) 4
5) 3

СОДЕРЖАНИЕ