**ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ**

**ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«САНКТ-ПЕТЕРБУРГСКИЙ**

**ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ**

**им. проф. М.А. Бонч-Бруевича»**

**(СПбГУТ)**

**Санкт-Петербургский колледж телекоммуникаций**

**Отчет о выполнении  
практического занятия №11-12-13**

Выполнил: Обучающиеся 4 курса, 581 группы,

Лепов Алексей Валерьевич

Проверил: преподаватель,  
Баталов Дмитрий Иннокентьевич

Санкт-Петербург

2021 год

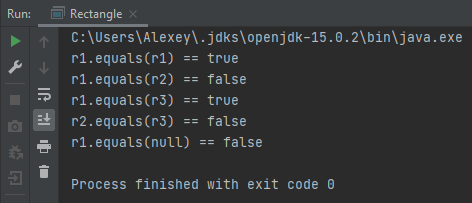
1. **Постановка задач.**

* Изучить работу с потоками.
* Посмотреть на результаты компиляции.
* Также нужно проверить свои навыки загрузки кода программ на хостинге проектов программного обеспечения github.com

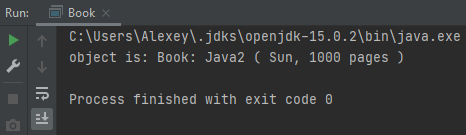
1. **Ход выполнения работ.**

* Выполнено Задание 1 практической работы. Запущены на выполнение коды программ, приведенных в конспекте лекционного занятия «Занятие № 10. Пакет java.lang».
* Выполнено Задание 2. практической работы. Запущены на выполнение коды программ, приведенных в конспекте лекционного занятия «Занятие № 11. Пакет java.util».
* Даны ответы на контрольные вопросы.

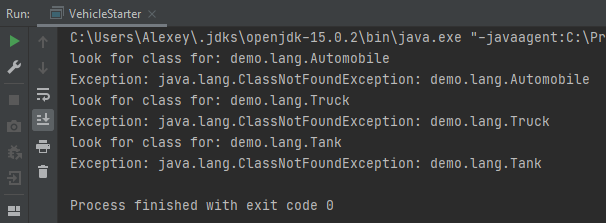
1. **Результаты компиляции программного кода.**



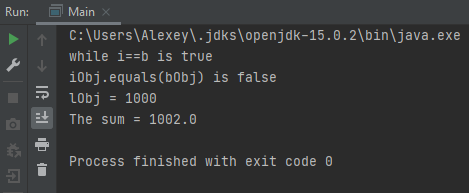
1. *Результат работы программы.* *Задание 1.*



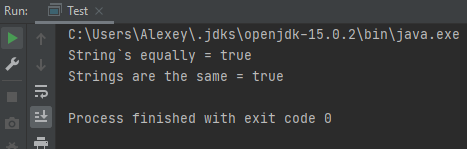
1. *Результат работы программы.* *Задание 1.*



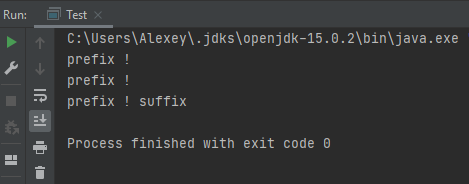
1. *Результат работы программы.* *Задание 1.*



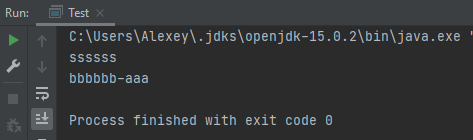
1. *Результат работы программы.* *Задание 1.*



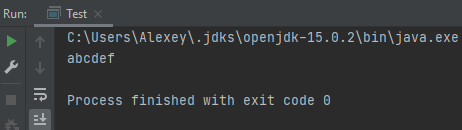
1. *Результат работы программы.* *Задание 1.*



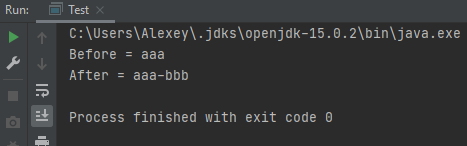
1. *Результат работы программы.* *Задание 1.*



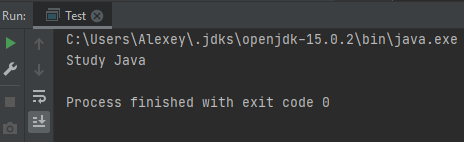
1. *Результат работы программы.* *Задание 1.*



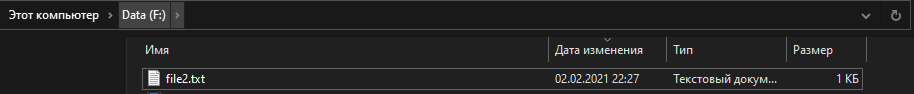
1. *Результат работы программы.* *Задание 1.*



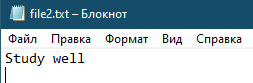
1. *Результат работы программы.* *Задание 1.*



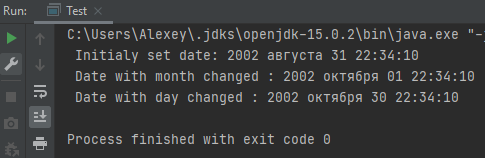
1. *Результат работы программы.* *Задание 1.*



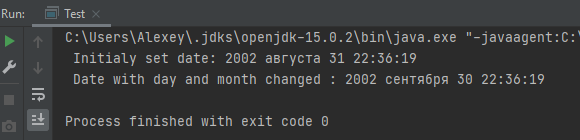
1. *Результат работы программы.* *Задание 1 (пояснение к рис. 10).*



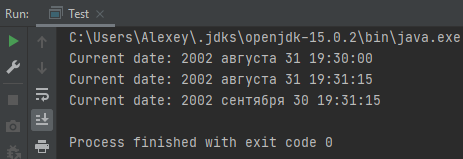
1. *Результат работы программы.* *Задание 1 (пояснение к рис. 10).*



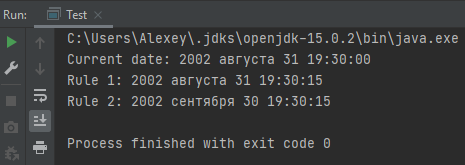
1. *Результат работы программы.* *Задание 2.*



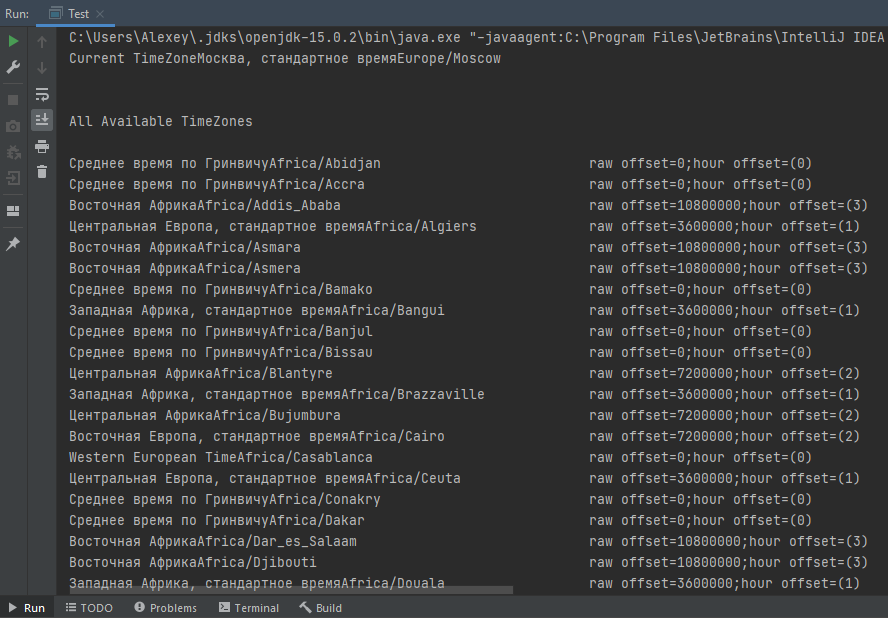
1. *Результат работы программы.* *Задание 2.*



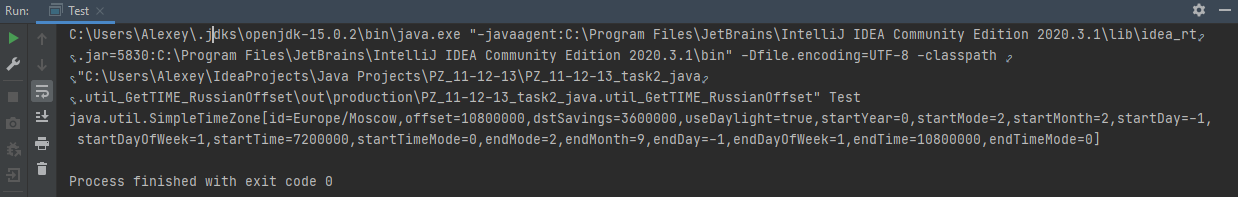
1. *Результат работы программы.* *Задание 2.*



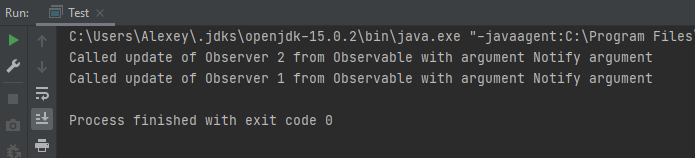
1. *Результат работы программы.* *Задание 2.*



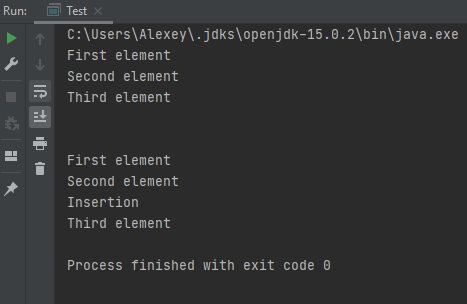
1. *Результат работы программы.* *Задание 2.*



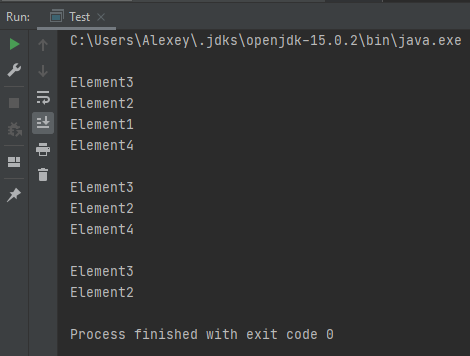
1. *Результат работы программы.* *Задание 2.*



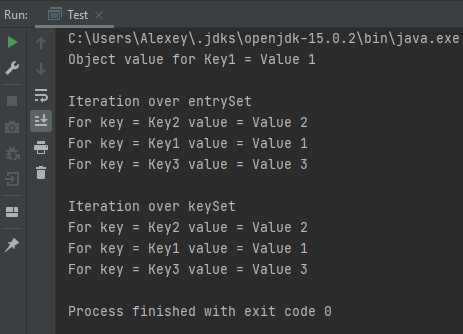
1. *Результат работы программы.* *Задание 2.*



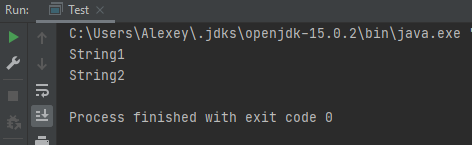
1. *Результат работы программы.* *Задание 2.*



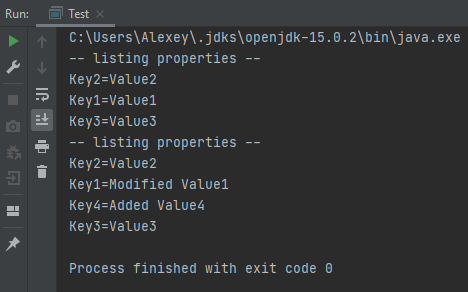
1. *Результат работы программы.* *Задание 2.*



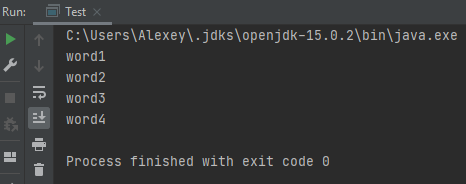
1. *Результат работы программы.* *Задание 2.*



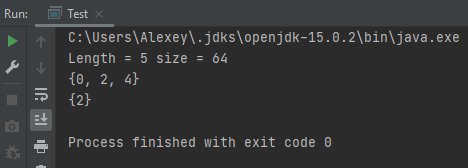
1. *Результат работы программы.* *Задание 2.*



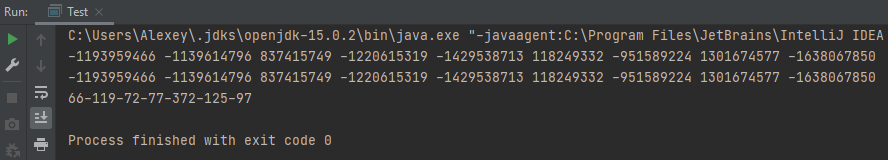
1. *Результат работы программы.* *Задание 2.*



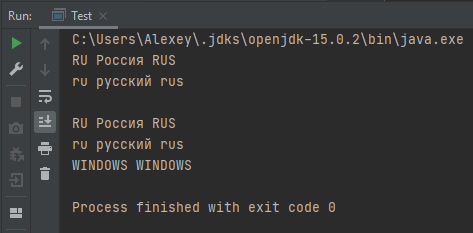
1. *Результат работы программы.* *Задание 2.*



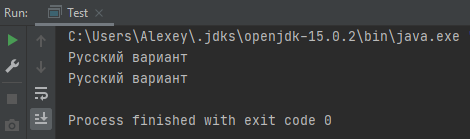
1. *Результат работы программы.* *Задание 2.*



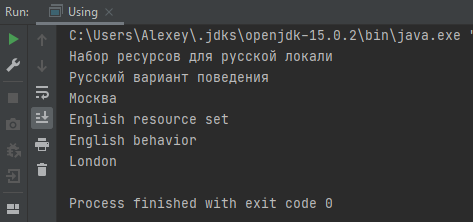
1. *Результат работы программы.* *Задание 2.*



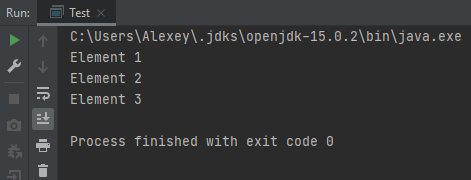
1. *Результат работы программы.* *Задание 2.*



1. *Результат работы программы.* *Задание 2.*



1. *Результат работы программы.* *Задание 2.*



1. *Результат работы программы.* *Задание 2.*
2. **Исходный код проектов.**

Выгрузка проекта была реализована с помощью онлайн инструментария GitHub: (<https://github.com/AlexeyLepov/SPbCK_Lepov_PZ_11-12-13.git>).

1. **Вопросы для защиты.**
2. Какие основные составные части должны присутствовать в каждой Java-программе, их функции (назначение)?

Все Java-программы содержат в себе четыре разновидности блоков: классы (classes), методы (methods), переменные (variables) и пакеты (package).

Методы есть не что иное как функции или подпрограммы. В переменных же хранятся данные. Данные понятия присутствуют так или иначе во всех языках программирования. С другой стороны, классы представляют собой фундамент объектно-ориентированных свойств языка. Для простоты на данном этапе изучения языка Jаva можно сказать, что класс - это некое целое, содержащее переменные и методы.

Наконец, пакеты содержат в себе классы и помогают компилятору найти те классы, которые нужны ему для компиляции пользовательской программы. Классы, входящие в один пакет, особым образом зависят друг от друга, пока же, опять-таки для простоты, можно рассматривать просто как наборы классов. Например, приложение Hello импортирует пакет java.util, в котором содержится класс Date.

Вышеперечисленные блоки присутствуют в любой Java-программе. Но программы могут включать в себя составные части и других видов (интерфейсы, исключения, потоки), которые не требуются обязательно для каждой программы, однако во многих программах без них не обойтись.

1. Что такое первичный класс приложения? Какой обязательный метод он должен содержать?

Java-программа может содержать в себе любое количество классов, но один из них всегда имеет особый статус и непосредственно взаимодействует с оболочкой времени выполнения (первичный класс). В таком классе обязательно должны быть определены один (для приложений) или несколько (для апплетов) специальных методов. Для приложений первичный класс должен обязательно содержать метод **main().**

1. Какие существуют виды переменных Java, чем они отличаются друг от друга?

В Java существует два вида переменных. Первый - примитивные типы (primitive types). К ним относятся стандартные, встроенные в язык типы для представления численных значений, одиночных символов и булевских (двоичных, логических) значений. Все примитивные типы имеют предопределенный размер занимаемой ими памяти. Ко второму виду переменных - ссылочные типы (reference type) - относятся типы, определенные пользователем (классы, интерфейсы) и типы массивов. Все ссылочные типы являются динамическими типами, для них выделяется память во время работы программы.

Примитивные и ссылочные типы также различаются по тому, как переменные этих типов передаются в качестве параметров методам (то есть функциям). Переменные примитивных типов передаются по значению, тогда как ссылочные переменные всегда передаются по ссылке.

1. Какие примитивные типы определены в Java, особенности булевского типа?

Всего в Java определено восемь примитивных типов: int (4b), short (2b), byte (1b), long (8b), float (4b), double (8b), boolean (true, false, 1 бит?), char (2b).

Первые шесть типов предназначены для хранения численных значений, с ними можно производить арифметические операции. Тип char предназначен для хранения символов в стандарте Unicode.

Булевские (логические, двоичные) переменные могут иметь одно из двух допустимых значений: true или false. Двоичные константы являются именно константами, а не строками.

Их нельзя преобразовать в строковый тип. Они также не являются целыми значениями нуль или единица, как двоичные значения в языках C/C++. Рассмотрим примеры использования булевских переменных:

*boolean b1=true, b2, b3; System.out.println(b1); // печать: true b2=(25==25); // b2 равно true b2=(25==10); // b2 равно false b3=(b1&b2); // b3 равно false*

*System.out.println("b1&b2="+b3); // печать: b1&b2=false b3=(b1!=b2); // b3 равно true*

*System.out.println("b1 != b2 - "+b3); // печать: b1 != b2 – true*

Булевским переменным можно присваивать не только булевские константы, но и результаты сравнения переменных различных типов. Операции !, !=, == работают с булевскими значениями так же, как одноименные операторы работают с целочисленными значениями в языке C/C++.

1. Что называется процессом реализации ссылочного типа?

Когда для переменной ссылочного типа выделяется память при помощи оператора new, то тем самым этот ссылочный тип реализуется. Таким образом, каждая переменная ссылочного типа является реализацией, объектом или экземпляром соответствующего типа.

1. Что делает конструктор класса? Должен ли он обязательно явно присутствовать в объявлении класса?

Конструктор - это специальный метод, который вызывается при создании нового объекта. Не всегда удобно инициализировать все переменные класса при создании его экземпляра. Иногда проще, чтобы какие-то значения были бы созданы по умолчанию при создании объекта. По сути конструктор нужен для автоматической инициализации переменных.

Конструктор инициализирует объект непосредственно во время создания. Имя конструктора совпадает с именем класса, включая регистр, а по синтаксису конструктор похож на метод без возвращаемого значения.

*private int Cat(); // так выглядит метод по имени Cat*

*Cat(); // так выглядит конструктор класса Cat*

В отличие от метода, конструктор никогда` ничего не возвращает.

Конструктор определяет действия, выполняемые при создании объекта класса, и является важной частью класса. Как правило, программисты стараются явно указать конструктор. Если явного конструктора нет, то Java автоматически создаст его для использования по умолчанию. Когда мы реализовывали класс Box, то никакого конструктора не создавали.

1. Какие существуют виды ссылочных типов?

**Ссылочные типы:**

* Тип object.
* Тип string.
* Классы ( class )
* Интерфейсы ( interface )

1. Что такое типы, определенные пользователем?
2. Что такое стандартные типы, определенные пользователем?

Ссылочные типы отличаются от примитивных тем, что они не определены в самом языке Java, и поэтому количество памяти, требуемое переменных этих типов, заранее знать невозможно. Пример одного из ссылочных типов - это тип массива. Массивы языка Java могут состоять из переменных любых типов, включая типы, определенные пользователем (классы).

1. В чем особенности строковых переменных?

Символьные переменные не стоит путать со строковыми, ‘a’ не идентично «a». Строковые переменные представляют объект String, который в отличие от char или int не является примитивным типов в Java.

1. Чем массивы Java отличаются от массивов других языков, их преимущества?

Массив — это структура данных, в которой хранятся элементы одного типа. Его можно представить, как набор пронумерованных ячеек, в каждую из которых можно поместить какие-то данные (один элемент данных в одну ячейку). Доступ к конкретной ячейке осуществляется через её номер. Номер элемента в массиве также называют индексом.

В случае с Java массив однороден, то есть во всех его ячейках будут храниться элементы одного типа. Так, массив целых чисел содержит только целые числа (например, типа int), массив строк — только строки, массив из элементов созданного класса будет содержать только его объекты. То есть в Java мы не можем поместить в первую ячейку массива целое число, во вторую String, а в третью – объект.

Для работы с массивами в Java есть класс java.util.Arrays (arrays на английском и означает “массивы”). В целом с массивами чаще всего проделывают следующие операции: заполнение элементами (инициализация), извлечение элемента (по номеру), сортировка и поиск. Поиск и сортировка массивов — тема отдельная. С одной стороны очень полезно потренироваться и написать несколько алгоритмов поиска и сортировки самостоятельно. С другой стороны, все лучшие способы уже написаны и включены в библиотеки Java, и ими можно законно пользоваться.

1. Как переменные различных видов передаются в качестве параметров методам?

Java передает параметры по значению. Всегда. Это означает — "скопировать значение и передать копию." При передаче параметра в метод (или конструктор) по значению в Java создаётся его локальная копия, которая действует только в пределах метода (конструктора). Для примитивных типов это легко:

*int х = 5;*

*doStuff (х); / / Передать копию х (значение 5) в метод doStuff*

Метод doStuff выглядит следующим образом:

*void doStuff (int у) {*

*/ / Действия с 'y'*

*}*

Копия значения x, тоесть 5, передается в метод doStuff (). Метод doStuff () имеет свою собственную переменную, которая называется y. Переменная y — новая, другая переменная. С копией того, что было в х на момент передачи его в метод. С этого момента, у и х не влияют друг на друга. При изменении у, вы не затрагиваете х.

1. Как ведут себя строковые переменные при передаче их в качестве параметров?

Строки в Java являются объектами, и строковые переменные являются ссылочными. Поэтому можно было бы предполагать возврат изменённого состояния строкового объекта, с которым связана переменная s3. Но всё обстоит совсем не так: при вызове

obj1.strAdd1(t1,t2,t3);

значение строковой переменной t3 не изменится. Дело в том, что в Java строки типа String являются неизменяемыми объектами, и вместо изменения состояния прежнего объекта в результате вычисления выражения s1+s2 создаётся новый объект. Поэтому присваивание s3=s1+s2 приводит к перещёлкиванию ссылки s3 на этот новый объект. А мы уже знаем, что это ведёт к тому, что новый объект оказывается недоступен вне подпрограммы – “внешняя” переменная t3 будет ссылаться на прежний объект-строку. В данном случае, конечно, лучше сделать функцию strAdd1 строковой, и возвращать получившийся строковый объект как результат вычисления этой функции.

1. Что такое совмещение методов?

Язык Java позволяет создавать несколько методов с одинаковыми именами, но с разными списками параметров. Такая техника называется совмещением методов (method overloading). В качестве примера при­ведена версия класса Point, в которой совмещение методов использовано для определения альтернативного конструктора, который инициализиру­ет координаты х и у значениями по умолчанию (-1).

class Point { int х, у;

Point(int х, int у) {

this.x = х;

this.у = у;

}

Point() {

х = -1;

у = -1;

} }

class PointCreateAlt {

public static void main(String args[]) {

Point p = new Point();

System.out.println("x = " + p.x + " у = " + p.y);

} }

В этом примере объект класса Point создается не при вызове первого конструктора, как это было раньше, а с помощью второго конструктора без параметров. Вот результат работы этой программы:

С:\> java PointCreateAlt

х = -1 у = -1

Решение о том, какой конструктор нужно вызвать в том или ином случае, принимается в соответствии с количеством и типом параметров, указанных в операторе new. Недопустимо объявлять в классе методы с одинаковыми именами и сигнатурами. В сигнатуре метода не учитываются имена формальных параметров учитываются лишь их типы и количество.

1. Что такое элементы класса и элементы экземпляра класса, чем они отличаются друг от друга? Как нужно указывать, что переменная или метод является элементом класса, а не экземпляра?

Каждый объект в Visual Basic определяется классом. Класс описывает переменные, свойства, процедуры и события объекта. Объекты являются экземплярами классов. Определив класс, вы можете создать из него любое количество объектов.

Взаимосвязь между объектом и его классом можно проиллюстрировать на примере печенья и формочки для печенья. Форма для печенья — это класс. Она определяет характеристики каждого печенья, то есть размер и форму. Класс используется для создания объектов. Отдельные печенья — это и есть объекты.

Необходимо создать объект, чтобы получить доступ к его членам, за исключением Shared элементов, к которым можно получить доступ без объекта класса.

1. Для чего используются модификаторы доступа? Какие существуют модификаторы доступа, как они ограничивают доступ к элементам?

Модификаторы доступа позволяют задать допустимую область видимости для членов класса, то есть контекст, в котором можно употреблять данную переменную или метод. В Java используются следующие модификаторы доступа:

* public: публичный, общедоступный класс или член класса. Поля и методы, объявленные с модификатором public, видны другим классам из текущего пакета и из внешних пакетов.
* private: закрытый класс или член класса, противоположность модификатору public. Закрытый класс или член класса доступен только из кода в том же классе.
* protected: такой класс или член класса доступен из любого места в текущем классе или пакете или в производных классах, даже если они находятся в других пакетах
* Модификатор по умолчанию. Отсутствие модификатора у поля или метода класса предполагает применение к нему модификатора по умолчанию. Такие поля или методы видны всем классам в текущем пакете.

Если другой класс имеет прямой доступ к этому полю, то есть вероятность, что в процессе работы программы ему будет передано некорректное значение, например, отрицательное число. Подобное изменение данных не является желательным. Либо же мы хотим, чтобы некоторые данные были достуны напрямую, чтобы их можно было вывести на консоль или просто узнать их значение. В этой связи рекомендуется как можно больше ограничивать доступ к данным, чтобы защитить их от нежелательного доступа извне (как для получения значения, так и для его изменения). Использование различных модификаторов гарантирует, что данные не будут искажены или изменены не надлежащим образом. Подобное сокрытие данных внутри некоторой области видимости называется инкапсуляцией.

1. Что позволяет делать процесс наследования?

Одним из ключевых аспектов объектно-ориентированного программирования является наследование. Процесс наследования позволяет объявить один класс наследником от другого.

1. Что такое суперкласс и подкласс?

Главный наследуемый класс в Java называют суперклассом. Наследующий класс называют подклассом. Получается, что подкласс - это специализированная версия суперкласса, которая наследует все члены суперкласса и добавляет свои собственные уникальные элементы. К примеру, в Android есть класс View и подкласс TextView.

Подкласс в свою очередь может быть суперклассом другого подкласса. Так например, упоминавший ранее класс TextView является суперклассом для EditText.

1. Что такое повторное использование кода?

Суть повторного использования кода состоит в том, что не нужно писать программный код еще раз. Намного эффективнее использовать ранее созданный и проверенный (протестированный) программный код. Для этого лучше всего подходят классы, которые вобрали в себя все преимущества объектно-ориентированного подхода к программированию.

До появления объектно-ориентированного программирования существовало процедурно-ориентированное программирование. В процедурно-ориентированном подходе для того, чтобы использовать повторно программный код, использовался метод копирования этого кода. Такой подход не был достаточно эффективен и, при увеличении кода программной системы, вызывал стремительный рост ошибок.

1. Какие заранее определенные переменные содержит каждый класс Java?

Каждый объект содержит свои копии переменных экземпляра, определенные в его классе. Следовательно, содержимое переменных в одном объекте может отличаться от содержимого тех же самых переменных в другом объекте. Между объектами нет никакой связи, за исключением того, что они относятся к одному и тому же типу.

1. Что можно сделать при помощи переменной this? Что можно сделать при помощи переменной super?

Ключевое слово this представляет ссылку на текущий экземпляр класса. Через это ключевое слово мы можем обращаться к переменным, методам объекта, а также вызывать его конструкторы.

C помощью this можно вызвать один из конструкторов класса и передать для его параметров необходимые значения:

*Person(String name)*

*{*

*this(name, 18);*

*}*

1. Что такое скрытие переменной, затемнение переменной и замещение метода?

Класс B наследует метод foo. Но переменная x в B скрывает x в A; он не заменяет его.

Это вопрос масштаба. Метод foo в A видит только переменные, которые находятся в области видимости. Единственной переменной в области видимости является переменная экземпляра x в A:

*class A*

*{*

*int x = 5;*

*void foo()*

*{*

*System.out.println(this.x);*

*}*

*}*

*class B extends A*

*{*

*int x = 6;*

*// some extra stuff*

*}*

*class C*

*{*

*public static void main(String args[])*

*{*

*B b = new B();*

*System.out.println(b.x);*

*System.out.println(((A)b).x);*

*b.foo();*

*}*

*}*

1. Как импортировать классы из пакетов?

Если нам надо использовать классы из других пакетов, то нам надо подключить эти пакеты и классы. Исключение составляют классы из пакета java.lang (например, String), которые подключаются в программу автоматически.

Например, знакомый по прошлым темам класс Scanner находится в пакете java.util, поэтому мы можем получить к нему доступ следующим способом:

*java.util.Scanner in = new java.util.Scanner(System.in);*

То есть мы указываем полный путь к файлу в пакете при создании его объекта. Однако такое нагромождение имен пакетов не всегда удобно, и в качестве альтернативы мы можем импортировать пакеты и классы в проект с помощью директивы import, которая указывается после директивы package:

*package study;*

*import java.util.Scanner; // импорт класса Scanner*

*public class Program {*

*public static void main(String[] args) {*

*Scanner in = new Scanner(System.in);*

*}*

*}*

1. Как использовать явные ссылки на классы из пакетов?

Директива import указывается в самом начале кода, после чего идет имя подключаемого класса (в данном случае класса Scanner).

В примере выше мы подключили только один класс, однако пакет java.util содержит еще множество классов. И чтобы не подключать по отдельности каждый класс, мы можем сразу подключить весь пакет:

*import java.util.\*; // импорт всех классов из пакета java.util*

Теперь мы можем использовать любой класс из пакета java.util.

Возможна ситуация, когда мы используем два класса с одним и тем же названием из двух разных пакетов, например, класс Date имеется и в пакете java.util, и в пакете java.sql. И если нам надо одновременно использовать два этих класса, то необходимо указывать полный путь к этим классам в пакете:

*java.util.Date utilDate = new java.util.Date();*

*java.sql.Date sqlDate = new java.sql.Date();*

1. Как добавить класс в пакет?

Чтобы поместить класс в пакет, нужно указать имя пакета в начале исходного файла перед определением класса. Например, файл Employee.java начинается следующими строками:

*package com.projava.corejava;*

*public class Employee {*

*.....*

*}*

1. Как загрузить графическое изображение из файла?

Загрузка графического изображения из файла, формируя объект класса Image:

*img = Toolkit.getDefaultToolkit().getImage(*

*szCurrentFilename);*

Далее мы дожидаемся завершения процесса загрузки:

*mt = new MediaTracker(this);*

*mt.addImage(img, 0);*

*try*

*{*

*mt.waitForAll();*

*}*

*catch (InterruptedException ex) { }*

Перед отображением содержимого загруженного файла мы проверяем поле sp:

*if(sp != null)*

*{*

*this.remove(sp);*

*}*

1. Как ускорить вывод графических изображений, загружаемых из файла?

Класс Java BufferedImage является подклассом класса Image. Он используется для обработки и управления данными изображения. BufferedImage сделан из ColorModel данных изображения. Все объекты BufferedImage имеют координату верхнего левого угла (0, 0). Данный класс имеет методы для ускорения вывода графического изображения.

* copyData (WritableRaster outRaster)

Он вычисляет произвольную прямоугольную область BufferedImage и копирует ее в указанный WritableRaster.

* getColorModel ()

Возвращает объект класса ColorModel изображения.

* getData (прямоугольник прямоугольник)

Он вычисляет и возвращает произвольную область BufferedImage .

* GetGraphics ()

Этот метод возвращает Graphics2D, сохраняет обратную совместимость.

* GetHeight ()

Возвращает высоту BufferedImage .

* getMinX ()

Возвращает минимальную координату x этого BufferedImage .

* getMinY ()

Возвращает минимальную координату y этого BufferedImage .

* getRGB (int x, int y)

Возвращает целочисленный пиксель в стандартной цветовой модели RGB (TYPE\_INT\_ARGB) и цветовом пространстве sRGB по умолчанию.

* GetType ()

Возвращает тип изображения.

1. Что называется событием, когда они генерируются?

Событие (event) в библиотеке AWT возникает при воздействии на компонент какими-нибудь манипуляциями мышью, при вводе с клавиатуры, при перемещении окна, изменении его размеров.

Объект, в котором произошло событие, называется источником (source) события.

Все события в AWT классифицированы. При возникновении события исполняющая система Java автоматически создает объект соответствующего событию класса. Этот объект не производит никаких действий, он только хранит все сведения о событии.

Во главе иерархии классов-событий стоит класс Eventobject из пакета java.utii — непосредственное расширение класса object. Его расширяет абстрактный класс AWTEvent из пакета java.awt — глава классов, описывающих события библиотеки AWT. Дальнейшая иерархия классов-событий показана на рис. 12.1. Все классы, отображенные на рисунке, кроме класса AWTEvent, собраны в пакет java.awt.event.

Генерация событий происходит при выполнении определенных действий:

* События типа windowevent возникают только в окнах: frame, dialog, filedialog, window.
* События типа textevent генерируются только в контейнерах textcomponent, textarea, textfield.
* События типа actionevent проявляются только в контейнерах button, list, textfield.
* События типа itemevent возникают только в контейнерах checkbox, choice, list.
* Наконец, события типа adjustmentevent возникают только в контейнере scrollbar.

1. Какие обработчики событий используются чаще всего?

Существует семь типов window-событий. Чтобы идентифицировать

их, класс WindowEvent определяет следующие целые константы:

* WINDOW\_ACTWATED. Окно активизировано.
* WINDOW\_CLOSED. Окно закрыто.
* WINDOW\_CLOSING. Пользователь потребовал, чтобы окно было закрыто.
* WINDOW\_DEACTWATED. Окно деактивизировано.
* WINDOW\_DEICONIFIED. Окно развернуто из пиктограммы.
* WINDOW\_ICONIFIED. Окно свернуто в пиктограмму.
* WINDOW\_OPENED. Окно открыто.

WindowEvent — подкласс ComponentEvent и имеет следующий конструктор:

WindowEvent (Window src, int type), где src — ссылка на компонент, который генерировал это событие; type указывает тип события.

В этом классе чаще всего используется метод getWindow(). Он возвращает window-объект, который сгенерировал событие. Его общая форма:

Window getWindow()