Java
Введение

© Составление, Гаврилов А.В., Будаев Д.С., Стефанов М.А. 2019

Костанян Мамикон
Каренович

t.me/makost96

NetCracker®

Лекция 1.1

Самара
2020

План лекции

- История языка Java и его особенности
- Объектно-ориентированное программирование, основные понятия
- Пакеты в Java
- Правила именования

Предыстория Java

- Старт проекта Green (1991)
  - Патрик Нотон, Джеймс Гослинг, Майк Шеридан
  - Идея Гослинга об "универсальном пульте"
  - Модификации Гослингом языка C++
  - Начало работ над OaK, "технология молотка"
- Первая демонстрация star7 (1992)

Предыстория Java

Идеи, заложенные в OaK, проект Green:

- Надежность и механизмы безопасности
- Работа на разных типах устройств
- Объектная ориентированность
- Объекты, доступные по сети

Предыстория Java

- **1991**
  Начало работ над Oak
- **1993**
  Работы в области интерактивного TV
  Появление браузера Mosaic
- **1994**
  Браузер WebRunner
- **1995**
  Официальное представление Java
  Включение в Netscape Navigator 2.0

Java timeline

- **1996** – JDK 1.0 (JLS, JVM, JDK)
- **1997** – JDK 1.1 (JIT, JavaBeans, JDBC, RMI-IIOP)
- **1998** – JDK 1.2 (изменения языка, policy/permission, JFC/Swing, EJB)
- **1999** – JSP + JINI + разделение развития
  - Java 2 Platform, Standard Edition (J2SE, JavaSE)
  - Java 2 Platform, Enterprise Edition (J2EE, JavaEE)
  - Java 2 Platform, Micro Edition (J2ME, JavaME)
- **2000** – JDK 1.3 (HotSpot (JIT) в составе JVM, JAXP), Java ME 1.0
- **2001** – Java EE 1.3 (EJB 2.0)
- **2002** – JDK 1.4 (новое API), Java ME 2.0
- **2003** – Java EE 1.4

Java timeline

- **2004** – JDK 1.5 (изменения языка)
- **2006** – JDK 1.6 (Java – opensourced, скриптовые языки, JDBC4) Java EE 5
- **2008** – Java FX 1.0
- **2009** – Java EE 6
- **2011** – JDK 7 (изменения языка…)
- **2013** – Java EE 7
- **2014** – JDK 8 (функциональные особенности…)
- **2017** – JDK 9 и Java EE 8
- **2018** – JDK 10
- **2018** – JDK 11
- **2019** – JDK 12, JDK 13
- **2020** – JDK 14

Особенности Java

- Строгая типизация
- Кросс-платформенность
- Объектная ориентированность
- Встроенная модель безопасности
- Ориентация на интернет-задачи, распределенные приложения
- Динамичность, легкость развития
- Легкость в освоении

Особенности Java

Строгая типизация

контроль типов производится как на этапе компиляции, так и выполнения

Объектная ориентированность

любая программа состоит хотя-бы из одного класса

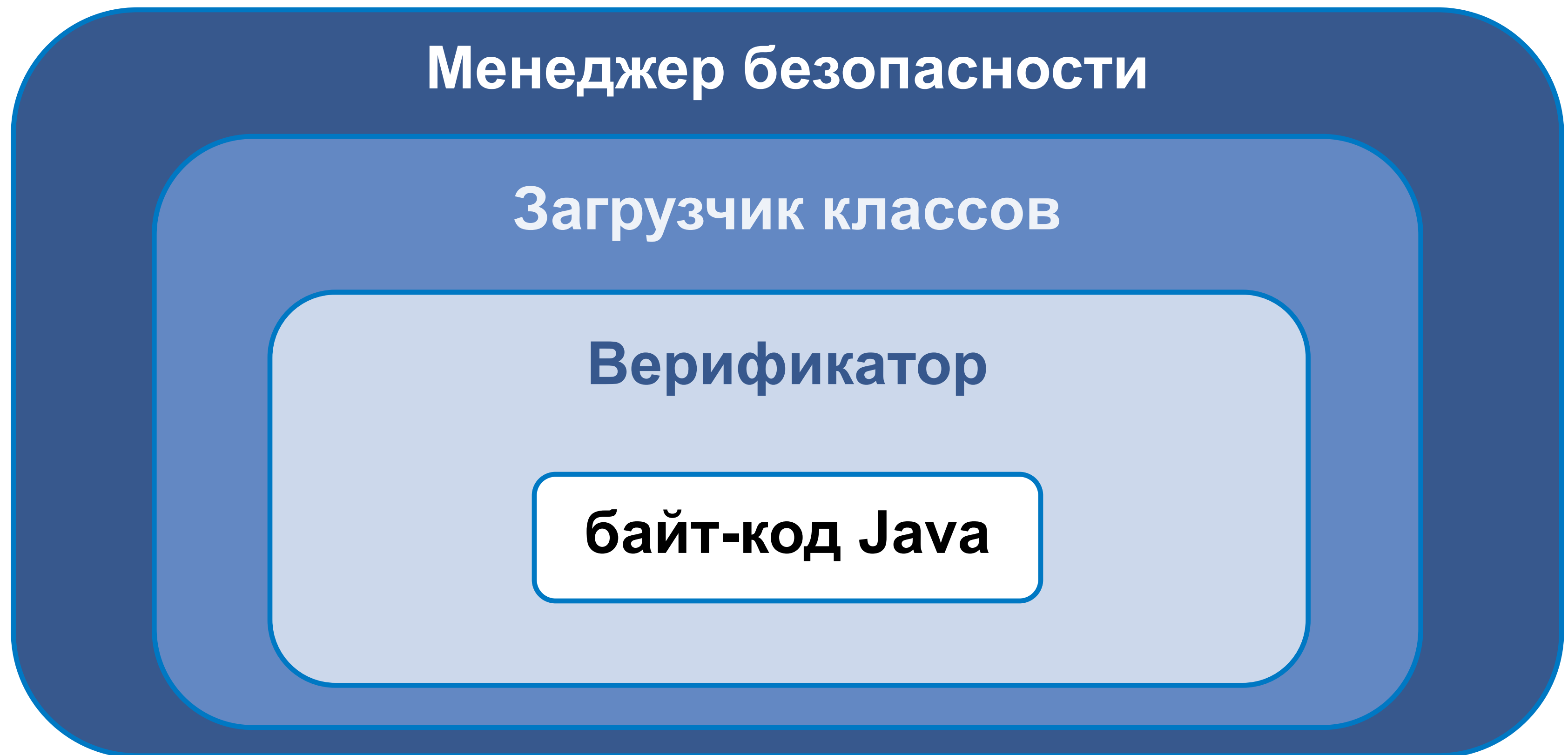
Особенности Java

Кросс-платформенность

исходный код программы **почти** не зависит от платформы (он, например, может зависеть от поставщика VM – есть реализации IBM, Oracle, Google…, или если программа на разных платформах должна вести себя по-разному)

Особенности Java

Встроенная модель безопасности



Модель безопасности

Верификатор

Проверяет байт-код на корректность

- Соответствие правилам языка Java
- Соответствие типов в любой точке программы (состояние типов стека)

Модель безопасности

Загрузчик классов

Отвечает за доставку байт-кода для классов Java в интерпретатор

- Каждый класс связан со своим загрузчиком
- Используется модель делегирования загрузки классов
- Есть базовый и системный загрузчики классов

Модель безопасности

Менеджер безопасности

Отвечает за принятие решений по безопасности на уровне приложения

- Контролирует доступ к ресурсам системы
- Основан на политиках безопасности (наборе правил, описывающих что и кому разрешено или запрещено)

Особенности Java

Ориентация на интернет-задачи и распределенные приложения

Язык изначально создавался для построения распределенных систем разнородных устройств

Динамичность

За 20 лет появилось 9 основных версий JDK и по сотне доработанных на каждую версию

Легкость в освоении

Синтаксис языка достаточно прост

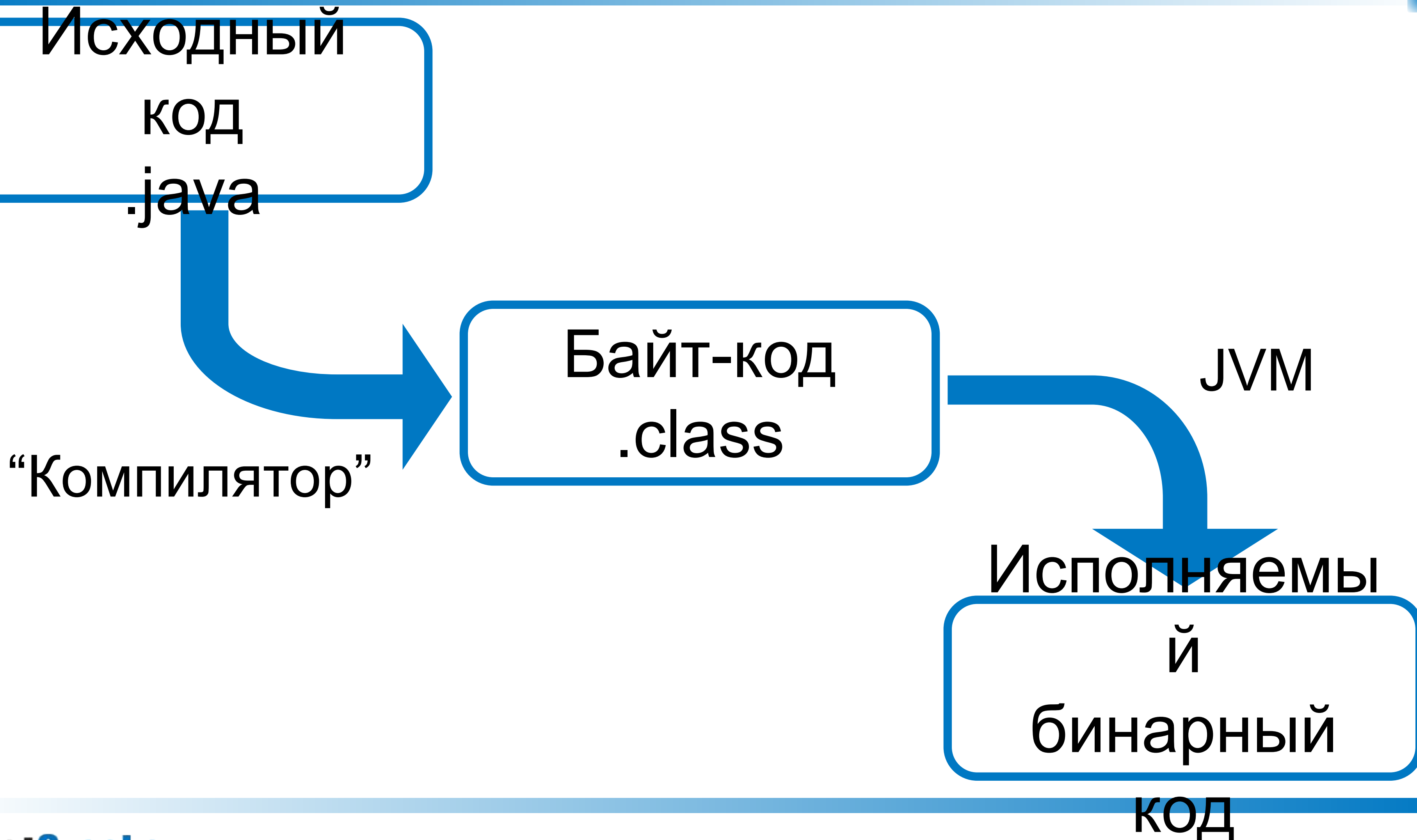
Java платформа

- Множество различных аппаратных систем
  - Intel x86, Sun SPARC, PowerPC и др.
- Множество разных программных систем
  - MS Windows, Sun Solaris, Linux, Mac OS и др.
- Возможность выполнения одного и того же кода на разных платформах реализуется за счет Java Virtual Machine (JVM)
  - Исходный код открыт с 1999 г.

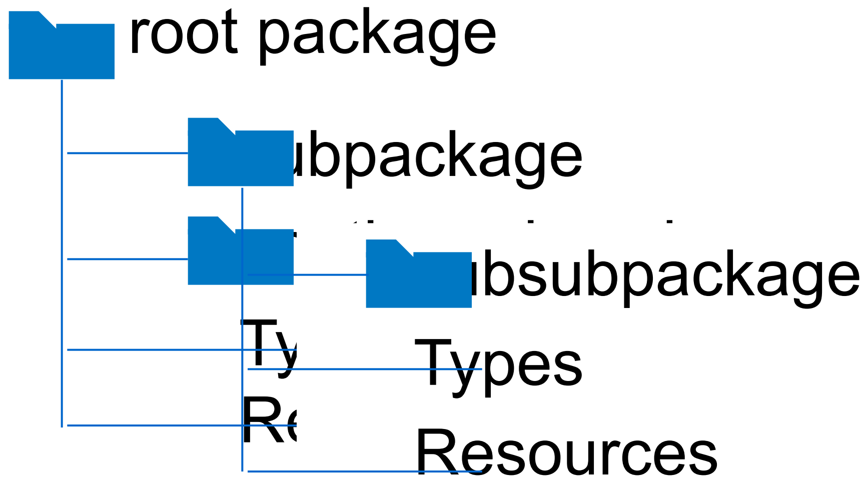
Именование установочных файлов

- Старый вариант
  `jdk-1_5_0_08-windows-i586-p.exe`
  - 1 – глобальная версия языка
  - 5 – номер версии языка
  - 0 – номер подверсии
  - 08 – номер модификации
  - windows-i586 – платформа
- Новый вариант
  `jdk-8u92-macosx-x64.dmg`
  - 8 – номер версии языка
  - 92 – номер модификации
  - macosx-x64– платформа

Разработка и запуск программ



Структура Java-приложения



Class

- Текстовые файлы
  - логи
  - бандлы i18n
  - файлы настроек
  - параметры менеджера безопасности

…

- Бинарные
  - изображения
  - A\V

Объекты и классы

Объект

- Состояние
- Поведение
- Уникальность

Класс

- Объекты имеют одинаковый набор свойств
- Объекты имеют общее поведение

Основные принципы ООП

- **Инкапсуляция**

  объединение данных и методов их обработки в одну сущность, приводящее к сокрытию реализации класса и отделению его внутреннего представления от внешнего

- **Наследование**

  отношение между классами, при котором один класс использует структуру или поведение другого (одиночное наследование) или других (множественное наследование) классов

- **Полиморфизм**

  способность объекта соответствовать во время выполнения двум или более типам. С другой стороны, способность различных объектов предлагать свои собственные реализации одного и того же интерфейса

Отношения между классами

- **Ассоциация**
  Объекты классов вступают во взаимодействие между собой
- **Наследование**
  Объекты дочернего класса наследуют состояние и поведение родительского класса
- **Агрегация**
  Объекты разных классов образуют целое, оставаясь самостоятельными
- **Композиция**
  Объекты одного класса входят в объекты (являются частью) другого, не обладая самостоятельностью
- **Класс-метакласс**
  Экземплярами класса являются классы

Достоинства ООП

- **Упрощение разработки**
  Разделение функциональности, локализация кода, инкапсуляция
- **Возможность создания расширяемых систем**
  Обработка разнородных структур данных, изменение поведения на этапе выполнения, работа с наследниками (обобщение алгоритмов, полуфабрикаты, каркасы)
- **Легкость модернизации с сохранением совместимости**
- **Повторное использование ранее созданных компонентов**

Недостатки ООП

- Неэффективность на этапе выполнения
- Неэффективность в смысле распределения памяти
- Излишняя избыточность
- Психологическая сложность проектирования
- Техническая сложность проектирования и документирования
- Сложность в изучении API

Объектный язык Java

- **Все** сущности в Java являются объектами, классами, интерфейсами или перечислениями
- **Строгая** реализация инкапсуляции
- Реализовано одиночное наследование от класса и **множественное** от интерфейсов

Понятие о пакетах

- Способ логической группировки **типов**
- Комплект ПО, распространяющийся независимо или применяющийся в сочетании с другими пакетами
- Пакеты содержат:
  - типы
  - вложенные пакеты,
  - дополнительные файлы ресурсов
- Все типы **должны** принадлежать какому-либо пакету

Функциональность пакетов

- Позволяют группировать взаимосвязанные типы
- Способствуют созданию пространств имен, позволяющих избежать конфликтов идентификаторов, относящихся к различным типам
- Обеспечивают дополнительные средства защиты элементов кода
- Формируют иерархическую систему

Способы реализации и доступ к пакетам

- Пакеты могут быть реализованы:
  - в виде структуры каталогов (каждый пакет - это каталог) с файлами (каждый файл – тип)
  - в виде jar-архива (zip архив структуры каталогов с дополнительными метаданными).
- Путь к используемым пакетам указывается:
  - непосредственно при запуске JVM (ключ –cp или –classpath),
  - через переменную окружения CLASSPATH (по умолчанию CLASSPATH=""). Пути в ней разделяются символом «;»
  - также используются пакеты «текущего каталога» (из которого запускается JVM)

Понятие имени

- Имена задаются посредством идентификаторов, указывают на компоненты программы
- Пространства имен
  - пакеты
  - типы
  - поля
  - методы
  - локальные переменные и параметры
  - метки
- Имена бывают
  составные или полные (`java.lang.Double`)
  и простые (`Double`)

Классический пример корректного кода

Пример зависимости имени от контекста

```
package Reuse;

class Reuse {
   Reuse Reuse (Reuse Reuse) {
      Reuse:
      for(;;) {
         if (Reuse.Reuse(Reuse) == Reuse)
            break Reuse;
      }
      return Reuse;
   }
}
```

Все тот же корректный код

Контексты подсвечены разными цветами

```
package Reuse;

class Reuse {
  Reuse Reuse (Reuse Reuse) {
    Reuse:
    for(;;) {
      if (Reuse.Reuse(Reuse) == Reuse)
        break Reuse;
    }
    return Reuse;
  }
}
```

Понятие модуля компиляции

- Модуль компиляции хранится в `.java` файле и является единичной порцией входных данных для компилятора.

- Состоит из:
  - объявления пакета
    ```
    package mypackage;
    ```
  - выражений импортирования
    ```
    import java.net.Socket;
    import java.io.*;
    ```
  - объявлений верхнего уровня – классов, интерфейсов, перечислений

Объявление пакета

- Первое выражение в модуле компиляции (например, для файла `java/lang/Object.java`)
  - `package java.lang;`
- При отсутствии объявления пакета модуль компиляции принадлежит безымянному пакету (не имеет вложенных пакетов)
- Пакет доступен, если доступен модуль компиляции с объявлением пакета
- Ограничение на доступ к пакетам нет

Выражения импорта

- Доступ к типу из данного пакета – по простому имени типа
- Доступ к типу из других пакетов – по составному имени пакета и имени типа
  - сложности при многократном использовании
- import-выражения упрощают доступ
  - импорт одного типа (`import java.net.URL;`)
  - импорт пакета с типами (`import java.net.*;`)

Выражения импорта

- Попытка импорта пакета, недоступного на момент компиляции, вызовет ошибку
- Дублирование импорта игнорируется
- Нельзя импортировать вложенный пакет
  - `import java.net; //ошибка компиляции`
- При импорте типов пакета вложенные пакеты не импортируются!

Выражения импорта

- Алгоритм компилятора при анализе типов:
  - выражения, импортирующие типы
  - другие объявленные типы
  - выражения, импортирующие пакеты
- Если тип импортирован явно, невозможны:
  - объявление нового типа с таким же именем
  - доступ по простому имени к одноименному типу в текущем пакете

Выражения импорта

- Импорт пакета не мешает объявлять новые типы или обращаться к имеющимся типам текущего пакета по простым именам
  - поиск типа сначала в текущем пакете
- Импорт конкретных типов дает возможность при прочтении кода сразу понять, какие внешние типы используются
  - но эти типы могут и не использоваться

Статический импорт

- Импорт элемента типа

```
import static pkg.TypeName.staticMemberName;
```

```
import static java.lang.Math.sqrt;
import static java.lang.Math.pow;
```

- Импорт всех элементов типа

```
import static pkg.TypeName.*;
```

```
import static java.lang.Math.*;
```

Пример

- Без статического импорта:

```
hypot = Math.sqrt(Math.pow(side1, 2)
              + Math.pow(side2, 2));
```

- `import static java.lang.Math.sqrt`
- `import static java.lang.Math.pow`

```
hypot = sqrt(pow(side1, 2)
           + pow(side2, 2));
```

Особенности статического импорта

- Повышает удобство написания программ и уменьшает объем кода
- Уменьшает удобство чтения программ
- Приводит к конфликтам имен
- Мораль: рекомендуется к использованию только когда действительно необходим

Объявление верхнего уровня

```
package first;
import …;
class MyFirstClass { … }
interface MyFirstInterface { … }
enum MyFirstEnumeration { … }
```

- Область видимости типа – пакет
- Доступ к типу извне его пакета
  - по составному имени
  - через выражения импорта
- Разграничение (модификаторы) доступа

Объявление верхнего уровня

- В модуле компиляции может быть максимум один `public` тип
- Имя публичного типа и имя модуля компиляции должны совпадать
- Другие не-`public` типы модуля должны использоваться только внутри текущего пакета
- Как правило, один модуль компиляции содержит один тип

Правила именования

- Пакеты
  `java.lang`, `javax.swing`, `ru.ssau.infokom`
- Типы
  `Student`, `ArrayIndexOutOfBoundException`
  `Cloneable`, `Runnable`, `Serializable`
- Поля
  `value`, `enabled`, `distanceFromShop`
- Методы
  `getValue`, `setValue`, `isEnabled`, `length`, `toString`
- Поля-константы
  `PI`, `SIZE_MIN`, `SIZE_MAX`, `SIZE_DEF`
- Локальные переменные
  `value`, `isFound`

Лексика языка Java

© Составление, Гаврилов А.В., 2016

Лекция 1.2

Самара
2020

NetCracker®

План лекции

- Структура исходного кода и его элементы
- Типы данных
- Описание классов
  - Общая структура
  - Поля
  - Методы
  - Конструкторы
  - Блоки инициализации
- Точка входа программы

Кодировка

- Java ориентирован на Unicode
- Первые 128 символов почти идентичны набору ASCII
- Символы Unicode задаются с помощью escape-последовательностей
  `\u262f`, `\uu2042`, `\uuu203d`
- **Java чувствителен к регистру!**

Исходный код

Исходный код разделяется на:

- Пробелы
  - ASCII-символ SP, \u0020, дес. код 32
  - ASCII-символ HT, \u0009, дес. код 9
  - ASCII-символ FF, \u000c, дес. код 12
  - ASCII-символ LF, символ новой строки
  - ASCII-символ CR, возврат каретки
  - символ CR, за которым сразу следует символ LF
- Комментарии
- Лексемы

Комментарии

- `// Комментарий`
  Символы после `//` и до конца текущей строки игнорируются
- `/* Комментарий */`
  Все символы, заключенные между `/*` и `*/`, игнорируются
- `/** Комментарий */`
  Комментарии документирования

Комментарии документирования (javadoc)

- Начинаются с `/**`, заканчиваются `*/`
- В строках начальные символы `*` и пробелы перед ними игнорируются
- Допускают использование HTML-тэгов, кроме заголовков
- Специальные тэги `@see`, `@param`, `@deprecated`

Лексемы

- Идентификаторы
- Служебные слова
  `class`, `public`, `const`, `goto`
- Литералы
- Разделители
  `{ } [ ] ( ) ; . ,`
- Операторы
  `= > < ! ? : == && ||`

Идентификаторы

- Имена, задаваемые элементам языка для упрощения доступа к ним
- Можно записывать символами Unicode
- Состоят из букв и цифр, знаков _ и $
- Не допускают совпадения со служебными словами, литералами **`true`**, **`false`**, **`null`**
- Длина имени не ограничена

Служебные (ключевые) слова

abstract	double	int	strictfp
boolean	else	interface	super
break	extends	long	switch
byte	final	native	synchronized
case	finally	new	this
catch	float	package	throw
char	for	private	throws
class	goto	protected	transient
const	if	public	try
continue	implements	return	void
default	import	short	volatile
do	instanceof	static	while

Типы данных

- **Ссылочные**
  - Предназначены для работы с объектами
  - Переменные содержат ссылки на объекты
  - Ссылка – это не указатель!
  - Тип переменной определяет контракт доступа к объекту

- **Примитивные (простые)**
  - Предназначены для работы со значениями естественных, простых типов
  - Переменные содержат непосредственно значения

Ссылочные типы

- К ссылочным типам относятся типы классов (в т.ч. массивов) и интерфейсов
- Переменная ссылочного типа способна содержать ссылку на объект, относящийся к этому типу
- Ссылочный литерал `null`

Примитивные типы

- Булевский (логический) тип
  - `boolean` – допускает хранение значений true или false
- Целочисленные типы
  - `char` – 16-битовый символ Unicode
  - `byte` – 8-битовое целое число со знаком
  - `short` – 16-битовое целое число со знаком
  - `int` – 32-битовое целое число со знаком
  - `long` – 64-битовое целое число со знаком
- Вещественные типы
  - `float` – 32-битовое число с плавающей точкой (IEEE 754-1985)
  - `double` – 64-битовое число с плавающей точкой (IEEE 754-1985)

Литералы

- Булевы
  `true false`
- Символьные
  `'a' '\n' '\\' '\377' '\u0064'`
- Целочисленные
  `29  035  0x1D  0X1d 0xffffL`
  `0b11110000 0B11110000 (Java 1.7)`
  - По умолчанию имеют тип `int`
- Числовые с плавающей запятой
  `1.  .1  1e1 1e-4D 1e+5f`
  - По умолчанию имеют тип `double`
- Строковые
  `"Это строковый литерал"   ""`

Подчеркивание в числовых литералах (Java 1.7)

- Можно использовать
  - В литералах любых числовых типов
    `765_324_213_434L`
  - В литералах в любых системах счисления
    `0xFF_00_FF_00`
  - В нужных местах числа
    `1_23_456_7890`
  - В нужном количестве
    `6______6`

Подчеркивание в числовых литералах (Java 1.7)

- Нельзя использовать
  - В начале и в конце числа
    `_123     123_`
  - Рядом с разделителем целой и дробной части
    `10_.01 10._01`
  - Перед суффиксами `L`, `F` и `D`
    `1_L     1.1_F     1.1_D`
  - В строковых литералах с числами
    `"6_____6"`

NetCracker®

Описание класса

Класс может содержать:

- поля,
- методы,
- вложенные типы (классы, интерфейсы, перечисления).

```
class Body {
  public long idNum;
  public String name;
  public Body orbits;

  public static long nextID = 0;
}
```

NetCracker®

Модификаторы объявления класса

- `public`
  Признак общедоступности класса. В одном файле может содержаться определение только одного public-класса
- `abstract`
  Признак абстрактности класса
- `final`
  Завершенность класса (класс не допускает наследования)
- `strictfp`
  Повышенные требования к операциям с плавающей точкой

Поля класса

- Объявление поля:

```
[модификаторы] <тип> {<имя> [=
<инициализирующее выражение>]};
double sum = 2.5 + 3.7;
public double val = sum + 2 *
Math.sqrt(2);
```

- Если поле явно не инициализируются, ему присваивается значение по умолчанию его типа (`0`, `false` или `null`)

Поля класса

- Модификаторы полей:
  - модификаторы доступа
  - **`static`**
    поле статично (принадлежит контексту класса)
  - **`final`**
    поле не может изменять свое значение после инициализации
  - **`transient`**
    поле не сериализуется (влияет только на механизмы сериализации)
  - **`volatile`**
    усиливает требования к работе с полем в многопоточных программах

Методы

- Объявление метода:
  [модификаторы] <тип> <сигнатура> [throws исключения] {<тело>}
- Если метод ничего не возвращает, то тип - **void**

```
class Primes {
  static int nextPrime(int current) {
    <Вычисление простого числа в теле метода>
    return nextPrime;
  }
  static void printPrimes(int bound, OutputStream out) {
    <Вывод простых чисел от 1 до bound в поток out>
  }
}
```

Модификаторы методов

- Модификаторы доступа
- `abstract`
  абстрактность метода (тело при этом не описывается)
- `static`
  метод принадлежит контексту класса
- `final`
  завершенность метода (метод не может быть переопределен при наследовании)
- `default` **(Java 1.8)**
  реализация метода по-умолчанию, используется в интерфейсах

Модификаторы методов

- `synchronized`
  синхронизированность метода (особенности вызова метода в многопоточных приложениях)
- `native`
  «нативность» метода (тело метода не описывается, при вызове вызывается метод из native-библиотеки)
- `strictfp`
  повышенные требования к операциям с плавающей точкой

Особенности методов

- Для нестатических методов вызов через ссылку на объект или в контексте объекта

```
reference.method();
methodReturningReference().method();
```

- Для статических методов вызов через имя типа, через ссылку на объект или в контексте класса

```
ClassName.staticMethod();
reference.staticMethod();
staticMethodReturningReference().method();
```

- Наличие круглых скобок при вызове обязательно, т.к. они являются оператором вызова метода

Особенности методов

- На время выполнения метода управление передается в тело метода
- Возврат управления осуществляется либо после выполнения оператора **`return`**, либо после выполнения последней инструкции (в случае, если метод ничего не возвращает)
- Возвращается одно значение простого или ссылочного типа **`return someValue;`**
- Аргументы передаются по значению, т.е. значения параметров копируются в стек:
  - для примитивных типов копируются сами значения
  - для ссылочных типов копируется значение ссылки
- Перегруженными являются методы с одинаковыми именами и различными входными параметрами

Переменное количество аргументов (Java 1.5)

- Синтаксис

```
[modifiers] type methName(type ... arg){…}
```

- Пример метода

```
int sum(int ... a) {
   int s = 0;
   for (int i = 0; i < a.length; i++)
      s += a[i];
}
```

- Пример вызова

```
int s2 = sum(1, 2, 3);
int s1 = sum(new int[] {1, 2});
```

Особенности переменного количества аргументов

- Внутри там все равно живет массив…
- Аргумент переменной длины в методе может быть только один
- Аргумент переменной длинны должен быть последним в списке аргументов метода
- В сочетании с перегрузкой методов способен приводить к изумительным ошибкам компиляции в виду неоднозначности кода

Создание объектов

- Создание ссылки и создание объекта – различные операции
- Используется оператор **new**, он возвращает ссылку на объект
- После оператора указывается имя конструктора и его параметры

```
Body sun;
sun = new Body();
sun.idNum = Body.nextID++;
sun.name = "Sun";
sun.orbits = null;

Body earth = new Body();
earth.idNum = Body.nextID++;
earth.name = "Earth";
earth.orbits = sun;
```

Конструкторы

- Память для объекта выделяет оператор `new`
- Конструкторы предназначены для формирования начального состояния объекта
- Правила написания конструктора сходны с правилами написания методов
- Имя конструктора совпадает с именем класса

Конструкторы

- Для конструкторов разрешено использование только модификаторов доступа
- Конструктор не имеет возвращаемого типа
- Оператор возврата **`return`** прекращает выполнение текущего конструктора
- Конструкторы могут быть перегружены
- Конструкторы могут вызывать друг друга с помощью ключевого слова **`this`** в первой строке конструктора

Конструкторы

- Если в классе явно не описан ни один конструктор, автоматически создается т.н. конструктор по умолчанию
- Если в классе описан хотя бы один конструктор, то автоматически конструктор по умолчанию не создается
- Конструктором по умолчанию называют конструктор, не имеющий параметров

Конструкторы

```
class Body {
  public long idNum;
  public String name;
  public Body orbits;

  private static String NAME_DEFAULT = "No Name";
  private static long nextID = 0;

  Body() {
    this(NAME_DEFAULT, null);
  }
  Body(String name, Body orbits) {
    idNum = nextID++;
    this.name = name;
    this.orbits = orbits;
  }
}
```

Деструкторы?

- В ряде языков деструкторы выполняют действия, обратные действию конструкторов: освобождают память, занимаемую объектом, и «деинициализируют» объект (освобождают ресурсы, очищают связи, изменяют состояние связанных объектов)
- Если после вызова деструктора где-то осталась ссылка (указатель) на объект, ее использование приведет к возникновению ошибки
- В Java деструкторов нет, вместо них применяется механизм автоматической сборки мусора

Автоматическая сборка мусора

- В случае нехватки памяти для создания очередного объекта виртуальная машина находит недостижимые объекты и удаляет их
- Процесс сборки мусора можно инициировать принудительно
- Для явного удаления объекта следует утратить все ссылки на этот объект и инициировать сбор мусора
- Взаимодействие со сборщиком осуществляется через системные классы `java.lang.System` и `java.lang.Runtime`

Блоки инициализации

- Если некоторые действия по инициализации должны выполняться в любом варианте создания объекта или для возможности обработки исключений, удобнее использовать блоки инициализации
- Тело блока инициализации заключается в фигурные скобки и располагается на одном уровне с полями и методами
- Порядок инициализации полей объекта

1. Все поля инициализируются значениями по-умолчанию
2. Выполняются инициализирующие выражения полей и блоки инициализации (в порядке их описания в классе)
3. Выполняется тело конструктора

Блоки инициализации

```
class Body {
  public long idNum;
  public String name = "No Name";
  public Body orbits;

  private static long nextID = 0;

  {
    idNum = nextID++;
  }

  Body(String name, Body orbits) {
    this.name = name;
    this.orbits = orbits;
  }
}
```

Статическая инициализация

```
class Primes {
  static int[] knownPrimes = new int[4];

  static {
    knownPrimes[0] = 2;
    for (int i=1; i<knownPrimes.length; i++)
      knownPrimes[i] = nextPrime(i);
  }

  //nextPrime() declaration etc.
}
```

- Статический блок инициализации выполняет инициализацию контекста класса
- Вызов статического блока инициализации происходит в процессе загрузки класса в виртуальную машину

Модификаторы доступа

- `private`
  Доступ только в контексте класса
- `package (none) (доступ по-умолчанию)`
  Доступ для самого класса и классов в том же пакете
- `protected`
  Доступ в пределах самого класса, классов-наследников
  и классов пакета
- `public`
  Доступ есть всегда, когда доступен сам класс

Перечисления

- В ранних версиях Java:

```
class Apple {
  public static final int JONATHAN = 0;
  public static final int GOLDENDEL = 1;
  public static final int REDDEL = 2;
  public static final int WINESAP = 3;
  public static final int CORTLAND = 4;
}
```

- Java 1.5:

```
enum Apple {
  Jonathan, GoldenDel, RedDel, Winesap, Cortland
}
```

Перечислимые типы

- Перечислимый тип
  `Apple`
- Константы перечислимого типа
  `Jonathan`, **`GoldenDel`**, **`RedDel`**…
- Объявление переменной

```
Apple ap;
```

- Присвоение переменной значения

```
ap = Apple.RedDel;
```

- Проверка равенства

```
if (ap == Apple.GoldenDel)
```

NetCracker®

А теперь отличия от классики

- Перечислимый тип – это класс!
- Да к тому же имеет методы!
  - `public static enumType[] values()` возвращает ссылку на массив ссылок на все константы перечислимого типа

```
Apple[] allApples = Apple.values();
```

  - `public static enumType valueOf(String str)` возвращает константу перечислимого типа, имя которой соответствует указанной строке, иначе выбрасывает исключение

```
Apple ap = Apple.valueOf("Jonathan");
```

И еще отличия…

- Можно определять конструкторы (только приватные), добавлять поля и методы, реализовывать интерфейсы

```
enum Apple {
  Jonathan(10), GoldenDel(9), RedDel, Winsap(15), Cortland(8);
  private int price;
  Apple(int p) {
    price = p;
  }
  Apple() {
    price = -1;
  }
  int getPrice() {
    return price;
  }
}
```

Особенности перечислимых типов

- Создавать экземпляры с помощью оператора `new` нельзя!
- Все перечислимые типы наследуют от класса `java.lang.Enum`
- Клонировать экземпляры нельзя, сравнивать и выполнять прочие стандартные операции – можно

Точка входа программы

- Метод
- `static`
- `public`
- С параметрами-аргументами
- Без возвращаемого значения

```
class Echo {
  public static void main(String[] args) {
    for (int i = 0; i < args.length; i++)
      System.out.println(args[i] + " ");
    System.out.println();
  }
}
```

Типы данных и операторы

© Составление, Гаврилов А.В., Будаев Д.С., 2016

Лекция 1.3

Самара
2020

NetCracker®

План лекции

- Типы данных в Java
- Операторы для работы с примитивными типами
- Операторы для работы со ссылочными типами
- Работа со строками
- Массивы
- Инструкции, управляющие ходом выполнения программы

Типы данных в Java

- Java – строго типизированный язык
  - тип известен на момент компиляции
  - выявление многих ошибок до выполнения
- Две группы типов данных
  - Примитивные или простые (primitive)
  - Ссылочные или объектные (reference)

Характеристики типов данных

- **Множество значений**
  - для примитивных типов – значения из диапазона этого типа
  - для ссылочных типов – ссылки на объекты, контракт которых включает в себя контракт, определяемый типом ссылки
- **Возможные операции со значениями**
  - для примитивных типов – операторы
  - для ссылочных типов – действия, входящие в контракт типа (вызов методов и обращение к полям), и операторы
- **Форма хранения и представления**
  - форма хранения определяется реализацией JVM
  - JVM гарантирует одинаковое представление, не зависящее от реализации

Переменные

- Именованные участки памяти, способные содержать значения определенного типа
- Могут быть объявлены в различных частях кода
  - поля объектов
  - поля классов (статические поля)
  - параметры методов
  - локальные переменные методов и блоков инициализации
- Объявление переменной состоит из наименования типа, идентификатора и инициализации
- Область видимости переменной определяется местом ее объявления
- Локальные переменные должны быть инициализированы перед их использованием

Операторы

Постфиксные	[] . (params) expr++ expr--
Унарные	++expr --expr +expr -expr ~ !
Создание и приведение	new (type)expr
Арифметика	* / %
Арифметика	+ -
Побитовый сдвиг	<< >> >>>
Сравнение	< > >= <= instanceof
Равенство	== !=
И (and)	&
Исключающее ИЛИ (xor)	^
Включающее ИЛИ (or)	|
Условное И (and)	&&
Условное ИЛИ (or)	||
Условный оператор	? :
Операторы присваивания	= += -= *= /= %= >>= <<= >>>= &= ^= |=

высокий

приоритет

низкий

Арифметические операторы примитивных числовых типов

- Арифметические операции
  - `+` – сложение двух значений
  - `-` – вычитание второго значения из первого
  - `*` – умножение двух значений
  - `/` – деление первого значения на второе
  - `%` – остаток от деления первого значения на второе

- Результат имеет тип, совпадающий с «наиболее широким» типом из типов операндов, но не меньше, чем `int`

Особенность примитивных вещественных типов

```
int a = 5, b = 0;
int c = a / b;
System.out.println(c);
```

```
Exception in thread "main"
java.lang.ArithmeticException
```

```
float a = 5, b = 0;
float c = a / b;
System.out.println(c);
```

```
Infinity
```

- Легальные значения
  - Positive Infinity (Infinity)
  - Negative Infinity (-Infinity)
  - Not a Number (NaN)

- Различаются значения `0`, `+0` и `-0`

Арифметические операторы примитивных числовых типов

- Инкременты и декременты – соответственно, увеличивают и уменьшают значение на 1
  - Постфиксная форма: `i++`, `i--`
    результатом оператора является прежнее (неизмененное) значение
  - Префиксная форма: `++i`, `--i`
    результатом оператора является новое значение
- Унарные `+` и `-`
  - Аналогичны случаю, когда первый операнд равен 0
  - Если знак + или - находится перед литералом, он может трактоваться как часть литерала

Побитовые операторы примитивных целых типов

- Логические операторы
  - `&` – побитовое «и» (and)
    ```
            1 &        3 ->        1
    00000001 & 00000011 -> 00000001
    ```
  - `|` – побитовое «или» (or)
    ```
            1 |        3 ->        3
    00000001 | 00000011 -> 00000011
    ```
  - `^` – побитовое «исключающее или» (xor)
    ```
            1 ^        3 ->        2
    00000001 ^ 00000011 -> 00000010
    ```
  - `~` – побитовое отрицание
    ```
    ~       1 ->       -2
    ~00000001 -> 11111110
    ```
- Вычисления производятся в типе `int` либо `long`

Побитовые операторы примитивных целых типов

- Операторы сдвига
  - `<<` – сдвиг влево

```
       1 << 2 ->        4
00000001 << 2 -> 00000100
```

  - `>>` – арифметический сдвиг вправо

```
       4 >> 2 ->        1
00000100 >> 2 -> 00000001

      -1 >> 2 ->       -1
11111111 >> 2 -> 11111111
```

  - `>>>` – логический сдвиг вправо

```
       4 >>> 2 ->        1
00000100 >>> 2 -> 00000001

      -1 >>> 2 ->                            1073741823
11111111 >>> 2 -> 00111111 11111111 11111111 11111111
```

- Вычисления производятся в типе `int` либо `long`

Операторы сравнения примитивных числовых типов

- `>` и `<` – строгое сравнение
- `>=` и `<=` – нестрогое сравнение
- `==` – определение равенства
- `!=` – определение неравенства
- Результат – логическое значение: `true` или `false`
- Сравнение проводится в наиболее широком типе из типов операндов

Операторы примитивного логического типа

- `==` – определение равенства
- `!=` – определение неравенства
- `!` – отрицание
- `&` – логическое «и» (and)
- `|` – логическое «или» (or)
- `^` – логическое «исключающее или» (xor)
- `&&` – условное «и»
  (может не вычислять второй операнд)
- `||` – условное «или»
  (может не вычислять второй операнд)

Операторы присваивания примитивных типов

- `=` – простое присваивание
  - Тип выражения справа должен допускать присваивание в переменную слева
- `+=`, `-=`, `*=`, `/=`, `%=`, `>>=`, `<<=`, `>>>=`, `&=`, `^=`, `|=`
  - Присваивание с действием
  - Выражение `a ?= b` эквивалентно `a = a ? b`, но выполняется быстрее
  - Типы операндов должны позволять совершить операцию

Преобразование примитивных числовых типов

- **Неявное преобразование типов**
  Преобразование к более широкому типу
- **Явное преобразование типов**
  Преобразование к указанному типу с помощью оператора
  `(type)expr`

```
short s1 = 29;
int i1 = s1;
float f1 = i1;

int i2 = 14;
short s2 = (short) i2;

short s = -134;
byte b = (byte) s; // b = 122;
```

Особенности преобразования примитивных числовых типов

- Более широким считается тип, переменные которого могут принимать большее количество значений
- Вещественные типы считаются шире целочисленных
- Однако возникают ошибки округления:

```
long orig = 0x7effffff00000000L;
float fval = orig;
long lose = orig - (long)fval;
```

```
orig = 9151314438521880576
fval = 9.1513144e18
lose = -4294967296
```

Операторы ссылочных типов

- `new` – создание объекта класса
- `=` – присвоение ссылки
  - Тип выражения справа должен допускать присвоение в тип переменной слева
- `==` и `!=` – сравнение ссылок
  - Сравниваются только ссылки, а не состояние объектов!
- `.` – разыменование ссылки
  - `reference.method()`
  - `reference.field`
- `()` – вызов метода
- У любого объекта можно вызвать методы, объявленные в классе `Object`

Преобразование ссылочных типов

- Преобразование типа возможно, только если контракт целевого типа является частью контракта приводимого типа

- Более широким считается тип, переменные которого могут принимать большее количество значений. Родительский тип считается более общим, чем дочерний.

- Неявное преобразование типов – преобразование от более узкого к более широкому

- Явное преобразование типов – преобразование от более широкого к более узкому с помощью оператора явного преобразования `(type)expr`

Преобразование и проверка ссылочных типов

```
Integer i = new Integer(5);
Object o = i;
i = (Integer) o;
```

- Если явное преобразование типов невозможно, возникает ошибка `java.lang.ClassCastException`
- Соответствие типа можно проверить с помощью оператора `instanceof`, возвращающего `true`, если тип применим к объекту и `false`, если нет
- Оператор `instanceof` не позволяет определить реальный тип объекта, а лишь проверяет его соответствие указанному типу

```
Integer i = new Integer(5);
Object o = i;
if (o instanceof Integer) {
  i = (Integer) o;
  ...
}
else { ... }
```

Оператор ветвления

- Формат:
  `<логическое выражение> ? <значение 1> : <значение 2>`

```
double factor = (a > b) ? 1 : 0.7;
```

- Если логическое выражение истинно, возвращается значение второго операнда, а если ложно – третьего операнда
- Типы второго и третьего операндов должны быть «совместимы»
- Оператор можно применять в выражениях присваивания вместо инструкции ветвления

```
boolean flag = ...;
...
factor = flag ? 1 : 0.7;
/*
if (flag)
  factor = 1;
else
  factor = 0.7;
*/
```

Работа со строками

- Для работы со строками существуют специальные классы `String`, `StringBuffer` (`StringBuilder` с Java 1.5)
- Каждый строковый литерал порождает экземпляр класса `String`
- Значение любого типа может быть приведено к строке
- Если хотя бы один из операндов оператора `+` является ссылкой на строку, то остальные операнды также приводятся к строке, а оператор трактуется как конкатенация строк

Массивы

- Массив – упорядоченный набор элементов одного типа
- Элементами могут быть значения простых и ссылочных типов
- Массивы сами по себе являются объектами и наследуют от класса `Object`
- Доступ к элементам по целочисленному индексу с помощью оператора `[]`

NetCracker®

Объявление одномерных массивов

- Объявление, инициализация, заполнение

```
int array1[], justIntVariable = 0;
int[] array2;
array2 = new int[20];
for (int i = 0; i < array2.length; i++)
  array2[i] = 1000;
```

- Способ «3 в 1»

```
byte[] someBytes = {0, 2, 4, 8, 16, 32};

someMethod(new long[] {1, 2, 3, 4, 5});
```

Работа с одномерными массивами

- Форма объявления ссылки на массив с квадратными скобками после типа элемента является более предпочтительной
- Объект массива создается с помощью оператора **`new`**
- Массив при этом заполняется значениями по умолчанию для типа его элементов (**`0`**, **`false`** или **`null`**)
- Нумерация в массивах начинается с 0
- Длина массива хранится в публичном неизменяемом поле **`length`**
- Изменить длину массива после создания его объекта нельзя

Многомерные массивы

- Состоят из одномерных массивов, элементами которых являются ссылки на массивы меньшей размерности
- При создании объекта необязательно указывать все размерности
- Массив необязательно должен быть «прямоугольным»

```
// Автоматическая
int[][] twoDimArr = new int[10][5];

// Вручную
int[][] twoDimArr = new int[10][];
for (int i = 0; i < 10; i++)
  twoDimArr[i] = new int[i];

// Явно
int[][] arr3 = { {0}, {0, 1}, {0, 2, 4} };
```

Виды инструкций

- Выражения присваивания
- Префиксные и постфиксные формы выражений с операторами инкремента и декремента
- Конструкции вызова методов
- Выражения создания объектов
- Составные инструкции
- Управляющие порядком вычислений

Блок

- Составная инструкция
- Может использоваться в любом месте, где допускается инструкция
- Определяет область видимости локальных переменных: объявленная внутри блока переменная не видна за его пределами

```
int a = 5;
int b = 10;
{
  int c = a + b;
  int d = a - b;
}
```

Ветвление

- Полная форма

```
if (ЛогическоеВыражение)
  trueStatement
else
  falseStatement
```

- Неполная форма

```
if (ЛогическоеВыражение)
  trueStatement
```

- `else` относится к ближайшему выражению `if`, поэтому настоятельно рекомендуется использование блоков инструкций

Блок переключателей

```
switch (expression) {
  case n: Statements; [break;]
  case m: Statements; [break;]
  ...
  default: Statements; [break;]
}
```

- Для типов `char`, `byte`, `short`, `int`, `enum (java 1.5)`, `String (java 1.7)`
- Выполняются инструкции, расположенные за меткой `case`, предложение которой совпало со значением параметра блока переключателей
- Если ни одно из предложений не подошло, выполняются инструкции, расположенные за меткой `default`
- Метка `default` является необязательной
- Метка `case` или `default` не служит признаком завершения блока переключателей
- Команда `break` передает управление первой инструкции, следующей за блоком переключателей

Условные циклы while

- Форма с предусловием
  - Выполняется пока условие истинно
  - Если при входе в цикл условие ложно, цикл не выполняется

```
while (ЛогическоеВыражение)
  Инструкция
```

- Форма с постусловием
  - Выполняется пока условие истинно
  - При первом входе в цикл проверка условия не производится

```
do
  Инструкция
while (ЛогическоеВыражение);
```

Цикл с предусловием for

- Формально цикл for в Java не является циклом со счетчиком
- Общий синтаксис

```
For(СекцияИнициализации; ЛогическоеВыражение; СекцияИзменения)
  Инструкция
```

- Все секции заголовка являются необязательными
- Тело также может быть пустым

```
for( ; ; );
```

Секции цикла for

- Секции инициализации и изменения могут быть представлены списком выражений, разделенных запятой

```
for (i = 0, j = 50; j >= 0; i++, j--) {
  //...
}
```

- Допустимо объявление переменных в секции инициализации

```
for (int i = 0, j = 50; j >= 0; i++, j--) {
  //...
}
```

Объявление переменных в цикле for

```
for (int i = 0, Cell node = head;
       i < MAX && node != null;
       i++, node = node.next) {
   //...
}
```

- При инициализации переменных различных типов они должны объявляться вне цикла

```
int i; Cell node;
for (i = 0, node = head;
       i < MAX && node != null;
       i++, node = node.next) {
   //...
}
```

Цикл for в стиле for-each (Java 1.5)

- Общая форма записи

```
for (type iterVar : iterableObj) statement;
```

- Тип элемента
- Переменная цикла
- Агрегат с элементами (коллекция)
- Тело цикла

Работа улучшенного цикла for

- В каждом витке цикла «извлекается» очередной элемент агрегата
- Ссылка на него (для ссылочных типов) или значение (для примитивных) помещается в переменную цикла
- Тип переменной цикла должен допускать присвоение элементов агрегата
- Цикл выполняется до тех пор, пока не будут перебраны все элементы агрегата

Обработка многомерных массивов

```
int sum = 0;
int nums[][] = new int[3][5];

for (int i = 0; i < 3; i++)
   for (int j = 0; j < 5; j++)
      nums[i][j] = (i + 1) * (j + 1);

for (int[] x: nums)
   for (int y: x)
      sum += y;
```

Особенности улучшенного цикла for

- Агрегат обязан реализовывать интерфейс `java.lang.Iterable<T>`
- Переменная цикла доступна только для чтения…
- Порядок обхода в целом не определен…
- Нет доступа к соседним элементам…
- Мораль:
  - Область применения обобщенного цикла for «несколько уже», чем у «необобщенной» версии
  - Зато для этого класса задач синтаксис обобщенного цикла существенно удобнее

Работа с метками

- Метка
  `label: Statement`

- Оператора `goto` в Java нет!!!

- Метками можно помечать блоки инструкций и циклы

- Обращаться к меткам разрешено только с помощью команд `break` и `continue`

break

- Применяется для завершения выполнения кода блока инструкций
- Завершение текущего блока (безымянная форма)
  ```
  break;
  ```
- Завершение указанного блока (именованная форма)
  ```
  break label;
  ```
- Завершить блок, который сейчас не выполняется, нельзя!

break

```
private float[][] matrix;

public boolean workOnFlag(float flag) {
  int y, x;
  boolean found = false;
  search:
    for (y = 0; y < matrix.length; y++) {
      for (x = 0; x < matrix[y].length; x++) {
        if (matrix[y][x] == flag) {
          found = true;
          break search;
        }
      }
    }
  //...
}
```

continue

- Применяется только в контексте циклических конструкций
- Производит передачу управления в конец тела цикла
- Завершение витка текущего цикла (безымянная форма)
  ```
  continue;
  ```
- Завершение витка указанного цикла (именованная форма)
  ```
  continue метка;
  ```
- Завершить виток цикла, который сейчас не выполняется, нельзя!

continue

```
static void doubleUp(int[][] matrix) {
  int order = matrix.length;
  column:
    for (int i = 0; i < order; i++) {
      for (int j = 0; j < order; j++) {
        matrix[i][j] = matrix[j][i] =
          matrix[i][j] * 2;
        if (i == j)
          continue column;
      }
    }
}
```

Возврат из метода

- Инструкция `return` прекращает выполнение метода и возвращает его результат
- С возвращаемым значением
  ```
  return value;
  ```
  - Значение должно быть приводимо к типу, возвращаемому методом
- Без возвращаемого значения
  ```
  return;
  ```
  - методы void
  - конструкторы

Спасибо за внимание!

NetCracker®

Дополнительные источники

- Нимейер, Патрик. Программирование на Java / Патрик Нимейер, Дэниэл Леук; [пер. с англ. М.А. Райтмана]. – Москва : Эксмо, 2014. – 1216 с.
- Шилдт, Г. Java 7- Полное руководство - 8th Edition. – М.: ООО «И.Д. Вильямс», 2012г. – 1104 с.
- Хорстманн, К. Java 2. Библиотека профессионала. Том 1. Основы [Текст] / Кей Хорстманн, Гари Корнелл. – М. : Издательский дом «Вильямс», 2010 г. – 816 с.
- Эккель, Б. Философия Java [Текст] / Брюс Эккель. – СПб. : Питер, 2011. – 640 с.
- JavaSE at a Glance [Электронный ресурс]. – Режим доступа: http://www.oracle.com/technetwork/java/javase/overview/index.html, дата доступа: 11.09.2020.
- JavaSE APIs & Documentation [Электронный ресурс]. – Режим доступа: http://docs.oracle.com/en/javase/JavaSE APIs & Documentation [Электронный ресурс]. – Режим доступа: http://docs.oracle.com/en/javase/1JavaSE APIs & Documentation [Электронный ресурс]. – Режим доступа: http://docs.oracle.com/en/javase/14JavaSE APIs & Documentation [Электронный ресурс]. – Режим доступа: http://docs.oracle.com/en/javase/14/, дата доступа: 11.09.2020.