СБЕРБАНК ТЕХНОЛОГИИ

Java introduction

Дата

Узнаем

- Изучить базовый синтаксис Java
- Изучить объекты, классы и пакеты в Java
- Сделать обзор основных типов (Object, String, примитивные типы, классы обертки)
- Показать Naming conventions

Именование переменных

- латинские буквы (большие и маленькие)*;
- цифры;
- знак доллара «$» и знак подчеркивания «_».

- имя не должно начинаться с цифры;
- длина имени не имеет ограничения (64k).

Объявление и инициализация переменных

Объявление:

<тип> имяПеременной;

int varName;

Object obj, obj2, obj3; // объявление П. одного типа

final int DAYS_IN_WEEK = 7; // Константа. (Нельзя просто объявить)

Инициализация:

varName = 10;

Объявление и инициализация:

double step = 10.0;

String name = "Peter";

Простые типы

Восемь примитивных типов данных: byte, short, int, long, char, float, double и boolean

- Целые числа: byte, short, int, long
- Числа с плавающей точкой: float, double
- Символы: char
- Логические значения: boolean

BOOLEAN

- Литералы: false, true
- Нет преобразования между boolean и другими примитивными типами
- Результат любого сравнения — boolean:
  <, >, ==, <=, >=, !=

Логические операторы

Оператор	Описание
&	Логическое AND (И)
&&	Сокращённое AND
|	Логическое OR (ИЛИ)
||	Сокращённое OR
^	Логическое XOR (исключающее OR (ИЛИ))
!	Логическое унарное NOT (НЕ)
&=	AND с присваиванием
|=	OR с присваиванием
^=	XOR с присваиванием
==	Равно
!=	Не равно
<если> **?** <то> **:** <иначе>	Тернарный (троичный) условный оператор

Таблица истинности

A	B	A && B	A || B	A ^ B	A == B	A != B
false	false	false	false	false	true	false
false	true	false	true	true	false	true
true	false	false	true	true	false	true
true	true	true	true	false	true	false

Логические операторы && и & (|| и |)

Различие между && и & (для || и | аналогично):
A && B && C, если A = false, то сравнение B && C выполнено не будет.
В случае A & B & C, даже если A или B – false, будут выполнены все сравнения.

Пример использования && (Сокращенное И):

```
Object a = null; Object b = null;
return a != null && a.equals(b);
```

Пример использования &:

```
while(true) {
    …
    if (a != null & i++ < 10) {
        return true;
    }
    …
}
```

Символьные значения

Тип char – гибридный тип. Его значения можно интерпретировать и как числа (их можно складывать и умножать) и как символы.

- размер — 16 бит, беззнаковый (0 .. 2^16 – 1)
- Представляет номер символа в кодировке Unicode
- Литералы:
  - символ в одинарных кавычках: ’a’
  - шестнадцатеричный код символа: ’\u78bc’
  - спецпоследовательности: ’\t’, ’\n’, ’\r’, ’\”, ’\\’

Свободно конвертируется в числовые типы и обратно

Целочисленные примитивные типы

Тип	Бит	Минимальное значение	Максимальное значение
byte	8	−128	+ 127
short	16	−2^15 (-32 768)	+ 2^15 − 1 (32 767)
int	32	−2^31 (-2 147 483 648)	+ 2^31 − 1 (2 147 483 647)
long	64	−2^63 (-9223372036854775808)	+ 2^63 − 1 (9223372036854775807)

- Размер фиксирован, одинаков для всех платформ
- Все типы знаковые, беззнаковых вариантов нет

Целочисленные литералы

- Десятичное число: 123
- Восьмеричное число: 0123
- Шестнадцатеричное число: 0x123
- Двоичное число: 0b101 (с Java 7)
- С подчеркиванием: 123_456_789 (с Java 7)
- С суффиксом L для long

Вещественные типы

Тип	Размер (бит)	Диапазон
float	32	от -1.4e-45 до 3.4e+38
double	64	от -4.9e-324 до 1.7e+308

Спросите меня как правильно сравниваются вещественные типы.

Вещественные литералы

- Обычная запись: -1.234
- Экспоненциальная запись: -123.4e-2 (−123.4 · 10−2)
- Шестнадцатеричная запись: 0xFFFFpFF (FFFF · 2FF)
- С суффиксом типа:

  38f
  3e19d
  123.4e-2f
  444.444d

Арифметические операции

- сложение + +=
- вычитание - -=
- умножение * *=
- деление / /=
- остаток % %=
- инкремент ++ (++x или x++)
- декремент -- (--x или x--)

```
int y = 10, x = 1;
y /= 2; // y = y / 2;
y += x++ - ++x;
```

y = ?

Неявное преобразование типов

A x = _; B y = _;

x = y;

Преобразование целочисленных типов в более емкие

- (**byte → short → int** → long)
- Преобразование char в int и long
- Преобразование целочисленных типов в типы с плавающей точкой

Явное преобразование типов

- Оператор приведения типа: (typename)
- При приведении более емкого целого типа к менее емкому

старшие биты просто отбрасываются

- При приведении типа с плавающей точкой к целому типу

дробная часть отбрасывается (никакого округления)

- Слишком большой double при приведении к float

превращается в Float.POSITIVE_INFINITY или

Float.NEGATIVE_INFINITY

Преобразование типов

- При вычислении выражения (a @ b) аргументы a и b преобразовываются в числа, имеющие одинаковый тип:
  - если одно из чисел double, то в double;
  - иначе, если одно из чисел float, то в float;
  - иначе, если одно из чисел long, то в long;
  - иначе оба числа преобразуются в int.

```
byte a = 1;
byte b = 1;
byte c = a + b; // compilation error
byte d = b + 1; // compilation error
```

КЛАСС ОБЕРТКА

Тип	**Класс-обертка**
byte	Byte
short	Short
int	Integer
long	Long
char	Character
float	Float
double	Double
boolean	Boolean

Boxing/unboxing

Autoboxing: примитивное значение → объект-обертка
Autounboxing: объект-обертка → примитивное значение

```
Integer i = 1;
Integer j = i + 1;
int k = i + j;
```

Обертки сравниваются как объекты!

Класс String. Работа со строками

В java нет отдельного примитивного типа для строковых данных, но есть специальный класс - java.lang.String.

- обертка над char[]
- утилитарные методы
  - substring, charAt, valueOf и т.д.
- Литералы в двойных кавычках
  - String str = "Строка“;
- неизменяемый (Immutable)
  - String str = "Строка " + "объект"; *
- Сравниваются как объекты, но есть исключения
- StringBuilder/StringBuffer(до 1.5, потокобезопасный)

*** но не все так просто…**

Управляющие конструкции

if

while

do while

for

switch

break и continue

Switch

```
switch ( digit ) {
        case 0:
                        text = " zero ";
                        break ;
        case 1:
                        text = "one";
                        break ;
                        // case 2 - case 9
        default :
                        text = "???";
}
```

- Без break исполнение продолжается
- Работает для примитивных типов byte, short, char, int, а также для enum
- В Java 7 добавлен switch для String

Naming convention

- Имена классов и интерфейсов – всегда с большой буквы, CamelCase.
- Всё пишется в Lower Camel Case.

```
int sheepCount; private int getWidth();
```

- Имена пакетов – всегда lowercase (в нижнем регистре).

```
package ru.sberbank.java_school.common;
```

- Константы — всегда в UPPERCASE.

```
public static final COLOR_WHITE = "#FFFFFF";
```

Naming convention

```
1. String Symbolconfigstorage = “example”;
2. public abstract int getNumberOfBranches();
3. private String NAME_OF_GREEK_GOD = “Zeus”;
4. public class Base_Quote
5. private interface Drivable {
6. package google.com.util;
7. private void sum(Integer param_a, Integer param_b)
8. public static final int MAGICNUMBER = 42;
9. String URL = null;
10. String pathXML = null;
```

- **Объект** - некоторая сущность, обладающая определённым состоянием и поведением, имеющая заданные значения свойств (атрибутов) и операций над ними (методов). Объект принадлежит одному классу (в java), который определяют поведение (являются моделью) объекта.

- **Объектно-ориентированное программирование** - парадигма программирования, в которой программа строится из взаимодействующих объектов

Свойства объекта

- Объект является экземпляром класса
- Объект имеет внутреннее состояние (в java – поля (переменные))
- Объект может принимать сообщения (в java - методы)

Объект - это данные и методы для работы с этими данными

- **Инкапсуляция** - Сокрытие деталей реализации за внешним интерфейсом
- **Наследование** - Создание производных классов, наследующих свойства базового
- **Полиморфизм** - Разная обработка сообщений в разных классах

Основы ООП

- Инкапсуляция, наследование и полиморфизм поддерживаются на уровне языка
- В Java все является объектом, кроме примитивных типов
- Исполняемый код может находиться только в классе
- Стандартная библиотека предоставляет огромное количество классов, и можно создавать свои

Модификаторы доступа

- **public** доступ для всех
- **protected** доступ в пределах пакета и дочерних классов
- по умолчанию (нет ключевого слова) доступ в пределах пакета
- **private** доступ в пределах класса

Объявление класса

```
package ru.sberbank.lesson2;

public class Person {

    // class content

}
```

Поля

```
package ru.sberbank.lesson2;

public class Person {
    public static final boolean RESIDENT = true;
    public static final boolean NOT_RESIDENT = false;

    private Long id;
    private String firstName;
    private String lastName;
    private int age;
    private boolean isResident;
}
```

Поля

- Поля класса инициализируются значениями по умолчанию
- Модификатор **final** — значение должно быть присвоено ровно один раз к моменту завершения инициализации экземпляра
- Модификатор **static** - поле относится к классу а не к экземпляру класса

Методы класса Object

- public boolean equals(Object obj)
  - рефлексивность - x.equals(x) == true
  - симметричность – x.equals(y) == true => y.equals(x)
  - транзитивность x.equals(y) && y.equals(z) && z.equals(x) == true
- public int hashCode()
- protected Object clone() throws CloneNotSupportedException

Конструкторы

```
public class Person {
    public static final boolean RESIDENT = true;
    public static final boolean NOT_RESIDENT = false;

    private Long id;
    private String firstName;
    private String lastName;
    private int age;
    private boolean isResident;

    public Person() {
    }

    public Person(String firstName, String lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }
}
```

Конструкторы

- Если не объявлен ни один конструктор, автоматически создается конструктор по умолчанию (без параметров)

- Может быть несколько конструкторов с разными входными параметрами

Деструктор

- В Java нет деструкторов, сбор мусора автоматический
- Есть метод void finalize(), но пользоваться им не рекомендуется (неизвестно, когда будет вызван)
- При необходимости освободить ресурсы заводят обычный метод void close() или void dispose() и вызывают его явно

Объявление метода:

[модификатор доступа] [static] <возвращаемый-тип>
<имя> ([параметры]) {
        [тело-метода;]
        [return [возвращаемое значение];]
}

методы

- модификатор static - статические методы относятся не к экземпляру класса, а ко всему классу
- модификатор final и abstract
- перегрузка (overload) методов
- переопределение (override) методов
- примитивные аргументы передаются по значению
- при передачи объекта, передается ссылка на объект

final

- Этот модификатор применяется только к классам, методам и переменным (также и к локальным переменным)
- Поля не могут быть изменены, методы переопределены
- Классы нельзя наследовать
- Аргументы методов, обозначенные как final, предназначены только для чтения, при попытке изменения будет ошибка компиляции
- Переменные final не инициализируются по умолчанию, им необходимо явно присвоить значение при объявлении или в конструкторе, иначе – ошибка компиляции
- Если final переменная содержит ссылку на объект, объект может быть изменен, но переменная всегда будет ссылаться на тот же самый объект
- Если класс объявлен final и abstract (взаимоисключающие понятия), произойдет ошибка компиляции
- Так как final класс не может наследоваться, его методы никогда не могут быть переопределены

```
public abstract class ClientImpl {
    private Long id;

    public abstract String getFullName();

    public Long getId() {
        return id;
    }
    public void setId(Long id) {
        this.id = id;
    }
}
```

Абстрактные классы

- Применяется только для методов и классов
- У абстрактных методов нет тела метода
- Является противоположностью final: final класс не может наследоваться, abstract класс обязан наследоваться

Класс должен быть объявлен как abstract если:

- он содержит хотя бы один абстрактный метод
- он не предоставляет реализацию наследуемых абстрактных методов
- он не предоставляет реализацию методов интерфейса, реализацию которого он объявил
- необходимо запретить создание экземпляров класса

```
public interface Client {
     Integer PERSON_TYPE = 1;
     Integer COMPANY_TYPE = 2;

     String getFullName();
}
```

Класс реализующий интерфейс Client:

```
public class ClientImpl implements Client {
    @Override
     public String getFullName() {
        return “Boris”;
     }
}
```

интерфейсы

- Методы всегда public и abstract, даже если это не объявлено
- Методы не могут быть static, final, strictfp, native, private, protected
- Переменные только public static final, даже если это не объявлено
- Переменные не могут быть strictfp, native, private, protected
- Может только наследовать (extends) другой интерфейс, но не реализовывать интерфейс или класс (implements).

Модификаторы

	Класс	Внутренний класс	Переменная	Метод	Конструктор	Логический блок
public	Да	Да (кроме локальных и анонимных классов)	Да	Да	Да	Нет
protected	Нет	Да (кроме локальных и анонимных классов)	Да	Да	Да	Нет
default	Да	Да	Да (и для локальной переменной)	Да	Да	Да
private	Нет	Да (кроме локальных и анонимных классов)	Да	Да	Да	Нет
final	Да	Да (кроме анонимных классов)	Да (и для локальной переменной)	Да	Нет	Нет
abstract	Да	Да (кроме анонимных классов)	Нет	Да	Нет	Нет
static	Нет	Да (кроме локальных и анонимных классов)	Да	Да	Нет	Да
native	Нет	Нет	Нет	Да	Нет	Нет
strictfp	Да	Да	Нет	Да	Нет	Нет

ПРИЛОЖЕНИЯ