Оглавление

[General Hints 1](#_Toc13515299)

[Basics 1](#_Toc13515300)

[Data types 1](#_Toc13515301)

[Exceptions 1](#_Toc13515302)

[Classes 1](#_Toc13515303)

[StringBuilder 1](#_Toc13515304)

[Arrays 1](#_Toc13515305)

[ArrayList 1](#_Toc13515306)

[Wrappers 1](#_Toc13515307)

[Time 1](#_Toc13515308)

[Variables 1](#_Toc13515309)

[Polymorphism 1](#_Toc13515310)

[Method overriding 1](#_Toc13515311)

[Method overloading 1](#_Toc13515312)

[Type casting 1](#_Toc13515313)

[Interfaces 1](#_Toc13515314)

[Reflections 1](#_Toc13515315)

[Controls 1](#_Toc13515316)

[loops 1](#_Toc13515317)

[switch 1](#_Toc13515318)

[if-else 1](#_Toc13515319)

[Unreachable statement 1](#_Toc13515320)

[Equality check 1](#_Toc13515321)

[== 2](#_Toc13515322)

[equals 2](#_Toc13515323)

[Keywords 2](#_Toc13515324)

[Static 2](#_Toc13515325)

# General Hints

# Basics and Benefits

* Java don’t have memory pointers
* Java got rid of operator overloading (almost, except + which can be used for both sum calculation and string concatenation)

# Data types

When a number is present in the code (without any calculations) – it is called a literal. Literals are mostly used for primitives, but there are reference literals: null, “string” and {1, 2, 3} for arrays

Long (long) – can be used as loop cursor, but can’t as array dimension size.

An object is a runtime instance of a class in memory. All the various objects of all the different classes represent the state of your program.

# Operators

UMARELSA

compound operator (+=, \*=, /= etc.) can only be applied to a variable that is already defined and cannot be used to declare a new variable

logical exclusive OR (XOR, ^): is only true if the operands are different

# Exceptions

# Classes

* **P**ackage-**I**mport-**C**lass (PIC). Fields and methods have to be inside of a class

Understand the effect of using packages and imports:

* packages contain Java classes (there is also ‘default package’ – implicit one, if no one explicitly set, but classes from such package can’t be imported to another explicit package)
* classes can be imported by class name or wildcard (\*). But the only purpose of the import statement is using class ‘short’ name, not fully qualified one, i.e. like aliases, not analog of #include
* wildcards do not look at subdirectories
* In the event of a conflict – class name imports take precedence over wildcard one

# StringBuilder

# Arrays

# ArrayList

# Wrappers

# Time

# Variables

Primitive

Reference:

* Do not confuse a reference with the object that it refers to, they are two different entities. The reference is a variable that has a name and can be used to access the contents of an object. A reference can be assigned to another reference, passed to a method, or returned from a method
* All references are the same size, no matter what their type is
* An object sits on the heap and does not have a name. Therefore, you have no way to access an object except through a reference. Objects come in all different shapes and sizes and consume varying amounts of memory. An object cannot be assigned to another

## class (with static specifier)

## instance

## method

## local

* defined in the block of code {}
* must be initialized before use

# Polymorphism

# Method overriding

# Method overloading

# Type casting and relevant stuff

* Note that integer division by 0 raises an exception, whereas floating-point division by 0 yields an infinite or NaN result

# Interfaces

# Reflections

# Controls

## Loops

* While the **for-each** statement is convenient for working with lists/arrays in many cases, it does hide access to the loop iterator variable

## switch

## if-else

# Unreachable statement

# Equality check

## ==

## equals

# Keywords

## Static

# GC

## finalize

Java allows objects to implement a method called *finalize()* that **might** get called. This method gets called if the GC tries to collect the object. If the GC doesn’t run, the method doesn’t get called. If the GC fails to collect the object and tries to run it again later, the method doesn’t get called a second time.

# Useful terms

## Overflow vs Underflow

* Overflow is when the absolute value of the number is too high for the computer to represent it. You can get overflow with both integers and floating-point numbers. To get an overflow, repeatedly multiply a number by ten.

**Overflow example:** If the variable x is a signed byte it can have values in the range -128 to +127, then

1. x = 127
2. x = x + 1

will result in an overflow. +128 is not a valid value for x.

* Underflow is when the absolute value of the number is too close to zero for the computer to represent it. You can only get underflow with floating point numbers. To get an underflow repeatedly divide it by ten.

**Underflow example:**

For floating point numbers, the range depends on their representation. If x is a single precision (32-bit IEEE) number, then

1. x = 1e-38
2. x = x / 1000

will result in an underflow. 1e-42 is not a valid value for x.