Syntax

Commands

Program is a list of commands. These commands are performed by JVM (Java Virtual Machine)

Command to print text to console (screen): System.out.println(“Hello world!”);

In Java every command ends with semicolon (;)
Program can`t contain only a list of the commands. Class contains functions. Functions contains commands. Functions = methods.

Class - Functions - Commands

Program Structure

Minimum program contains 1 class. Every class is in a separate file. The name of this file is the same as the name of the class.

The code of the class contains the name of the class (Book) and the body of the class.

```
public class Book

{

    Class body

}
```

Class body can contain variables (data of class) and methods/functions of the class.

Method main()

Minimum program with class main(). From the class main() the program begins to execute.

```
public class Book

{

    public static void main (String[] args)

   {

   }


}
```

```
public class Book {

       public static void main (String[] args) {

              System.out.println(“This is a book”);

     }


}
```

“This is a book” is a parameter which is passed to the command System.out.println.

Java IS register sensitive.

System.out.println - a command to output a parameter (for example, text) to the console from THE NEW LINE.

System.out.print - … in THE SAME LINE.

If you need a gap in the text to print to the console just add a gap within “” (“ This is a book: “)

Variables

A variable is like a box for data. For example, you wrote a note and put it into the box. The bos is a variable. A note is like a meaning.

In Java every variable has:

1) Type (int (Integer), String, double, byte, short, long, float)
2) Name
3) Meaning

Name is for differentiating one variable from the other. Name is like an inscription on the box to define what is inside this box.

Type means what type of meanings or data can be saved in this variable.

Meaning is data or info which is storing in the variable.

Creating a variable: type name; (int x;)

Creating several variables: int n, m, k;

Types: int (1, 2, 3…), String (text), double (½, ⅔)
Writing a meaning to the variable: name = meaning;

== is about comparing

One variable has one meaning.

In one method one can`t create 2 variables with the same names.

Creating a variable + giving a meaning: int x = 5; int y = x + 1; String e = “OK”;

Creating several variables + giving meanings: int c = 3, b = 4, d = c + b;

Combining lines: String k = “This is “ + “my friend”;

int y = 2;
String m = “Age: “ + y;

String n = “How are you?”;
System.out.println(n);

System.out - link to the data of putting info to the console
System.in - link to the data for reading info which is entering to the console
System.err - for outgoing errors.
. (dot) in these examples is for addressing to the element which is within the other element

Comments

/* a lot of letters */

// just 1 line of comment

Operators

int x = (3+2) * 5;
int y = 4 / 2;

Reminder of the division (%): int d = 9 % 5; (d = 4; 1 and 4 as a remainder)

The number is EVEN: (x % 2) == 0; (2, 4, 6…)

The number is ODD: (y % 2) == 1; (1, 3, 5…)

To get the last digit of the number:

int number = 337;

int last = number % 10;

(output: 7)

Increment (increasing the number a variable on 1): i++;
Decrement (opposite): k- -; (minusminus)

i++ is the same as: i = i + 1;

int y = 2; (2) (giving apples)
y++; (3)
y++; (4)
y++; (5)

Int m = 6; (6) (cutting trees)
m- - ; (5)
m - - ; (4)
m - - ; (3)

If one creates a variable and did not give it any meaning and then TRY to USE this variable it will be an ERROR!

For example: int x; System.out.println (x); ERROR

Concatenation is sticking lines together: “John” + “John”

“Tom” + “ “ + “Tom” (output: Tom Tom) with a space

In programming all counting begins with Zero (0)

Number To String

To make string of the number just add this number to the space line:

String m = “ “ + 5;

String To Number

Only for strings which contains numbers!!!

Use class Integer with the method parseInt():

String n = “787”;
int number = Integer.parseInt(n);

Methods To Work With Strings

Method length()

To get to know the length of the line:

String wordLength = “home”;
int countLength = wordLength.length();

(output: 4)

Method toLowerCase()

To turning all symbols to small letters:

String title = “MMM”;
String title2 = title.toLowerCase();

(output: mmm)

Method toUpperCase()

To turning all symbols to big letters:

String title = “mmm”;
String title2 = title.toUpperCase();

(output: MMM)

Reading From The Console

For entering data one use object: System.in
It helps to read data from the console (1 symbol per 1 time)

Class Scanner

Full name: java.util.Scanner) can read data from different sources: console, files, the Internet, keyboard.

To read data from the keyboard one need to pass to class Scanner the object System.in like a parameter (the source of data)

```
import java.util.Scanner;
public class Book {
    public static void main (String[]  args)
    {

        Scanner keyboard = new Scanner(System.in);

        String text = keyboard.nextLine();  // to read a string from the keyboard
        int number = keyboard.nextInt();  // to read a number from the keyboard
        double number2 = keyboard.nextDouble(); // to read 0.5

        System.out.println("Text:  " + text);
        System.out.println("Number:  " + number);
        System.out.println("Half:  " + number2);

    }

}
```

Creating an object Scanner is like creating a variable:

Scanner keyboard = new Scanner(System.in);

Scanner keyboard - like creating a variable with type Scanner with name “keyboard”

newScanner - creating the new object (word “new”) with type Scanner and passing to it as a parameter where new created object Scanner takes data - object System.in

Check Types of Not Entered Data

hasNextInt(); (Is there a type int? Can it be converted to type int?)
hasNextFloat();
hasNextDouble();
hasNextBoolean();
hasNext(); (Is there another 1 word?)
hasNextLine(); (Is there another 1 line?)

Calling methods

To call method of the object, on which a variable addresses:

variable.method(parameters);

For example: System.out.println(2);

If one plans to pass parameters to the function just leave () empty:

variable.method();

Operator IF - ELSE

If is a conditional operator which helps to execute different blocks of the commands depending on true of the condition

```
if (condition)
   command1;
else
   command2;
```

If condition is true command1 is executed, otherwise - command2

If condition is true

    Execute command1

Otherwise

    Execute command2

```
int age = 3;
if (age < 18)
  System.out.println(“You are not a student”);
else
   System.out.println(“You are a student”);
```

(output: You are not a student)

```
import java.util.Scanner;
public class Person {
     public static void main(String[] args) {
          String message = “, go to a play ground”;
          Scanner keyboard = new Scanner(System.in);
          String name = keyboard.nextLine();
          int age = keyboard.nextInt();

          if (age >= 1 && age <= 18)
          {
              System.out.println(name + message);
              System.out.println(“Have fun!”);
          }

    }
}
```

if() {} to add more commands

To execute a command if the condition is true and if false NOTHING TO DO:

```
if(condition)
   command1;
```

Else IF

```
int price = 5;

if(price >= 8)
  System.out.println(“Buy it!”);
else if(price > 20)
   System.out.println(“Think about buying”);
else if(price > 1000)
    System.out.println(“Do not buy it!”);
else
    System.out.println(“Stop thinking about it!”);
```

If price >= 8 - buy
If price more than 20 but not more than 8 - think
If price more than 1000 but not more than 20 - not buy
If price less than 1000 - stop thinking

If to use if-else without {} , word “else” is about previous (the closet to it) if

Logical Operators

AND = &&
OR = ||
NOT = !

(!x) && (!y) = not x and not y

true && true = true
true && false = false
false && true = false
false && false = false

true || true = true
true || false = true
false || true = true
false || false = false

!true = false
!false = true

a && !a = false
a || !a = true
!(a && m) = !a || !m
!(a || m) = !a && !m

Ternary (Triple) Operator

Brief version of if-else operator:

condition ? expression1 : expression2;

If condition is true - execute expression1, otherwise - expression2

The difference from if-else operator is that ternary operator is an expression. That`s why its result can be giving to something

```
int number = 5;
int house;

if (number > 10)
    house = 2;
else
    house = 1;
```

OR

```
int number = 5;
int house = number > 10 ? 2 : 1;
```

Comparing Lines

To compare lines use method equals():

line1.equals(line2);

This method returns true if lines are the same and false if they are different

To compare lines without taking into account the register:

line1.equalsIgnoreCase(line2);

Cycle While

In Java there are 4 types of cycles: while, do-while, for, for-each

```
while(condition)
        command;
```

Cycle body of while cycle is being executed again and again WHILE the condition is true

If the condition is false cycle body will not be executed

```
int x = 5;
while(x > 0)
{

 System.out.println(x);
 x- - ;

}
```

(output (5 lines): 5 4 3 2 1)

To output lines from the keyboard till the line “exit” is entered:

```
Scanner keyboard = new Scanner(System.in);
boolean isStopWord = false;
while(!isStopWord)
{

      String m = keyboard.nextLine();
      isStopWord = m.equals(“exit”);

}
```

Using cycle in cycle output into the console rectangle 5 height, 10 width filled with letters U:

```
UUUUUUUUUU
UUUUUUUUUU
UUUUUUUUUU
UUUUUUUUUU
UUUUUUUUUU
```

```
public class Shape {
    public static void main(String[] args) {

        int vertical = 0;

        while (vertical < 5)

        {

            int horizontal = 0;

            while (horizontal < 10)

            {

                System.out.print(“U”);
                horizontal++;

            }

            System.out.println();
            vertical++;

            }
    }
}
```

Breaking Cycle

If one executs command break inside the cycle it will be stopped and program starts to execute commands after the cycle.

```
Scanner keyboard = new Scanner(System.in);

while(true)
{

     String m = keyboard.nextLine();
     if (m.equals(“exit”))

         break;

}
```

Command continue

If one executes command continue within the cycle current loop of the cycle will stop. Command continue stops a loop not a cycle.

Use command continue if you want to “skip” executing of cycle body

To print to the console 20 numbers from 1 to 20, but skip numbers which are divided by 5:

```
int k = 0;

while (k < 20)
{

  k++;
  if ( (k % 5) == 0)
     continue;
System.out.println(k);

}
```

Cycle For

If you know how many times the command will be executed you can use for cycle

The same code in while and for cycles:

```
int i = 0;

while (i < 10)
{

    System.out.println(i);
    i++;

}
```

```
for (int i = 0; i < 10; i++)
{

    System.out.println(i);

}
```

So all code about the counter i was put in one place

Cycle for is just more convenient way to write cycle while

To output only even numbers from 1 to 11:

```
for (int i = 1; i < 11; i++)
{

    if  (i % 2 == 0)

    System.out.println(i);

}
```

Cycle Do - While

Cycle body is being executed again and again while the condition is true.
It`s like while cycle but inverse.

```
do
    command;
while (condition);
```

While cycle will be executed: condition - cycle body - condition - cycle body …

Do - while cycle will be executed: body - condition - body - condition …

Body in do - while cycle will be executed at least 1 time

```
String m;

do
{

    m = keyboard.nextLine();

}

while (!m.equals(“exit”));
```

Dividing Integers

If to divide integer on integer the reminder will be cutted (as in Math.floor)

To prevent it just add .0 to at least one of divided numbers:

double n = 5 / 2.0 (output: 2.5)

Dealing with dividing variables:

```
int x = 5;
int y = 2;
double m = x * 1.0 / y;
```

Rounding Numbers

Class Math with methods: rount(), ceil(), floor()

int m = (int) Math.round(4.1); (output: 4)
int m = (int) Math.round(4.5); (output: 5)

int m = (int) Math.ceil(4.1); (output: 5)
int m = (int) Math.ceil(4.5); (output: 5)

int m = (int) Math.floor(4.1); (output: 4)
int m = (int) Math.floor(4.5); (output: 4)

Arrays

Array is a special object (table) where one can save SEVERAL meanings.

type[] name = new type[quantity];

int[] myArray = new int[10];
myArray[0] = 5;
myArray[5] = myArray[2] + myArray[3];

new int[10] - creating the new object for 10 elements

myArray[0] =5; - filling cell with index 0 with the meaning 5

Size of array can not be changed after creation

type[] name;
name = new type[quantity];

int m = 10;
int[] k = new int[m];

If there is not such cell with an index it will be the error:
ArrayIndexOfBoundException

To get to know quantity of array elements: array.length;

To fill array with 10 elements with numbers from 0 to 9:

```
int[] m = new int[10];
for(int i = 0; i < 10; i++) {
    m[i] = i;
}
```

```
int[] classes = new int[5];
classes[0] = 4; // 4 pupils in the first class
classes[1] = 6;
```

The same briefly:

int[] classes = new int[5] {4, 6};

The most briefly:

int[] classes = {4, 6};

Two-Dimensional Arrays (Table)

```
int[] [] name = new int[height][width];
```

Like in chess, sea battle…

```
int [] [] table = {
  {2,3,4,5,6},
  {2,3,4,5,6}
};
```

2 lines and 5 columns

Not Even Arrays

```
int[] [] table = new int[2] [ ];
table[0] = new int[5];
table[1] = new int[7];
```

Height: 2
First line: array with 5 elements
Second line: array with 7 elements

Class Arrays

Full name: java.util.Arrays

Method Arrays.toString()

```
int[] myArray = {3,4,5};
String m = Arrays.toString(myArray);
```

(output: 3, 4, 5)

In two-dimensional array use instead **Arrays.deepToString()**

To compare arrays: **Arrays.equals(firstArray, secondArray);**
And with two-dimensional arrays: Arrays.deepEquals()

Filling an array with the same meanings: **Arrays.fill()**

```
Arrays.fill(name, meaning)
```

```
int[] m = new int[10];
Arrays.fill(m, 55);
```

```
int[] k = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
Arrays.fill(k, 4, 8, 77);
String my = Arrays.toString(k);
```

Filling cells 4, 5, 6, 7 with meaning 77

Arrays.sort(nameOfArray); - to sort for increasing meanings

Creating Your Own Functions

Method is a group of commands with a unique name

Without a method:

```
class Book {
     public static void main (String[] args)
    {

        System.out.print(“My”);
        System.out.println(“name”);

        System.out.print(“My”);
        System.out.println(“name”);

        System.out.print(“My”);
        System.out.println(“name”);

    }
}
```

With your own method:

```
class Book {
     public static void main (String[] args)
    {

          printMyName();  // calling a method
          printMyName();
          printMyName();

     }

        public static void printMyName ()

       {
            System.out.print(“My”);
            System.out.println(“name”);

        }
}
```

Passing Parameters

Method with parameters:

```
public static void name (parameters)
{
    code of method
}
```

Method with different parameters:

```
class Book {
    public static void printNames (String name, int count)
    {
        for (int i = 0; i < count; i++)
            System.out.println(name);
    }
    public static void main (String[] args) {
        printNames("Anne", 4);
        printNames("Gilbert", 8);
    }
}
```

Calling method printNames with different parameters

Meaning which are being passed to the method during its calling are arguments of method

Operator/Command Return

After executing operator return; current method will be finished

Functions With Result Void

Methods can not only do something depending on the parameters but also calculate something and return the result of the calculation

```
Scanner keyboard = new Scanner(System.in);
int m = keyboard.nextInt();
```

(method nextInt returns meaning of type int)

Each method can return only one meaning of one before defined type. Type of returning a meaning is defined during creation of a method:

```
public static type name (parameters)

{

    code of the method

}
```

Where type is a type of the result which is returned by the method

For methods which return nothing there is a stopping type: void

```
int min(int x, int y)
{

    if (x < y)
      return x;
   else
      return y;

 }
```

Access Modificators

Access Modificators to limit access to the method from the other classes:

1) public
2) protected
3) private

Private - the method can be called only from THE SAME class in which it was created

Public - the method can be called from ANY methods of any class

Without the modificator - can be seen to all class of the PACKAGE where it was created

Protected - the method can be called from the same class, the same package, and child-classes

Keyword Static

Key word static return a method to static

Static method is not connected to any object but it belongs to the class where it was created

Calling a static method:

NameOfClass.nameOfMethod()

For example:

Math.abs()

Arrays.sort()

The name of the class before the name of the static method can not be written if you call the method from the same class

The static method can not address to non static methods of its own class

The static method can address only static methods. That`s why all methods which you want to call from the method main we define as static

public static void main(Strings[] args):

This method main receives parameters: the array of lines. args means arguments of the program. During starting the program to it one can pass parameters - array of strings. They will be in the array args of the method main()

args is a variable of type of the array of strings

Keyword Throws

public static type name(parameters) throws Exception

{

    code of method

}

In this method can be errors of type Exception (in this class and its child classes)

Types Of Variables

Local Variables

Local variables are within methods. A local variable exists only in the block of code {} where it was created

It is impossible to create 2 local variables with the same names in the 1 method

Variables - Parameters

Variables - Parameters can be seen within in the whole body of the method where they were created

Class Variables

Class variables are common variables for all methods of the class

Static Variables

Static methods can address only to static variables. Static variable are connected not to the object of the class where they were created but to the class

To address to static variables from the other classes can be done like:

ClassName.variableName

public class Book {

```
    public void add(int info)

    {

        Saving.sum = Saving.sum + info;
        Saving.count++;

    }

public class Saving {

     public static int count = 0;
     public static int sum = 0;

 }

}
```

To public static variables one can address from any method of the program (and not only from the method)

Constants

Constant is a variable the meaning of which one can not change (for example, number pi)

```
final type name = meaning;
```

Method and class also can be final. Final method can not be overrided. Final class can not be as a basic class for inheritance

Global constants

To create global constants one has to create static variables of the class and also define them as public and final:

```
Class Book {

    public static final String SOURCE_ROOT = “c:\\projects\\folder1\\”;
    public static final int DISPLAY_WIDTH = 1010;
    public static final int DISPLAY_HEIGHT = 500;


}
```

Shadowing Variables

One can not in one method create several local variables with the same names. In different methods - can.

Class variables and local variables of the method can have the same name:

```
public class Book {

    public int count = 0;
    public int sum = 0;


   public void add(int info)
   {

        sum = sum + info;
        int sum = info * 4;
        count++;

    }
}
```

In add method local variable **sum** was created and it to the end of the method action covers/shadows class variable sum

To address to the class variable add keyword this before its name:

this.name

```
public void add(int info)

{

        int sum = info * 2;
        this.sum = this.sum + info;

}
```

If there is shadowing not a class variable but STATIC class variable use class name instead of **this**:

ClassName.name

Book.sum = Book.sum + info;

Variables within cycle for

Variables like counter i++ can be seen only in cycle body and in the title for(int i….) of the cycle for

Objects

Object is data (variables) and methods which was grouped together for processing this data

Objects variables are “object data” or “object state”

Object methods is “object behavior”

Each object like each variable has a type. This type is defined only one time during object creation and then can not be changed. Object type is its class

To work with an object just save the link to the object in the variable and then call methods for this variable:

variable.method()

To create an object of the certain class use **operator new**:

Class variable = new Class(parameters);

For example:
Scanner keyboard = new Scanner(System.in); OR
int[] info = new int[5];

new Scanner - object of the class/example of the class
The class is called the class of the object

Objects - Constants or Immutable Objects (not changing)

Object Oriented Programming

Principles of OOP:

1) Abstraction
2) Encapsulation
3) Inheritance
4) Polymorphism

Abstraction - extracting main features and components

Encapsulation - hiding complex detail (like one can see only a search box on the page: use private methods and remain only 2-3 public methods)

Inheritance is about not inventing a bicycle from the very beginning (parent - child class)

Polymorphism - one interface - different backend (wheel … other parts of the car are the same, but it can be fire engine car, an ambulance, a lorry…)

OOP is about dividing a program on objects. Objects contain variables and functions (staff and actions with this staff). It is useful for changing the first version of the product, to add new features…

Inheritance:

```
Class Child
    Class Parent
```

In the parent class we have common methods for child classes. And child classes has their special features

```
Class Parent
{

    int x;
    int y;

}
```

```
Class Child extends Parent
{

   Int m;

}
```

Inheritance can be only from 1 class

Class Math

Method:

double sin(double m) - returns sinus of the angle m

Variables - Constants:

double Math.PI
double Math.E

Square root from k: double sqrt(double k)
Cube root from k: double cbrt(double k)
K in degree M: double pow(double k, double m)

To return minimum of two numbers: Math.min(k, m);
To return maximum of two numbers: Math.max(k, m);
To return minimum of several numbers: Math.min(Math.min(k, m), Math.min(n, t))

To return random number: Math.random()

Packages

Names for a lot of classes can not be unique. That's why in Java all classes are grouped in packages. It is like files and folders on the computer

The full name of class folder with all subfolders is a package of the class

The name of the package (way to the file) is separated with dots (not with / like): myProject.src

The folder src (source) contains all classes of 1 program. It is the first folder after the folder with the name of the project: myProject\src

Classes are put into the packages but not to the src folder

A library is a set of classes

A file contains:

```
package packageName;

import The name of the package;

public class className
{

}
```

Package name is the same as the folder name. File name is the same as the public class

Method Thread.sleep()

```
Thread.sleep(4000); // stop the program on 4 seconds
Thread.sleep(500); // stop the program on half of a second
Thread.sleep(60* 60 * 7000); // stop the program on 7 hours
```

Properties: Getter & Setter

All class fields are private!

To give access to other classes to get or change data within the objects of your class add to your class 2 methods: get-method and set-method

```
class Pupil {

    private String name;  // private - field name

    public Pupil (String name)

    {

        this.name = name; // field initiating using the constructor

}


public String getName()

{

      return name; // getName() - method RETURNS the meaning of the field name

}

public void setName(String name)

{

     this.name = name;  // setName() - method CHANGES
                        //   the meaning of the field name

}

}
```

Any other class can not change the meaning of the field name

It is because a lot of variables are private not public (like private String name)

Objects Comparison

```
String x = new String(“Oh”);
String y = new String(“Oh”);

System.out.println(x.equals(y));
```

In Java all classes are inherited from the class Object

Popular Errors

1) Missing semicolon (;)
2) Not closed “ “
3) Missing + in concatenation
4) Not closed { } (write right { exactly under the left })
5) Missing ( ) (if-else condition)
6) Incorrect title of “main” method (psvm + Tab)
7) File name is different from class name (if in Java file there are several classes only 1 class is public)
8) Missing keyword “package” (in the beginning of each class put package name to which this class belongs)
9) Missing keyword “import” (to use the class from the other program: import + full class name)
10) Do not compare object using operator == (instead use equals.)
11) Object elements without meaning (int array = **new int[];**)
12) Several public classes in 1 file (1 file = 1 public class)
13) Missing “static” in variable/method (method “main” can addresses only to static variable/method: public **static** int = 4; - not in main, but in other method, but “main” method takes this variable to its method from this not main method)
14) Class constructor as a method (constructor name is the same as class name & constructor does not have type of the result)
15) Incorrect inheritance of the interfaces (class is inherited from the class: class Table **extends** Furniture. Class is inherited from the interface: class Table **implements** Wood. Interface is inherited from the interface: **interface** Material **extends** Wood)
16) Missing break in switch operator (switch {**case** - **break**; **case** - **break**; }

Classes - Wrappers

Primitive types have their “twins”: classes - wrappers. Classes - wrappers (int - **Integer**, float - **Float**, double - **Double**, char - **Character**, boolean - **Boolean**, byte - **Byte**, short - **Short**, long - **Long**) which save within 1 field with a certain type of meaning. They “wrap” primitive meanings with the classes

All objects of classes - wrappers are immutable (unchanged)

To turn int type to Integer class: Integer name = new Integer(meaning);

To turn Integer class to int type: int name = variable.intValue();

```
Integer price = new Integer(30);
int y = price.intValue();
```

(y == 30)

Autoboxing: int => Integer

Unboxing: Integer => int

Before auto/unboxing:

To create object of Integer class:

```
Integer x = Integer.valueOf(5); // wrap 5 with Integer class
int y = x.intValue(); // get the meaning from Integer object
Integer m = new Integer(y + 5) // creating the new meaning Integer == 10
```

After auto/unboxing:

```
Integer x = 5;
int y = x;
Integer m = x + y;
```

Collections in Java: Class ArrayList

ArrayList works the same as the array, but can CHANGE ITS SIZE

To create ArrayList object:

```
ArrayList<Type>  name = new ArrayList<Type>();
```

Type - type of elements in ArrayList collection

```
ArrayList<Integer> list = new ArrayList<Integer>();
```

From the beginning the length of a new list = 0

Methods:

1) void add (type value) // add passed element to the list
2) void add (int index, type value) // add the element to the certain place in the list
3) type get (int index) // returns the element with number of index
4) void set (int index, type value) // changes the meaning of the element with number of index to value
5) type remove (int index) // removes the element with number of index. Returns deleted element
6) type remove (type value) // removes the element: the element should be passed to the list. If there are several such elements the first element of them will be deleted
7) void clear() // clears the list = deletes all elements from the list
8) boolean contains (type value) // checks if there is the element value in the list
9) boolean isEmpty() // checks if the list is empty
10) int size() // returns list size - quantity of list elements
11) type[ ] toArray(type[ ] array) // returns the array which contains the same elements as the list. The array should be passed to the method

Switch Operator

Switch is an operator of multiple choice

```
switch(expression)

{

   case meaning1: code1;
   case meaning2: code2;
   case meaning3: code3;

}
```

If the meaning of the expression = meaning1, code1 is executed …

```
int price = 10;

switch(price)

{

    case 8: System.out.println(“Low”);
    case 9: System.out.println(“Medium”);
    case 10: System.out.println(“High”);

}
```

Without break operator all cases will be written in the console.

```
int price = 10;

switch(price)

{

    case 8:
        System.out.println(“Low”);
        break;
    case 9:
         System.out.println(“Medium”);
         break;
    case 10:
         System.out.println(“High”);

}
```

After the last case break operator should not be written because it is the last block of code

Analog of else (if any of cases match the condition): default (default action):

```
int price = 150;

switch(price)

{

    case 8:
        System.out.println(“Low”);
        break;
    case 9:
         System.out.println(“Medium”);
```

```
        break;
    case 10:
        System.out.println(“High”);
        break;
    default:
        System.out.println(“Go out of the shop!”);


}
```

Switch is like if-else. But if-else should be used within the if condition for each separate case there are different complex expressions

In switch can be used only these types as a parameter:

1) byte, short, int
2) char
3) String
4) all enum types

Exceptions

Objects - exceptions are inherited from Throwable class

Catching exceptions: block try - catch:

```
try
{
    code where an error can appear
}
catch(ExceptionType name)
{
    code of exception processing
}
```

If during the execution of basic code (in try block) exceptions do not appear, code in the catch block will not be executed. If exception appears - will be executed if exception type is the same as the type of the variables in ( )

There can be several catch blocks

Throwing exceptions:

throw exception;

```
try
{
    int x = 5/0;
}
catch(Exception m)
{
   System.out.println("Exception was caught");
   throw m;
}
```

To execute code if/if not exception appears add to try-catch block **finally {}** block. Finally block will be executed always

Java Date Time API - set of classes to solve all time problems