**after class 2:**

**13.**

a) attributes: kolor, rozmiar, model, ilość głośników, typ wejścia do ładowania

behaviors: włącz, wyłącz, zadzwoń

b) attributes: stan konta, czy ma blika, czy jest otwarte, typ konta, jaką walutę przechowuje

behaviors: zablokuj, wykonaj przelew, pokaż stan

**before class 3:**

**1.** static fields and methods - static fields can be accessed without an instance of the class. To make a static field, you simply place the static keyword before the declaration of a class’s member variable, or field

A static method of a class can be called without an instance of that class. As a result, static methods don’t have any access to instance variables, or instance fields, because instance variables store information about an instance of a class.

Instance variables are created when an object is created with the use of the keyword 'new' and destroyed when the object is destroyed. Static variables are created when the program starts and destroyed when the program stops.

Use static when you want to provide class level access to a method, i.e. where the method should be callable without an instance of the class.

**2.** from the "main()" method

the statement is located in class

**3.** constructor - a special type of subroutine called to create an object. It prepares the new object for use, often accepting arguments that the constructor uses to set required member variables.

**4.** You can either use array declaration or array literal (but only when you declare and affect the variable right away, array literals cannot be used for re-assigning an array).

For primitive types:

int[] myIntArray = new int[3];

int[] myIntArray = {1, 2, 3};

int[] myIntArray = new int[]{1, 2, 3};

For classes, for example String, it's the same:

String[] myStringArray = new String[3];

String[] myStringArray = {"a", "b", "c"};

String[] myStringArray = new String[]{"a", "b", "c"};

The third way of initializing is useful when you declare an array first and then initialize it, pass an array as a function argument, or return an array. The explicit type is required.

String[] myStringArray;

myStringArray = new String[]{"a", "b", "c"};

----------------------------------------------------

Syntax for default values:

int[] num = new int[5];

Or (less preferred)

int num[] = new int[5];

Syntax with values given (variable/field initialization):

int[] num = {1,2,3,4,5};

Or (less preferred)

int num[] = {1, 2, 3, 4, 5};

Note: For convenience int[] num is preferable because it clearly tells that you are talking here about array. Otherwise no difference. Not at all.

**5.** Java provides the following ITERATIVE STATEMENTS:

- while statement

The while statement is used to execute a single statement or block of statements repeatedly as long as the given condition is TRUE. The while statement is also known as Entry control looping statement. The syntax and execution flow of while statement is as follows.

"ipublic class WhileTest {

public static void main(String[] args) {

int num = 1;

while(num <= 10) {

System.out.println(num);

num++;

}

System.out.println("Statement after while!");

}

}"

- do-while statement

The do-while statement is used to execute a single statement or block of statements repeatedly as long as given the condition is TRUE. The do-while statement is also known as the Exit control looping statement. The do-while statement has the following syntax.

"public class DoWhileTest {

public static void main(String[] args) {

int num = 1;

do {

System.out.println(num);

num++;

}while(num <= 10);

System.out.println("Statement after do-while!");

}

}"

- for statement

The for statement is used to execute a single statement or a block of statements repeatedly as long as the given condition is TRUE. The for statement has the following syntax and execution flow diagram.

"public class ForTest {

public static void main(String[] args) {

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

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

}

System.out.println("Statement after for!");

}

}"

- for-each statement

The Java for-each statement was introduced since Java 5.0 version. It provides an approach to traverse through an array or collection in Java. The for-each statement also known as enhanced for statement. The for-each statement executes the block of statements for each element of the given array or collection.

"public class ForEachTest {

public static void main(String[] args) {

int[] arrayList = {10, 20, 30, 40, 50};

for(int i : arrayList) {

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

}

System.out.println("Statement after for-each!");

}

}"

CONDITIONAL STATEMENTS:

- if Statement

It is one of the most simple and significant conditional statements which will execute only if the condition is true. It must be specified in the lowercase letters “if” and uppercase “IF” wouldn’t work as Java is a case sensitive language and hence it will generate an error. The below-given figure shows the basic syntax of “if-statement” in Java:

"if (condition)

{

statement(s); //executes if condition is true

}"

example:

"int a=75, b=100;

if (a < b)

{

System.out.println("The value of a is less than b");

}"

- else Statement

The if statement executes the code only if the condition is true and to tackle the falsy conditions an else statement will be used. Following will be the syntax for the else statement in java:

"if (condition)

{

statement(s); //executes if condition is true

}

else

{

statement(s) //executes if condition is false

}"

example:

"int a=175, b=100;

if (a < b)

{

System.out.println("The value of a is less than value of b");

}

else

{

System.out.println("The value of b is less than value of a");

}"

- else if Statement

It is used when we have to tackle more than two conditions, it determines a new condition and executes the statements if the condition specified within the “else if” statement is true. The below-given snippet shows the working of the else-if statement in java:

"if (condition)

{

statement(s); //executes if condition is true

}

else if(condition)

{

statement(s) //executes if condition is true

}

else

{

statement(s) //executes if condition is false

}"

example:

"int a=75;

if (a >= 80)

{

System.out.println("Excellent! You Got A Grade");

}

else if(a >60 && a<80)

{

System.out.println("Good.. You Got B Grade");

}

else

{

System.out.println("You Got F Grade.. Better Luck next Time!");

}"

**6**. method overloading - when a class have multiple methods by same name but different parameters