

Objects in memory

Constructors

Returning value from methods



Objects in memory

- There are two types of memory in Java static and dynamic (heap)
- Primitives are stored into the static memory
- Objects are reference data types and are stored into the heap
- The reference to the object is kept in the static memory



Objects in memory

- Objects are created via constructors operator new allocates memory in the heap
- The Garbage collector destroys the unused objects clears the heap
- The destruction of objects is not a programer task –
 the garbage collector does it for you





- Objects are referent types
- Primitives are not referent types
- Dealing with objects is always dealing with its reference
- Declaration of an object creates a reference but it points to nothing – i.e. null





Using = with objects deals only with the reference

What will be printed into the console?

```
Person joro = new Person();
Person mitko = new Person();
joro.age = 18;
mitko = joro;
mitko.age = 21;
System.out.println(joro.age);
```

```
Person joro = new Person();
Person mitko = new Person();
mitko = joro;
```

What happens with the object mitho in the memory?



Constructor

- Constructor is responsible for creating an object
- Constructors don't have a return type it should always return the newly created object
- To constructors can be passed parameters
- Constructors should have a body
- Constructors are always named to the class name



Constructor

Default constructor

Constructor with

Parameters for age and name

```
Person() {

Person(int ageParam, String nameParam) {
   age = ageParam;
   name = nameParam;
}
```



We will start writing example with Car and Person (the classes from the previous lesson) and a new class

CarShop

- 1. First start with adding the fields price and isSportsCar to the class Car
- 2. Write constuctor in class Car:





- This always refers to the current object
- Using this in constructors is good practice

In the following case using this is obligatory.

If this is not used, the scope of age and name is restricted only for the constructor i.e when referencing them, we reference the passed parameters but not the fields

```
public class Person {
   int age;
   String name;

Person(int age, String name){
     this.age = age;
     this.name = name;
   }
}
```

In constuctor

3. Use this and change the parameters' names



More about constructors

- Default constructor a constructor without parameters
- Default constructor is always available if no other constructors are defined
- Each class can have more than one constructor
- If a constructor with parameters is defined, the default constructor is not available



More about constructors

 The constructors can be invoked in the body of another constructor



More about constructors

```
public class Person {
   int age;
   String name;
   double height;
   Person(){}
   Person(int age){
       this();
       this.age = age;
   Person(int age, String name){
       this(age);
       this.name = name;
   Person(int age, String name, double height){
       this(age, name);
       this.height = height;
```

This constructor uses
the default constructor

This constructor uses
the another constructor
which uses the
default constructor



4. Write constuctor in class Car:

Car(String model, boolean isSportCar, String
color, double price, int maxSpeed)

it calls the other constructor and then set the other parameters to the fields. It also checks if the car is sport before setting its maxSpeed to more than 200



In class Person add 2 constructors:

5. Default constructor - it sets age to 0 and weight to 4.0

Change class Person to contain array of Friends instead of one friend

6. Person(String name, long personalNumber,
 boolean isMale)

it calls the default constructor first, then set the values and initialize the friends array with new array with 3 elements

7. Create class Demo with main method and test the constructors of class Car and Person



Declaration

- return type (boolean, int, String, <any other class>)
- Method name (starts with lowerCase, use camelCase convension)
- Brackets (mandatory)
- List with parameters in the brackets (not mandatory)
- Body starts with { and ends with }



Returned types

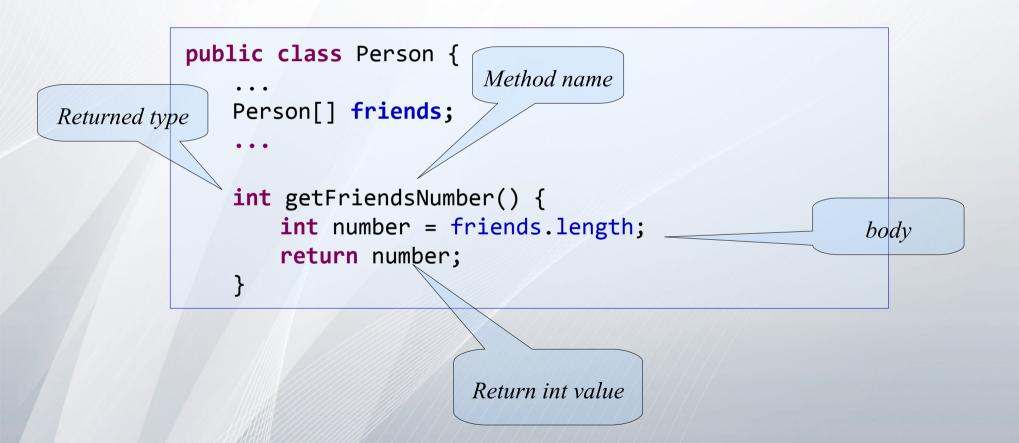
- void the method do not return value
- Any other type object(or primitive) with this type must be returned in the body of method

Keyword *return*

- Used for break the execution of the method and return a value
- Void methods can execute return but without value



Example of method with returned type int





8. Create method in class Car

boolean isMoreExpensive(Car car)

9. Test it in class Demo



10. Create method in class Car

double calculateCarPriceForScrap(double metalPrice)

The price = metalPrice * coef

The coefficient starts from 0.2 and depends of the car's color and if it's sport:

If the color is black or white, 0.05 is added to the coefficient If the car is sport, 0.05 is added to the coefficient

11. Test it in class Demo

To the class Person add fields:

11. money – money of the Person

12. car - reference to his own car



To the class Person add method:

13. void buyCar(Car car)

the person buys the car if he has enough money

To the class Car add method:

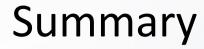
14. void changeOwner(Person newOwner)



To the class Person add method:

15. double sellCarForScrap()

the method returns the money to the person after the car is sold for scrap





- Objects and referent types
- What is a reference
- Constructors
- Default constructor and how to use constructors
- How to call constructor in constructor
- Methods with returned types not void
- How to use return keyword