Constructiors

Encapsulation

Access modifiers

Getters and Setters

Static and final keywords

IT TALENTS
Training Camp

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

IT TALENTS
Training Camp

Constructor

- The constructor is responsible for creating an object
- Constructors don't have a return type – they always return the newly created object
- Parameters can be passed to constructors
- 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;
    }
```

Car, Person, CarShop Example

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 `isSportCar` to the class Car

2. Write constuctor in class Car:

```
Car(String modelParam, boolean isSportCarParam,
String colorParam)
```

it sets the parameters to the fields and sets default values to currentSpeed and gear

Keyword **this**

- This always refers to the current object
- Using **this** in constructors is a 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

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

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

}
```

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

IT TALENTS
Training Camp

More about constructors

- The constructors can be invoked in the body of another constructor
- If a constructor with parameters is defined, the default constructor is not available

More about constructors

```java
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(name, age);
        this.height = height;
    }
}
```

This constructor uses the default constructor

This constructor uses the another constructor which uses the default constructor

Car, Person, CarShop Example

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

Car, Person, CarShop Example

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

Car, Person, CarShop Example

8. Create method in class Car

```
    boolean isMoreExpensive(Car car)
```

9. Test it in class Demo

Car, Person, CarShop Example

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

Car, Person, CarShop Example

To the class Person add fields:

11. `money` – money of the Person

12. `car` – reference to his own car

Car, Person, CarShop Example

To the class Person add method:

13. `void buyCar(Car car)`

the person buy the car if has enough money

To the class Car add method:

14. `void changeOwner(Person newOwner)`

Car, Person, CarShop Example

To the class Person add method:

15. `double sellCarForScrap()`

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

Encapsulation

IT TALENTS
Training Camp

Packages

Packages in java:

- Hierarchical units identical to folders – on the file system the packages are presented as folders
- Provide grouping of related types(classes)
- Provide access protection and space management

```java
package lesson16;

public class Car {
    String model;
    double price;
    boolean isSportCar;
    double maxSpeed;
}
```

Encapsulation

- One of the four fundamental OOP concepts
- The ability of an object to be a container (or capsule) for related properties (fields) and behaviours (methods).
- A protective barrier that prevents the code and data being randomly accessed by other code defined outside the class.
- Benefits:
  - Main benefit is the ability to use the implemented code without breaking its logic and constraints
  - It gives maintainability, flexibility and extensibility

IT TALENTS
Training Camp

Access modifiers

- Access modifiers are used to

  Control access to classes (top level), methods, constructors or fields (bottom level) from outside the class

- For top level (classes) there are *public, package* and in some cases *private(inner classes)*
- For bottom level: *public*, *protected*, *package* and *private*

IT TALENTS
Training Camp

Access modifiers example

public modifier for the class

```java
package lesson06;

public class Person {
    public String name;
    private int age;
    private long personalNumber;
    boolean isMale;
```

public modifier

private modifier

package(default) modifier

Explaining *public*, *private* and *default*

- public – gives access to the class, field or method from everywhere outside the class
- private – access is restricted only within the class
- default/package – visible from within the class and all other classes in the package
-
- Protected – we'll talk about it in the next lessons because it's related to inheritance

IT TALENTS
Training Camp

Purpose of access modifiers

- Problem: If all fields of class Person are public they will be accessible from everywhere which evaluates the Encapsulation principle of OOP
- Accessibility directly to fields is dangerous and unsecure
- For accessing private fields outside the class are used public methods called „getter“ and „setter“

Getters and setters

- Getters are used for getting the value of private field outside the class.
- It should be implemented only if is neccessary
- Setters are void methods and are used for setting the value of private field outsite the class
- Validation can be implemented as part of the setter's body

```
private int age;

public int getAge() {
    return age;
}

public void setAge(int age) {
    if(age >= 0) {
        this.age = age;
    }
}
```

Using keyword *final* for fields

- Can be used for fields, parameters, local variables and classes.
- Used for field, it indicates that the field is constant Once a value is assigned, it cannot be changed during the whole program execution.
- Convension – use uppercase and "_" to separate words(for static final fields)
- Constants must be initialized either after declaring, or in the constructor

```
    private final String NAME = "Ivan";
    private int age = 14;
```

Using keyword *final* for method's parameters

- The same logic as when using with fields - the parameter cannot be changed in the method's body

```
public void setAgeFromOtherPerson(final Person person) {
    this.age = person.getAge();
}
```

- Be careful with fields and parameters of some reference type:

Setting fields or argument of some reference type as final don't guarantee that its state won't be changed. It only guarantee that the reference won't be changed.

Using keyword *final* for a variable in some block of code

Compile error

```
public class Demo {
    public static void main(String[] args) {
        Car bmw = new Car("BMW 330", true, "Red");
        Car ford = new Car("Ford Fiesta", false, "Black", 2000, 330);
        final Car myCar = bmw;
        myCar = ford;

        final int myAge = 20;
        myAge = 21;
    }
}
```

Compile error

Static fields

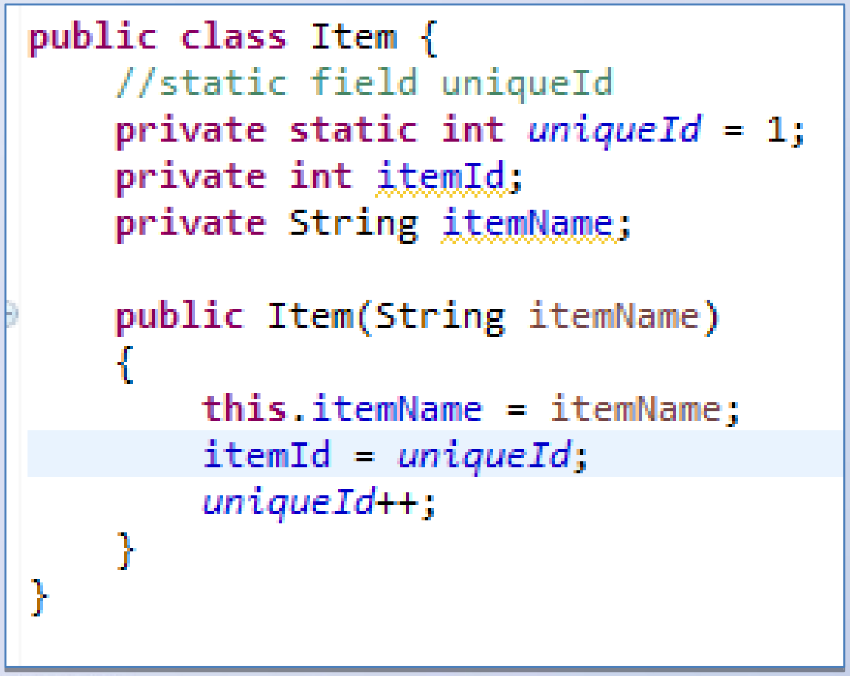
- Keyword *static* indicate the field as static
- Static fields belong to the class – not the instances of a class
- Static fields are shared between the objects because they belong to the class
- Static reference can be and should be referenced via class' name

Static fields

If some object change the value of a static fields, its changed in all object of this class

Try it with few simple classes!

Example – ”uniqueId” field



```
public class Item {
    //static field uniqueId
    private static int uniqueId = 1;
    private int itemId;
    private String itemName;

    public Item(String itemName)
    {
        this.itemName = itemName;
        itemId = uniqueId;
        uniqueId++;
    }
}
```

Exercise

```
public class A {
    public static int x = 0;
    public int y = 4;

    public A(int x, int y){
        this.x = x;
        this.y = y;
    }

    public static void main(String[] args) {
        A a1 = new A(2,3);
        A a2 = new A(7, 9);

        System.out.println(a1.x);
        System.out.println(a2.y);
        a2.y++;
        a1.x += a2.y;
        System.out.println(a1.x);
        a2.y = a1.y - 1;
        System.out.println(a2.y);
    }
}
```

What will be the output from the main method?

Static methods

- Again static keyword is used
- Static method can be and should be called via class name, not via instance of its class
- Static methods CANNOT use non static fields of the class
- main method is example of static method

```
public class Test {
    public static void main(String[] args) {
        double c = Math.pow(2, 10);
        System.out.println(c);
    }
}
```

main method is static

Calling static method of class Math

Accessing static/non static members from static/non static context

context	non static fields or methods	static fields or methods
non static	Yes	Yes
static	No	Yes

IT TALENTS
Training Camp

Christian and God example

- Create class God
- Add some methods to it
- Create class Christian
- Add static field for christian's god
- Add some methods to the class Christian
- Create class Demo, access god and call some methods of object God

IT TALENTS
Training Camp