Encapsulation

Access modifiers

Keyword *final*

Static fields and methods

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 lesson06;

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.
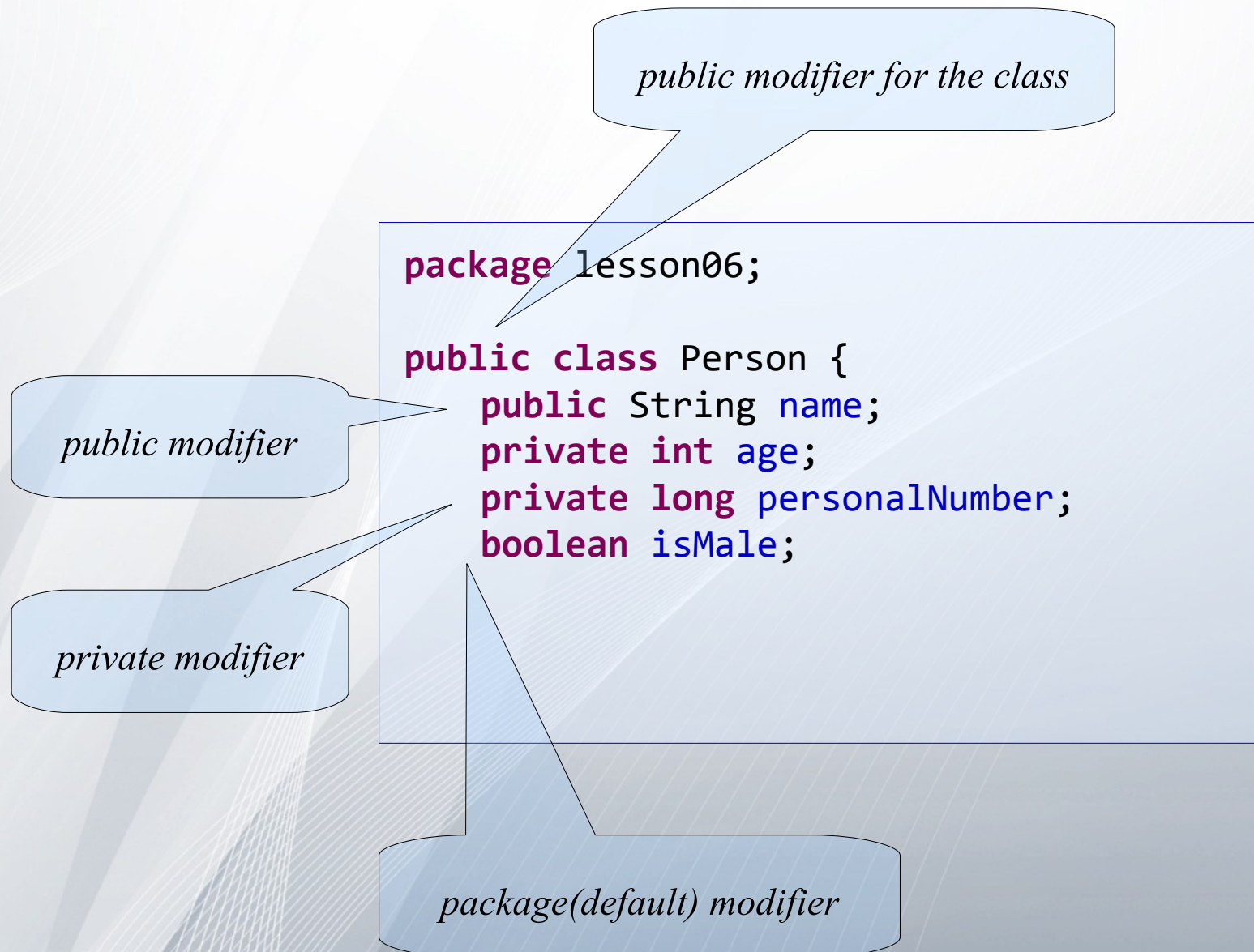
Benefits:

- Main benefit is the ability to use the implemented code without breaking its logic and constraints

- It gives maintainability, flexibility and extensibility

- 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*

Access modifiers example

public modifier for the class

public modifier

private modifier

```
package lesson06;

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

package(default) modifier

- 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

- 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 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

```java
private int age;

public int getAge() {
    return age;
}

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

- 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

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

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

```java
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.

Compile error

```java
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

- 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

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

Try it with few simple classes!

```java
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?

- 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 used non static fields of the class

- main method is example of static method

```java
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

- 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

- What is package

- What is encapsulation and how to achieve it

- Access modifiers

- Getters and setters – purpose and usage

- Final keyword – purpose and initialization of final fields

- Static fields and methods

- How to refer static fields and call static methods