Key Concepts (2)

Object-Oriented programming

Antonius Camara

Last week...

- Classes and Objects
- Constructors
- Exercise: implementing a simple Class, its constructor method and other methods

Today...

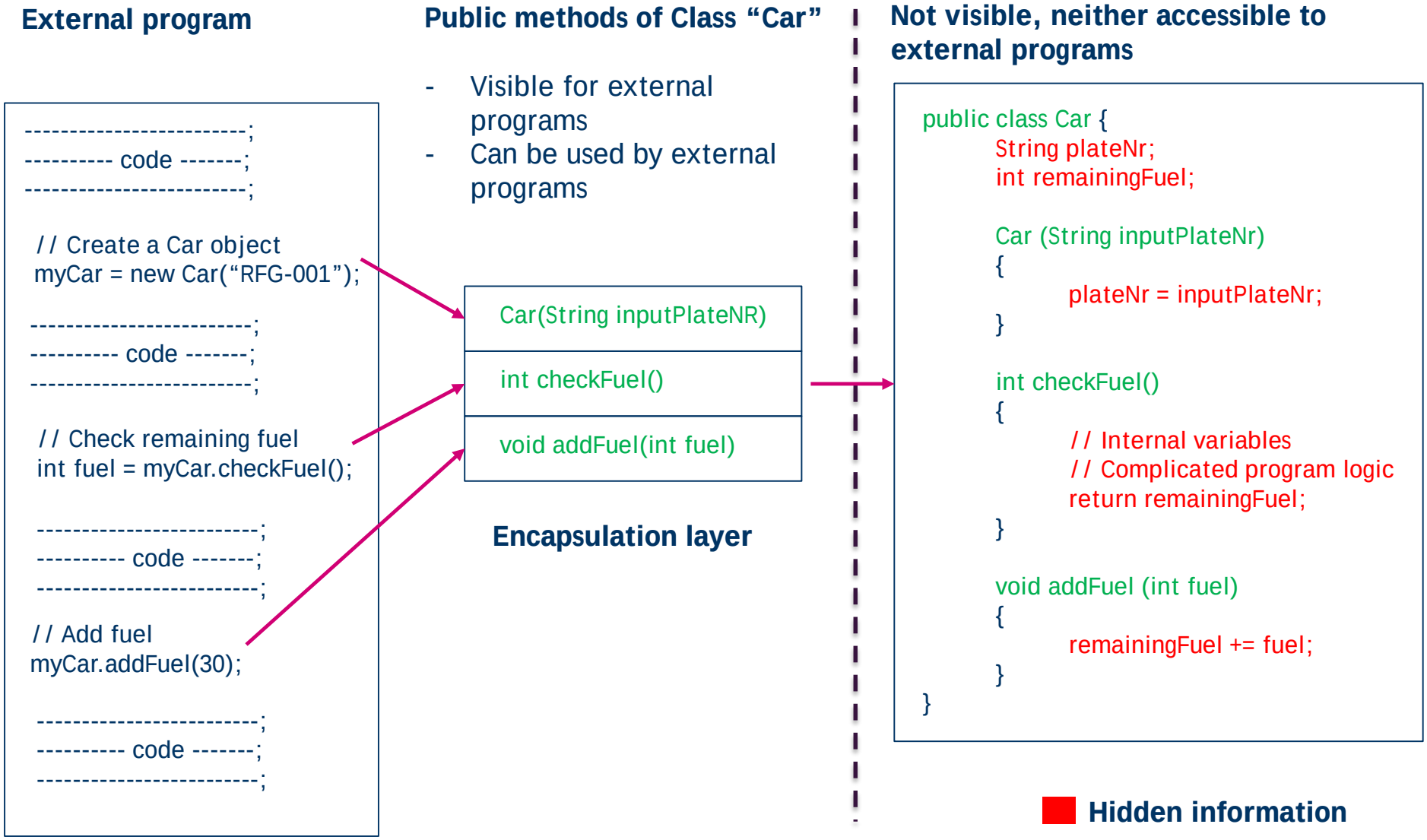
- Encapsulation and Information Hiding
- Access modifiers
- Setters and Getters
- Instance variables vs. Class variables
- The keyword “this”
- Hands-on exercise

Encapsulation and Information Hiding

LAUREA
UNIVERSITY OF APPLIED SCIENCES
Together we are stronger

- Very important concepts in Object-Orientation
  - Allow for better software maintenance and re-usability
- An object's attributes can only be modified and accessed via the object's methods → Encapsulation
- Therefore, it is not possible for an external program to directly manipulate the values of an object's attributes
- External programs do not need to have visibility of what are all the attributes (variables) of a class, neither how a method is actually implemented → Information Hiding
- An external program only needs to know:
  - The methods that can be used in that Class (parameters and return values)

Encapsulation and Information hiding: example



Why encapsulation is important

LAUREA
UNIVERSITY OF APPLIED SCIENCES
Together we are stronger

- External programs don't need to know details on how a class is implemented. It is simpler for programmers just to know and use a limited set of methods that perform some well defined actions. Programmers using a Class don't need to spend time to understand how the Class is implemented
- By hiding access to a Class or object's variables, we avoid "misbehaving" programs to incorrectly modify the values of these variables
- If in the future, we need to modify the programming logic of a method, the external program does not need to be modified, provided that the method name, return value, and input parameters remain the same. All that's needed is a new version/upgrade of the Class files, and the whole program will continue working normally

Encapsulation is not implemented by default!

LAUREA
UNIVERSITY OF APPLIED SCIENCES
Together we are stronger

- Encapsulation and Information Hiding are not automatically implemented by default
- Programmers need to properly design and implement a Class to incorporate Encapsulation and Information Hiding functionality
- The following elements/concepts need to be implemented in a Class:
  - Access modifiers for variables and methods: public, private
  - Setter methods
  - Getter methods

Access modifiers for methods and variables

LAUREA
UNIVERSITY OF APPLIED SCIENCES
Together we are stronger

Access modifier	Scope
public	Can be accessed by any class
protected	Can be accessed by subclasses or classes in the same package
<no modifier>	Can be accessed by classes in the same package
private	Can be accessed only from within the same class

Access modifiers: example

```
public class Car {
        String plateNr;
        int remainingFuel;

        Car (String inputPlateNr)
        {
                plateNr = inputPlateNr;
        }

        int checkFuel()
        {
                // Internal variables
                // Complicated program logic
                return remainingFuel;
        }

        void addFuel (int fuel)
        {
                remainingFuel += fuel;
        }
}
```

Instance variables are explicitly set as private, meaning that these variables can not be accessed from outside the class Car

Applying access modifiers to Class variables and methods

The class constructor methods and other methods are set to public, i.e they are supposed to be accessed from outside the class

```
public class Car {
        private String plateNr;
        private int remainingFuel;

        public Car (String inputPlateNr)
        {
                plateNr = inputPlateNr;
        }

        public int checkFuel()
        {
                // Internal variables
                // Complicated program logic
                return remainingFuel;
        }

        public void addFuel (int fuel)
        {
                remainingFuel += fuel;
        }
}
```

Setters and Getters (methods)

Set and Get methods allow external programs to get and set values of private variables of a class

Get the speed of a Car object

Set the speed of a Car object

Setters methods also have also another important job: making sure the values provided by the external program are valid and according to the properties of a Class. In this example, objects of the Car class have a maximum speed of 240.

```
public class Car {
        private static final int MAX_SPEED = 240;

        private String plateNr;
        private int currentSpeed;

        public Car (String inputPlateNr) {
                plateNr = inputPlateNr;
                currentSpeed = 0;
        }

        public int getSpeed () {
                return currentSpeed;
        }

        public Boolean setSpeed (int speed) {
                if (speed < 0 || speed > MAX_SPEED){
                        return false;
                }
                else {
                        currentSpeed = speed;
                        return true;
                }
        }
}
```

Class variables vs. Instance variables

Static: denotes a **Class variable**. Class variables are shared by all instances of a class and are not bound to a specific instance/object of the Class. Class variables exist even if you don't create any instance/object of the class

Final: the value of the variable can be assigned only once and never again. This is used to define "**Constant**" values used in a Class. Constant variables are written with capital letters.

Instance variables: Instance variables are bound to instances/objects of the class. Each object of a class will have its own set of values for instance variables

```
public class Car {
        private static final int MAX_SPEED = 240;

        private String plateNr;
        private int currentSpeed;

        public Car (String inputPlateNr) {
                plateNr = inputPlateNr;
                currentSpeed = 0;
        }

         public int getSpeed () {
                return currentSpeed;
        }

        public Boolean setSpeed (int speed) {
                if (speed < 0 || speed > MAX_SPEED){
                        return false;
                }
                else {
                        currentSpeed = speed;
                        return true;
                }
        }
}
```

The keyword “this”

LAUREA
UNIVERSITY OF APPLIED SCIENCES
Together we are stronger

Used only inside:

- **Constructor methods**
- **Other class methods**

“this” is used to refer to the object being refered, i.e “this object”

In the previous examples there was not a need to use the keyword “this”, but in this example “this” is needed for example because **plateNr** is both the name of the instance variable and also the name of the input argument in the constructor method. “this” is used to specify the variable being used:

this.plateNr means the plateNr instance variable

```
public class Car {
        private static final int MAX_SPEED = 240;

        private String plateNr;
        private int currentSpeed;

        public Car (String plateNr) {
                this.plateNr = plateNr;
                this.currentSpeed = 0;
        }

         public int getSpeed () {
                return this.currentSpeed;
        }

        public Boolean setSpeed (int speed) {
                if (speed < 0 || speed > MAX_SPEED){
                        return false;
                }
                else {
                        this.currentSpeed = speed;
                        return true;
                }
        }
}
```

Exercise (1/2)

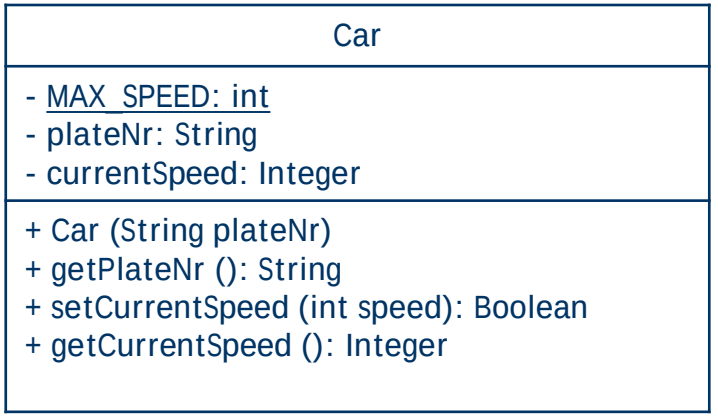
- Create a new Java project in Eclipse
- Create a Class Car according to the following UML Class diagram. Set MAX_SPEED to 240.
- Note the name of the constructor's argument

In UML:

(-) → The minus sign means a **private** method or variable

(+) → The plus sign means a **public** method or variable

An underlined name refers to a **Class variable** or method (static variable or method)



Exercise 2/2

- Under the same project, create a Class CarSpeeds (program). Define a main method for this class
- Under this main method do the following operations:
  - Initialize an array with 3 Car objects. Use the constructor to set the plateNr. Hardcode the plate numbers to: (“HGR-987”, “EFX-395”, “EFX-395”)
  - After the array is initialized, the program asks from the user via the command prompt the speed for each car, and sets the speed for the car calling the method setCurrentSpeed
  - If the user has provided an invalid speed (setCurrentSpeed returns “false”), the program will notify that the speed is invalid and will ask the speed again
  - After all the speeds are successfully set, the program will calculate the speed average of all three cars and print to the screen:
    - “The average speed of cars is xxx Km/h”
- Run the program “CarSpeeds” and check if you got it right. Check the example execution below:

```
Enter the speed of car HGR-987:
80
Enter the speed of car EFX-395:
260
Invalid speed!
Enter the speed of car EFX-395:
240
Enter the speed of car ACW-900:
120
The average speed of cars is 146 Km/h
```