Object-Oriented Analysis & Java

Lecture 2

Florentin Bekier

Classes

Classes and objects

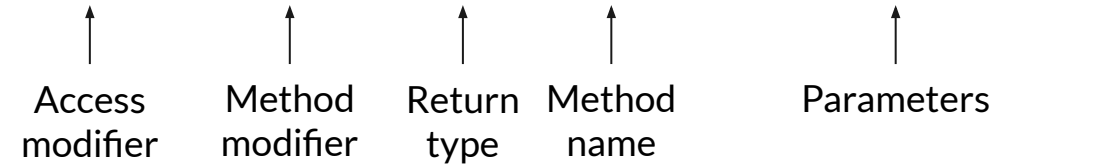
- Programs are composed of *classes*
- Classes are “a blueprint for objects”
  - It defines a new type in the language
  - It defines the state and behaviour of this class’ objects
- An object is an instance of a class
  - It has attributes (state), methods (behaviour) and is unique

Attributes and methods

- Attributes are variables specific to a class
  - Also called *fields*
- Methods are functions/procedures

Methods in Java

```
public static void main(String[] args)
```



Access modifiers

- `public`: Anyone can access it
- `protected`: Can be accessed by the subclasses and classes from the same package
- `package-private`: Classes from the same package can access it (default access level)
- `private`: Only the class itself can access it

Static methods

- Also called *class methods*
- Called on the class itself
- Don’t need a specific instance to be called
    - Cannot use `this`

Example: `Math.sqrt(36.0);`

Static attributes

- Also called *class variables*
- Used to represent a value that is not linked to a specific instance
- Can be accessed the same way as static methods

Example: `System.out`

Keyword `this`

- Inside a class, this keyword is a reference to the current instance
- Implicitly used when calling an attribute or a method inside a class method
  - `speed = sp; // Same as this.speed = sp`

Constructors

- Special methods to create a new instance of a class
- Same name as the class
- Can take parameters
- Don’t return anything (`void`)
- All classes must have at least one constructor

Constructors

```
public class Car {
    private double speed;

    public Car(double sp) {
        this.speed = sp;
    }
}
```

Getters and setters

- Getters (or accessors) are methods used to get the value of an attribute
- Setters (or mutators) are methods used to set the value of an attribute

Getters and setters

```
public class Car {
    private double speed;

    public double getSpeed() {
        return this.speed;
    }

    public void setSpeed(double sp) {
        this.speed = sp;
    }
}
```

Using an object

- Create a new object:
  - `Car myCar = new Car(30.0);`
- Use methods on an object:
  - `myCar.getSpeed();`

Method `toString`

- Automatically called when an object has to be converted to a string
- By default, all classes have a `toString` method inherited from the `Object` class
- Make it a habit to redefine it

```
@Override
public String toString() {
    return "Car with a speed of " + getSpeed();
}
```

Equality

- For primitive types, == compares their value
- For reference types, == compares the reference itself (i.e., if they reference the same object)

```
Car c1 = new Car(10.0);
Car c2 = new Car(10.0);
Car c3 = c1;
```

- Use the method `equals` to test the equality of two objects semantically

Method `equals`

- Return `true` if `obj` “equals to” `this`
- By default, all classes have an `equals` method with the same behaviour as ==
- Make it a habit to redefine it

```
@Override
public boolean equals(Object obj) {
    return ...;
}
```

Does the equality make sense?

- Reflexivity: `a.equals(a)` must be true
- Symmetry: `a.equals(b)` must be equivalent to `b.equals(a)`
- Transitivity: If `a.equals(b)` is true and `b.equals(c)` is true, then `a.equals(c)` must be true
- Nullability: `a.equals(null)` must be false

Method equals

```
@Override
public boolean equals(Object obj) {
    if (obj instanceof Car) {
        Car c = (Car) obj;
        return c.getSpeed() == this.getSpeed();
    }

    return false;
}
```

Arrays

Creating an array

- In Java, arrays are objects
- Declaration: `String[] names;`
- Initialization:
  - `String[] names = new String[10];`
  - `String[] names = {"Frank", "John", "Paul"};`

Arrays of arrays

```
int[][] matrix = new int[20][50];
int[][] reducedMatrix = {{1, 2, 3}, {4, 5}, null};
```

Copying an array

Use the method `clone` to duplicate an array:

```
int[] nums = new int[33];
int[] numsCopy = (int[]) nums.clone();
```

Be careful: the method won't copy the objects or the sub-arrays!