Java

Controll Statements and OOP

Tobias Hanf, Manik Khurana

8. November 2021

Java-Course

Overview

Recalling last session

Conclusion

Control Structures

- taking input
- If-Then-Else
- for and while loop
- Conditions

Na Na Na Na Na Na Na Na Batman

OOP in Java

Object Oriented Programming

What is OOP?

- OOP - Object Oriented Programming
- want to model the real world
- take things and create digital copy
- two main concepts - Objects and Classes

What is a class?

A blueprint for a series of objects with common attributes/methods
Example Car:

What is a class?

A blueprint for a series of objects with common attributes/methods
Example Car:

- attributes
  - wheels
  - windows
  - color
  - engine
  - ...

What is a class?

A blueprint for a series of objects with common attributes/methods
Example Car:

- attributes
  - wheels
  - windows
  - color
  - engine
  - ...
- methods
  - accelerate
  - break
  - toggle turn signal
  - ...

What is an object?

An object is an instantiation of a class

- a class does not really exist
- only objects exist
- taking a class and filling it with data
- can be created and destroyed

Class Student

```java
public class Student {

    // Attributes
    private String name;
    private int matriculationNumber;


    // Methods
    public void setName(String name) {
        this.name = name;
    }

    public int getMatriculationNumber() {
        return matriculationNumber;
    }

}

```

private, public, protected?

- there is an outside and an inside

private, public, protected?

- there is an outside and an inside
- not everything should be visible from the outside

private, public, protected?

- there is an outside and an inside
- not everything should be visible from the outside
- have to tell the compiler what to show and what not

private, public, protected?

- there is an outside and an inside
- not everything should be visible from the outside
- have to tell the compiler what to show and what not
- keywords
  - private - only visible from inside the class
  - public - visible from everywhere
  - protected - visible to child classes

private, public, protected?

- there is an outside and an inside
- not everything should be visible from the outside
- have to tell the compiler what to show and what not
- keywords
  - private - only visible from inside the class
  - public - visible from everywhere
  - protected - visible to child classes
- attributes should always be private (or protected)

private, public, protected?

- there is an outside and an inside
- not everything should be visible from the outside
- have to tell the compiler what to show and what not
- keywords
  - private - only visible from inside the class
  - public - visible from everywhere
  - protected - visible to child classes
- attributes should always be private (or protected)
- methods which are for internal use should be private

private, public, protected?

- there is an outside and an inside
- not everything should be visible from the outside
- have to tell the compiler what to show and what not
- keywords
  - private - only visible from inside the class
  - public - visible from everywhere
  - protected - visible to child classes
- attributes should always be private (or protected)
- methods which are for internal use should be private
- every other methods can/should be public

Class Student

```java
public class Student {

    // Attributes
    private String name;
    private int matriculationNumber;


    // Methods
    public void setName(String name) {
        this.name = name;
    }

    public int getMatriculationNumber() {
        return matriculationNumber;
    }

}

```

Creation

We learned how to declare and assign a primitive datatype.

```
1
2
3
```

```
int a; // declare a
a = 273; // assign 273 to a
```

The creation of an object works similar.

```
1
2
3
```

```
Student example = new Student();
// create an instance of Student
```

The **object** derived from a **class** is also called **instance**. The variable is called the **reference**.

Class Student

```java
public class Student {

    // Attributes
    private String name;
    private int matriculationNumber;


    // Methods
    public void setName(String name) {
        this.name = name;
    }

    public int getMatriculationNumber() {
        return matriculationNumber;
    }

}

```

What is a method?

What is a method?

- piece of code which can be reused

What is a method?

What is a method?

- piece of code which can be reused
- can take input from the outside

What is a method?

What is a method?

- piece of code which can be reused
- can take input from the outside
- can return data to the outside

What is a method?

What is a method?

- piece of code which can be reused
- can take input from the outside
- can return data to the outside

Other names:

- function
- procedure
- subroutine

Why use methods?

Why use methods?

- programmers are lazy $\rightarrow$ do more with less code

Why use methods?

Why use methods?

- programmers are lazy $\rightarrow$ do more with less code
- better structure and less changes

Why use methods?

Why use methods?

- programmers are lazy $\rightarrow$ do more with less code
- better structure and less changes
- reduces errors

Why use methods?

Why use methods?

- programmers are lazy $\rightarrow$ do more with less code
- better structure and less changes
- reduces errors
- important for OOP

Calling a Method

```
1   public class Student {
2
3       private String name;
4
5       public String getName() {
6           return name;
7       }
8
9       public void setName(String newName) {
10          name = newName;
11      }
12
13  }
14
```

The class *Student* has two methods: *void setName()* and *String getName()*.

Calling a Method

```java
public class Main {

    public static void main(String[] args) {
        Student example = new Student(); // creation
        example.setName("Jane"); // method call
        String name = example.getName();
        System.out.println(name); // Prints "Jane"
    }

}

```

You can call a method of an object after its creation with **reference.methodName();**.

Calling a Method

```java
public class Student {

    private String name;

    public void setName(String newName) {
        name = newName;
        printName();   // Call own method
        this.printName(); // Or this way
    }

    public void printName() {
        System.out.println(name);
    }

}

```

You can call a method of the own object by simply writing **methodName();** or **this.methodName();**

Methods with Arguments

```
1   public class Calc {
2
3       public void add(int summand1, int summand2) {
4           System.out.println(summand1 + summand2);
5       }
6
7       public static void main(String[] args) {
8           int summandA = 1;
9           int summandB = 2;
10          Calc calculator = new Calc();
11          System.out.print("1 + 2 = ");
12          calculator.add(summandA, summandB);
13          // prints: 3
14      }
15
16  }
17
```

Methods with Return Value

A method without a return value is indicated by **void**:

```
1            public void add(int summand1, int summand2) {
2                System.out.println(summand1 + summand2);
3            }
4
```

A method with an **int** as return value:

```
1            public int add(int summand1, int summand2) {
2                return summand1 + summand2;
3            }
4
```

Calling Methods with a return value

```java
public class Calc {

    public int add(int summand1, int summand2) {
        return summand1 + summand2;
    }

    public static void main(String[] args) {
        Calc calculator = new Calc();
        int sum = calculator.add(3, 8);
        System.out.print("3 + 8 = " + sum);
        // prints: 3 + 8 = 11
    }

}

```

Constructors

```
1   public class Calc {
2
3       private int summand1;
4       private int summand2;
5
6       public Calc() {
7           summand1 = 0;
8           summand2 = 0;
9       }
10
11  }
12
```

A constructor gets called upon creation of the object

Constructors with Arguments

```
1  public class Calc {
2
3      private int summand1;
4      private int summand2;
5
6      public Calc(int x, int y) {
7          summand1 = x;
8          summand2 = y;
9      }
10
11 }
12
```

```
1  [...]
2  Calc myCalc = new Calc(7, 9);
3
```

A constructor can have arguments as well!

Conclusion

An Example

You want to program an enrollment system, for a programming course.

Your classes are:

student	who wants to attend the course
lesson	which is a part of the course
tutor	the guy with the bandshirt
room	where your lessons take place
...	

Class Student

```
1                public static void main(String[] args) {
2                    Student peter = new Student();
3                    peter.changeName("Peter");
4                }
5
```