

Java

Inheritance

Julius Felchow (Mail - julius.felchow@mailbox.tu-dresden.de),
Benjamin Weller (Mail - benjamin.weller@tu-dresden.de)

25. November 2019

Java-Kurs

Overview

1. Arrays

Multi-Dimensional Array

2. Recall OOP

3. Visibilities

4. Inheritance

Inheritance

Constructor

Implicit Inheritance

Arrays

Array

An array is a data-type that can hold a **fixed number** of elements. An Element can be any simple data-type or object.

```
1 public static void main(String[] args) {  
2  
3     int[] intArray = new int[10];  
4     intArray[8] = 7; // assign 7 to the 9th element  
5     intArray[9] = 8; // assign 8 to the last element  
6  
7     System.out.println(intArray[8]); // prints: 7  
8 }
```

You can access every element via an index. A n-element array has indexes from 0 to (n-1).

Array Initialization

You can initialize an array with a set of elements.

```
1  public static void main(String[] args) {  
2  
3      int[] intArray = {3, 2, 7};  
4  
5      System.out.println(intArray[0]); // prints: 3  
6      System.out.println(intArray[1]); // prints: 2  
7      System.out.println(intArray[2]); // prints: 7  
8  }
```

Alternative Declaration

There two possible positions for the square brackets.

```
1    public static void main(String[] args) {  
2  
3        // version 1  
4        int[] intArray1 = new int[10];  
5  
6        // version 2  
7        int intArray2[] = new int[10];  
8    }
```

2-Dimensional Array

Arrays work with more than one dimension. An m-dimensional array has m indexes for one element.

```
1 public static void main(String[] args) {  
2  
3     // an array with 100 elements  
4     int[][] intArray = new int[10][10];  
5  
6     intArray[0][0] = 0;  
7     intArray[0][9] = 9;  
8     intArray[9][9] = 99;  
9 }
```

Assignment with Loops

Loops are often used to assign elements in arrays.

```
1 public static void main(String[] args) {  
2  
3     int[][] intArray = new int[10][10];  
4  
5     for(int i = 0; i < 10; i++) {  
6         for(int j = 0; j < 10; j++) {  
7             intArray[i][j] = i*10 + j;  
8         }  
9     }  
10 }
```


Arrays with objects

Loops are often used to assign elements in arrays.

```
1 public static void main(String[] args) {  
2  
3     Student[][] studentArray = new Student[10][10];  
4  
5     for(int i = 0; i < 10; i++) {  
6         for(int j = 0; j < 10; j++) {  
7             intArray[i][j] = new Student();  
8         }  
9     }  
10 }
```

Recall OOP

Classes

- Like blueprints
- Contain definitions of attributes and functions

```
1 public class Student {  
2  
3     // Attributes  
4     private String name;  
5     private int matriculationNumber;  
6  
7     // Methods  
8     public void setName(String name) {  
9         this.name = name;  
10    }  
11  
12    public void setMatriculationNumber(int number) {  
13        this.matriculationNumber = number;  
14    }  
15 }  
16
```

Objects

- An object (instance) can be created from a class
- The structure of the object follows the class definitions
- Multiple objects can be created

We learned how to declare and assign a primitive datatype.

```
1  int a; // declare a
2  a = 273; // assign 273 to a
3
```

The creation of an object works similar.

```
1  Student example = new Student();
2
```

Student example

```
1 // Create a new Student
2 Student anna = new Student();
3
4 // Set name and matriculation number of our new student
5 anna.setName("Anna");
6 anna.setMatriculationNumber(12345678);
7
8 // Create another Student
9 Student berta = new Student();
10
11 berta.setName("Berta");
12 berta.setMatriculationNumber(12345679);
13
```

Visibilities

Visibilities

- public
- private
- protected

Visibilities

```
1      public class Student {  
2          public String getName() {  
3              return "Peter";  
4          }  
5  
6          private String getFavouritePorn() {  
7              return "...";  
8          }  
9      }  
10  
11      // [...]   
12      exampleStudent.getName(); // Works!  
13      exampleStudent.getFavouritePorn(); // Error
```


Inheritance

Students and Tutors

Our class *Tutor* is a kind of *Student* denoted by the keyword **extends**.

- *Tutor* is a **subclass** of the class *Student*
- *Student* is the **superclass** of the class *Tutor*

```
1  public class Tutor extends Student {  
2  
3  }  
4
```

As mentioned implicitly above a class can have multiple subclasses. But a class can only inherit directly from one superclass.

Example

```
1  public class Student {
2
3      // Attributes
4      private String name;
5      private int matriculationNumber;
6
7      // Methods
8      public void setName(String name) {
9          this.name = name;
10     }
11
12     public void printName() {
13         System.out.println("This is student " + this.
14         name);
15     }
16 }
```

Inherited Methods

The class *Tutor* also inherits all methods from the superclass *Student*.

```
1  public class Faculty {  
2  
3      public static void main(String[] args) {  
4  
5          Tutor tutor1 = new Tutor();  
6  
7          tutor1.setName("Benny");  
8  
9          tutor1.printName();  
10         // prints: This is student Benny  
11     }  
12 }  
13
```

Override Methods

The method `printName()` is now additionally defined in *Tutor*.

```
1 public class Tutor extends Student {  
2  
3     @Override  
4     public void printName() {  
5         System.out.println("This is tutor " + this.name)  
6     }  
7 }  
8
```

`@Override` is an annotation. It helps the programmer to identify overwritten methods. It is not necessary for running the code but improves readability. What annotations else can we discuss in a future lesson.

Override Methods

Now the method `printName()` defined in *Tutor* will be used instead of the method defined in the superclass *Student*.

```
1  public class Faculty {  
2  
3      public static void main(String[] args) {  
4  
5          Tutor benny = new Tutor();  
6  
7          benny.setName("Benny");  
8  
9          benny.printName();  
10         // prints: This is tutor Benny  
11     }  
12 }  
13
```

Super()

If we define a **constructor with arguments** in *Student* we have to define a constructor with the same list of arguments in every subclass.

```
1      public class Student {
2
3          private String name;
4          private int matriculationNumber;
5
6          public Student(String name, int number) {
7              this.name = name;
8              this.matriculationNumber = number;
9          }
10
11         public void printName() {
12             System.out.println("This is student " + name);
13         }
14     }
15
```

Super()

For the constructor in the subclass *Tutor* we can use `super()` to call the constructor from the superclass.

```
1  public class Tutor extends Student {
2
3      public Tutor(String name, int matriculationNumber) {
4          super(name, matriculationNumber);
5      }
6
7      @Override
8      public void printName() {
9          System.out.println("This is tutor " + name);
10     }
11 }
12
```


Super() - Test

```
1 public class Faculty {  
2  
3     public static void main(String[] args) {  
4         Tutor benny = new Tutor("Benny", 12345678);  
5  
6         benny.printName();  
7         // prints: This is tutor Benny  
8     }  
9 }  
10
```

Every class is a subclass from the class *Object*. Therefore every class inherits methods from *Object*.

See <http://docs.oracle.com/javase/7/docs/api/java/lang/Object.html> for a full reference of the class *Object*.

toString()

Tutor is a subclass of *Object*. Therefore *Tutor* inherits the method `toString()` from *Object*.

`System.out.println(argument)` will call `argument.toString()` to receive a printable String.

```
1      public class Faculty {  
2  
3          public static void main(String[] args) {  
4              Tutor benny = new Tutor("Benny", 12345678);  
5  
6              System.out.println(benny);  
7              // prints: Tutor@_some_HEX-value_  
8              // for example: Tutor@4536ad4d  
9          }  
10     }  
11
```

Override toString()

```
1      public class Tutor extends Student {  
2  
3          public Tutor(String name, int matriculationNumber) {  
4              super(name, matriculationNumber);  
5          }  
6  
7          @Override  
8          public String toString() {  
9              return "This is Tutor " + name;  
10         }  
11     }  
12
```

Override toString() - Test

```
1  public class Faculty {  
2  
3      public static void main(String[] args) {  
4          Tutor benny = new Tutor("Benny", 12345678);  
5  
6          System.out.println(benny);  
7          // Prints: This is tutor Benny  
8      }  
9  }  
10
```