Java

Abstract

Vincent Gerber, Tilman Hinnerichs

Java Kurs

6. Juni 2018

1. An Addtition to Control Statements: Switch-Statements
2. Abstract classes and methods
3. Interfaces
   - Overview
   - Example
   - Multiple Interfaces

Differentiate

A common problem:

```
1   public static void main (String[] args) {
2
3   int address = 2;
4
5   if (address == 1) {
6   System.out.println("Dear Sir,");
7   } else if (address == 2) {
8   System.out.println("Dear Madam,");
9   } else if (address == 4) {
10  System.out.println("Dear Friend,");
11  } else {
12  System.out.println("Dear Sir/Madam,");
13  }
14  }
15
```

Differentiate with Switch

How to write it using the Switch-statement:

```
1   public static void main (String[] args) {
2
3   int address = 2;
4
5   switch(address) {
6   case 1:
7   System.out.println("Dear Sir,");
8   break;
9   case 2:
10  System.out.println("Dear Madam,");
11  break;
12  case 4:
13  System.out.println("Dear Friend,");
14  break;
15  default:
16  System.out.println("Dear Sir/Madam,");
17  break;
18  }
19  }
20
```

Differentiate with Switch

Depending on a variable you can switch the execution paths using the keyword **switch**. This works with `int`, `char` and `String`.

The variable is compared with the value following the keyword case. If they are equal the program will enter the corresponding case block. If nothing fits the program will enter the default block.

```
1     public static void main (String[] args) {
2     switch(intVariable) {
3     case 1:
4     doSomething();
5     break;
6     default:
7     doOtherThings();
8     break;
9     }
10    }
11
```

Exercises

- Create a simple calculator with +,-,/ and *
- Extend our wheel class with a pressure state function (low, medium, full,...)

Abstract Class

The keyword **abstract** denotes an abstract class.

```
1  public abstract class AbstractExample {
2
3  }
4
```

- You can not create objects from an abstract class.
- Abstract classes can extend other abstract classes and can implement interfaces [1].
- Abstract classes can be extended by normal and abstract classes.

[1] Interfaces will be discussed later

An Example

We want to model some furniture.

Hence, we want some chairs, tables, ... as classes which extend the furniture class (UML diagram on whiteboard)

We do **NOT** want an object of type furniture. In this case we label the class furniture as abstract.

Methods

An abstract class may has concrete methods and may has abstract methods.

```
1  public abstract class AbstractExample {
2
3      public void printHello() {
4          System.out.println("Hello");
5      }
6
7      public abstract String getName();
8  }
9
```

An abstract method forces the class to be abstract as well.

Subclasses

The subclass has to implement abstract methods or has to be abstract as well. All concrete methods will be regularly inherited.

```
1  public class Example extends AbstractExample {
2
3      @Override
4      public String getName() {
5          return "Example";
6      }
7  }
8
```

Why using Abstract?

- Lets you build a template for the children classes
- Lets you add functionality classes
- Able to avoid redundancy of code in different classes with similar functionality
- Ability to specify default implementations of methods

Interfaces

An **interface** is a well defined set of constants and methods a class have to **implement**.

$\Longrightarrow$ Makes the handling of other objects much easier

Interfaces

An **interface** is a well defined set of constants and methods a class have to **implement**.

$\Longrightarrow$ Makes the handling of other objects much easier

For Example: A post office offers to ship letters, postcards and packages. With an interface *Trackable* you can collect the positions unified. It is not important how a letter calculates its position. It is important that the letter communicate its position through the methods from the interface.

Interface Trackable

An interface contains method signatures. A signature is the definition of a method without the implementation.

```
1    public interface Trackable {
2
3    public int getStatus(int identifier);
4
5    public Position getPosition(int identifier);
6    }
7
```

Note: The name of an interface often (not always) ends with the suffix *-able*.

Letter implements Trackable

```
1   public class Letter implements Trackable {
2
3   public Position position;
4   private int identifier;
5
6   public int getStatus(int identifier) {
7   return this.identifier;
8   }
9
10  public Position getPosition(int identifier) {
11  return this.position;
12  }
13  }
14
```

The classes *Postcard* and *Package* also implement the interface *Trackable*.

Overview

Time for some UML!

Access through an Interface

```
1   public static void main(String[] args) {
2
3   Trackable letter_1 = new Letter();
4   Trackable letter_2 = new Letter();
5   Trackable postcard_1 = new Postcard();
6   Trackable package_1 = new Package();
7
8   letter_1.getPosition(2345);
9   postcard_1.getStatus(1234);
10  }
11
```

Two Interfaces

A class can implement multiple interfaces.

```
1  public interface Buyable {
2
3  // constant
4  public float tax = 1.19f;
5
6  public float getPrice();
7  }
8
```

```
1  public interface Trackable {
2
3  public int getStatus(int identifier);
4
5  public Position getPosition(int identifier);
6  }
7
```

Postcard implements Buyable and Trackable

```
1  public class Postcard implements Buyable, Trackable {
2
3  public Position position;
4  private int identifier;
5  private float priceWithoutVAT;
6
7  public float getPrice() {
8  return priceWithoutVAT * tax;
9  }
10
11 public int getStatus(int identifier) {
12 return this.identifier;
13 }
14
15 public Position getPosition(int identifier) {
16 return this.position;
17 }
18 }
19
```

Access multiple Interfaces

```
1   public static void main(String[] args) {
2
3   Trackable postcard_T = new Postcard();
4   Postcard postcard_P = new Postcard();
5   Buyable postcard_B = new Postcard();
6
7   postcard_T.getStatus(1234);
8   postcard_B.getPrice();
9   postcard_P.getStatus(1234);
10  postcard_P.getPrice();
11  }
12
```

postcard_P can access both interfaces.
postcard_T can access Trackable.
postcard_B can access Buyable.

Abstract Class vs. Interface

Abstract class
- Template for children classes
- Able to add functionality
- No multi-inheritance
- Default implementations of methods

Interface
- Template for implementations
- **NO** functionality in interface itself
- Sort of multi-inheritance
- **NO** default implementation

The usage of both can overlap.

Exercises

Let's start with a small drawing application

- Create a base class for every drawable object (with functions like draw, getPosition, ...)
- Create two different drawable classes (e.g circle, rectangle)
- Implement an interface Size with getVolume and getArea as functions. Every Drawable should implement this interface
- Declare a function with an object as input and the volume and area as output.