Java OOP

Konrad Raue, Oliver Scholz

26. November 2019

Gliederung

1. Nachtrag
2. Enum
3. innere Klassen
4. abstrakte Klassen
5. Interface
6. Keywords
7. Iterator
8. Observer
9. Nächste Woche

Nachtrag

Collections

Collections:

statische Klasse, die Funktionen für Klasse Collection und deren Unterklassen bietet
https://docs.oracle.com/javase/8/docs/api/?java/util/Collections.html

wichtige Object-Methoden:

- hashCode()
- toString()

Enum

enum

```
public enum Enum{
    MONTAG,
    DIENSTAG,
    MITTWOCH,
    DONNERSTAG,
    FREITAG,
    SAMSTAG,
    SONNTAG
}
```

```
public class Klasse{
    private Enum day;

    public Klasse(Enum day){
        this.day = day;
    }

    public void dayForward(){
        if(day == Enum.MONTAG){
            day = Enum.DIENSTAG;
        }
        ...
    }
}
```

innere Klassen

innere Klassen

```
1  public class Klasse{
2      public enum Enum{
3          MONTAG(0), DIENSTAG(1), MITTWOCH(2), DONNERSTAG(3),
4          FREITAG(4), SAMSTAG(5), SONNTAG(6);
5
6          private int value;
7
8          Enum(int value){
9              this.value = value;
10         }
11     }
12
13     private Enum day;
14
15     public Klasse(Enum day){
16         this.day = day;
17     }
18
19     public void dayForward(){
20         int dayInt = day.ordinal();
21         dayInt = (dayInt + 1) % 7;
22         for(Enum e : Enum.values()) {
23             if(e.value == dayInt) {
24                 day = e;
25                 break;
26             }
27         }
28     }
29 }
```

Von außerhalb z.B. mit Klasse.Enum.MONTAG ansteuerbar.

abstrakte Klassen

abstract

```java
public abstract class AbstrakteKlasse{
    private int value;

    public AbstrakteKlasse(int value){
        this.value = value;
    }

    public int getValue(){
        return value;
    }

    public abstract void work();
}
```

```java
public class Klasse extends AbstrakteKlasse{
    public Klasse(int value){
        super(value);
    }

    public void work(){
        //do something
    }
}
```

Alle abstrakten Methoden müssen implementiert werden.

Interface

interface

```java
public interface Person{
    public void work();
}
```

```java
public class Student implements Person{
    private String name;
    private String studiengang;
    private int fachsemester;

    public Student(String name, String studiengang, int fachsemester
    ){
        this.name = name;
        this.studiengang = studiengang;
        this.fachsemester = fachsemester;
    }

    public void work(){
        //do something
    }
}
```

Alle Methoden müssen implementiert werden.

abstract vs interface

Interface with Default Methods	Abstract Class
Inside interface every variable is Always public static final and there is No chance of instance variables	Inside abstract class there may be a Chance of instance variables which Are required to the child class.
Interface never talks about state of Object.	Abstract class can talk about state of Object.
Inside interface we can't declare Constructors.	Inside abstract class we can declare Constructors.
Inside interface we can't declare Instance and static blocks.	Inside abstract class we can declare Instance and static blocks.
Functional interface with default Methods Can refer lambda expression.	Abstract class can't refer lambda Expressions.
Inside interface we can't override Object class methods.	Inside abstract class we can override Object class methods.

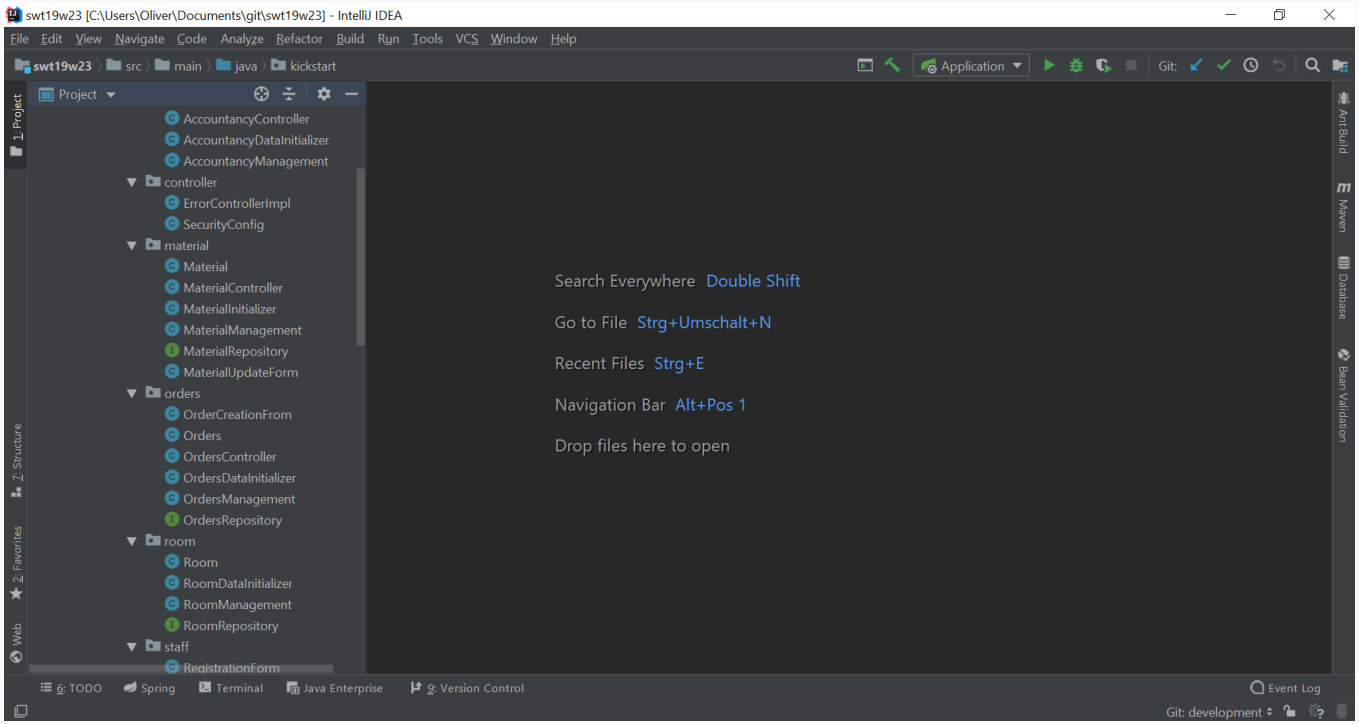
oops interface vs abstract class

Interface	Abstract class
Interface support multiple inheritance	Abstract class does not support multiple inheritance
Interface does'n Contains Data Member	Abstract class contains Data Member
Interface does'n contains Cunstructors	Abstract class contains Cunstructors
An interface Contains only incomplete member (signature of member)	An abstract class Contains both incomplete (abstract) and complete member
An interface cannot have access modifiers by default everything is assumed as public	An abstract class can contain access modifiers for the subs, functions, properties
Member of interface can not be Static	Only Complete Member of abstract class can be Static

Keywords

Sichtbarkeiten

- private: nur Klasse selbst kann darauf zugreifen
- public: alle Klassen können darauf zugreifen
- protected: alle Vererbungen dieser Klasse können darauf zugreifen + package-private
- package-private: standardmäßig, alle Klassen im selben Package können darauf zugreifen



static, final

- static
  - für Methoden, Attributen oder Blocks verwendet
  - bei Methoden können diese auch ohne Objekt mit Klasse.Methode aufgerufen werden, beinhalten keine nicht statischen Methoden und Attribute, können statische Attribute ändern
  - bei Attributen werden diese zu Klassenvariablen, in allen Objekten gleich, können auch ohne Objekt mit Klasse.Attribut aufgerufen werden
- final
  - für Klassen, Methoden, Attribute, Methodenparametern verwendet
  - bei Klassen können keine Klassen von dieser vererbt werden
  - bei Methoden können diese nicht mehr überschrieben werden bei vererbten Klassen
  - bei Attributen können diese einmal Wert zugewiesen bekommen (entweder direkt oder im Konstruktor), danach sind sie nicht mehr "veränderbar"(konstant bis auf inneren Zustand)
  - bei Methodenparametern sind diese nach Übergabe nicht mehr veränderbar

Iterator

Iterator

```java
import java.util.ArrayList;

public class IteratorImplementation implements java.util.Iterator<String>{
    private ArrayList<String> res;
    private ArrayList<String> cache;

    public IteratorImplementation(ArrayList<String> res){
        this.res = new ArrayList<>(res);
        cache = new ArrayList<>(res);
    }

    public boolean hasNext(){
        cache = new ArrayList<>(res);
        if(!res.isEmpty()){
            return true;
        }
        return false;
    }

    public String next(){
        if(hasNext()){
            for(String s : cache){
                if(!s.contains('s')){
                    res.remove(s);
                }else{
                    res.remove(s);
                    return s;
                }
            }
        }
        return null;
    }
}
```

Observer

Observer

```
1  import java.util.Observer;
2  import java.util.Observable;
3
4  public class KlassenObserver implements Observer{
5      private Klasse klasse;
6
7      public void update(Observable observable, Object arg){
8          klasse = (Klasse) observable;
9          System.out.println(klasse.getContent());
10     }
11 }
```

Observable

```
1  import java.util.Observable;
2
3  public class Klasse extends Observable{
4      private String content;
5
6      public Klasse(String content){
7          this.content = content;
8      }
9
10     public String getContent(){
11         return content;
12     }
13
14     public void setContent(String content){
15         this.content = content;
16         setChanged();
17         notifyObservers();
18     }
19 }
```

Nutzung

```java
public class Nutzung{
    public static void main(String[] args){
        Klasse klasse = new Klasse("");
        KlassenObserver observer = new KlassenObserver();
        klasse.addObserver(observer);
        klasse.setContent("Hello");
        klasse.setContent("World!");
    }
}
```

Nächste Woche

Nächste Woche

- Comparator
- Exception
- Testen mit JUnit
- UML