**PR2 – Formular für Lesenotizen**

**SS2021**

|  |  |  |  |
| --- | --- | --- | --- |
| Nachname  Lushaj | Vorname  Detijon | Matrikelnummer  1630149 | Abgabedatum:  29.04.21 |

**L.6.6 Abstrakte Klassen**

**Abstrakte Klasse**: Eine nicht **instanziierbare Klasse**, die dazu dient, gemeinsamen Code zu implementieren und als Superklasse beerbt zu werden. Eine abstrakte Klasse kann...

* einige Methoden konkret implementieren
* andere Methoden abstrakt, d. h. ohne Methodenrumpf, belassen
* nicht instanziiert werden (Compilerfehler)

• Interface:

* ausschließlich abstrakte Methoden\*.
* Man kann alle Interface-Methoden abstract deklarieren. Ist aber unnötig und unüblich.

• Abstrakte Klasse:

* Mischung aus abstrakten und nicht-abstrakten Methoden.

• Konkrete Klasse:

* Instanziierbar
* alle Methoden mit Rumpf.

*UML Klassenname der abstrakten Klasse kursiv.*

Syntax:

**public** **abstract** **class** classname **implements** HasArea { ... }

**public** **abstract** **class** Shape **implements** HasArea {

**public** **abstract** **double** perimeter();

@Override **public** **abstract** **double** area(); //nicht noetig

//Subklassen erben auch die abstrakte area-Methode von HasArea

**public** **double** compactness() {

... }

}

* Die Klasse Shape wurde nur eingeführt, um gemeinsamen Code in einer “künstlichen” Klasse zu vereinen – nicht, um tatsächlich Objekte dieser Klasse zu erzeugen
* Wir sollten nur verhindern, dass Instanzen (Objekte) gebildet werden.

🡪 Klassen können nur **von einer Klasse erben**, aber **beliebig viele Interfaces** implementieren.

**L.6.9 Functional Interfaces / Lambda-Ausdrücke**

@FunctionalInterface

**public** **interface** Comparator<T>

* Es gibt Interfaces, die nur eine abstrakte Methode besitzen.

**Lambda-Ausdruck**: Angabe von Parametern und Rückgabewert zur Beschreibung einer Funktion.

Beispiele:

UnaryOperator<Integer> f= (x)->x\*x ; // f ist eine Funktion

UnaryOperator<Integer> f= **new** UnaryOperator<Integer>() {

@Override **public** Integer apply(Integer x) {

**return** x \* x;

}

};

Auch denkbar: Rumpf ohne Rückgabe bei void-Methode:

Consumer<String> printer= s -> System.out.println(s);

printer.accept("Hello"); // prints Hello

Es sind auch mehrere Parameter möglich:

BinaryOperator<Double> fsum= (x,y) -> x + y ;

System.out.println(fsum.apply(4.0, 5.0)); // prints 9.0

Oder auch keine Parameter:

Supplier<Double> r = () -> Math.random() ;

System.out.println(r.get());

// prints 0.20374821418062938 or other random value

Syntax:

(parameters ) -> expression

(Integer x) -> x \* x

(x) -> {

int sq= x \* x;

return sq;

}

**Externe vs. Interne Iteration**

Beispiel für externe Iteration:

**for** (Integer i : list) { System.***out***.println(i); }

internen Iteration: Methoden-Referenz

list.forEach(i -> System.out.println(i));

list.forEach(System.out::println); // kürzer

**L.6.8 Sortieren / lokale und anonyme innere Klassen**

Comparator<OBJEKT> implementieren Collections.sort(list, Comparator);

|  |  |
| --- | --- |
| **public** **class** MyAreaComparator **implements** Comparator<HasArea> {  @Override **public** **int** compare(HasArea a1, HasArea a2) {  **return** Double.*compare*(a1.area(), a2.area());  **if** (a1.area() < a2.area()) **return** -1;  **if** (a1.area() > a2.area()) **return** 1;  **return** 0;  }  } | java.util.Comparator  ArrayList<HasArea> liste= **new** ArrayList<HasArea>();  liste.add(**new** Rectangle(2.0, 3.0));  // Flächeninhalt 6.0  liste.add(**new** Circle(1.0));  // Flächeninhalt 3.14  Collections.sort(liste, **new** MyAreaComparator()); |

**Lokale Klasse**: Eine innerhalb einer Methode deklarierte Klasse.

**Syntax:**

... methodname(...) {

statement(s);

**class** classname {

attribute(s) and/or method(s);

}

statement(s);

}

* Comparator<OBJEKT> kann in der Klasse implementiert  
  werden!

**Anonyme Klasse**: Eine innerhalb einer Methode deklarierte   
Klasse ohne Namensangabe.

**public** **static** **void** main(String[] args) {

ArrayList<Person> liste= **new** ArrayList<Person>();

liste.add(**new** Person("kek",12));

liste.add(**new** Person("hans",16));

Comparator<Person> cmp= **new** Comparator<Person>() {

@Override **public** **int** compare(Person a, Person b) {

**return** Integer.*compare*(a.getAlter(), b.getAlter());

}

};

//Beispiel für Lambda-Ausdruck

Comparator<Spieler> cmp= (a, b) -> Integer.*compare*(a.getnum(), b.getnum()); **//Lambda-Ausdruck**

Collections.sort(liste, cmp);

}

Collections.sort(liste, **new** Comparator<Person>() {

**...**

});

**L.6.8.4 Natürliche Sortierung / Comparable**

* Eine Standardsortierung (natürliche Sortierung) kann durch erben von java.lang.Comparable implementiert werden.

**public** **class** Ding **implements** Comparable<Ding> {

**private** String a;

**private** **int** b;

// natürliche Sortierung: aufsteigend nach a.

@Override **public** **int** compareTo(Ding other) {

// Impl. abstützen auf String.compareTo:

**return** **this**.a.compareTo(other.a);

}

...

}

Syntax:

**public** **interface** Comparable<T> {

**public** **int** compareTo(T other);

}

|  |  |
| --- | --- |
| import java.io.File;  import java.io.FileInputStream;  import java.io.FileNotFoundException;  import java.io.IOException;  import java.io.InputStreamReader;  public class DetectZip {  /\*\*  \* returns zip oder no zip anhand des Bytecodes  \* soll mit 50hex 4Bhex beginnen -> ZIP  \* genau die beiden ersten Bytes überprüfen  \* keine Annahme über den Dateinamen  \*  \* Falls die Datei mit dem übergebenen Dateinamen nicht existiert oder  \* nicht lesbar ist oder falls gar keiner oder zwei oder mehr Kommandozeilenparameter  \* angegeben wurden, soll Ihr Programm den Text error ausgeben. \*/  } | public static void main(String[] args) throws IOException {  String error = "error";  if (args.length != 1) {  System.out.println(error);  return;  }  FileInputStream fis = null;  try {  File file = new File(args[0]);  fis = new FileInputStream(file);  } catch (FileNotFoundException e) {  System.out.println(error);  return;  }  String a = Integer.toHexString(fis.read());  String b = Integer.toHexString(fis.read());  fis.close();  if (!a.equals("50") || !b.equals("4b")) {  System.out.println("no zip");  return;  }  System.out.println("zip");  } |