Karlsruher Institut für Technologie

Programmieren – Wintersemester 2016/2017

Architecture-driven Requirements Engineering (ARE)
`https://sdqweb.ipd.kit.edu/wiki/Programmieren`
Jun.-Prof. Dr.-Ing. Anne Koziolek

Beispielaufgaben Präsenzübung 2

Allgemeine Hinweise[1]

Am 25.01.2017 findet die Präsenzübung statt. Informationen zur Hörsaaleinteilung und zum Zeitplan werden rechtzeitig auf der Vorlesungshomepage und in der Vorlesung bekanntgegeben. Im Folgenden finden Sie Aufgaben, die beispielhaft für die Aufgaben in der Präsenzübung sind.

Hinweis: Die Aufgaben werden in der Präsenzübung nur auf Deutsch gestellt. Falls Sie ein Wörterbuch benötigen, geben Sie dieses **bis spätestens 18.01.2017** im Sekretariat von Prof. Reussner (Gebäude 50.34, Raum 328. Öffnungszeiten Montag bis Freitag, 9:00 Uhr bis 13:00 Uhr) zur Überprüfung ab. **Nicht durch uns geprüfte Wörterbücher können nicht verwendet werden.** Sie erhalten das Wörterbuch in der Präsenzübung von uns zurück. Falls Sie nicht an der Präsenzübung teilnehmen, können Sie es wieder im Sekretariat abholen.

All questions in the written test („Präsenzübung“) will be in German. If you need a dictionary, hand it in for checking at the secretary's office of Professor Reussner (building 50.34, room 328. Opening hours: Monday to Friday, 9:00 to 13:00) **at the latest on 18.01.2017**. **If we have not checked the dictionary, you will not be allowed to use it during the test.** We will return the dictionary to you immediately before the written test. If you don't participate in the written test you can pick the dictionary up at the secretary's office.

Die Beispielaufgaben werden nicht abgegeben oder korrigiert.

Hinweis: Für eine korrekte Lösung müssen nicht unbedingt alle Lücken ausgefüllt werden. Die Größe der Lücken steht weiterhin nicht unbedingt in Relation zur Länge des Texts, der eingefüllt werden muss.

Hinweis: Sie müssen bei der Beantwortung der Präsenzübungsaufgaben keine Vorgaben zum Code-Style einhalten wie sonst bei der Bearbeitung der Übungsblätter. Insbesondere können Sie nichtsprechende und kurze Methoden- und Variablennamen wählen.
Den Bezeichner `System.out.println` können Sie in Ihrer Lösung mit `sysout` abkürzen.

[1] Die Hinweise sind bis auf den letzten die gleichen wie auf Übungsblatt 4.

Aufbau der Präsenzübung

Für die Präsenzübung werden Sie ca. 20 Minuten Bearbeitungszeit haben. Die Präsenzübung prüft nur Wissen ab, dass sie bei der eigenständigen Bearbeitung der Übungsblätter erworben haben. Die Präsenzübung ist auf den Umgang mit der Sprache Java und Implementierungswissen fokusiert. Das bedeutet, Sie müssen wissen, wie Konstrukte wie Klassen, Methoden, Schnittstellen, Vererbungsbeziehungen, Variablendeklarationen, Array-Zugriffe oder Attribute in Java syntaktisch ausgedrückt werden und was die Bedeutung dieser Konstrukte ist. Um dieses Wissen nachzuweisen, müssen Sie einerseits kleine Codeausschnitte schreiben oder ergänzen, als auch das Verhalten von Code oder eventuelle Fehler in kleinen Codeausschnitten erkennen.

Das bedeutet insbesondere, dass *kein* in der Vorlesung behandeltes Hintergrundwissen, wie beispielsweise Details zur IEEE-Gleitkommazahlendarstellung oder Wissen über die Koch-Kurve, abgeprüft wird. Weiterhin müssen Sie keine Schnittstellen von Methoden der Java-API oder Namen von Methoden auswendig lernen. Falls nötig, werden diese in den Aufgaben angegeben. Die Präsenzübung wird Aufgaben aus verschiedenen Kategorien enthalten:

- **Java-Basics**. Grundlegende Java-Syntax wie Deklaration von Variablen, Klassen, Methoden, Sichtbarkeit, Überschattung und Gültigkeitsbereiche von Variablen, Zugriff auf Felder, Logische Operatoren (`&&`, `||`, `!`), primitive Typen (`int`, `double`, `boolean`). Einfache Arithmetik, Modulo (%), String-Konkatenation. ...
- **Kontrollfluss**. `for`-, `while`- und `do-while`-Schleifen, `if`, `else if`, `else`. Kombination: in `for`-Schleife Eigenschaft von Element prüfen, verschachtelte `for`-Schleifen. Iterieren über Einträge in einem Feld. Einfache Rekursion. ...
- **Objekt-Orientierung**. Polymorphie, Interface-Implementierung. Überschreiben und Überladen von Methoden. Getter-/Setter-Methoden. Konstruktoren. Sichtbarkeit. ...

Die Beispielaufgaben in diesem Dokument sind jeweils einer der Kategorien zugeordnet.

Hinweis: Die Anzahl der Aufgaben in diesem Beispieldokument ist ausdrücklich nicht gleich der in den 20 Minuten zu bearbeitenden, sondern mehr.

Aufgaben

1. Aufgabe: Java-Basics (Beispiel)

(Lösung auf Seite 11)

Ergänzen Sie die folgende Klasse so, dass gilt: `theText` ist ein öffentliches Attribut (*field*), das eine Zeichenfolge enthält, `numbers` ist ein privates Attribut, das ein Feld (*array*) von Ganzzahlen enthält.

```
class ValueHolder {
    [          ] [                ] theText;
    [          ] [                ] numbers;
}
```

2. Aufgabe: Java-Basics (Beispiel)

(Lösung auf Seite 11)

Ergänzen Sie die folgende Schnittstelle um eine Methode `doSomething`, die ein Feld (*array*) von Ganzzahlen entgegennimmt und keine Rückgabe hat.

```
interface Worker {
    [                                                  ]
}
```

3. Aufgabe: Kontrollfluss (Beispiel)

(Lösung auf Seite 11)

Ergänzen Sie den folgenden Code so, dass beim Aufruf von `doNTimes(int)` die Methode `doSomething()` `n` mal aufgerufen wird. Es gilt immer $n \geq 0$.

```
public static void doSomething() { /* ... */ }

public static void doNTimes(int n) {
    [                                                  ]
}
```

4. Aufgabe: Kontrollfluss (Beispiel)

(Lösung auf Seite 12)

Ergänzen Sie den folgenden Code so, dass jedes zweite Element des Felds (*array*) `values` ausgegeben wird, beginnend mit dem zweiten Eintrag. Für das Feld `new int[] { 42, 5, 17, 7, 3, 1, 5, 10 }` wäre dies beispielsweise `5, 7, 1, 10`. Hinweis: Die Anzahl der Elemente eines Arrays `values` erhalten Sie mit `values.length`.

```
public static void printSkip(int[] values) {
  for (int i =                ;                          ;                ) {

    System.out.println(values[i]);

  }
}
```

5. Aufgabe: Kontrollfluss (Beispiel)

(Lösung auf Seite 12)

Ergänzen Sie den folgenden Code so, dass nacheinander die Werte in `values` ausgegeben werden, die an den in `positions` definierten Stellen stehen. Sie können davon ausgehen, dass `positions` nur Werte zwischen einschließlich 0 und `values.length - 1` enthält. Hinweis: Die Anzahl der Elemente eines Arrays `arr` erhalten Sie mit `arr.length`.

```
public static void printValuesAt(String[] values, int[] positions) {
  for (int i =                ;                          ;                ) {
    System.out.println(                                                  );
  }
}
```

6. Aufgabe: Kontrollfluss (Beispiel)

(Lösung auf Seite 12)

Ergänzen Sie den folgenden Code so, dass alle ungeraden Werte in `values` ausgegeben werden (mit `System.out.println`). Sie können davon ausgehen, dass `values` nicht `null` ist. Hinweis: Die Anzahl der Elemente eines Arrays `arr` erhalten Sie mit `arr.length`.

```
public static void printOdds(int values[]) {
  for (int i =                ;                          ;                ) {

  }
}
```

7. Aufgabe: Kontrollfluss (Beispiel)

(Lösung auf Seite 13)

Es sei die Methode `evaluate()` mit dem Rückgabetyp `boolean` gegeben. Ergänzen Sie den folgenden Code so, dass die Methode `evaluate()` so lange aufgerufen wird, bis sie zum ersten Mal `true` zurückgibt. Zählen Sie in der Variable `i`, wie oft in Folge `false` zurückgegeben wurde.

```
int i = 0;
while (                                          ) {
    
    
}
System.out.println(i);
```

8. Aufgabe: Kontrollfluss (Beispiel)

(Lösung auf Seite 13)

Ergänzen Sie den folgenden Code der Methode `doSomething(int)` so, dass:

- Wenn `i` ungerade ist, erhöhe `i` um 1,
- wenn dies nicht galt und wenn `i` gerade und größer als oder gleich 3 ist, gebe den Text „Fall 2“ aus.

```
public static int doSomething(int i) {
  if (                                           ) {
    
    
    
    
    
    
    System.out.println("Fall 2");
  }

  return i;
}
```

9. Aufgabe: Objekt-Orientierung – Objekterzeugung (Beispiel)

(Lösung auf Seite 13)

Es sei eine Klasse `Date` mit den beiden Konstruktoren `Date(int dayOfYear)` und `Date(int day, int month)` gegeben. Ergänzen Sie den folgenden Code so, dass `date0` mit dem 42-sten Tag des Jahres (also `dayOfYear = 42`) belegt wird und `date1` mit dem Datum mit `day = 7` und `month = 10`.

```
Date date0 =                                      ;
Date date1 =                                      ;
```

10. Aufgabe: Objekt-Orientierung – Attribute (Beispiel)

(Lösung auf Seite 13)

Ergänzen Sie die Klasse `SuperCollider` mit einem öffentlichen statischen Attribut (*field*) `myValue` das eine Ganzzahl (*integer*) enthalten kann und einem privaten Attribut `myText`, das einen String enthalten kann.

```
class SuperCollider {

}
```

11. Aufgabe: Objekt-Orientierung – Getter und Setter (Beispiel)

(Lösung auf Seite 14)

Ergänzen Sie die folgende Klassendefinition um einen Getter und einen Setter für das Attribut `value`. Setzen Sie im Setter nur neue Werte, die größer gleich dem Attribut (*field*) `min` und kleiner gleich dem Attribut `max` sind.

```
class MagicSuperCompiler {
  private int min;
  private int max;

  private int value;

  // Getter for value

  // Setter for value

}
```

12. Aufgabe: Objekt-Orientierung – Getter und Setter (Beispiel) (Lösung auf Seite 14)

Ergänzen Sie die folgende Klassendefinition um den Setter `setValueAt` und `getValueAt`, die jeweils auf das Attribut (*field*) `values` und den Wert an der Stelle `pos` zugreifen. Sie können davon ausgehen, dass `values` immer ungleich `null` ist und für das Argument gilt: `0 <= pos < values.length`.

```
class MagicSuperCompiler {
  private String[] values;

  public void setValueAt(int pos, String newValue) {
    
  }

  public String getValueAt(int pos) {
    
  }
}
```

13. Aufgabe: Objekt-Orientierung – Vererbung (Beispiel)

(Lösung auf Seite 15)

Geben Sie für jede markierte Zeile den Rückgabewert des Ausdrucks an. Falls die Zeile zu einem Kompilierfehler führt, füllen Sie die Lücke mit „Fehler“.

```
class Parent {
  public String getFoo() { return "parentFoo"; }
}
class Child extends Parent {
  public String getFoo() { return "childFoo"; }
  public String getBar() { return "childBar"; }
}
class GrandChild extends Child {
  public String getFoo() { return "grandChildFoo"; }
}

Parent p0 = new Child();
p0.getFoo();
p0.getBar();
Child c0 = new GrandChild();
c0.getFoo();
c0.getBar();
GrandChild gc0 = new GrandChild();
gc0.getFoo();
gc0.getBar();
```

14. Aufgabe: Objekt-Orientierung – Vererbung (Beispiel)

(Lösung auf Seite 15)

Markieren Sie im folgenden Code die Zeilen, in denen **unzulässige** Aufrufe oder Zuweisungen stehen, d.h. Aufrufe oder Zuweisungen, die zu Kompilierfehlern führen.

```
interface IfA { void doA(); }
interface IfB { void doB(); }

class Parent implements IfA {
  public void doA() {}
  public void doC() {}
}

Parent p0 = new Parent();
IfA ifa0 = new Parent();
  p0.doA();
  p0.doB();
  p0.doC();
  ifa0.doA();
  ifa0.doB();
  ifa0.doC();
```

15. Aufgabe: Objekt-Orientierung – Schnittstellen und Schnittstellenimplementierung (Beispiel)

(Lösung auf Seite 16)

Schreiben Sie Java-Code für eine Schnittstelle, die eine Methodensignatur Ihrer Wahl enthält und eine Klasse, die diese Schnittstelle implementiert. Sie müssen keine Funktionalität oder ähnliches beschreiben (d.h. Sie können Methodenrümpfe leer lassen).

16. Aufgabe: Objekt-Orientierung – Konstruktor (Beispiel)

(Lösung auf Seite 16)

Ergänzen Sie die folgende Klassendefinition mit einem Konstruktor, der ein Feld (*array*) `String[]` entgegennimmt und die Werte mit dem Operator `+` verbindet (konkateniert) und in das Attribut (*field*) `value` schreibt.

```
class MagicSuperCompiler {
  private String value;

  public MagicSuperCompiler(String value) {
    this.value = value;
  }

  public MagicSuperCompiler(String[] values) {
    
    for (int i = 0; i < values.length; i++) {
      
    }
    
  }
}
```

Lösungen

1. Lösung: Java-Basics (Beispiel)

(Aufgabe auf Seite 3)

Ergänzen Sie die folgende Klasse so, dass gilt: `theText` ist ein öffentliches Attribut (*field*), das eine Zeichenfolge enthält, `numbers` ist ein privates Attribut, das ein Feld (*array*) von Ganzzahlen enthält.

```
class ValueHolder {
  public String theText; // oder CharSequence
  private int[] numbers; // oder long[], Integer[], ...
}
```

2. Lösung: Java-Basics (Beispiel)

(Aufgabe auf Seite 3)

Ergänzen Sie die folgende Schnittstelle um eine Methode `doSomething`, die ein Feld (*array*) von Ganzzahlen entgegennimmt und keine Rückgabe hat.

```
interface Worker {
  void doSomething(int[] values);
  // kein Fehler, falls public, oder long[], Integer[], ...
}
```

3. Lösung: Kontrollfluss (Beispiel)

(Aufgabe auf Seite 3)

Ergänzen Sie den folgenden Code so, dass beim Aufruf von `doNTimes(int)` die Methode `doSomething()` `n` mal aufgerufen wird. Es gilt immer $n \geq 0$.

```
public static void doSomething() { /* ... */ }

public static void doNTimes(int n) {
  for (int i = 0; i < n; i++) {
    doSomething();
  }
}
```

4. Lösung: Kontrollfluss (Beispiel)

(Aufgabe auf Seite 4)

Ergänzen Sie den folgenden Code so, dass jedes zweite Element des Felds (*array*) `values` ausgegeben wird, beginnend mit dem zweiten Eintrag. Für das Feld `new int[] { 42, 5, 17, 7, 3, 1, 5, 10 }` wäre dies beispielsweise `5, 7, 1, 10`. Hinweis: Die Anzahl der Elemente eines Arrays `values` erhalten Sie mit `values.length`.

```
public static void printSkip(int[] values) {
  for (int i = 1; i < values.length; i += 2) {
    System.out.println(values[i]);
  }

  // oder

  for (int i = 0; i < values.length; i++) {
    if ((i % 2) == 1) {
      System.out.println(values[i]);
    }
  }
}
```

5. Lösung: Kontrollfluss (Beispiel)

(Aufgabe auf Seite 4)

Ergänzen Sie den folgenden Code so, dass nacheinander die Werte in `values` ausgegeben werden, die an den in `positions` definierten Stellen stehen. Sie können davon ausgehen, dass `positions` nur Werte zwischen einschließlich 0 und `values.length - 1` enthält. Hinweis: Die Anzahl der Elemente eines Arrays `arr` erhalten Sie mit `arr.length`.

```
public static void printValuesAt(String[] values, int[] positions) {
  for (int i = 0; i < positions.length; i++) {
    System.out.println(values[positions[i]]);
  }
}
```

6. Lösung: Kontrollfluss (Beispiel)

(Aufgabe auf Seite 4)

Ergänzen Sie den folgenden Code so, dass alle ungeraden Werte in `values` ausgegeben werden (mit `System.out.println`). Sie können davon ausgehen, dass `values` nicht `null` ist. Hinweis: Die Anzahl der Elemente eines Arrays `arr` erhalten Sie mit `arr.length`.

```
public static void printOdds(int values[]) {
  for (int i = 0; values.length; i++) {
    if ((values[i] % 2) == 1) {
      System.out.println(values[i]);
    }
  }
}
```

7. Lösung: Kontrollfluss (Beispiel)

(Aufgabe auf Seite 5)

Es sei die Methode `evaluate()` mit dem Rückgabetyp `boolean` gegeben. Ergänzen Sie den folgenden Code so, dass die Methode `evaluate()` so lange aufgerufen wird, bis sie zum ersten Mal `true` zurückgibt. Zählen Sie in der Variable `i`, wie oft in Folge `false` zurückgegeben wurde.

```
int i = 0;
while (!evaluate()) {
  i++;
}
System.out.println(i);
```

8. Lösung: Kontrollfluss (Beispiel)

(Aufgabe auf Seite 5)

Ergänzen Sie den folgenden Code der Methode `doSomething(int)` so, dass:

- Wenn `i` ungerade ist, erhöhe `i` um 1,
- wenn dies nicht galt und wenn `i` gerade und größer als oder gleich 3 ist, gebe den Text „Fall 2“ aus.

```
public static int doSomething(int i) {
  if ((i % 2) == 1) {
    i++;
  } else if (i >= 3) { // Hinweis: nur } if { ist hier nicht korrekt
    System.out.println("Fall 2");
  }

  return i;
}
```

9. Lösung: Objekt-Orientierung – Objekterzeugung (Beispiel)

(Aufgabe auf Seite 5)

Es sei eine Klasse `Date` mit den beiden Konstruktoren `Date(int dayOfYear)` und `Date(int day, int month)` gegeben. Ergänzen Sie den folgenden Code so, dass `date0` mit dem 42-sten Tag des Jahres (also `dayOfYear = 42`) belegt wird und `date1` mit dem Datum mit `day = 7` und `month = 10`.

```
Date date0 = new Date(42);
Date date1 = new Date(7, 10);
```

10. Lösung: Objekt-Orientierung – Attribute (Beispiel)

(Aufgabe auf Seite 6)

Ergänzen Sie die Klasse `SuperCollider` mit einem öffentlichen statischen Attribut (*field*) `myValue` das eine Ganzzahl (*integer*) enthalten kann und einem privaten Attribut `myText`, das einen String enthalten kann.

```
class SuperCollider {
  public static int myValue;
  private String myText;
}
```

11. Lösung: Objekt-Orientierung – Getter und Setter (Beispiel)

(Aufgabe auf Seite 6)

Ergänzen Sie die folgende Klassendefinition um einen Getter und einen Setter für das Attribut `value`. Setzen Sie im Setter nur neue Werte, die größer gleich dem Attribut (*field*) `min` und kleiner gleich dem Attribut `max` sind.

```
class MagicSuperCompiler {
  private int min;
  private int max;

  private int value;

  // Getter for value
  public int getValue() {
    return this.value; // oder: return value;
  }

  // Setter for value
  public void setValue(int newValue) {
    if ((min <= newValue) && (newValue <= max)) {
      // oder alternative korrekte Bedingungen
      this.value = newValue; // oder: value = newValue;
    }
  }
}
```

12. Lösung: Objekt-Orientierung – Getter und Setter (Beispiel)

(Aufgabe auf Seite 7)

Ergänzen Sie die folgende Klassendefinition um den Setter `setValueAt` und `getValueAt`, die jeweils auf das Attribut (*field*) `values` und den Wert an der Stelle `pos` zugreifen. Sie können davon ausgehen, dass `values` immer ungleich `null` ist und für das Argument gilt: `0 <= pos < values.length`.

```
class MagicSuperCompiler {
  private String[] values;

  public void setValueAt(int pos, String newValue) {
    this.values[pos] = newValue;
  }

  public String getValueAt(int pos) {
    return this.values[pos];
  }
}
```

13. Lösung: Objekt-Orientierung – Vererbung (Beispiel)

(Aufgabe auf Seite 8)

Geben Sie für jede markierte Zeile den Rückgabewert des Ausdrucks an. Falls die Zeile zu einem Kompilierfehler führt, füllen Sie die Lücke mit „Fehler“.

```
class Parent {
  public String getFoo() { return "parentFoo"; }
}
class Child extends Parent {
  public String getFoo() { return "childFoo"; }
  public String getBar() { return "childBar"; }
}
class GrandChild extends Child {
  public String getFoo() { return "grandChildFoo"; }
}

Parent p0 = new Child();
p0.getFoo(); childFoo
p0.getBar(); Fehler
Child c0 = new GrandChild();
c0.getFoo(); grandChildFoo
c0.getBar(); childBar
GrandChild gc0 = new GrandChild();
gc0.getFoo(); grandChildFoo
gc0.getBar(); childBar
```

14. Lösung: Objekt-Orientierung – Vererbung (Beispiel)

(Aufgabe auf Seite 8)

Markieren Sie im folgenden Code die Zeilen, in denen **unzulässige** Aufrufe oder Zuweisungen stehen, d.h. Aufrufe oder Zuweisungen, die zu Kompilierfehlern führen.

```
interface IfA { void doA(); }
interface IfB { void doB(); }

class Parent implements IfA {
  public void doA() {}
  public void doC() {}
}

Parent p0 = new Parent();
IfA ifa0 = new Parent();
   p0.doA();
✗  p0.doB();
   p0.doC();
   ifa0.doA();
✗  ifa0.doB();
✗  ifa0.doC();
```

15. Lösung: Objekt-Orientierung – Schnittstellen und Schnittstellenimplementierung (Beispiel)

(Aufgabe auf Seite 9)

Schreiben Sie Java-Code für eine Schnittstelle, die eine Methodensignatur Ihrer Wahl enthält und eine Klasse, die diese Schnittstelle implementiert. Sie müssen keine Funktionalität oder ähnliches beschreiben (d.h. Sie können Methodenrümpfe leer lassen).

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

public class HardWorker implements Worker {
  public void work() {
  }
}
```

16. Lösung: Objekt-Orientierung – Konstruktor (Beispiel)

(Aufgabe auf Seite 10)

Ergänzen Sie die folgende Klassendefinition mit einem Konstruktor, der ein Feld (*array*) `String[]` entgegennimmt und die Werte mit dem Operator `+` verbindet (konkateniert) und in das Attribut (*field*) `value` schreibt.

```java
class MagicSuperCompiler {
  private String value;

  public MagicSuperCompiler(String value) {
    this.value = value;
  }

  public MagicSuperCompiler(String[] values) {
    this.value = ""; // oder value = "" oder andere Variable String val2
    for (int i = 0; i < values.length; i++) {
      this.value += values[i];
      // oder value += values[i] oder val2 += values[i]
    }
    // falls andere Variable dann this.value = val2;
  }
}
```