

# Prüfung Grundlagen der Programmierung 2

## Prüfung Objektorientierte Programmierung

### Prüfung Praktische Informatik 2

### Prüfung Software-Entwicklung 2

Prüfer: Glavina, Grauschopf, Windisch  
Prüfungsdauer: 90 Minuten  
Hilfsmittel: keine

#### Aufgabe 1: Programmverständnis (etwa 15%)

a) Welche Ausgabe erzeugt das folgende Programm?

```
class Fahrzeug {
    Fahrzeug() {
        System.out.println("Fahrzeug erzeugt");
    }
    Fahrzeug(String name) {
        this();
        System.out.println("Fahrzeug mit Name erzeugt");
    }
    public String toString() {
        return "Fahrzeug";
    }
}

class Motorwagen extends Fahrzeug {
    Motorwagen() {
        this("Spider");
        System.out.println("Motorwagen erzeugt");
    }
    Motorwagen(String name) {
        System.out.println(this + " mit Name");
    }
}

class MonsterTruck extends Motorwagen {
    MonsterTruck(int sitze) {
        System.out.println("MonsterTruck");
    }
}
```

```

class Ala {
    public static void main(String[] args) {
        System.out.println("---");
        new Fahrzeug();
        System.out.println("---");
        System.out.println(new Fahrzeug());
        System.out.println("---");
        System.out.println(new Motorwagen());
        System.out.println("---");
        System.out.println(new MonsterTruck(2));
    }
}

```

b) Welche Ausgabe erzeugt das folgende Programm?

```

class A {
    private static int i;
    A(int zahl) {i = zahl;}
    int get() {return i;}
}

class Alb {
    public static void main(String[] args) {
        A a1 = new A(47);
        A a2 = new A(11);
        System.out.println(a1.get());
        System.out.println(a2.get());
        System.out.println(a1.get() - a2.get());
    }
}

```

## Aufgabe 2: Basisalgorithmen und Container-Klassen (ca. 20%)

- a) Gegeben sei eine Java-Klasse Student mit den beiden Attributen name (String) und matrikelnummer (int). Beide Attribute seien über entsprechende Zugriffsoperationen ("Getter") lesbar. Geben Sie eine Java-Methode istEingetragen an, die eine Student-Referenz und eine **unsortierte** Studentenliste (Typ List<Student>) übergeben bekommt und "true" liefert, wenn ein Student mit der Matrikelnummer des übergebenen Studenten in der Liste existiert. Andernfalls wird "false" zurückgegeben.

Hinweis: Lösen Sie die Aufgabe ohne die Methode contains() zu verwenden!

- b) Geben Sie eine Java-Prozedur eintragen an, die eine Student-Referenz und eine **sortierte** Studentenliste übergeben bekommt und das übergebene Studentenobjekt in die Liste einsortiert. Die Liste soll nach Matrikelnummern aufsteigend sortiert sein.

Hinweis: Verwenden Sie die Methode add(int index, E element) aus dem

Interface zu `List<E>`, welches das übergebene Objekt an die `index`-te Stelle der Liste einfügt.

- c) Geben Sie nun eine geringfügig ergänzte – jedoch performantere – Version der Methode aus Teilaufgabe 2.a) an, die die Sortiertheit der Liste ausnutzt.

### Aufgabe 3: Klassenmodellierung und Polymorphie (ca. 32%)

Ein so genanntes logisches Gatter verfügt über eine Reihe von booleschen Eingängen und berechnet daraus mit der Methode `output()` einen booleschen Wert, seinen Ausgang. Ein Gatter hat folgende Attribute:

- `input`: eine Reihung von Wahrheitswerten (Eingänge des Gatters)
- `bezeichnung`: eine Zeichenkette (Bezeichnung des Gatters)

Als konkrete Gatter betrachten wir nun die Gattertypen `Nand`, `Nor` und `Xor`:

- `Nand` liefert genau dann `false`, wenn alle `input`-Werte `true` sind.
- `Nor` liefert genau dann `false`, wenn mindestens ein Wert in `input` `true` ist.
- `Xor` liefert genau dann `true`, wenn genau ein einziger Eingangswert `true` ist.

- a) Zeichnen Sie ein Klassendiagramm (inklusive Attribut- und Methodennamen) mit einer abstrakten Klasse `Gatter` und den drei davon abgeleiteten Klassen `Nand`, `Nor` und `Xor`. Die Klassen haben neben den bereits beschriebenen Attributen und der Methode `output` folgende weitere Methoden:

- einen Konstruktor mit zwei Parametern, wobei der erste die Bezeichnung des Gatters und der zweite die Anzahl der Eingänge darstellt. Alle Eingänge sind initial mit `false` zu besetzen.
- einen Getter für die Bezeichnung
- eine Methode `getInput`, die die `Input`-Reihung in `String`-Darstellung zurückgibt (ganze Reihung als `String`, `true` als Ziffer 1, `false` als Ziffer 0)
- eine Methode `setIndexedBit(int index, boolean value)`. Diese setzt den indizierten `input`-Wahrheitswert auf `value`.

- b) Implementieren Sie die Klassen `Gatter` und `Xor`.

- c) Schreiben Sie eine Anwendungsklasse, in der Sie zunächst ein `Gatter`-Array der Länge 3 anlegen. Belegen Sie anschließend das Array mit einem zweistelligen `Nor`-Gatter, einem fünfstelligen `Nand`-Gatter und einem dreistelligen `Xor`-Gatter.

Die Anwendungsklasse soll außerdem eine Schleife enthalten, die für jedes Gatter den ersten Eingangswert auf `true` setzt und dann den Wert der Attribute und den Ausgangswert ausgibt.

#### Aufgabe 4: GUI-Programmierung (ca. 33%)

Wir programmieren Teile eines einfachen „Vier Gewinnt“-Spiels (s. Abbildung), bei dem zwei Spieler gegeneinander an einem Rechner antreten.

Wer die Regeln kennt, kann die nächsten beiden Absätze überlesen.

Die Spieler „gelb“ und „rot“ belegen abwechselnd Felder mit ihrer Farbe. Wer als erster vier Felder horizontal, diagonal oder vertikal mit seiner Farbe belegt hat, hat gewonnen.

Das Spielbrett ist vertikal angeordnet, in jeder Spalte soll (im realen Leben schwerkraftbedingt) nur das unterste unbelegte Feld auswählbar sein.



- Jedes einzelne Feld des Spielbretts kann eine der Farben blau (Anfangszustand), gelb (Spieler 1) oder rot (Spieler 2) haben. Definieren Sie eine Enumeration „State“, deren drei mögliche Werte `BLUE`, `YELLOW` und `RED` sind.
- Das Spielbrett ist eine zweidimensionale Anordnung von Einzelfeldern. Diese sind Objekte des Typs `VgButton`. Ein `VgButton` ist ein `JButton`. Er hat als Attribute
  - einen `State`,
  - eine Zeilennummer und
  - eine Spaltennummer.

Zeilen- und Spaltennummer werden vom einzigen vorhandenen Konstruktor mit per Parametern übergebenen Werten belegt. Den State setzt der Konstruktor auf den Anfangszustand. Dies macht er mit einem Aufruf der Setter-Methode `setState(State s)`. Diese Setter-Methode bewirkt auch eine Farbänderung des aufrufenden `VgButton`'s. Sie muss nicht definiert werden, und kann wie alle anderen Setter und Getter als existent vorausgesetzt werden.

Definieren Sie die Klasse `VgButton` mit Attributen und Konstruktor wie vorstehend beschrieben.

Im folgenden Rahmenprogramm der Klasse `VgGui`, die das Spielbrett implementiert, sind 2 Stellen markiert: `TODO c)` + `TODO d)`.

In den Teilaufgaben c) und d) wird beschrieben, welcher Code hier einzufügen wäre. Lesen Sie das Rahmenprogramm gründlich durch:

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

class VgGui extends JFrame implements ActionListener {

    // Dimensionen des Spielbretts
    private final static int ROWS = 6;
    private final static int COLS = 7;

    // Spieler, der dran ist - Gelb fängt an
    private State turn = State.YELLOW;

    // button[0][0] ist links oben,
    // button[ROWS-1][COLS-1] rechts unten
    private VgButton buttons[][] = new VgButton[ROWS][COLS];

    // Für die Siegmeldung
    private JLabel label = new JLabel();

    VgGui() {
        /*****
         * TODO c) *
         *****/
    }

    public void actionPerformed(ActionEvent e) {
        VgButton b = (VgButton) (e.getSource());
        /*****
         * TODO d) *
         *****/
    }

    // gibt true zurück, wenn der Spieler, der
    // dran ist, mit seinem letzten Zug gewonnen hat
    private boolean hasWon(int row, int col) {
        // ...
    }

    public static void main(String[] args) {
        new VgGui();
    }
}
```

c) Geben Sie hier den Konstruktor von `VgGui` an. Der Konstruktor soll eine GUI wie in

der Abbildung oben dargestellt erzeugen.

- Für die Anordnung der Buttons aus dem zweidimensionalen Array `buttons` ist ein Container mit passendem `LayoutManager` zu wählen.
- Das Array `buttons` muss noch fertig initialisiert werden. Zusätzlich soll dabei jedem `VgButton` aus dem Array `this` als `ActionListener` zugeordnet werden. Ferner soll der `VgButton` dem im ersten Punkt beschriebenen Container hinzugefügt werden.
- Sonstige nötige Methodenaufrufe, um das vorgeschriebene Aussehen zu verwirklichen.

- d) In `actionPerformed` kann davon ausgegangen werden, dass das `ActionEvent` immer von einem `VgButton` ausgelöst wurde.

In den folgenden beiden Fällen ist der Klick unwirksam und `actionPerformed` wird sofort verlassen, insbesondere wird der `State` des geklickten `VgButton's` nicht geändert:

- Wenn der `State` des geklickten Felds nicht `BLUE` war.
- Wenn das geklickte Feld keine Auflage hat, d.h. ein darunter liegendes Feld existiert, das blau ist.

Treffen diese beiden Fälle nicht zu, wird der `State` des geklickten `VgButton's` auf `turn` geändert.

Weiter wird dann mit der vorgegebenen, nicht zu implementierenden Methode `hasWon` überprüft, ob der Spieler mit dem letzten Zug gewonnen hat. Wenn ja, so soll `label` den Text „Gewonnen“ anzeigen.

Schließlich wird `turn` auf den Wert des anderen Spielers gewechselt, d.h. von gelb nach rot bzw. von rot nach gelb.