# Vertiefung Softwareentwicklung

## BEISPIEL KLAUSUR – mit Antwortbeispielen

## <DATUM>

**Name:**_____

**Student ID (Matrikelnummer):** _____

| | Aktuellen Semester | WS 2022/23 | UE SE2 im SS 2022 |
|---|---|---|---|
| **"UE gemacht im"** | | | |

**Hints:**

1. The exam duration is **90 minutes**.
2. Use permanent writing material (e.g., a pen; "Dokumentenechtes Schreibmaterial").
3. Write your **name and student ID** first.
4. This is a closed-book exam, **no additional material** is allowed.
5. **Read comments in the code parts carefully**.
6. Enter your code solution in the given fields on the sheet – only those will be considered. (Notes can be used on the back sides of the sheets).
7. In case you write comments, both English and German may be used.

**Points:**

| A 1 | A2 | A 3 | A 4 | A 5 | Overall |
|---|---|---|---|---|---|
| 20 | 20 | 15 | 20 | 25 | 100 |
| | | | | | |

## A 1 – Basic Theory on Java (20 Points)

For each statement, select true or false (2 points for each **correct answer** AND **explanation**):

| | **true** | **false** |
|---|---|---|
| Classes class B, class C are given. The declaration of class A is correct:<br>class A extends B, C { }<br>**Explain your answer:**<br>**In Java ist Multiple Inheritance nicht unterstützt.** | | **X** |
| Interfaces interface B, interface C are given. The declaration of class A is correct:<br>class A extends B, C { }.<br>**Explain your answer:**<br>**Um ein oder mehrere Interfaces zu implementieren wird "implements" verwendet, nicht "extends".** | | **X** |
| When a class member field is declared protected, only the class and sub-classes of this class can access it.<br>**Explain your answer:**<br>**Ein als protected deklariertes Feld ist auch in Klassen desselben Pakets zugreifbar.** | | **X** |
| *etc.* | | |

## A 2 – Applying Knowledge (20 points)

Select the correct statements (note that **incorrect answers will lead to deduction of points**):

**A) The following classes and interfaces are given:**

```
class Animal { }
interface Flying { }

class Tiger extends Animal { }
class Frog extends Animal { }
class Mockingbird extends Animal implements Flying {}
```

**Which of the following assignments are correct? (4 P)**

**X** Animal animal = new Tiger();

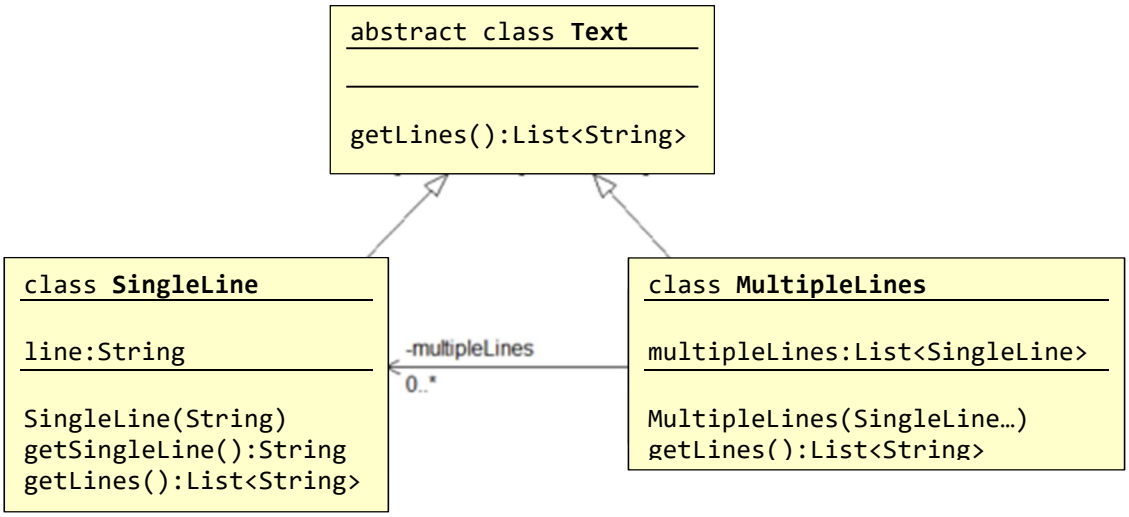
☐ Animal animal = new Flying();
☐ Tiger tiger = new Animal();

**X** Flying bird = new Mockingbird();

*etc.*

## A3 – Inheritance and Dynamic Binding (15 Points)

The following class system allows to compose a text. `Text` is the abstract base class of the derived classes `SingleLine` (text with just one line) and `MultipleLines` consisting of a list of SingleLine texts.




`Text` defines the abstract method `getLines()`, which returns a list of text Strings. **You should now implement the classes `SingleLine` and `MultipleLines` and all methods as defined in the UML class diagram.**

```
public abstract class Text {
  public abstract List<String> getLines();
}
```

**Implement the classes SingleLine and MultipleLines by writing the missing code parts below:**

**class SingleLine:** Implement a **constructor, a getter method returning the line,** and the method **getLines():**

```
public class SingleLine extends Text {
  private final String line;

  public SingleLine(String line) {
     this.line = line;
  }

  public String getSingleLine(){
    return line;
  }

  @Override
  public List<String> getLines() {
     return List.of(line);
  }
}
```

**class MultipleLines:** Implement a **constructor** and the method **getLines():**

```
public class MultipleLines extends Text {
  private final List<SingleLine> multipleLines;

  public MultipleLines(SingleLine...elems) {
     multipleLines = List.of(elems);
  }

  @Override
  public List<String> getLines() {
    return multipleLines.stream()
                        .map( l -> l.getSingleLine())
                        .collect (Collectors.toList());
  }
}
```

The code can be used to construct a message consisting of multiple single lines, as shown below.

```
puclic class Main {
   public static void main(String[] args) {
   Text prologOfDune = new MultipleLines(
      new SingleLine("A beginning is a very delicate time. "),
      new SingleLine("Know then, that it is the year 10191. "),
      new SingleLine("The known universe is ruled by the Padisha Emperor Shaddam IV,
my father. "),
      new SingleLine("In this time, the most precious substance in the Universe is
the spice melange."),
      new SingleLine("\n"),
      new SingleLine("[Princess Irulan]")
   );

   prologOfDune.getLines().stream()
                          .forEach(System.out::print);
}
```

*Output:*

A beginning is a very delicate time. Know then, that it is the year 10191. The known universe is ruled by the Padisha Emperor Shaddam IV, my father. In this time, the most precious substance in the Universe is the spice melange.

[Princess Irulan]

## A 4 - Functional Programming – HOF, Streams (20 Points)

### A - HOF (Higher Order Functions) – 10 Points:

The functional interfaces Predicate and Consumer are given as follows:

```
@FunctionalInterface
public interface Predicate<T> {
    boolean test(T t);
}
```

```
@FunctionalInterface
public interface Consumer<T> {
    void accept(T t);
}
```

In class HofDoForEach, **implement the HOF**

**void doForEach(Predicate<String> pred, Consumer<String> user)**

that uses the predicate (pred) to test each element of list movies and – if the condition is true – uses the consumer (user) method on this element.

```
public class HofDoForEach {
  List<String> movies = List.of("Blade Runner", "IRobot", "Terminator");

  public void doForEach(Predicate<String> pred, Consumer<String> user) {
    for (String m : movies) {
      if (pred.test(m)) user.accept(m);
    }
  }
}
```

Use the method doForEach() to printout movie titles with a title length larger than 7 characters:

```
doForEach(m->m.length() > 7, System.out::println);
```

*Output:*

Blade Runner

Terminator

**B – Streams - 10 Points**

The following list of words is given:

```
List<String> words =
List.of("This", "list", "contains", "the", "movies", "Titanic", "and",  "Avatar");
```

Implement the following operation by **USING THE STREAM API!**

**The following Stream operations may be used:**

- `Stream<T> filter(Predicate<? super T> predicate)`
- `<R> Stream<R> map(Function<? super T,? extends R> mapper)`
- `Stream<T> sorted()`
- `void forEach(Consumer<? super T> action)`
- `<R, A> R collect(Collector<? super T, A, R> collector) … in combination with Collector<T, ?, List<T>> toList() of class Collectors`

**Operation 1 (5 Points):**

Use stream operations to filter words that start with an uppercase letter, map them to lower case words and return a list containing these words.

Hints:

- The method `boolean Character.isUpperCase(char c)` returns true if the parameter is an upper case letter, else it returns false; the String method toLowerCase() can be used to change a String into lower case only characters.

```
List<String> toLowerCaseWords =
        words.stream()
            .filter(w -> Character.isUpperCase(w.charAt(0)))
            .map(w->w.toLowerCase())
            .collect(Collectors.toList());
```

**Operation 2 (3 Points):**

Use stream operations to order the words alphabetically and to print them out.

```
words.stream()
     .sorted()
     .forEach(System.out::println);
```

**Operation 3 (2 Points):**

What is the output of the following stream statement?

```
System.out.println(  words.stream()
                        .filter(w->w.length() > 4)
                        .flatMap(w->Stream.of(w.length()))
                        .findFirst()
                        .isPresent());
```

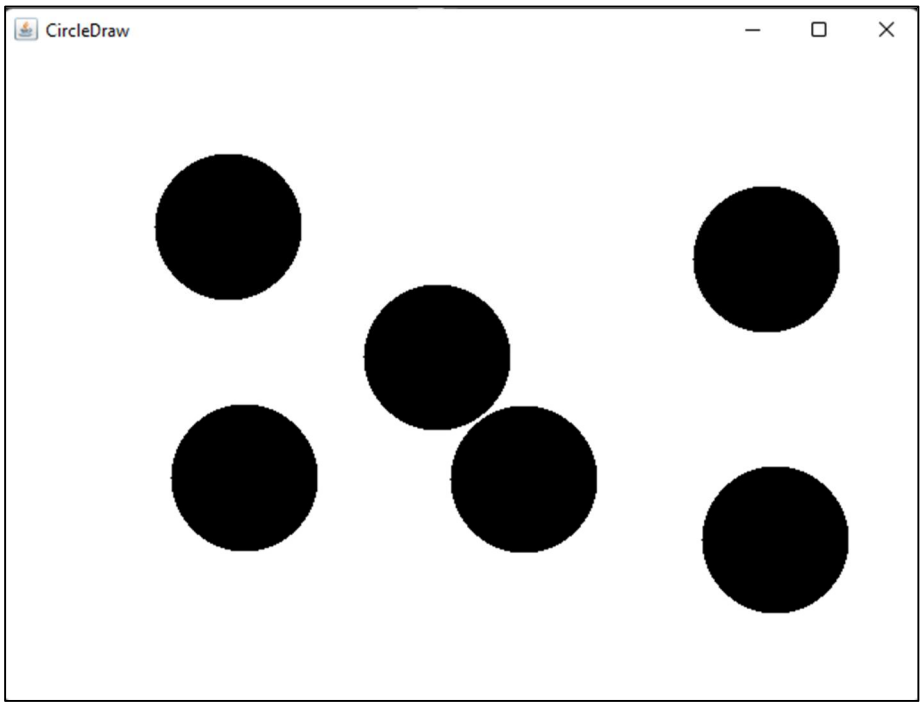
```
true
```

## A 5: MVC – 25 Points

We want to draw circles on a 2D area. A list of all circles should be managed by the model. The user may add circles to the model by clicking on a free space in the area or remove circles when clicking at the position of a circle. The view displays the circles in the area. **Use the MVC architecture to implement this application:**

- **Model**
  - `class Circle`: contains the x/y coordinates of the center of the circle and the radius and a method `contains(int x1, int y1)` that checks whether the coordinates x1/y1 are within the area of the circle
  - `class CircleModel` contains a list of Circles
  - `class CircleEvent` is the model change event of kind DELETED (when a circle is removed from the model) or ADDED (when a circle is added to the model)
  - `interface CircleModelListener` is the model listener
- **View (incl. Controller)**
  - `class CircleVew` is the view of the application – upon mouse click, new circles should be added or removed.

The application looks as follows:



Class Circle is given as follows:

```
public class Circle {

  private final int x; private final int y; private final int radius;

   public int getX() { return x;    }

   public int getY() { return y;    }

   public int getRadius() { return radius;}

   public Circle(int x, int y, int radius) {
         this.x = x; this.y = y; this.radius = radius;
   }

   // returns true if x/y values are within the are of the circla
   public boolean contains(int x, int y) {
         double xDiff = Math.abs(this.x - x);
         double yDiff = Math.abs(this.y - y);
         return Math.sqrt(xDiff * xDiff + yDiff * yDiff) <= radius;
   }

   @Override
   public String toString() {
      return "Circle [x=" + x + ", y=" + y + ", radius=" + radius + "]";
   }
}
```

Class CicleEvent is given as follows:

```
public class CircleEvent extends EventObject{
   private static final long serialVersionUID = 1L;

   public enum Kind { ADDED, DELETED };
   private final Circle circle;
   private final Kind kind;

   public CircleEvent(CircleModel source, Circle circle, Kind kind) {
         super(source);
         this.circle = circle;
         this.kind = kind;
   }

   public Kind getKind() { return kind; }

   public Circle getCircle() { return circle;}

}
```

The interface CircleModelListner is given as follows:

```
public interface CircleModelListener extends EventListener{
     public void circleEvent(CircleEvent event);
}
```

**Extend the class CircleModel with model listeners, adding and removing model listeners, and firing an event. The following is given:**

```
public class CircleModel {
   private List<Circle> circles = new ArrayList<>(); // List of circles

   // List of model listeners
   private List<CircleModelListener> listeners = new ArrayList<>();

   // Add model listeners
   public void addCircleModelListener(CircleModelListener listener) {
      listeners.add(listener);
   }

   // Remove model listener
   public void removeCircleModelListener(CircleModelListener listener) {
      listeners.remove(listener);
   }

   public List<Circle> getCircles(){
      return circles;
   }
```

**// Now add: a method to add a circle, a method to remove a circle, and a "fire method" to inform model listeners about the change (adding or removing of circles):**

```
   // Implement a method to add a new circle to the list of circles

   public void add(Circle circle) {
      final boolean added = circles.add(circle);
      if (added) {
         fireModelEvent(CircleEvent.Kind.ADDED, circle);
      }
   }


   // Implement a method to remove a circle from the list of circles
    public void remove(Circle circle) {
      final boolean removed = circles.remove(circle);
      if (removed) {
          fireModelEvent(CircleEvent.Kind.DELETED, circle);
      }
   }


   // Implement the fire method
   // Parameters should be the CircleEvent.Kind and the circle

   private void fireModelEvent(CircleEvent.Kind kind, Circle circle) {

      CircleEvent event = new CircleEvent(this, circle, kind);
      for (CircleModelListener listener : listeners) {
         listener.circleEvent(event);
      }
   }

}
```

**The view CircleView handles all mouse events to control adding and deleting of circles – add a MouseListener to the View and handle mouse release events (note the in-code comments!).:**

```
  public class CircleView extends JComponent {
  …

  private Color circleColor;
  private CircleModel model;

  public CircleView(CircleModel model, Color circleColor) {
        this.model = model;
        this.circleColor = circleColor;

       // Add a MouseListener to the current object that reacts to the mouse event
       // mouseReleased (callback method: void mouseReleased(MouseEvent e); From
       // the MouseEvent e, you get x/y mouse position with e.getX() and e.getY()
       // Differentiate between:
       //         Mouse position on a circle: remove the circle at position
       //                                else: add a circle at position, radius 50

      addMouseListener(new MouseListener() {

        @Override
        public void mouseReleased(MouseEvent e) {
          // REMOVE
          for (Circle circle : model.getCircles()) {
            if (circle.contains(e.getX(), e.getY())) {
              model.remove(circle);
              return;
           }}

          // ADD a new circle around the current mouse position
          model.add(new Circle(e.getX(), e.getY(), 50));
        } });

        // ADD a listener to the model that calls repaint()
       model.addCircleModelListener(new CircleModelListener() {

         @Override
         public void circleEvent(CircleEvent event) {
           repaint(); }
       });

      // alternatively also correct:
      //  model.addCircleModelListener (event->repaint());
    }
```

```
  // Method called when paint(), repaint() is called for the circle view
    @Override
    protected void paintComponent(Graphics g) {
      super.paintComponent(g); Graphics g2d = g.create();g2d.setColor(circleColor);
      // Clear all content in this circle view
      g2d.clearRect(0, 0, getWidth(), getHeight());
      // Draw each circle that is contained in the model / list of circles
      for (Circle circle : model.getCircles()) {
       int diameter = circle.radius * 2;
       g2d.fillOval(circle.x - circle.radius, circle.y - circle.radius, diameter,
diameter);}}}}
```