TECHNISCHE UNIVERSITEIT EINDHOVEN

Examination 2IP91 and 2IBP90
8 November 2016, 9:00–12:00/12:30

This exam consists of 3 questions on 4 pages.

- Put your name, number, and the date of today at the top of *every Java file* you submit.
- Add comments to your code only where clarification is needed.
- Don't make the lines in your code longer than 80 characters. Longer lines will mess up the layout of the printed program and make it harder to read for the graders.
- Before you submit your solutions, check that you have included all the files you want to submit and that you have saved them.
- Submit your solutions as compilable `.java` files in the folder `2IP91submission` on your desktop (non-Windows users choose a name themselves).
- Do not use named packages for your code. Use only letters, digits, and underscores in your file names.
- **When you leave the exam room, report to a supervisor to verify that your work has been submitted. The supervisor for the Auditorium rooms will be outside the room Aud 9. The supervisor for the Study Hub sections will be in the Study Hub.**
- You are allowed to consult on your laptop the course material (lecture slides, reader, programs you have made during the course) and the Java API. You are allowed to bring a printed copy of the reader to the exam.
- Use of the internet or other means of communication is *not* allowed during the examination.

---

1 Miscellaneous *(24 pt)*

Submit your answers to these questions in the file Miscellaneous.java. First copy the file to the submission folder, then start adding your answers to the copy.

Insert your questions Submit a compilable Java file.

(4) 1. True or false? A variable of type Garden (see question 1.5) can only store instances of the class Garden.

(4) 2. True or false? An instance variable in a class can not be changed by a method in that class.

(4) 3. Consider the following method.

```
void fragment() {
    int x = 0;
    try {
        x += 1;
        // here a NullPointerException is thrown
        x += 2;
    } catch (NullPointerException e) {
```

```
            x += 4;
        }
        System.out.println(x);
    }
```

Suppose this method is called and the commented part will produce a `NullPointerException`. What will be printed?

a) 0 b) 1 c) 2 d) 3
e) 4 f) 5 g) 6 h) 7 i) nothing

(8) 4. The intention of the following method is to sort the array `a` by swapping two neighboring elements when they are positioned in the wrong order.

There are two mistakes, however (not counting possible inefficiencies). Indicate in comment what the mistakes are and repair them, maintaining the algorithm. You have to change only a few lines.

```
void sort(int[] a) {
    for (int i = 0; i < a.length; i++) {
        for (int j = 0; j < a.length; j++) {
            if (a[j] > a[j+1]) {
                a[j] = a[j+1];
                a[j+1] = a[j];
            }
        }
    }
}
```

(4) 5. What is the number of objects (instances) of class Garden created by the following method?

```
void create() {
    Garden aap;
    Garden noot;
    Garden[] street;
    aap = new Garden();
    noot = aap;
    street = new Garden[10];
}

class Garden {
    int length;
    int width;

    int getArea() {
        return length * width;
    }
}
```

2 Getting digital *(40 pt)*

Submit your answers to these questions in the file `Digital.java` . First copy the file to the submission folder, then start adding your answers to the copy.

Note:

- For a positive int `n`, the expression `n % 10` gives the last digit of `n`.
- For an int `n`> 9, the expression `n / 10` gives `n` without the last digit.

(16) 1. Write a *recursive* method `int digitSum(int n)` that calculates the sum of the digits (in decimal notation) of the number `n` for `n`≥ 0.

   Do not compute an array with the digits of `n`. Do not use instance variables. Do not use loops. A non-recursive method will earn you at most half the points.

   Examples:
   digitSum(7) returns 7
   digitSum(193) returns 13.

(10) 2. Write a method `int digitalRoot(int n)` that repeatedly applies the sum of the (decimal) digits of `n` until the result is one digit.

   Use the method `digitSum`.

   Example:
   digitalRoot(193) returns 4.

(14) 3. Write a method `String reverse(int n)` that reverses the (decimal) digits of `n` and returns them as a String, for positive `n`. Do not compute an array with the (decimal) digits of `n`. Do not use instance variables. Example:
   `reverse(12)` returns `"21"`
   `reverse(2)` returns `"2"`
   `reverse(100)` returns `"001"`.

3 Soccer *(36 pt + 10 bonus)*

Submit your solution in the file `Soccer.java`. First copy the file to the submission folder, then start adding your answers to the copy. The file `Soccer.java` contains:

- a class Soccer that is a subclass of JPanel where balls and other shapes can be drawn and interacted with;
- an abstract class Ball that represents shapes on the screen that react when kicked (i.e., when the button Kick is pressed);
- a class BlueBall, subclass of Ball, that is represented by blue circles that move to the right when kicked.

The class contains a method `buildIt` that creates the GUI components necessary for the Soccer application and a method `createBalls` that creates the Ball objects. Initially, it creates one BlueBall in the middle of the screen.

There is a button Kick that, when pressed, should cause every ball on the screen to be kicked.

(9) 1. Add another BlueBall object in the bottom left of the window. Make sure that it reacts to kicking, just as the first BlueBall.

(9) 2. Add a class PinkBall, subclass of Ball, where a ball is represented by a solid pink rounded rectangle that moves diagonally to the right and down when kicked. See the excerpt from the API on the class `Graphics` below on how to draw rounded rectangles.

   Create a PinkBall object somewhere near the top left corner of the window.

(9) 3. Add a class DottedBall, *subclass of BlueBall*, where a ball is represented by a solid blue circle with a black dot in the center that, as a BlueBall, moves to the right when kicked and furthermore increases a little bit in size each time it is kicked. Do not unnecessarily copy code from BlueBall. You may want to use `super`.

Create a DottedBall in the middle of the top half of the window.

(9) 4. Add a class ChameleonBall, subclass of Ball. An object of this class does not move, but changes to a random color each time a kick occurs.

Create a ChameleonBall somewhere near the bottom left corner of the screen.

(10B) 5. If you answer this bonus question, make a copy of your file `Soccer.java`, with the answers to questions 3.1–3.4, and name it `SoccerB.java`. Close the file `Soccer.java`. Make your changes for this question to the file `SoccerB.java`. Change Soccer to SoccerB in the class name and in the method `main`.

Create an interface `Undoable` that represents the capability to undo the effect of a kick. It contains one method, `public void kickBack()`.

Have BlueBall and PinkBall implement this interface. Implement `kickBack()` in such a way that it does the opposite of what a kick does in such objects. E.g., a call to `kickBack` on a BlueBall will make the circle move to the left, the same distance as a call to `kick` would move it to the right.

Add an instance variable `ArrayList<Undoable> undoables` to the class Soccer. Add all objects that implement the interface Undoable to the ArrayList `undoables`.

Add a button with name "Kick Back" to the bottom of the JFrame. Clicking this button should have all objects that are referenced in the ArrayList `undoables` to get a call to `kickBack`. Note that you have to adapt the method `actionPerformed` in Soccer.

Excerpt from the API description of the class `java.awt.Graphics`:

```
public abstract void fillRoundRect(int x,
                                   int y,
                                   int width,
                                   int height,
                                   int arcWidth,
                                   int arcHeight)
```

Fills the specified rounded corner rectangle with the current color. The left and right edges of the rectangle are at x and x + width - 1, respectively. The top and bottom edges of the rectangle are at y and y + height - 1.

Parameters:

x - the *x* coordinate of the rectangle to be filled.

y - the *y* coordinate of the rectangle to be filled.

width - the width of the rectangle to be filled.

height - the height of the rectangle to be filled.

arcWidth - the horizontal diameter of the arc at the four corners.

arcHeight - the vertical diameter of the arc at the four corners.

Good luck!

Grading: The total number of points achieved divided by 10 is the grade g for this examination, maximized to 10. The final grade is the result of the following formula rounded to the nearest integer number: $0.6 \cdot g + 0.4 \cdot h$. Here g is the grade for this exam and h is the grade for the homework assignments. The grade g has to be at least 5.0 to pass.