EINDHOVEN UNIVERSITY OF TECHNOLOGY
Department of Mathematics & Computer Science

Final Exam 2IP90 and 2IBP90 (Programming), Tuesday, 30 October 2018, 9:00–12:00 (12:30)

This exam consists of 4 questions on 5 pages.

- Put your name, number, and the date of today at the top of *every 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 and text files in the provided folder `2IP90-submission` on your desktop (Windows users). Non-Windows users: make a folder named `2IP90-YourName` (and fill in your own name).
- Do not use named packages for your code.
- **When you leave the exam, report to a supervisor to verify that your work has been submitted.**
- 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.

Grading: The grade g for this examination is the total number of points achieved plus 2 divided by 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.

1 Miscellaneous (20 pt)

Submit your answers to these questions in the enclosed file answersMiscellaneous.txt.

1. Mention all the local variables (not including parameters) in the following program. (4)

```
import java.util.*;

class Columnism {
    Scanner scanner = new Scanner(System.in);

    void run() {
        int height; // no of rows in sheet, including row of sums
        int width; // no of columns in sheet
```

```
 9
10          // read dimensions
11          height = scanner.nextInt();
12          width = scanner.nextInt();
13          // read values
14          int[][] sheet = new int[height][width];
15          for (int r = 0; r < height; r++) {
16              for (int c = 0; c < width; c++) {
17                  sheet[r][c] = scanner.nextInt();
18              }
19          }
20      }
21  }
```

.

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

```
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;
    }
}

```

3. Consider the following recursive method that is intended to reverse a String. There are two errors. Give the line numbers of the lines with the errors, explain them, and propose repairments. (8)

```
1   String reverse(String s) {
2       if (s=="") {
3           return "";
4       } else {
5           return s.substring(s.length−1, s.length)
6                   + reverse(s);
7       }
8   }
9
```

4. Consider the following program fragment. When this is executed, how many calls to the method `rhubarb` will take place? (4)

```
        int[][] matrix = new int[5][5];
        for (int i = 0; i < matrix.length; i++) {
            for (int j = i; j < matrix.length; j++) {
                rhubarb();
            }
        }

```

Submit your answers in the provided file Miscellaneous.txt.

2 Making a difference (26 pt)

Extend the provided class Difference with the methods described below. In all these methods, you may assume that the parameter `nums` is not null.

Difference already contains a demo method for your, and the graders, convenience. Leave this method as it is, possibly commenting out functions that you have not implemented. You may add other demo or test methods yourself. Submit the file Difference.java.

1. Write a method `boolean allZero(int[] nums)` that returns `true` if all elements of `nums` are zero; if `nums` has no elements, it should return `false`. (4)
2. Write a method `int[] difference(int[] nums)` that returns the *difference array* of `nums`. This is an array that is one element shorter than `nums` and contains the differences between successive elements of `nums`. You may assume that `nums` contains at least two elements. For example, if `numbers` is the array {`1, 4, 9, 16`}, then `difference(numbers)` should return the array {`3, 5, 7`}. (4)
3. *Using the methods above*, write a method `boolean isConstant(int[] nums)` that returns `true` if the argument is a constant array, i.e., if all elements in `nums` have the same value. Both an empty array and an array with one element are considered constant. (4)
4. *Using the methods above*, write a method `boolean isLinear(int[] nums)` that returns `true` if the elements are a linear function of the indices, such as {`0, 2, 4, 6, 8`} and {`10, 11, 12`}. Note that an array is linear if the difference array is constant. (4)
5. Write a method `void printFunction(int[] nums)` that checks if the argument is a linear array (not constant) and then prints the function of x that defines the elements as a function of the indices. For example, `printFunction(new int[]{1, 4, 7, 10})` should print `3x + 1`. When the argument is not a linear array, it should print an appropriate message. (4)
6. The *degree* of an integer array is the number of times one can apply the difference operator until the array is constant. Hence, considering the definitions above, a constant array has degree 0 and a linear array has degree 1. A quadratic array, where the elements are a quadratic function of the indices, has degree 2, etc. Write a *recursive* method `int degree(int[] nums)` that returns the degree of `nums` as defined above. You may assume that `nums` has at least one element. Do not use instance variables. (6)

3 Eggs (32 pt)

Consider the provided file `EggBasket.java`. When run, it will show a window with a button *Move* and an egg. (Actually, the shape is an ellipse, which is not the same as an egg shape, but we should make things not too complicated.)

1. Add another Egg object to the scene. It should not overlap with the other egg. (6)
2. Add a method `void move()` to the class `Egg` that changes the position of the egg on the screen `moveDistance` pixels to the right. If the egg is near the edge of the window, it may move out of sight. This is intended. (5)
3. Have clicking the button *Move* call the method `move` on all Egg objects on the screen. (8)

   Hint: implement the ActionListener on the EggPanel, not on the EggBasket.
4. Add a class `EasterEgg` that is a subclass of Egg. It has the same shape as an Egg, but is decorated with a design of your liking. (8)

   Add a constructor that initializes the position of the egg.

   Place an EasterEgg in the window, at least 100 pixels from the left edge.
5. Adapt the class EasterEgg such that moving it will look a bit like rolling by turning the egg 90 degrees at each move step. Turning can be implemented without the use of Math.cos or Math.sin. (5)

4 Folder (20 pt)

The code to calculate the sum of an array is very similar to the code to calculate the product of an array. We want to exploit that similarity and write a single function that can perform all these calculations, depending on the operator (addition, multiplication, etc.) that is passed as an argument.

Given is the interface `BinaryOperation` that represents a oparation such as addition that takes two int numbers and produces an int number. The method `apply` performs the operation. The method `neutralElement` provides the neutral element of the operation, i.e., the number n with the property that $apply(n, a) = a$ for any a. For addition, this is 0.

```
interface BinaryOperator {
    public int apply(int a, int b);

    public int neutralElement();
}

```

This interface is in the provided file `Folder.java`. You should add your code to this file and submit it. Keep the class `Folder` the first in the file. This way, running it in DrJava will execute the main method of this class.

1. An outline of the `Plus` class is given in the Folder.java file. The method bodies of `apply(int a, int b)` and `neutralElement()` contain return statements that should be replaced by actual implementations (these returns are only there to make the class compile). Implement both methods such that they represent the binary operation *addition*. (4)
2. Add a method `int fold(int[] a, BinaryOperation op)` to the Folders class that “folds” the array with the operation `op`. Starting with the neutral element, it applies the operation repeatedly to the elements of the array. For example, when the array `a` is {`1, 2, 3, 4`}, `fold(a, new Plus())` should return 10. Add code to the `demo` method of the class `Folder` to demonstrate this method. Demonstrate it by printing out the result of `fold(a, new Plus())`. (8)

3. Add a class `Times` that represents the binary operation of multiplication. For example, `fold(a, new Times())` should return 24. Like with the previous step, add code to the demo method that prints out the result of `fold(a, new Times())`. (4)

4. Add a class `Max` that represents the binary operation of taking the greatest of two numbers. For example, `fold(a, new Max())` should return 4. Add code to the `demo` method to print out the result of `fold(a, new Max())`. (4)

   Hint: `Integer.MAX_VALUE` and `Integer.MIN_VALUE` give the largest respectively smallest possible value of the type `int`.

Good luck!