TECHNISCHE UNIVERSITEIT EINDHOVEN

Examination 2IP90 and 2IBP90
7 November 2017, 9:00–12:00/12:30

This exam consists of 4 questions on 6 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 room, 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.

---

1 Miscellaneous *(28 pt)*

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

(3) 1. True or false? There can be only one method named *foo* in a class.

(3) 2. True or false? There can be only one local variable named *x* in a class.

(3) 3. Consider the following code fragment (located in a method body).

```
A a;
B b;
a = b;
```

The first two lines are accepted by the compiler (so `A` and `B` are defined and visible). What should hold about `A` and `B` to have the compiler accept the assignment?

(3) 4. Describe in words what the following method returns. Look carefully!

```
 1 boolean m(int[] numbers) {
 2     boolean result = false;
 3     for (int i = 0; i < numbers.length - 1; i++) {
 4         if (numbers[i] * numbers[i] == numbers[i+1]) {
 5             result = true;
 6         } else {
 7             result = false;
 8         }
 9     }
10     return result;
11 }
```

5. What is wrong with the following code? There is one line where something goes wrong. Give the line number and tell what is wrong.

   (3) It is possible to repair this by changing one line. Give line number and what one has to change. (That there is no main method is not considered wrong.)

```
 1 class Square {
 2     double size;
 3 
 4     void setSize( double s ) {
 5         size = s;
 6     }
 7 
 8     double getArea() {
 9         return size * size;
10     }
11 }
12 
13 class SquareDemo {
14     Square mySquare;
15 
16     void demo() {
17         mySquare.setSize( 5 );
18         System.out.println( mySquare.getArea() );
19     }
20 }
```

(3) 6. Consider the following program.

```
 1 class Course {
 2     String title;
 3     int[] grades;
 4 
 5     void printGrades( int threshold ) {
 6         int grade;
 7         for (int i=0; i<grades.length; i++) {
 8             grade = grades[i];
 9             if ( grade >= threshold ) {
10                 System.out.print( grade + "  " );
```

```
                }
                System.out.println();
            }
        }
}
```

Mention all the local variables (not including parameters) in this program.

(10) 7. What follows is a *recursive* method `int power(int a, int b` that computes a^b in a very efficient way. It is incomplete and contains some holes. The holes are marked with A, B, C, and D. Complete it by mentioning for each of the holes what should be filled in there. In hole A you should fill in the minimal condition on the parameters under which the method works correctly.

```
// assume /* A */
int power( int a, int b ) {
    if ( b == 0 ) {
        return /* B */;
    } else if ( b % 2 == 0 ) {
        return power( a * a, /* C */ );
    } else { // b % 2 == 1
        return /* D */;
    }
}
```

2 Kaprekar *(30 pt)*

Submit your answers to this question in the file Kaprekar.java .

Kaprekar's routine (after the Indian mathematician D.R. Kaprekar (1905 – 1986)) is as follows.

1. Take any integer number ≥ 0 with at most four digits.
2. Add leading zeros, if necessary, to make it into a four digit representation of the number. Arrange the digits in ascending and in descending order to get two numbers (i.e., make the largest and the smallest number that can be composed from the digits).
3. Subtract the smallest number from the largest number.
4. If the new number is not equal to the previous one go back to step 2, else stop.

For example:

Starting with 1211, the following numbers are produced.

```
2111 - 1112 = 999
9990 - 999 = 8991
9981 - 1899 = 8082
8820 - 288 = 8532
8532 - 2358 = 6174
7641 - 1467 = 6174
```

It has been proven that for all integers ≥ 0 with at most four digits the routine will after at most 7 iterations reach either 0 (in case the starting number consists of four equal digits) or 6174, Kaprekar's constant.

Write a class `Kaprekar` that, for any number with at most four digits supplied as console input, outputs the results of Kaprekar's routine, as in the example above.

Adhere to the following requirements.

1. (10 pt) It is useful to have the number available as integer as well as in the form of separate digits stored in an array. Write a helper method `int[] int2array(int n)` that converts the integer representation to array representation with leading zeros added if necessary. Hint: `n%10` provides the last digit of `n`, `n/10` provides `n` with the last digit removed. Write a helper method `int array2int(int[] a)` that converts the array representation to integer representation.
2. (10 pt) Using these helper methods, write a method `int nextKaprekar(int n)` that for an integer `n` returns the next number according to the routine *and* prints the line with the computation as in the example.

   Hint:
   The method `void Arrays.sort( int[] a )` in `java.util` sorts the array `a` in ascending order.
3. (10 pt) Using `nextKaprekar`, write a method `void doKaprekar()` that inputs a number from the console and produces the output according to Kaprekar's routine.

3 Music *(22 pt)*

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

We are going to develop some classes to keep track of music tracks, songs, etc.

Consider the class `Music` as provided in the file Music.java

```
class Music {
    void demo() {
        demo1();
        System.out.println("--------------------");
        demo2();
    }

    void demo1() {
        Piece piece;
        // TODO
        System.out.println( piece );
    }

    void demo2() {
        // TODO
    }

    public static void main(String[] a) {
        (new Music()).demo();
    }
}
```

(3) 1. Add a class `Piece` to the file Music.java that represents a piece of music. It has instance variables `title` and `artist` (both Strings).

(2) 2. Add a constructor `Piece()` that sets the title to "Zonder titel" and the artist to "NN".

(3) 3. Add a constructor `Piece(String title, String artist,)` that sets the corresponding instance variables. Use this constructor to complete the method `demo1`.

(3) 4. Add a method `public String toString()` that produces a text representation in the following form:

```
The Fifth -- London Symphony Orchestra
```

Notes:

- Remember, every class inherits directly or indirectly from the class `Object`, where the method `public String toString()` is defined.
- The newline character can be denoted by `\n` in Strings. So, when you would print the String `"One\nTwo\nThree"` you would see

  ```
  One
  Two
  Three
  ```

(4) 5. Add a class `Song` to the file Music.java that is a subclass of `Piece` and has an instance variable `int duration` (duration of the song in seconds).

Add a constructor to set title, artist, and duration.

(3) 6. Override the method `toString` and have it return a text representation of a Song object in the following form:

```
One More Cup Of Java -- Bob Dylan
duration 3:47
```

Note that the duration is shown in minutes and seconds.

Don't copy code from the `toString` method in `Piece` to this method.

(4) 7. Add an `ArrayList` instance variable `pieces` to store `Piece` and `Song` objects. Add a method `void printAll()` to `Music` that prints the text representations of the objects in `pieces`.

Write a method `void demo2()` to the class `Music` that fills `pieces` with:

(a) the Piece shown in question 4,

(b) the Song shown in question 6, and

(c) a Song of your choice.

4 Colors *(20 pt)*

For this question, submit the file `Colors.java`, with your adaptations.

Consider the file Colors.java. It defines the classes `Colors` and `FlyingLabel` that show a grid of colored rectangles (labels) in shades of pink.

Adapt the class `FlyingLabel` in such a way that

(10) 1. A rectangle turns yellow (`Color.YELLOW` in `java.awt`) when the user clicks on it.

(10) 2. A rectangle turns to its original color when clicked again.

Some information from the API

The interface `Mouselistener` contains the following methods.

```
public void mousePressed( MouseEvent e)
public void mouseReleased( MouseEvent e)
public void mouseClicked( MouseEvent e)
public void mouseEntered( MouseEvent e)
public void mouseExited( MouseEvent e)
```

The class `Component` contains a method `public Color getBackground()` that returns the current color of the background of the Component it is called on. Its counterpart `setBackground` is used in the template code to set the background color of a Component.

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.