TECHNISCHE UNIVERSITEIT EINDHOVEN

# Examination Programming (2IP91)
# Tuesday 5 November 2013, 9:00–12:00 uur

This exam consists of 3 questions and an appendix on 4 pages.

You are allowed to consult on your laptop the course material, including programs you have made during the course. Use of the internet is *not* allowed during the exam (except for downloading the exercises at the beginning and uploading your solutions at the end of the exam).

---

*Grading:* The total number of points achieved is the grade $g$ for this examination. 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 average of the 6 highest grades for the homework assignments (an assignment that was not submitted is graded with a 0). Furthermore, this grade $g$ has to be at least 5.0 to pass.

---

*Submit your solutions as* `.java` *files in the folder 2IP91 as described in the instruction leaflet. Do not put your code in a package (i.e., use the default package).*

## 1 Stars *(30 pt)*

(15) 1. Write a method to print a rectangle on standard output (System.out) of given width, height, and type (filled or non-filled). (See examples).

```
* * * * * *   width=6, height=3, non-filled
*           *
* * * * * * *

* * * * * *   width=6, height=3, filled
* * * * * *
* * * * * *
```

(15) 2. Write a method `void barChart( int[] values )` that prints a horizontal bar chart of `numbers`, where the bars are sequences of stars and their lengths are equal to the values in the array `values`. For example, `barChart( new int[] 2, 5, 0, 4, 3 )` will result in the following output.

```
**
*****

****
***
```

## 2 Differentiation *(35 pt)*

Make a class Differentiation and add the following methods to it. In all these methods, you may assume that the parameter `nums` is not `null`.

(5) 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 `true` as well.

(6) 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, `difference( new int[]{1, 4, 9, 16} )` should return the array {`3, 5, 7`}.

(6) 3. Using the methods above, write a method `isConstant( int[] nums )` that returns `true` if all elements in `nums` have the same value. You may assume that `nums` has at least one element.

(6) 4. We call an integer array

   - `constant` if all elements have the same value;
   - `linear` if the difference array is constant;
   - `quadratic` if the difference array is linear;
   - `other` if none of the above apply.

   For example, {`1, 3, 5, 7` } is linear, {`-4, -1, 0, -1, -4` } is quadratic, {`1, 2, 4, 8, 16, 32, 64` } is other.

   Write a method `String getType( int[] nums )` that returns the type of the array as defined above as a String. You may assume that `nums` has at least one element.

(12) 5. 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, a linear array has degree 1, 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.

## 3 Monster Mania *(35 pt)*

See the enclosed program. It creates a window with a "monster" in it (a yellow rectangle with a mouth that opens and closes).

(7) 1. Change the method `void addMonsters()` to add two other monsters on different positions.

(8) 2. Add a subclass `SeeingMonster` of `Monster` that has two eyes (colored circles). Include a constructor that sets the eye color and the position of the monster. Change `addMonsters` to a `SeeingMonster`.

(8) 3. Add a subclass `MovingMonster` of `Monster` that represents a monster that moves horizontally from left to right across the screen by itself (use the Timer). Change `addMonsters` to add a moving monster.

   Note that a MovingMonster will move off the screen when it reaches the edge. This is the intended behavior.

(12) 4. Add a subclass `BouncingMonster` that represents a monster that has instance variables `int speedX` and `speedY` that denote the speed of the monster in the horizontal and vertical direction respectively. When it reaches the edge of the panel, it should bounce back as a ball. I.e., the angle of entry should be equal to the angle of exit (you do not have to calculate angles, simple changes of `speedX` and/or `speedY` suffice).

Include a constructor that sets the initial position and the horizontal and vertical speed of the monster. Change `addMonsters` to add a bouncing monster.

---

*Good luck!*

## Appendix – Some methods from the Java API

In `java.awt.Component`:

```
public int getWidth()
```

Returns the current width of this component. This method is preferable to writing `component.getBounds().width`, or `component.getSize().width` because it doesn't cause any heap allocations.

**Returns:**

the current width of this component

```
public int getHeight()
```

Returns the current height of this component. This method is preferable to writing `component.getBounds().height`, or `component.getSize().height` because it doesn't cause any heap allocations.

**Returns:**

the current height of this component

In `java.awt.Graphics`:

```
public abstract void fillOval(int x,
            int y,
            int width,
            int height)
```

Fills an oval bounded by the specified rectangle with the current color.

**Parameters:**

`x` - the *x* coordinate of the upper left corner of the oval to be filled.

`y` - the *y* coordinate of the upper left corner of the oval to be filled.

`width` - the width of the oval to be filled.

`height` - the height of the oval to be filled.

```
public abstract void drawOval(int x,
            int y,
            int width,
            int height)
```

Draws the outline of an oval. The result is a circle or ellipse that fits within the rectangle specified by the `x`, `y`, `width`, and `height` arguments.

The oval covers an area that is `width + 1` pixels wide and `height + 1` pixels tall.

**Parameters:**

`x` - the *x* coordinate of the upper left corner of the oval to be drawn.

`y` - the *y* coordinate of the upper left corner of the oval to be drawn.

`width` - the width of the oval to be drawn.

`height` - the height of the oval to be drawn.

```
public abstract void setColor(Color c)
```

Sets this graphics context's current color to the specified color. All subsequent graphics operations using this graphics context use this specified color.

**Parameters:**

`c` - the new rendering color.