APS101 lab 2 – week 3

This document contains the instructions for the week 3 APS101 lab. To earn your lab marks, you must actively participate in the lab (*You don't need to finish in the time alloted, you just need to try hard*).

1 Objectives

- Make sure you have email forwarding set up properly.
- Write subclasses of class `JFrame`.
- Write methods and method calls.
- Experiment with typecasts.
- Solve some simple geometry problems.
- Use local variables.
- Rewrite code to eliminate local variables.

2 Email forwarding

Make sure you have email forwarding set up properly (e.g. send an email to your `@ecf.utoronto.ca` email and see if it is forwarded to your other email). Ask your TA for help if you need it.

Driver and navigator

driver: The person typing at the keyboard.

navigator: The person watching for mistakes, and thinking ahead.

Throughout the lab, you'll be switching back and forth between the driver and navigator roles. We repeat the most important rule for this lab: **The navigator must not touch the keyboard or mouse.** If the navigator does type or click when they are not supposed to, the navigator will get a zero for the lab.

3 Screen size

Sit down with your partner. The rest of these instructions call you two `s1` and `s2`. Pick which one is which. **Important Note: `s1` should log in and start up DrJava, and be the first driver.**

Type the following into the Interactions pane. This gets the screen height and width and saves them in `int` variables `screenWidth` and `screenHeight`.

```
import java.awt.*;
Dimension d= Toolkit.getDefaultToolkit().getScreenSize();
int screenWidth= (int) d.getWidth();
int screenHeight= (int) d.getHeight();
d
screenWidth
screenHeight
```

Hints: Whenever a class you are defining needs to know how big the screen is, you can use the method calls and variables above. Remember that `(int)` is a typecast: it converts a `double` value to an `int` value. Why do we typecast? Because `d.getWidth()` and `d.getHeight()` return `double`s, but `JFrame`'s `setSize` method needs `int`s. See what happens if, for example, you leave out the typecast in the assignment to `screenWidth`.

4 Working with the screen size

Write a subclass of `JFrame` called `MaxWindow`. A subclass is what we have also called a "customized" class. It has the same methods as `JFrame`, plus some extra ones: `MaxWindow`'s methods are described in the instructions below.

The next couple of paragraphs describe how you should work in this lab. This is a useful way to proceed whenever you are writing a `Java` class.

Compile very frequently, even if you have barely written anything: For example: write only the part that says you are about to define a `MaxWindow` class: "`public class ... extends ...`" (with the proper elements instead of the dots!), a curly brace, a closing curly brace, and leave the class body empty (nothing between the curly braces). Save this much (in a file called `MaxWindow.java`). Compile. You may find errors - we often mistype things without noticing. The compiler will notice, though. Fix the errors.

Next, write a `method header` with an empty `method body`; compile, and fix errors. Then write part of the method body, compile, and fix any errors. Write the rest of the body; compile, fix, and so on. (If you are not sure what is meant by "header" and "body", ask your TA.) Continue this way until you have written all the methods.

Test each method from the Interactions pane right after you write it. (Again, if you are not sure how to do this, ask your TA.)

Here are the method descriptions:

1. **Switch roles: `s2` drives and `s1` navigates.** Write a `void` method called `maximizeHeight`, which moves the window to the top of the screen and makes it as tall as the screen. Methods `setLocation` and `setSize` will come in handy. Use the `Toolkit` code from step 3 to get the screen dimensions.
2. Write a `void` method called `maximizeWidth`, which moves the window to the left of the screen and makes it as wide as the screen.
3. **Switch roles: `s1` drives and `s2` navigates.** Write a `void` method called `maximize`, which moves the window to the top left of the screen and makes it as big as the screen. You must do this by calling `this.maximizeHeight` and `this.maximizeWidth`.
4. Write an `int` method called `fitIntoScreenWidth`, which returns how many whole times this `MaxWindow` object can fit across the screen. Again, you will need to use the code from step 3 to get the screen dimensions.
5. Write an `int` method called `fitIntoScreenHeight`, which returns how many whole times this `MaxWindow` object can fit along the screen from top to bottom.

5 Parameters and return values

In this section you will write another subclass of JFrame. This subclass is independent of `MaxWindow` that you just wrote: it is a brand new subclass, starting from scratch:

Write a subclass of `JFrame` called `TilingWindow` that has the methods described in the following paragraphs. As before, **compile frequently**. Test each method from the Interactions pane after you write it. Save the class in a file called `TilingWindow.java`.

You may not have time to write all of these methods, but try your best and do as many as you can.

1. **Switch roles: `s2` drives and `s1` navigates.** Write an `int` method called `widthRatio` that has one parameter, a `JFrame otherWindow`, and returns the value obtained by dividing the width of `this` window by `otherWindow`'s width.

2. **Switch roles: `s1` drives and `s2` navigates.** Write a `boolean` method called `canTileSideways` that has two parameters, an `int i` and a `JFrame j`, and returns `true` if `i` copies of `j` will fit side by side inside this window. For example, if `i` is 8, `j` is 50 pixels wide, and this window is 430 pixels wide, then the result is `true` because 8 copies of `j` can fit side by side. However, if `i` is 9, then the result is `false` because 9 copies of `j` won't fit: $9 \times 50 > 430$. You'll need to use the `Toolkit` code from step 3 to get the screen dimensions.

3. **Switch roles: `s2` drives and `s1` navigates.**

   If you wrote `canTileSideways` so that it uses a local `boolean` variable, rewrite it without such a variable.

4. Write a `boolean` method called `canTile` that has two parameters, an `int i` and a `JFrame j`, and returns `true` if `i` copies of `j` will fit inside this window in a grid pattern. Hint: figure out how many times `j` fits horizontally and how many times `j` fits vertically, and go from there. Can you do this without declaring any variables?