SOFTENG 251
Object-Oriented Software Construction
Laboratory A Exercise

Focus: Unit Testing and JUnit tool (SPACE-SHAPE I)
Instructor: Paramvir Singh
Tutors: Timo Van Veenendaal, Frank Wang Ma, Sanuri Gunawardena
Course Coordinator: Ewan Tempero

April 25, 2019

1 Goals

The goal of this lab is to:

- reinforce skills in object-oriented programming,
- learn JDK graphics libraries for developing basic java-based animations, and
- practice unit testing and JUnit tool

2 Learning Outcomes

On successful completion of this lab, you should be able to:

- use Graphics and Color java classes
- build, extend and test class hierarchies
- write basic unit test cases in JUnit in Eclipse
- learn changing code safely (without side effects)

3 Overview

This laboratory serves to reinforce skills in object-oriented programming. You will work with JDK libraries and the JUnit unit-testing tool.

In completing the given tasks, you will further develop the Space-Shapes application introduced in lectures. Essentially, Space-Shapes involves an animation comprising an extensible set of shapes in space. These shapes have in common knowledge of their position, velocity, direction and a rectangular bounding space zone (bounding box), while each type of shape has a specific way of rendering itself.

You will need to investigate a couple of java.awt classes: Graphics and Color to complete this assignment. Browsing the online JDK API should be sufficient for the needs of this exercise.

The source code for this exercise is available from Canvas. You will probably have lots of questions when looking through the source code, as there will be references to JDK entities that you have not seen before along with use of design techniques yet to be covered. For this exercise, however, you are essentially constrained to building and testing the Shape class hierarchy. As the course progresses, you will learn about the other aspects.

4 Assessed Lab

This is an assessed lab. If you complete it satisfactorily you will earn 2% towards your final grade.

To complete the lab satisfactorily you must have:

- Correctly implemented at least three out of four given tasks, where the correctness of your implementation is determined by SOFTENG 251 teaching staff. (Implementation)
- Written up a brief report reflecting on your performance in the lab. (Report)
- Demonstrate that you have understood the work by answering some verbal questions in your scheduled lab. (Oral questioning)

5 Submission and Due Date

In Week 9 practical session, you must demonstrate your work to a lab supervisor and convince the supervisor that you have met the assessment criteria for each task. Please note:

- You must attend the practical session to have your work assessed. Failure to demonstrate work in the practical session will result in a mark of zero. Your mark for this assignment is based purely on what you demonstrate (tasks, oral questions, and report) in the lab session.
- You should be in your scheduled practical session when it starts. If you arrive late there is no guarantee that lab supervisors will have time to assess your work.
- Prior to asking for your work to be assessed, you should have everything ready you will have no more than 5 minutes to convince the lab supervisor that you have met the stated criteria.
- You should also submit your work, in a single zip file, using **ADB by 17:00hrs** on **Friday May 17, 2019**.
- Your report must be assessed by the end of the **Wednesday/Thursday/Friday May 15-17, 2019 Lab** in order to receive the marks.

6 Planning and Notes

You do not need to wait until the lab to start the exercises. In general, you should come to all classes (especially labs) having done some preparation. It is much easier to understand new material and concepts if you have some idea of what to expect.

It is not mandatory to attend this lab, however it is strongly recommended that you come along, and get introduced to the JUnit tool and the lab exercise tasks. This will also give you an opportunity to ask questions and discuss the lab exercise in-person with the instructor and the tutors. If you do come, you are welcome to bring your own device (e.g. laptop).

It is worth recording what you do in your Journal, especially any details that do not appear in this description. As this is an assessed lab, you will need to provide a report on what you did (see the Report section below). These notes may be useful when writing the report, and it would be reasonable to keep the report in your Journal.

7 Preparation

- Read about unit testing for object-oriented software.
- *JUnit tool installation:* Most Eclipse releases already come equipped with JUnit. So, you should not need any kind of additional installation in Eclipse. For general JUnit configuration instructions in Eclipse, see the end of this document.

8 Resources

Source code java files for the Space-Ship application, description of the lab tasks, and a summary of steps to configure JUnit tool in Eclipse are provided at the end of this document.

9 Lab Location

Labs are in 301-1062 (Science Chem, Room 1062) also known as UG4 (Undergraduate Lab 4)

10 Report

There is good evidence that reflection significantly improves learning. Your report for this lab should consist of a short (100-200 words is sufficient but feel free to write more) discussion reflecting on how well you performed in the lab.

You can perform this self-reflection in three phases: *before*, *during* and *after* you have completed this lab exercise. As your knowledge and skill about software testing in general, unit testing, JUnit tool, and Java GUI libraries evolves through the underlying lab tasks, you will be able to write about this journey from knowing little to knowing more by the end of the lab. Note that here "end of the lab" relates to the completion of lab exercises by the given deadline, which is Week 9 for Lab 5.

Further, from your experience of making notes in Part 1 of this course, you are expected to have at least some understanding of the kind of self-reflection questions you can answer in your journal.

11 Exercise Details

11.1 Constraints

The changes you make to the Shape class hierarchy should not break any existing code (e.g. AnimationViewer class) that has been written to use the hierarchy.

11.2 Task 1: Add an Oval space-shape

Write unit tests for Oval, a new type of space-shape that moves like the other shapes but displays itself as an oval that fits into its bounding box. It must be possible to create an Oval using the same set of creation options as offered by class Shape. Implement Oval and include such a space-shape in your animation.

Assessment criteria

- The class hierarchy should be developed sensibly and in accordance with good object-oriented programming practice.
- The unit test(s) should demonstrate that an Oval instance paints itself correctly.
- The space-shape application should demonstrate correct functioning of an Oval instance.

11.3 Task 2: Add a Hexagon space-shape

Write unit tests for a Hexagon, a new type of Robot which moves like the other robots but displays itself as a hexagonal shape that fits into its bounding box. Again, it must be possible

to give values for instance variables at construction time or rely on default values. Implement Hexagon and include such a space-shape in your animation.

The Hexagon is to be painted to fit within the given width and height, using drawLine() calls to the Painter. The lines of the gem are to be painted starting with the left-most vertex and proceeding in a clock-wise direction.

The definition of the coordinates of the points is phrased here in terms of (x,y,width,height): The top-left and bottom-left vertices of a Hexagon are normally 20 pixels to the right of the left hand side of the bounding box. Similarly, the top-right and bottom- right vertices of a Hexagon are normally 20 pixels to the left of the right hand side of the bounding box. However, if the width of a Hexagon is less than 40 pixels, the top-left and top-right vertices are both positioned at point (x+width/2, y). Similarly, the bottom-left and bottom-right vertices are both positioned at point (x+width/2, y+height). In other words, small Hexagons are four-sided figures.

Hints: When developing the unit tests for Hexagon, you should include test cases to demonstrate that the small and regular Hexagon instances are constructed correctly. One way of writing these tests is to use a MockPainter and to check that its log contains the correct line sequences for Hexagon objects.

Assessment criteria

- The class hierarchy should be developed sensibly and in accordance with good object-oriented programming practice.
- The test cases should show that a Hexagon paints itself correctly based on its width and height.
- The Shape application should demonstrate correct functioning of a Hexagon instance.

11.4 Task 3: Add a DynamicShape

Write unit tests for a DynamicShape, a new type of space-shape that paints itself similarly to a Hexagon. At construction time, an additional java.awt.Color argument can be supplied. If not given, the new Hexagon objects color should default to black.

A DynamicShape moves like the other robots, but it sometimes changes its appearance when it hits the walls. After it bounces off the left or right wall it paints itself as a solid figure, in the color specified at construction time. After it bounces off the top or bottom wall it switches its appearance to that of a Hexagon, i.e. rendering itself with an outline. If it bounces off both walls, the vertical (left or right) wall determines its appearance.

Implement it and include such a space-shape in your animation.

Hints: To thoroughly test DynamicShape, you should prepare 4 test cases that verify that a DynamicShape instance renders itself correctly when bouncing off one wall: top, bottom, left and right. In addition you should develop a further four test cases to check that following a bounce of two walls (top-right, top-left, bottom-right, bottom-left) a DynamicShape objects appearance is consistent with its specification.

To add support for color, introduce three new methods to the Painter interface: fillRect(),getColor() and setColor(). fillRect() should take the same arguments as awt.Graphics.fillRect().getColor() should return a java.awt.Color value and setColor() should take a java.awt.Colorargument. Using these methods, you can query the current color before setting a new color and drawing a filled rectangle. After drawing the filled rectangle, you can reset the current color to its original value.

The MockPainter implementation of Painter should be extended to log color changes and drawing of filled rectangles. Class GraphicsPainter can implement the new methods to use its java.awt.Graphics object.

Assessment criteria

- The test cases should be sufficiently thorough to demonstrate that a DynamicShape meets the above specification.
- The implementation of DynamicShape and any changes you might make to Shape should be in accordance with good object-oriented programming practice. Note that this criterion will not be satisfied if you duplicate code in Shape and DynamicShape.
- The Space-Shape application should demonstrate correct functioning of a DynamicShape instance.

11.5 Task 4: Be imaginative!

Add a new and interesting kind of space-shape. Possible suggestions include (but are not limited to):

A Container space-shape that contains two member space-shapes. On bouncing, the Container shape toggles between its members and paints itself accordingly.

An Image space-shape that displays an image. For this, you could investigate class java.awt.Image from AWT library. There are several options for loading Image instances, e.g. javax.imageio.ImageIO.

A Shield that displays a border around an existing shape. A really cool implementation would allow a space-shape to be shielded an arbitrary number of times (borders around borders). On bouncing, none of the borders should pass beyond the space boundaries.

A starry twinkling night sky background. You can draw little white solid circles or solid star shapes to fulfill this requirement. Can you think of how to add such a twinkling effect to these little shapes?

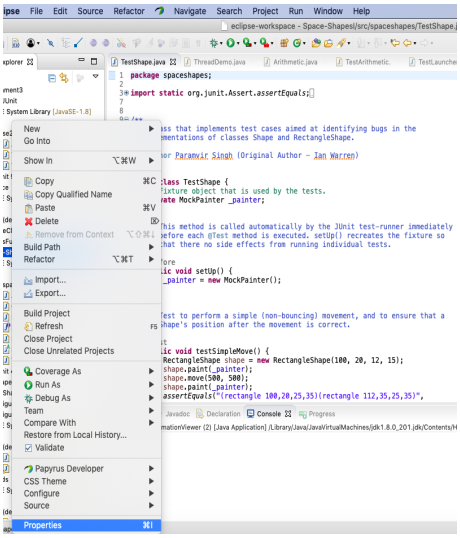
Assessment criteria

For this task, you simply need to demonstrate an interesting space-shape that makes appropriate use of object-oriented principles. Of course, it is good practice to write unit tests, but they are not required for assessment purposes in this case.

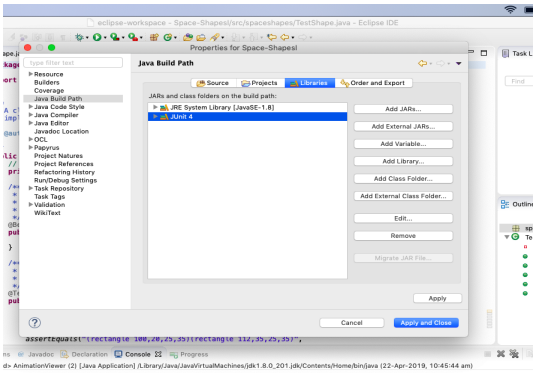
11.6 Configuring Eclipse with JUnit

JUnit is built into Eclipse, but it needs to be enabled as part of your projects settings. To enable JUnit:

1. Select Properties from the Project menu.



2. Select Java BuildPath from the list entries on the left.



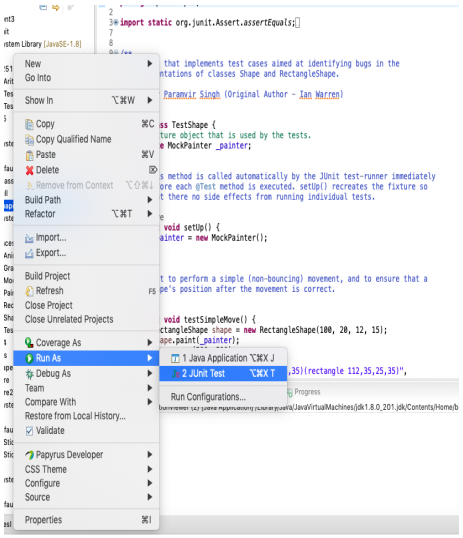
3. Click the Add Library button and select JUnit.

4. For JUnit library version, select JUnit 4.

5. Click the Finish button.

6. Finally, click the OK button in the Properties dialog.

To run a JUnit test class, e.g. TestShape, right-click on the class within the Project Explorer pane, and select "Run As" followed by "JUnit Test".



JUnit Quick Guide: https://www.tutorialspoint.com/junit/junit_quick_guide.htm