COMPSCI 230 (S1 2020) Assignment

Bounce: part I/III

Introduction

The aim of this assignment project (named Bounce) is to develop and reinforce skills in object-oriented programming. The context of this project is similar to what you handled in A1, which should help you get going with the rest of the assignments. You will develop an application of some complexity that is designed around design patterns and which leverages the JDK library classes. **The assignment project is split into three parts, and each part has a number of tasks.** You will generally need to complete one part before continuing on to the next.

This very assignment (A2 or Bounce-I) requires you to complete the first iteration of this project. In completing the assignment, you will further develop the Bounce application introduced to you in lectures. Essentially, Bounce involves an animation comprising an extensible set of shape types. Shapes have in common knowledge of their position, velocity, direction and bounding box, while each kind of special shape has a specific way of painting itself.

You will need to refer to JDK API when completing this assignment. Browsing the online JDK API documentation should be sufficient for the needs of this assignment.

The source code for this assignment 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 part of the assignment, however, you are essentially constrained to building and testing the *Shape* class hierarchy.** As the course progresses, you will learn about the other aspects.

Assessment criteria

Each task is associated with assessment criteria. To test that you have met the criteria, each task is associated with CodeRunner tests. The tests include a mix of:

- tests that run your compiled code to check that it is functionally correct and,
- other tests that examine your source to ensure that conforms to object-oriented principles.

CR3 submissions carry **85%** of the marks, whereas the ADB submission carry the rest **15%**.

Marking scheme for CR3 questions is mentioned against each task. *ADB submission will be marked based on whether your entire submitted code compiles and runs successfully to show the expected GUI as per specifications given in assignment brief.* As such, if your code passes all CR3 tests for all six questions, and your code is showing the expected GUI in *your* IDE, submitting the full iteration code should work fine at marker's end too. You can ensure this by carefully packaging, zipping and submitting the verified code to ADB as per the submission instructions given below.

Submission

For part 1 of the assignment (A2 – Bounce I), you must:

- (8.5 marks) pass the *CodeRunner3* tests by Week 8 (Friday *9:00*pm, 15 May 2020).
    - visit **coderunner3.auckland.ac.nz.**
    - under 'Assignments', you should access 'A2 – Bounce I' Quiz.
    - the quiz has a total of 6 questions, each of which would require you to paste the source code of one of your classes' that you wrote in your IDE to complete each task.

    - each assignment task has two CR questions (one question for passing tests and another for static analysis of class code)
    - You may click on 'Check' for each question multiple times without any penalty.
    - You may however make only one overall CR submission for the whole quiz.
- (1.5 marks) submit your entire **source code (including the original Canvas code)** to the Assignment Drop Box (ADB – ***adb.auckland.ac.nz***) by Week 8 (Friday *10:00*pm, 15 May 2020).
    - The code submission must take the form of **one zip file** that *maintains the Bounce package structure*.
    - You must name your zip file as "**YourUPI_230a2_2020.zip**".
    - You may make more than one ADB submission, but note that every submission that you make replaces your previous submission.
    - Submit ALL your files in every submission.
    - Only your very latest submission will be marked.
    - Please double check that you have included all the files required to run your program in the zip file before you submit it. **Your program must at least compile and run (through Java 1.8) to gain any marks.**

NOTE: It would be ideal to first complete all tasks (that you are able to complete) in an IDE like Eclipse, and then only proceed to answer CR3 questions once you have finished writing working code for this iteration in your IDE.

Constraints

For all three parts of the assignment project, any changes you make to the Shape class hierarchy must not break existing code (e.g. the `AnimationViewer` class). In particular, **class `Shape` must provide the following public interface** *(note this is not the Java "interface"; here the term "interface" refers to the set of public methods in a class that may act as entry points for the objects of other classes)* **at all times**:

- `void paint(Painter painter)`
- `void move(int width, int height)`

Task 1: add class OvalShape

An `OvalShape` is a special kind of `Shape` that moves like other shapes but which paints itself as an oval that fits within its bounding box.

Specific requirements

- Class `OvalShape` must be a subclass of `Shape`.
- It must be possible to create an `OvalShape` object using the same set of creation options as offered by class `Shape`. In other words, class `OvalShape` must provide 4 constructors, similarly to class `Shape`.
- An `OvalShape` should use Painter's `drawOval()` method when painting itself.

Once created, edit the `AnimationViewer` class to add an instance of `OvalShape` to your animation.

Testing

From the CodeRunner quiz named *Bounce I*, complete the first two questions:

- (1.5 marks) *Bounce I OvalShape*, which runs a series of tests on your `OvalShape` class.
- (1 mark) *Bounce I OvalShape (static analysis),* which examines the source code for your `OvalShape` class.

Your code will need to pass all tests.

Task 2: add class DynamicRectangleShape

A `DynamicRectangleShape` is a new kind of Shape that paints itself similarly to a `RectangleShape`. At construction time, a fill color may be specified that is used when painting a `DynamicRectangleShape`. Depending on how it bounces, a `DynamicRectangleShape` alters its appearance.

Specific requirements

- Class `DynamicRectangleShape` must be a subclass of `RectangleShape`.
- Class `DynamicRectangleShape` must provide two constructors:

```
public DynamicRectangleShape(int x, int y, int deltaX, int deltaY,
                                                    int width, int height)

public DynamicRectangleShape(int x, int y, int deltaX, int deltaY,
                                        int width, int height, Color color)
```

- If a `DynamicRectangleShape` object is created without specifying a color, it should default to red.
- Prior to any bouncing, a `DynamicRecantangle` paints itself as a solid figure, in the color specified at construction time or the default red color. After bouncing off the left or right wall, a `DynamicRectangleShape` always paints itself as a solid shape. After it bounces off the top or bottom wall, it switches its appearance to that of a `RectangleShape`, i.e. painting itself with a black outline. If it bounces off both walls, the vertical (left or right) wall determines its appearance.
- In implementing `DynamicRectangleShape`'s movement/bouncing behaviour, use the object-oriented programming technique of **overriding method `move()` for refinement**.
- After making any calls to a `Painter` to effect a color change, a `DynamicRectangleShape` object should restore the painting color to what it was before it made the change.
- Make only necessary calls to `Painter`'s `setColor()` method.

Implement class `DynamicRectangleShape` and include such a shape in your animation.

Testing

From the CodeRunner quiz named Bounce I, complete the following questions:

- (2 marks) *Bounce I DynamicRectangleShape*, which runs a series of tests on your DynamicRectangleShape class.
- (1 mark) *Bounce I DynamicRectangleShape (static analysis),* which examines the source code for your DynamicRectangleShape class.

Your code will need to pass all tests.

Task 3: add class ImageRectangleShape

`ImageRectangleShape` is a special type of Shape that displays an image. At construction time, an `ImageRectangleShape` takes a `java.awt.Image` parameter that represents an image, e.g. in a format such as JPEG, PNG or GIF. In an animation, an `ImageRectangleShape` objects displays its image as it moves and bounces.

Specific requirements

- Class `ImageRectangleShape` must be a subclass of `RectangleShape`.

- Class `ImageRectangleShape` must provide one constructor:

```
public ImageRectangleShape(int deltaX, int deltaY, Image image)
```

- Class `ImageRectangleShape` must provide a static factory method:

```
public static Image makeImage(String imageFileName, int
                                                    shapeWidth)
```

  This method loads the image specified by `imageFileName` and scales it. Specifically, if the width of the image is greater than `shapeWidth`, which is the desired width of the `ImageRectangleShape` object that will display the image, the image must be scaled so that its width equals `shapeWidth`.

Implement class `ImageRectangleShape` and include such a shape in your animation.

Testing

Complete the CodeRunner quiz named Bounce I by working through the remaining questions:

- (2 marks) *Bounce I ImageRectangleShape*, which runs a series of tests on your ImageRectangleShape class.
- (1 mark) *Bounce I ImageRectangleShape*, which examines the source code for your ImageRectangleShape class.

Your code will need to pass all tests.

Hints

- To implement the **static** `makeImage()` method, you will need to use image processing classes from the JDK: `java.awt.Image`, `java.awt.Graphics2D`, `java.awt.image.BufferedImage` and `java.io.File`. Consult the JDK API documentation.
- Download a suitable image file, in JPEG, PNG or GIF format, from the Internet that you can use for this task.
- The image file should be stored in a directory that is accessible to the AnimationViewer application. You may want to use your login directory. On Windows machines, this is C:/Users/<username>, where <username> is replaced with your username. The following code shows how you can generate the full path name for an image named Holden.jpg that is stored in your login directory.

```
String path = System.getProperty("user.home");
String imageFileName = path + "/" + "Holden.jpg"
```

  The `imageFileName` variable can then be suppled as the first argument to `ImageRectangleShape`'s static `makeImage()` method, and should be a *fully qualified path* to the image file.

- A possible algorithm for `makeImage()` is as follows:

  variables:

sw	*width of new shape (int)*
sh	*height of new shape (int)*
f	*instance of File (java.io.File)*
b	*instance of BufferedImage (java.awt.image.BufferedImage)*

w *width of loaded image (int)*
h *height of loaded image (int)*
sf *scale factor (double)*
g *instance of Graphics2D (java.awt.Graphics2D)*

algorithm:

1. *Let f = new java.io.File object, supplying a string whose value represents the name of the image file*
2. *Let b = new BufferedImage object, using javax.imageio.ImageIO's read() method with f as a parameter*
3. *Let w = b's width, obtained using b's getWidth() method*
4. *Let h = b's height, obtained using b's getHeight() method*
5. *If w > sw:*
    a. *Let sf = sw / w*
6. *Let sh = h * sf*
7. *Let b = new BufferedImage object, supplying sw and sh as parameters*
8. *Let g = new Graphics2D object by calling method createGraphics() on b*
9. *Draw the image onto g by calling g's drawImage() method*
10. *Return b*

- With the above algorithm, note:
    - When calculating *sf* and *h,* you will need to use casting to type double.
    - `Graphics2D` inherits a six-argument `drawImage()` method that is suitable to use. The last parameter is an `ImageObserver` that can be used to track the loading of an image; you can simply provide a **null** value for this parameter.

Debugging

I strongly recommend that you use **Eclipse** for all assignments, and use **Eclipse debugging** feature for your assistance.

ACADEMIC INTEGRITY

The purpose of this assignment is to help you develop a working understanding of some of the concepts you are taught in the lectures.

We expect that you will want to use this opportunity to be able to answer the corresponding questions in the exam.

We expect that the work done on this assignment will be your own work.

We expect that you will think carefully about any problems you come across, and try to solve them yourself before you ask anyone for help.

This really shouldn't be necessary, but **note the comments below about the academic integrity related to this assignment**:

The following sources of help are acceptable:

- Lecture notes, tutorial notes, skeleton code, and advice given by us in person or online, with the exception of sample solutions from past semesters.
- The textbook.

- The official Java documentation and other online sources, as long as these describe the general use of the methods and techniques required, and do not describe solutions specifically for this assignment.
- Piazza posts by, or endorsed by an instructor.
- Fellow students pointing out the cause of errors in your code, without providing an explicit solution.

The following sources of help are **NOT** acceptable:

- Getting another student, friend, or other third party to instruct you on how to design classes or have them write code for you.
- Taking or obtaining an electronic copy of someone else's work, or part thereof.
- Give a copy of your work, or part thereof, to someone else.
- Using code from past sample solutions or from online sources dedicated to this assignment.

The Computer Science department uses copy detection tools on all submissions. Submissions found to share code with those of other people will be detected and disciplinary action will be taken. To ensure that you are not unfairly accused of cheating:

- Always do individual assignments by yourself.
- Never give any other person your code or sample solutions in your possession.
- Never put your code in a public place (e.g., Piazza, forum, your web site).
- Never leave your computer unattended. You are responsible for the security of your account.
- Ensure you always remove your USB flash drive from the computer before you log off, and keep it safe.

Concluding Notes

Not everything that you will need to complete this assignment has or will be taught in lectures of the course. You are expected to use the Oracle Java API documentation and tutorials as part of this assignment.

Post any questions to Piazza or ask via Zoom (avoid emails) – this way, the largest number of people get the benefit of insight!

We may comment along the lines on Piazza to deal with any widespread issues that may crop up.