# COMPSCI 230 Assignment

Bounce: part III/III

## Introduction

This is the final instalment of the Bounce project, and perhaps the most challenging. The emphasis of Bounce III is on working with design patterns and Java's Swing API. You are required to complete and extend the application in a way that makes appropriate use of design patterns. Bounce III is a model/view application that presents three views of a shared model. Such applications are commonplace and introduce the need for views to be mutually consistent and synchronised with a model whose state changes at run-time. Once you have completed the assignment tasks, you should have an application that looks similar to that shown in **Appendix 1**.

## 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 **90%** of the marks, whereas the ADB submission carry the rest **10%**.

The marking scheme for CR3 questions is mentioned against each task. *ADB submissions 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 four CR3 tests, and your code is showing the expected GUI in *your* IDE, submitting the full code should work fine at the 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 3 of the assignment (*A4 – Bounce III*), you must:

- **(9 marks)** pass the *CodeRunner* tests by Week 12 (Friday 11:59pm, 30 October 2020)
    - visit **coderunner3.auckland.ac.nz.**
    - under 'Assignments', you should access 'A4 – Bounce III' Quiz.
    - the quiz has a total of four questions, each of which would require you to paste the source code of one or more of your classes' that you wrote in your IDE to complete each task.
    - Task 1 and Task 2 have one CR question each, and Task 3 has two CR questions.
    - 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 mark)** submit your **source code (including the original Canvas code)** to the Assignment Drop Box (ADB – ***adb.auckland.ac.nz***) by Week 12 (Friday 11:59pm, 30 October 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_230a4_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 most recent 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 compile and run (through Java 1.8).**

**NOTE: It is highly recommended to first complete all tasks 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. However, if you did not complete at least one of the previous Bounce assignments, then this may not be advisable. Read the preparation section below for more.**

## Constraints

For all 3 parts of the of the assignment, 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)`

## Preparation

You will need to download the **bounce-3.zip** file from Canvas. The zip file includes the source code and a properties file. The properties file allows you to specify application properties that are read by the application.

You are strongly advised to **construct a model** of the application that captures the key classes, instances and their interactions. The model need not be "proper" UML – informal diagrams that allow you to discover, document and visualise the application's structure would be sufficient to obtain the required understanding. *It is common in industry to work on existing software projects, and an ability to understand their structure is an important skill.*

You should copy all the files from the bounce package that you have developed for **Bounce I** and **II** into the bounce package of **Bounce III**. Specifically, these (Java) files are: *Shape*, *RectangleShape*, *OvalShape*, *DynamicRectangleShape*, *ImageRectangleShape and NestingShape*. You should replace your **ImageRectangleShape** class with that contained in the bounce-III.zip download.

***For students who did not complete Bounce I and II**:- You can still proceed with Bounce III by implementing the required Bounce III classes, and then testing and submitting them using CodeRunner. However, without the classes from Bounce I and II, your Bounce application won't compile outside of CodeRunner. Also, without the classes for Bounce I and II, you won't be able to develop an executable Bounce application. This means that you will forfeit ADB (**1 out of 10 marks**) for this assignment.*

If you did not complete any of the first two assignments, and **you intend to write code to include those previous shapes** you should study the **A2 and A3 review lectures** on Canvas pages.

## Supplied code

The Bounce III project is organised around five packages:

- **bounce:** This includes the Shape hierarchy classes, Painter and the classes and interfaces concerned with the Shape model: ShapeModel, ShapeModelListener and ShapeModelEvent.
- **bounce.bounceApp:** This package contains the main Bounce application class and BounceConfig, which provides a convenient way of accessing application properties.
- **bounce.forms:** Package bounce.forms contains GUI form classes that allow the user to specify Shape attribute values. These include ColourFormElement for specifying a Shape's colour, ImageFormElement for specifying a Shape's image file, and ShapeFormElement for specifying general Shape attributes such as width and height. In addition, this package contains FormHandler implementations that read form data and which instantiate particular Shape subclasses.
- **bounce.forms.util:** This contains application-independent artefacts for building forms. Form, FormElement and FormHandler are basic interfaces for representing forms, form elements containing data fields, and form processing respectively. FormComponent and FormElementComponent are Swing component implementations of Form and FormElement. FormUtility is convenience class for laying out forms. Of the five Bounce packages, this is the one that you need to be least familiar with.
- **bounce.views:** This package includes the view classes for this application and their associated adapters that link them to a ShapeModel.

The supplied code makes the following assumptions about your bounce package code:

- **Class Shape and all shape subclasses** (other than ImageRectangleShape) have a 7-argument constructor that takes the following parameters in order: x, y, deltaX, deltaY, width, height and text. The first 6 parameters are of type int, while text is of type String.
- **Class DynamicRectangleShape** has an 8-argument constructor that takes the following parameters in order: x, y, deltaX, deltaY, width, height, text and colour. The first 6 parameters are of type int, text is a String and colour's type is java.awt.Color.

Once your code conforms to the above expectations, the project will still contain compilation errors because the tasks have yet to be completed. Specifically, the following units will have compilation errors:

- bounce.bounceApp.Bounce. This is the main application class. It requires bounce.views.TreeModelAdapterWithShapeModelListener to be implemented.
- bounce.forms.TestImageShapeFormHandler. This test case requires you to implement class bounce.forms.ImageShapeFormHandler in Task 3.
- bounce.views.TreeViewer requires that bounce.views.TreeModelAdapter is implemented.

# Task 1: implement class bounce.views.TreeModelAdapter

A key part of a ShapeModel object is its composite structure of Shapes. When completed, the Bounce application should include a hierarchical view of this composition structure. Rather than reinvent the wheel, it makes sense to reuse Swing's JTree component that is tried and tested and which is intended for visualising hierarchical structures. However, there is an interface mismatch problem in that a JTree component is not able to render ShapeModel objects directly. Rather, a JTree component can render any TreeModel object. This problem – of wanting to make incompatible objects work together – is common in object-oriented software development and can be solved using the **Adapter design pattern**. Apply this design pattern so that you can display a ShapeModel's shape composition using a JTree component.

Once you have completed this task, you can run the demo program bounce.views.TreeViewer. Assuming your TreeModelAdapter class is correct, this program will display a ShapeModel's shape composition.

## Hints

- Carefully study the API documentation for Swing's TreeModel interface. You will spend less time on this task overall if you invest in some up-front activity to understand the TreeModel interface.
- Many of the TreeModel methods have arguments of type Object that you will need to cast to Shape and NestingShape. In implementing class TreeModelAdapter, be sure to use the **instanceof** operator so that only safe casts are performed.
- Class TreeModelAdapter should define one constructor that takes as argument – a reference to an object of type ShapeModel.
- TreeModel's valueForPathChanged() method can simply be implemented with an empty method body. This method is an event notification method that can safely be implemented like this in the context of this task.

## Testing

From the CodeRunner quiz named *Bounce III*, complete the first question:

- **(3 marks)** *Bounce III TreeModelAdapter*, which runs a series of tests on your TreeModelAdapter class.

Your code will need to pass all tests.

## Task 2: Implement class - bounce.views.TreeModelAdapterWithShapeModelListener

Your TreeModelAdapter class has leveraged the Swing framework, and with very little effort you have a robust means of visualising a ShapeModel's shape composition. Awesome. However, the TreeModelAdapter class suffers from a limitation that prevents the Bounce application from working correctly. Bounce is a model/view application, with the GUI showing three different views of a ShapeModel instance. The problem is that while the AnimationView and TableViewAdapter views respond to changes in the state of ShapeModel, an instance of the TreeModelAdapter class does not. Consequently, when a new Shape instance is added or an existing Shape object is removed from the ShapeModel, the change is not reflected in the JTree component. Obviously, this is most undesirable as all views should offer consistent representations of a common model.

Introduce a new class, TreeModelAdapterWithShapeModelListener, that extends your TreeModelAdapter class such that a TreeModelAdapterWithShapeModelListener instance can both render a ShapeModel's shape composition and respond to changes that occur in the ShapeModel. Using a TreeModelAdapterWithShapeModelListener instance, its connected JTree component will update its display whenever the ShapeModel changes. It is helpful to think of a TreeModelAdapterWithShapeModelListener object as follows:

- A TreeModelAdapterWithShapeModelListener object plays two roles: the model of a JTree component, and the (non-visual) view (listener) of a ShapeModel.
- A TreeModelAdapterWithShapeModelListener object essentially transforms ShapeModel events that is hears about into TreeModel events that it fires to its TreeModelListeners. In the case of the Bounce application there is one such listener, the JTree component.

Implement class TreeModelAdapterWithShapeModelListener to do the necessary. Once completed you should have a functioning Bounce application. Once you have edited and deployed the application's properties file, discussed below, run the application and enjoy the fruits of your labour.

On startup the Bounce application attempts to read configuration information from the file named *bounce.properties*. You should locate this file within your login directory. For home PCs running Windows, the login directory is C:/Users/<username>, where <username> is replaced with your username. You should ensure that the shape classes property lists the fully qualified names of Shape subclasses that you want the application to use. Essentially, the *combo box* on the GUI is populated with the names of the classes you specify for the shape classes property. Hence, when using the application you can specify the kind of shape you want to add to the animation. In specifying class names for this property, each name should be separated by whitespace; note however that if names are spread across multiple lines a \ character should end each line except the last. For example, the following specifies that two classes should be loaded:

```
shape classes = bounce.RectangleShape \
    bounce.NestingShape
```

### Hints

- Carefully study the Java 8.0 API documentation for Swing's TreeModelListener interface and classes TreeModelEvent and TreePath. Similarly to before, you will spend **significantly less time** on this task if you **study the API pages** first rather than rushing in and trying to hack something together.

- There is a reasonable amount of complexity in the TreeModelListener and TreeModelEvent entities. Some of this complexity lies in the ability for a TreeModelEvent to describe multiple node additions/removals/changes in a TreeModel. Note that your TreeModelAdapterWithShapeModelListener class need only generate TreeModel events that describe a single addition or removal of a Shape at a time. Hence, the childIndices and children arrays that form part of the state of a TreeModelEvent will always have a length of 1.
- Each node shown in the Bounce application's JTree component represents a composite or simple shape instance within the ShapeModel object. Each node is labelled with the name of the Shape subclass that the node is an instance of. The JTree acquires the label value for each node by calling method toString() on each Shape instance it discovers through making TreeModel calls on its model. Given that these label values are not subject to change in the Bounce application, your TreeModelAdapterWithShapeModelListener class need not make any treeNodesChanged() calls on registered TreeModelListeners. In other words, your TreeModelAdapterWithShapeModelListener class has only to make treeNodesInserted() and treeNodesRemoved() calls on registered listeners, thereby communicating new and removed nodes, respectively.
- Class TreeModelAdapterWithShapeModelListener should define one constructor that takes as argument a reference to an object of type ShapeModel.

## Testing

From the CodeRunner quiz named *Bounce III*, complete the second question:

- **(3 marks)** *Bounce III TreeModelAdapterWithShapeModelListener*, which runs a series of tests on your class.

Your code will need to pass all tests.

# Task 3: add class bounce.forms.ImageShapeFormHandler

The Bounce III application allows images to be loaded and added to the animation. Key classes are:

- bounce.ImageRectangleShape. This is a Shape subclass that paints a given image.
- bounce.forms.ImageFormElement. An ImageFormElement allows the user to select a particular image from disk that is to be displayed by an ImageRectangleShape.
- bounce.forms.SimpleImageShapeFormHandler. An instance of SimpleImageShapeFormHandler processes data entered in form elements ShapeFormElement and ImageFormElement and uses this to instantiate an ImageRectangleShape object.

Class SimpleImageShapeFormHandler scales the image selected by a user based on the ShapeFormElement's width field value. ShapeFormElement's height is ignored as the scaling operation maintains the image's aspect ratio.

The problem with SimpleImageShapeFormHandler is that the loading and scaling operations – which are expensive for large images – are performed by the Event Dispatch thread. The image processing thus causes the application to freeze and become unresponsive when working with large images.

For this task, you should create a new class (bounce.forms.ImageShapeFormHandler) to more appropriately load and scale the image using a background thread, and make the new ImageRectangleShape instance available to the Event Dispatch thread. Like SimpleImageShapeFormHandler, ImageShapeFormHandler should have a constructor that takes ShapeModel and NestingShape parameters and should implement the FormHandler interface.

## Specific requirements

- Class ImageShapeFormHandler must use class SwingWorker as part of its implementation.
- Class ImageShapeFormHandler must implement the FormHandler interface.

Once you have implemented ImageShapeFormHandler, you should edit the FormResolver class to change method getFormHandler() to return an ImageShapeFormHandler rather than a SimpleImageShapeFormHandler. You can then run the Bounce application and have large images loaded and scaled in the background when creating ImageRectangleShape objects.

## Hints

- The logic for acquiring form data, loading and scaling the image, and instantiating ImageRectangleShape is provided in the existing SimpleImageShapeFormHandler class. You can reuse this without modification in ImageShapeFormHandler. Don't consider this to be a case of duplicating code – in reality you would discard the naive SimpleImageShapeFormHandler class.
- You will need a large image file (greater than 20MB) to effectively demonstrate your ImageShapeFormHandler implementation. One source of suitable images is https://effigis.com/en/solutions/satellite-images/satellite-image-samples/

## Testing

From the CodeRunner quiz named *Bounce III*, complete the remaining questions:

- **(2 marks)** *Bounce III ImageShapeFormHandler*, which runs a series of tests on your class.
- **(1 mark)** *Bounce III ImageShapeFormHandler (static analysis),* which analyses your code.

Your code will need to pass all tests.

## 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.

# Appendix 1 Screen shot of final Bounce application