Assignment 02 Java Best Practices

12 November 2020

Assignment: **35 marks**

Type: Team Assignment (code and report)

OVERVIEW

There is a large amount of legacy code that has been written for older versions of the JDK. Legacy systems, implemented using legacy code will still run, but there are advantages to modernising these applications to take advantage of modern libraries and language features. These legacy systems are often monolithic (meaning they have little or no reusable code "snippets") which can make these systems difficult to understand and modify.

As Java evolved, new language features have been added to enhance the language, some examples are

- JDK 1.5 Annotations, enhanced for loop, generics
- JDK 7 Type inference (declarations), try with resources, catch multiple exception types, Objects class
- JDK 8 Functional interfaces, lambda expressions, Optional class
- JDK 9 Modules, Optional class enhancements
- JDK 10 Local variable type inference
- JDK 11 Type inference for lambda parameters

In this assignment you will work on a legacy codebase and identify where it can be modernised, develop a plan to modernise it and rewrite the code as a well-documented modern API.

Deliverables

An updated group repository on `csgate.ucc.ie` and a report submitted via Canvas.

Due Date

Thursday 10th December 2020.

THE MODERNISATION TASK

Introduction

I have located some code from 1998 that implements a ray tracing application. The original code implemented a Java applet that read a scene description file and applied a ray tracing algorithm to render the objects described in the scene. The code written by Leonard McMillan is available at http://groups.csail.mit.edu/graphics/classes/6.837/F98/Lecture20/RayTrace.java and was written as a single Java file.

There has been an attempt to modernise the code. The code has been modularised (see module-info.java), the code has been updated to use JavaFX (instead of Java applet), however many occurrences of AWT do still exist, requiring methods to map between JavaFX rand AWT representations of colour. Also, some use of the `Vector` type have been replaced with `List<>`.

As it is currently presented the application presents the same functionality as the original. It reads a scene description file and renders an image based on its contents. An example scene description file is

```
# Camera
eye 1.5 10.5 -1.5
lookat -0.5 0 -0.5

# Lights
light 1.0 1.0 0.981 ambient
light 0.9 0.9 0.9 ambient
light 0.745 0.859 0.224 ambient
light 0.6 0.6 0.6 directional -1 -1 -1

# Objects
surface 0.2 0.8 0.2 0.5 0.9 0.4 10.0 0 0 1
sphere -0.4 0.375 -0.4 0.375

surface 0.7 0.3 0.2 0.5 0.9 0.4 6.0 0 0 1
sphere -0.6 1.05 -0.6 0.3

surface 0.2 0.3 0.8 0.5 0.9 0.4 10.0 0 0 1
sphere -0.8 1.575 -0.8 0.125
```

The only shape allowed is a sphere, the appearance of which is determined by the last defined surface. All other elements relate to the position of the observer and the position and type of the lights.

ASSIGNMENT REQUIREMENTS

The current implementation uses an external scene description file. The re-written application should present an API suitable for ray tracing. In this new API a client should be able to programmatically create the elements required for ray tracing to construct a render job. You should consult the existing ReadInput() method, to discover how the current solution constructs the rendering job from the input

```
// adds an object to the objectList
objectList.addElement(new Sphere(currentSurface, v, r));
// adds a light to the lights list
lightList.addElement(new Light(Light.AMBIENT, null, r, g, b));
```

When the render job is run the result can be accessed as an image. The way this functionality is presented is a decision your group must make.

The modernised application should

- Remove the monolithic structure of the original application.
- Use Java coding conventions.
- Restructure the application as an API for ray tracing.
- Remove outdated Java feature from the API and present a stable API.
- Document the API using Javadoc.

You are not required to (but may choose to do so)

- Re-implement the functionality of the methods.
- Read in an external scene description file (it is intended to be replaced by a programmatic API)
- Present the image as a part of the application (I.e. write a JavaFX application).

The assignment requires teamwork and using a shared (for each group) repository. How well the git repository is managed is part of the assessment.

ASSIGNMENT RESOURCES

You have been divided into groups (details on Canvas) on csgate.ucc.ie there is a folder on `/users/shared/cs3318` containing a git repository. The groups are numbered from group01 to group22 (note the leading zero). Each member of the group should clone a copy of the repository for their group, so for example if I am in group 00, I would open IntelliJ and create a new project from version control using

```
username@csgate.ucc.ie:/users/shared/cs3318/group00/raytracing.git
```

As the location of the git repository. Replace `username` with your lab username and you will be prompted for your lab password. Make sure that each commit to your repository has a single purpose and is documented using a commit comment.

To update the git repository on `csgate.ucc.ie` you must push content from your local repository to the shared repository. You will need to manage how to organise updates to the shared repository to avoid and resolve conflicts.

You should also note that work you intend to do may already be done by a member of your team and if it has been pushed to the shared repository can be pulled from there.

OUTCOMES

The assignment outcomes are

- Updated project on the group's repository, containing a well documented API for ray tracing
- A 500 word report describing the choices your team made about the API.