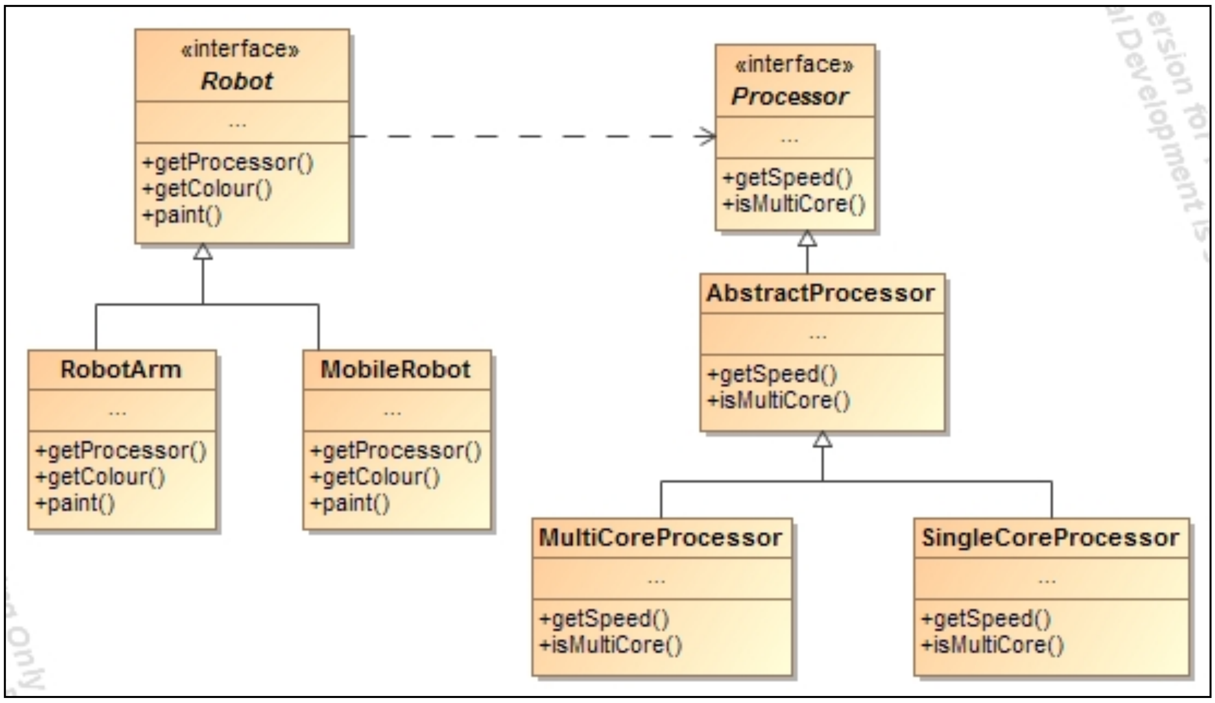
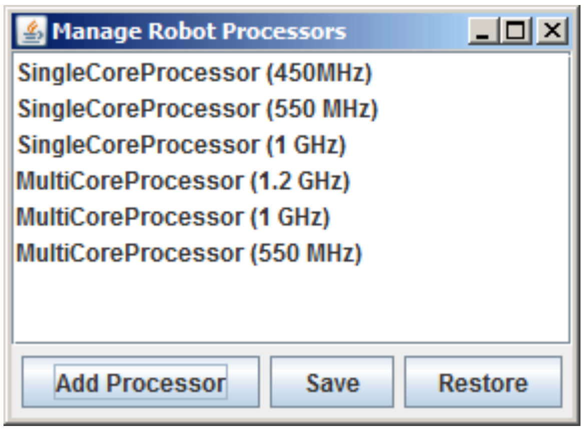
Design Patterns Practical Test CSC3031/CSC7055

This practical will involve amending and part-writing a small graphical application making use of a number of design patterns. The application is for a Robot factory that makes two types of robot: A MobileRobot and a RobotArm. All types of robots have a processor which can be SingleCore or MultiCore. The diagram below illustrates part of the domain structure of the application.

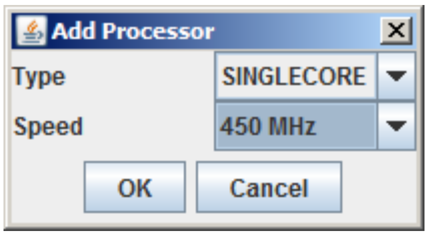


The part of the application which we are most concerned with is the part that manages the processor used in a robot. Our application displays a list of processors that are in stock and provides the functionality to add new processors as they become available. It can also save the list of available processors to persistent storage and restore back to the last saved point.

The finished application would, if completed, contain the following screens.



SingleCoreProcessor (450MHz)
SingleCoreProcessor (550 MHz)
SingleCoreProcessor (1 GHz)
MultiCoreProcessor (1.2 GHz)
MultiCoreProcessor (1 GHz)
MultiCoreProcessor (550 MHz)



Type: SINGLECORE
Speed: 450 MHz

The ‘Add Processor’ button brings up the second screen where you can choose the Type (Singlecore or Multicore) and the Speed (450MHz, 550MHz, 1 GHz or 1.2 GHz). ‘Save’ saves the list of available processors, ‘Restore’ takes the list back to the last save point.

Setting Up

Download **FirstnameLastname.docx** and **practicaltest.zip** from QOL. Unzip the zip file to a directory from where you can load it into whatever environment you want to program in.

Open the document FirstnameLastName.docx and save it, replacing FirstName and LastName with your name – this is where you will write or paste your answers. Try to keep to the template and when you are pasting – keep the IDE formatting if possible (Paste Special – Formatted Text).

Questions

Answer each question in the answer document, which you will email when finished. When pasting code, try paste special and keep format. That makes it easier for the marker to read.

Q1. You will notice that java files are grouped into 4 packages. One of these is called ‘`runner`’ – this is just for testing the application. The 3 other packages group java files in a way that partitions the systems into areas of concern. Look through the code and in your answers document; write in the architectural pattern that is being used to modularise these concerns. Explain your answer.

Q2. The package persistence is all about storage. A class **`EntityTable`** is responsible for storing and retrieving the processor objects using a numerical key. Explanation:

- Its constructor takes an `EntityKeyGenerator` object which it uses to generate a unique numerical key.
- The processor objects are stored in a Map, with the key from `EntityKeyGenerator` acting as a primary key. The map will point that key to a processor object.
- getByKey() allows us to retrieve a processor object
- getAll() retrieve all of the processors
- addEntity allows us to add a processor with a new key generated from `EntityKeyGenerator`
- restore() will retrieve the last saved list of processors

`EntityKeyGenerator` which returns the next key needs to be a Singleton. Rewrite this as an enum that will ensure that we can only have a single key generator and a single point of access to it (i.e. Singleton pattern). Copy the code you have written to your answer document – it would be helpful if you could keep formatting if using an IDE like Eclipse (Paste Special – Formatted text). Label it “Q2”.

Q3. EntityTable has this line:

```java
private Collection<EntityListener> listeners;
```

The purpose of this is to implement the Observer pattern .

Complete the methods:

- `addEntityListener`
- `removeEntityListener`
- `fireEntityAdded`
- `fireEntityRestored`

to make the Observer pattern work.

Copy these methods to your answer document.

Q4. The only point of access from classes in our domain package to those in our persistence package is via the enum `DatabaseWrapper`. Similarly all domain objects are accessed via `BusinessWrapper`. Which pattern are these cases an example of? Write the answer in your answer document.

Q5. In saving processors we want to be able to save as CSV format or as serialized objects. The classes with this responsibility are `EntityCSVSave` and `EntitySerializationSave`, respectively. Use the already written `AbstractProcessorSave` class as well as amending the `EntityCSVSave` and `EntitySerializationSave` classes to implement the Strategy pattern. Copy the three classes to your answers document.

Q6. In your answer document, use the examples of the save() and restore() methods in the EntitySerializationSave class to explain the Decorator pattern.

Q7. In the domain package there are two types of processor: MultiCoreProcessor and SingleCoreProcessor.

The BusinessWrapper has a method that calls the ProcessorFactory to create the type that it passes in.

```
    public Object addProcessor(String speed, Object type) {
        Processor engine = ProcessorFactory.create(speed, type ==
                                 ProcessorFactory.Type.MULTICORE);
        DatabaseWrapper.INSTANCE.addProcessor(engine);
        return engine;
    }
```

Complete the Factory class ProcessorFactory which will return the correct processser. Copy the ProcessorFactory class to your answer document.

Q8. The AddProcessorDialog class has this code associated with the OK button.

```
    public void actionPerformed(ActionEvent event) {

      BusinessWrapper.INSTANCE.addProcessor((String)speedCombo.getSelecteItem()
      , typeCombo.getSelectedItem());
             setVisible(false);
```

In your answer document state which pattern you would use to remove this tight dependency of the responsibility of addProcessor with the Button. Just write the name of the pattern.

Q9. Suppose we want to have a robot consist of other robots both mobile and robot arm types. Write a class that demonstrates the use of a suitable pattern that allows the new class to be treated as just another Robot. Assume the new class has its on controlling Processor. Write any required data members and a constructor that allows us to pass in a collection of robots and a reference to its controlling processor. Ignore getters and setters. Paste in the new class to your answer document.

Please make sure you have saved your *answers* as [FirstnameLastName].docx

Convert this file to a PDF and upload the PDF to the Assignment Tool on QOL.

Zip up your Java files and upload these to the Assignment Tool on QOL.