Name: Adam Smith

Student Number: 40056108

Q1

|  |
| --- |
| The architectural pattern being used is known as the layers pattern. The application is divided into three distinct packages which are all responsible for distinct areas of the system. The views package is responsible for the user interface, the domain package hosts the classes responsible for representing the physical entities of the system such as robots and processors and the persistance package is reponsible for saving and restoring the data in the system. By looking at the imports in the files in these packages it becomes clear the view package only imports and communicates with the domain package and the domain package only imports and communicates with the persistance package. |

Q2

|  |  |
| --- | --- |
| BEFORE | AFTER |
| **package** persistence;  **public** **class** EntityKeyGenerator {  // Q2. change this class to make EntityKeyGenerator a Singleton using the enum method  **private** **int** nextKey;    **public** **int** **getNextKey**() {  **return** ++nextKey;  }  } | **package** persistence;  **public** **enum** EntityKeyGenerator {  // Q2. change this class to make EntityKeyGenerator a Singleton using the enum method  ***PROCESSOR***;    **private** **int** nextKey;    **public** **synchronized** **int** **getNextKey**() {  **return** ++nextKey;  }  } |

Q3

|  |  |
| --- | --- |
| BEFORE | AFTER |
| **void** **addEntityListener**(EntityListener listener) {  listeners.add(listener);  }    **void** **removeEntityListener**(EntityListener listener) {  listeners.remove(listener);  }    **void** **fireEntityAdded**(Integer key, Object value) {  EntityEvent **event** = **new** EntityEvent(key, value);  // Q3 add code here to notify observers of the event  }    **void** **fireEntityRestored**() {  EntityEvent **event** = **new** EntityEvent();  // Q3 add code here to notify observers of the event  } | **void** **addEntityListener**(EntityListener listener) {  listeners.add(listener);  }    **void** **removeEntityListener**(EntityListener listener) {  listeners.remove(listener);  }    **void** **fireEntityAdded**(Integer key, Object value) {  EntityEvent **event** = **new** EntityEvent(key, value);  // Q3 add code here to notify observers of the event  **for** (EntityListener **listener** : listeners) {  listener.entityAdded(event);  }  }    **void** **fireEntityRestored**() {  EntityEvent **event** = **new** EntityEvent();  // Q3 add code here to notify observers of the event  **for** (EntityListener **listener** : listeners) {  listener.entityRestored(event);  }  } |

Q4

|  |
| --- |
| These cases are an example of the façade pattern. They are used to hide the internal structure of the packages from any outside accessors and also promote loose coupling. The singleton pattern is also used to provide a single point of access. |

Q5

|  |  |
| --- | --- |
| BEFORE | AFTER |
| **package** persistence;  **import** java.io.\*;  **public** **class** EntityCSVSave {    String **getFileSuffix**() {  **return** ".csv";  }    String **getFileName**(EntityTable table) {  **return** table.getClass().getSimpleName();  }      **void** **save**(EntityTable table) **throws** IOException {  // code to save table data in CSV format (omitted)  }      EntityTable **restore**(EntityTable table) **throws** IOException {  // code to restore table data from CSV format (omitted)  **return** table;  }  }  **package** persistence;  **import** java.io.\*;  **public** **class** EntitySerializationSave {    String **getFileSuffix**() {  **return** ".ser";  }    String **getFileName**(EntityTable table) {  **return** table.getClass().getSimpleName();  }    **void** **save**(EntityTable table) **throws** IOException {  File **file** = **new** File(getFileName(table) + getFileSuffix());  FileOutputStream **fos** = **new** FileOutputStream(file);  BufferedOutputStream **bos** = **new** BufferedOutputStream(fos);  ObjectOutputStream **oos** = **new** ObjectOutputStream(bos);  oos.writeObject(table);  oos.close();  }      EntityTable **restore**(EntityTable table) **throws** IOException {  File **file** = **new** File(getFileName(table) + getFileSuffix());  FileInputStream **fis** = **new** FileInputStream(file);  BufferedInputStream **bis** = **new** BufferedInputStream(fis);  ObjectInputStream **ois** = **new** ObjectInputStream(bis);  **try** {  table = (EntityTable) ois.readObject();  } **catch** (ClassNotFoundException **ex**) {  **throw** **new** IOException(ex);  }  ois.close();  **return** table;  }  } | **package** persistence;  **import** java.io.\*;  **public** **class** EntityCSVSave **extends** AbstractProcessorSave {    String **getFileSuffix**() {  **return** ".csv";  }    String **getFileName**(EntityTable table) {  **return** table.getClass().getSimpleName();  }      **void** **save**(EntityTable table) **throws** IOException {  // code to save table data in CSV format (omitted)  }      EntityTable **restore**(EntityTable table) **throws** IOException {  // code to restore table data from CSV format (omitted)  **return** table;  }  }  **package** persistence;  **import** java.io.\*;  **public** **class** EntitySerializationSave **extends** AbstractProcessorSave {    String **getFileSuffix**() {  **return** ".ser";  }    String **getFileName**(EntityTable table) {  **return** table.getClass().getSimpleName();  }    **void** **save**(EntityTable table) **throws** IOException {  File **file** = **new** File(getFileName(table) + getFileSuffix());  FileOutputStream **fos** = **new** FileOutputStream(file);  BufferedOutputStream **bos** = **new** BufferedOutputStream(fos);  ObjectOutputStream **oos** = **new** ObjectOutputStream(bos);  oos.writeObject(table);  oos.close();  }      EntityTable **restore**(EntityTable table) **throws** IOException {  File **file** = **new** File(getFileName(table) + getFileSuffix());  FileInputStream **fis** = **new** FileInputStream(file);  BufferedInputStream **bis** = **new** BufferedInputStream(fis);  ObjectInputStream **ois** = **new** ObjectInputStream(bis);  **try** {  table = (EntityTable) ois.readObject();  } **catch** (ClassNotFoundException **ex**) {  **throw** **new** IOException(ex);  }  ois.close();  **return** table;  }  } |

Q6

|  |
| --- |
| Inside both the save and restore methods a FileOutputStream is wrapped inside a BufferedOutputStream which in turn is wrapped inside an ObjectOutputStream. Each successive wrapping adds additional functionality to the object without requiring the need to subclass. This is an example of the decorator pattern. |

Q7

|  |  |
| --- | --- |
| BEFORE | AFTER |
| **package** domain;  **public** **class** ProcessorFactory {  **public** **enum** Type {***SINGLECORE***, ***MULTICORE***};    // Q7 factory code in here    **static** Processor **create**(String size, **boolean** multicore) {  **return** ProcessorFactory.*create*(multicore ? Type.***MULTICORE*** : Type.***SINGLECORE***, size);  }    **private** **ProcessorFactory**() {}  } | **package** domain;  **public** **class** ProcessorFactory {  **public** **enum** Type {***SINGLECORE***, ***MULTICORE***};    // Q7 factory code in here    **static** Processor **create**(String size, **boolean** multicore) {  **return** ProcessorFactory.*create*(multicore ? Type.***MULTICORE*** : Type.***SINGLECORE***, size);  }    **private** **ProcessorFactory**() {}    **public** **static** Processor **create** (Type processorType, String size) {  **if** (processorType == Type.***SINGLECORE***) {  **return** **new** SingleCoreProcessor(size);  }  **else** **if** (processorType == Type.***MULTICORE***) {  **return** **new** MultiCoreProcessor(size);  }    **return** **null**;  }  } |

Q8

|  |
| --- |
| Command. |

Q9

|  |
| --- |
| **package** domain;  **import** java.util.List;  **public** **class** CompositeRobot **implements** Robot {  **private** Processor processor;  **private** Robot.Colour colour;  **private** List<Robot> parts;    **public** **CompositeRobot**(Processor p, List<Robot> parts) {  **this**(p, Robot.Colour.***UNPAINTED***, parts);  }    **public** **CompositeRobot**(Processor p, Robot.Colour colour, List<Robot> parts) {  **this**.processor = p;  **this**.colour = colour;  **this**.parts = parts;  }  @Override  **public** Processor **getProcessor**() {  **return** processor;  }  @Override  **public** Robot.Colour **getColour**() {  **return** colour;  }  @Override  **public** **void** **paint**(Robot.Colour colour) {  **this**.colour = colour;  }    @Override  **public** String **toString**() {  **return** getClass().getSimpleName() + " (" + processor + ", " + colour + ")";  }    } |