ADVANCED JAVA PROGRAMMING AJP-P8-2012-2013

INTRODUCTION

This exercise will test your knowledge of the composite pattern.

SET-UP

Import the *Netbeans* project folder named 'AJP-P8-2012-2013-STUDENT'. Open it up. You have been given a set of unit tests. Initially, there will be LOTS of syntax errors in these unit tests. This is expected. The errors will disappear as you write the required classes.

INSTRUCTIONS FOR BRONZE TASK

You are working on a strategy game. In this game, users can create s*oldiers*. Soldiers can be added to *squads*. Large squads can be built by combining two or more smaller squads. Using the composite pattern, implement a solution that allows soldiers and squads of soldiers to be treated identically. Bear the following notes in mind:

- You must create an interface called *Deployable* with a single method *getStrength*(). This method returns an *int*.
- You must create a concrete implementation of *Deployable* called *Soldier.* When the *getStrength*() method of class Soldier is called, the method should return the value 1.
- You must create a concrete implementation of *Deployable* called *Squad.*
- The *Squad* class should declare an *ArrayList*
- The *Squad* class should have an *add*() method that allows you to add a *Deployable* object to its *ArrayList*
- The *Squad* class should have an *remove*() method that allows you to remove a *Deployable* object from its *ArrayList*
- *When the getStrength()* method of the *Squad* class is called, the method should iterate through all the *Deployable* objects in the *ArrayList*, calling *getStrength*() on each member of the list, and returning the sum.
- For example, if a *Squad* contains 4 *Soldiers*, the *getStrength*() method should return 4 when it is called.

Make sure that your solution to the BRONZE task does not have any *CheckStyle* errors. You cannot pass this task if your code contains *CheckStyle* errors

INSTRUCTIONS FOR SILVER TASK

In uk.ac.tees.username.silver, extend your solution to the BRONZE task in the following way:

- Create a concrete implementation of *Deployable* called *Grenadier.* When the *getStrength*() method of this class is called, the method should return 3.
- Create a concrete implementation of *Deployable* called *MachineGunner.* When the *getStrength*() method this class is called, the method should return 5.

- Change *Deployable* from an interface to an abstract class
- Add a new method to *Deployable* called *attack*(). The full signature of this method should be

```
boolean attack(Deployable deployable)
```

This method is invoked when one *Deployable* object attacks another. The outcome of the attack is decided by comparing the *strength* attributes of the two objects. The object with the highest strength attribute wins. Ties are resolved in favour of the attacking party.

For example, if a *Grenadier* (strength 3) attacks a *Soldier* (strength 1), the method should return true (successful attack). If a *Grenadier* (strength 3) attacks a *MachineGunner* (strength 5), the method should return false (failed attack). If Soldier x (strength 1) attacks *Soldier* y (strength 1), the method should return true (successful attack).

Make sure that your solution to the SILVER task does not have any *CheckStyle* errors. You cannot pass this task if your code contains *CheckStyle* errors

INSTRUCTIONS FOR GOLD TASK

This task involves material that we have not covered in class and will require research.

Furthermore, to spice up the medal race, I will be providing ***NO HELP*** *with this task.*

All *EasyFlap* planes have storage lockers. These lockers can be filled with various items of in-flight equipment. In this exercise you will design and implement a java class that describes these lockers. This class will use *generics*.

In the first step we will build a class library that describes the things we can put into a *Locker*.

- You must create an abstract class called *Gear*. This class has two instance variables. The first is an int which stores the *weight* of the item in kg. The second is a char which stores a *code* identifying the type of the item – (b)edding, (f)ood, (m)edical or (s)anitary products.
- Referring to *InfoTest*.java, develop an enum called *Info*.
- Implement a concrete class called *Blankets* that extends *Gear*. This class should be initialised using values taken from the enum *Info*.
- Implement a concrete class called *Food* that extends *Gear*. This class should be initialised using values taken from the enum *Info*.
- Implement a concrete class called *Medkit* that extends *Gear*. This class should be initialised using values taken from the enum *Info*.
- Implement a concrete class called *Napkins* that extends *Gear*. This class should be initialised using values taken from the enum *Info*.
- Implement a concrete class called *Oxygen* that extends *Gear*. This class should be initialised using values taken from the enum *Info*.
- Implement a concrete class called *Pillows* that extends *Gear*. This class should be initialised using values taken from the enum *Info*.

- Implement a concrete class called *SicknessBag* that extends *Gear*. This class should be initialised using values taken from the enum *Info*.

When *InfoTest* and *GearTest* are passing, we can move on to the next stage.

- You must create a class called *Locker*.
- The *Locker* class should be parameterized with a generic type T, where T is the type of item the locker contains
- The *Locker* class should have a generic method called *add*() that adds an item to the *Locker*.
- The Locker class should declare an *ArrayList* of type T that stores anything which is successfully added to the *Locker*
- Here are the rules to apply when adding items to a *Locker*
  - You cannot mix food and bedding
  - You cannot mix medicine and bedding
  - You cannot mix medicine and food
  - Sanitary products can be mixed with anything
  - You cannot exceed 20kg per locker

Make sure that your solution to the GOLD task does not have any *CheckStyle* errors. You cannot pass this task if your code contains *CheckStyle* errors.