Java Programming 2 – Lab Exam

This document is provided to allow you to prepare your solution to the JP2 lab exam in advance. You will sit the actual lab exam during your scheduled lab session during Week 11 (26/27 November), or during a different time by consultation with the teaching office.

During the lab exam, you will use Eclipse it its default configuration on one of the lab machines. Aside from the provided starter code, you will also have access to the complete Java 10 Javadoc in HTML, as well as all of the JP2 lecture slides (and to a paper language dictionary if appropriate).

Set up

Note that the instructions below are designed to be used while preparing your solution in the lab before the exam. During the exam itself, you will use a different process to create the initial Eclipse project to do your work in – the code during the exam will be identical to the code provided here.

1. Launch Eclipse as in previous labs (see the Laboratory 3 lab sheet for details)
2. In Eclipse, select **File** → **Import** … (Shortcut: **Alt-F, I**) to launch the import wizard, then choose **General** → **Existing Projects into Workspace** from the wizard and click **Next** (Shortcut: **Alt-N**).
3. Choose **Select archive file** (Shortcut: **Alt-A**), click **Browse** (Shortcut: **Alt-R**), go to the location where you downloaded `LabExam.zip`, and select that file to open.
4. You should now have one project listed in the Projects window: **LabExam**. Ensure that the checkboxes beside this project is selected (e.g., by pressing **Select All** (Shortcut: **Alt-S**) if it is not), and then press **Finish** (Shortcut: **Alt-F**).

Submission material[1]

In this lab, you will create a set of classes representing playable characters in a video game, as well as a class representing a player of that video game. The following are the main features that you will implement – the remainder of this lab sheet describes in more detail how the classes should be created.

Every **Game Character** has three properties: a **name** (e.g., "Superman"), a **cost** (i.e., the amount of in-game coins required to purchase that character), and a set of **powers** (e.g., for Superman, this might include Flying, Invincibility, Speed, and Energy Blast).

A **Player** of a video game has a set of game characters which they have already purchased, as well as a supply of in-game coins that can be used to purchase more game characters if necessary.

During the game, a player will encounter a series of levels – and each level requires a set of characters with particular powers. For example, one level might need characters with powers including Flight, Strength, and Science. To begin such a level, a player must choose a set of

[1] You will not be submitting this lab through Moodle. The lab exam software will also provide a mechanism for submitting your code – details will be provided as part of the exam pack on the day.

characters that supply all of the necessary powers. For example, if a level needs character with Flight, Strength, and Science, the player might choose Superman (to provide Flight and Strength) along with Doctor Octopus (to provide Science). If a player does not have characters that are able to supply the necessary powers, they can also use their coins to purchase additional characters to provide the remaining powers, as long as the cost is not too high.

Example

The player has 150 coins, along with the following characters in their set:

- Robin (providing Weapons)
- Starfire (providing Strength, Flight, and Energy Blast)

In addition, the following characters are available for purchase:

- Beast Boy (cost 150, providing Transformation)
- Cyborg (cost 200, providing Strength, Computer, Flight, and Weapons)
- Raven (cost 100, providing Magic)

The following are the game characters the player should choose in various circumstances

- If the required powers are Weapons and Strength, they should choose Robin and Starfire.
- If the required powers are Flight, Strength, and Transformation, they should purchase Beast Boy and then choose Starfire and Beast Boy.
- If the required powers are Transformation, Magic, and Flight, then this player cannot start the level as they do not have the required powers and cannot purchase enough characters to provide them.

Task 1: Power and GameCharacter

*Note about implementation: all classes created in this exam should be put in the **superhero** package.*

You should create an enumerated type **Power** with the following values:

> CLONING, COMPUTER, ENERGY_BLAST, FLIGHT, INVINCIBILITY, MAGIC, MAGNETISM, SCIENCE, SMALL, SPEED, STRENGTH, TELEPATHY, TRANSFORMATION, WEAPONS, WATER

You should then create a class **GameCharacter** representing a game character including the following properties:

- name (a String)
- cost (an integer)
- powers (a set of Power objects)

You should create a constructor to set the value of the above fields with the following signature (note that this constructor uses **varargs** for its final parameter):

> **public GameCharacter(String name, int cost, Power... powers)**

In addition, your **GameCharacter** class must satisfy the following requirements:

- Your class must be **immutable** – that is, it should not be possible to change any of the fields of the class in any way after it is created.
- The class must have a complete set of **get** methods for all parameters, but no **set** methods
    - Be sure that none of your **get** methods violate the immutability of your class
- You should provide appropriate implementations of **equals()**, **hashCode()**, and **toString()**.
    - It is fine to use the auto-generated versions of **equals()** and **hashCode()** if you want, but you must write your own (non-auto-generated) **toString()** method.
- Your class must implement **Comparable** and provide a **compareTo** method that sorts **GameCharacter** objects in increasing order of cost.
- You must write **descriptive Javadoc comments** for the class and for all class members (fields and methods).

The test cases in **TestGameCharacter.java** provide tests for all of the above functionality.

Task 2: Player

Once your **GameCharacter** class is finished, you should implement a **Player** class to represent a player of the game, making use of the **GameCharacter** class defined above. **Player** should have two fields:

- The number of coins that they have (an integer)
- The set of game characters that they currently own (represented as **GameCharacter** objects) – you can choose whatever data type you want for this field

Player should also have a constructor that initialises both of the above fields appropriately, as well as **get** methods for both fields (no **set** methods). The signature of the **get** methods should be:

- **public int getCoins()**
- **public Set<GameCharacter> getCharacters()**

You should then implement two additional public methods, as follows – the details of both methods are provided below:

- **public void buy(GameCharacter gc) throws IllegalArgumentException**
- **public Set<GameCharacter> chooseCharacters(Power... neededPowers)**

Note that you are free to add any number of **private** helper methods that you want, but the above should be the only **public** methods in the class other than the constructor and the **get** methods.

buy() method

The **buy()** method should behave as follows:

- If the player already has the given character, throw an **IllegalArgumentException** with an appropriate message
- If the player does not have enough coins to buy the given character, throw an **IllegalArgumentException** with an appropriate message

- Otherwise, add the given character to the player's set and deduct the cost from their total number of coins

chooseCharacters() method

This method should implement the behaviours shown above under the example. In particular, given a set of powers, this should return the set of characters required to provide all of the requested powers, or **null** if it is not possible to provide the powers. If providing the requested powers required buying more characters, those characters should be bought using the **buy()** method.

The suggested behaviour for this method is as follows:

1. Go through the characters that the player already has and choose those that satisfy at least one of the required powers.
2. If the selected set of characters covers all of the required powers, return that set from the method and do not go any further.
3. Otherwise, use **GameCharacters.getAllCharacters()** (provided) to obtain the full list of characters. Sort this list in increasing order of cost, and then go through the list one-at-a-time as follows:[2]
    a. If the user already has that character, ignore it
    b. If the character does not provide a power that we still need, ignore it
    c. If the character cannot be bought with the available coins, ignore it
    d. Otherwise, store the character as one that might be bought and update the set of powers that we could obtain
4. At the end, if we have found a set of characters which can cover the remaining required powers:
    a. Buy all of the new characters
    b. Return the full set of characters to fulfil the required powers
5. If no set of characters can be found to cover all of the required powers, this method should return **null.**

Important note: **the Player should not buy any characters until it is clear that buying those characters will lead to a solution of the problem!** That is, in the loop above (3a—d), no calls to **buy()** should be included – you should only call **buy()** after the loop.

The test cases in **TestPlayer.java** provide tests for all of the above functionality.

Hints and tips

In general, this lab exam gives you more freedom to design your own classes than on the weekly lab assignments: as long as the required behaviour is implemented, you are free to include any additional **private** methods in any class as long as it meets the specification.

In addition to the JUnit tests, you can also use the provided **SuperheroMain** method to test the functionality of your classes – feel free to modify that method to test other aspects of your program.

[2] Note that this "greedy" process will not necessarily find a solution to all possible sets of powers and characters. **This is all right** – the test cases will check only that your solution is valid and will not require any more complex reasoning.

The output of the initial version of that method on my implementation is shown below – note that this is not the only possible result for this problem.

Sample output

The full output from **SuperheroMain** on my solution is given below:

```
Player coins at start: 150
Player characters at start:
Robin (100): [WEAPONS]
Starfire (200): [STRENGTH, ENERGY_BLAST, FLIGHT]
Cyborg (150): [STRENGTH, COMPUTER, FLIGHT, WEAPONS]
Beast Boy (150): [TRANSFORMATION]
Raven (100): [MAGIC]

Chosen character(s):
Cyborg (150): [STRENGTH, COMPUTER, FLIGHT, WEAPONS]
Aqualad (100): [WATER, TELEPATHY]
Raven (100): [MAGIC]
Gizmo (50): [SMALL, SCIENCE]

Player coins now: 0
Player characters now:
Robin (100): [WEAPONS]
Starfire (200): [STRENGTH, ENERGY_BLAST, FLIGHT]
Cyborg (150): [STRENGTH, COMPUTER, FLIGHT, WEAPONS]
Beast Boy (150): [TRANSFORMATION]
Aqualad (100): [WATER, TELEPATHY]
Raven (100): [MAGIC]
Gizmo (50): [SMALL, SCIENCE]
```

Indicative mark sheet

This lab exam will be marked out of 20. The following summary shows how those 20 marks will be allocated.

- 2 marks: Correctness of **Power** enumeration
- 6 marks: Correctness of **GameCharacter** class
- 3 marks: Correctness of **Player** class (except for **chooseCharacters()** method)
- 7 marks: Correctness of **chooseCharacters()** implementation
- 2 marks: Overall quality of code and presentation