CSSE2002/7023 Assignment 2

Due: Friday, Sep. 29 15%

Extend either your solution or the provided solution for Assignment 1 to include exception-handling and file I/O. Only the classes *Character* and *SuperCharacter* will be used in this assignment. The goal of this assignment is to provide a database of characters which can be searched and updated, and is stored in a file. Here, the term database simply means a data object stored in a file. Revise the lecture slides on reading objects from files and writing objects to files.

1 Solving the illegal power ranking problem

Exceptions are a better way to solve the problem of illegal power rankings for super characters.

1. Create an exception class *IllegalPowerRankingException*. A minimal class along the lines of those in the lectures notes is all that is required.
2. Have your *SuperCharacter* class throw said exceptions when an illegal power ranking value is given for creating or modifying a super character.

2 Restricting exposure and cloning

The reason class *Character* does not have accessor and mutator methods for the *traits* attribute is because that would expose the protected *traits* attribute externally to class *Character*, thus violating its required protection and access only through the desired *add* and *remove* interface. The same holds for the *powers* attribute of class *SuperCharacter*.

Again for the same reason, when searching the database for a character object by name it is not acceptable to simply return the object when it is found, because the object reference so returned would allow the character object in the database to be changed arbitrarily.

The solution to this is to return a complete copy, or **clone**, of the object in the database. Cloning is a standard paradigm for this. The process for cloning an object is to declare a new object of the same type and copy the values of the attributes into the new object. To make a *deep* copy (which is necessary for this assignment), any attribute of an object that is itself an object must be cloned. All objects in Java support cloning, and classes that require specific cloning strategies override the *clone* method.

The signature for the clone method **must** be: **public** Object clone()

Add this method to class *Character*:

```
/**
 * @return a deep copy clone
 */
public Object clone() {
    Character clone = new Character(name, description);

    // !! Note: the clone() method of the traits object CANNOT be used here
    //          traits are added individually to the clone
    for (String t : traits)
        clone.traits.add(t);

    return clone;
}
```

Make sure you fully understand this code before adding it to class *Character*!

Assignment task: Write a *clone* method for class *SuperCharacter*, overriding the inherited method.

3 Character database

Create a class *CharacterDatabase* to represent a collection of *Character* objects. A character database is a *HashSet* of *Character* objects, stored in a file.

Implement these methods for the character database:

Constructor: The constructor takes a String parameter, which is the name of the file where the database is stored.

load: Read the *HashSet* database from the file. If the file does not exist, throw a *FileNotFoundException* exception.

save: Write the *HashSet* database to the file

add: Add a character to the database

remove: Remove a character from the database

search: Search for a character in the database by *name*. If found, return a **clone** of the character, otherwise return **null**

I/O exceptions may be raised during execution of *load* and *save*. Handle them appropriately: if there is some sensible behaviour that lets the program keep running, do it, otherwise terminate the program using the *printStackTrace* method inherited from class *Exception*.

4 Specification and Validation

Provide, in comments, a specification for the behaviour of the search method. Write a JUnit test case that performs white-box testing with path coverage of the search method. You may, of course, use JUnit to construct other test cases. Note that *HashSet* offers no guarantee of the order in which elements are stored. Factor this into your path coverage testing.

5 Implementation Details

Add these two methods to class *Character*:

```
public boolean equals(Object other) {
    try {
        Character c = (Character) other;
        return name.equals(c.getName());
    }
    catch (ClassCastException cce) {
        return false;
    }
}

public int hashCode( ) {
    return name.hashCode( );
}
```

Style and Marking

As for Assignment 1. Only the *clone* method in *SuperCharacter*, the *CharacterDatabase* class, and the testing described above will be assessed.

Notes

Some aspects of this assignment are not fully specified. This is largely a learn-by-doing exercise. Try things out, see what happens, and learn.