CSC 171 LAB 06

Lab Instructions

The goal of this lab is to give you an opportunity to practice with packages, arrays, multidimensional arrays, loops, lists, and inheritance.

For each section below write an appropriate test of some kind. You may do this in any way that you think is appropriate, but remember that some exercises require you to test the same class or classes as a previous exercise, so a single **`main`** method may not be appropriate.

****PLEASE NOTE**** that many of the exercises below require you to implement a **`toString`** method. A **`toString`** method creates and returns a String. It ***does not*** print anything to the command line. In other words, you should not call **`System.out.println`** in your **`toString`** method. Instead, you should write test methods that call your **`toString`** method and print the **`String`** that it returns.

1. Create a Java class called "Product" in a package called "csc171.lab06.products". A product has a **`String`** name, a **`double`** price, and a **`String`** serial number. Your Product should have a **`protected`** constructor that requires all three values to be provided as parameters. For clarity, your parameters should use the same name as your instance variables (e.g. both the parameter and instance variable should be named "**`serialNumber`**"). Use the **`this`** reference to set your instance variables to the parameter values. Write appropriate accessors and mutators for each of the instance variables, again using the **`this`** reference when appropriate.

2. Use the **`new`** operator to create a new Product and pass it as a parameter to **`System.out.println`** to print it to the console. Notice that the output looks something like this:

   ```
   csc171.lab06.products.Product@15db9742
   ```

   The **`java.lang.Object`** class provides a method called **`toString`** that returns a String representation of the Object. This method is called *automatically* in many cases, including when you add your object to a **`String`** using the concatenation operator, and when you pass your object as a parameter to **`System.out.println`**. The default implementation returns **`Strings`** that look like the output above, which is not very useful most of the time. **Override** the **`toString`** method by creating your own version in the Product class with this signature: **`public String toString()`**. You may find it useful to use the **`@Override`** annotation (as discussed in class) as this will perform some error checking for you. Your method should return a more useful description of your Product including the name, current price, and serial number. Now, if you print your Product, it should produce output that looks something like this:

```
LEGO Playset(54321), $24.99
```

3. Create a new Java class called “Toy” in the “csc171.lab06.products” package. A Toy *is a* Product, and should *inherit* the state and behavior defined by the Product class. In addition, toys also have an age range that indicates the appropriate age of children who might play with the toy. Your toy class should have two instance variables that indicate the minimum age and maximum age in the age range. Implement a `public` constructor for the Toy class, and the appropriate accessors and mutators for the age range. Remember that Product does not have a parameterless constructor, so you will need to use the `super` reference to call the constructor on Product. This means that your Toy’s constructor will take a total of 5 parameters (name, serial number, price, minimum age, and maximum age) and pass 3 of them to the parent constructor.

4. Create a new Java class called “Book” in the “csc171.lab06.products” package. A Book *is a* Product, and should *inherit* the state and behavior defined in the Product class. In addition, a Book also has an author, and a number of pages. Choose appropriate data types for each and add them to the Book class along with a `public` constructor and the appropriate accessors and mutators. Like the Toy class’s constructor, the Book constructor will need to take additional parameters and will need to use the `super` reference to call the constructor on the parent class, Product.

5. Implement a `toString` method in the Toy and Book classes. Do NOT copy and paste or rewrite the code from your Product class. Remember that a main advantage of inheritance is *code reuse*, so use the `super` reference to call the `toString` method on the superclass, and then add additional information to the end of the `String` that it returns to describe the Toy or Book. The Strings returned by Toy and Book should look something like this:

   ```
   LEGO Playset(54321), $24.99, suitable for ages 3 to 9
   The Name of the Wind(12345), $5.99, by P Rothfuss, 800 pages
   ```

6. Create a Java class called “Warehouse” in the “csc171.lab06.warehouse” package (note that this is ***not*** the same package that was used in the previous examples). The Warehouse class should have two strongly typed `ArrayList` instance variables, one each for Toys and Books. Remember that you make an `ArrayList` strongly typed by putting a type in angle brackets, e.g. `ArrayList<Book>`. You will then only be able to add objects of that type to the list. Because these classes are in a different package, you should use `import` statements to access them from your Warehouse. The class should also have methods to add Toys or Books one at a time, and two accessors to get the `ArrayList` that contains all of the Toys or Books (e.g. `public ArrayList<Book> getBooks()`). As an experiment in your test, import the Product class and try to use the `new` operator to create an instance of the class. Describe what happens and why you think it happened in your README.

7. Implement a **`toString`** method in your Warehouse class that creates a **`String`** by using a **`for each`** loop over each of your lists. Remember that when you use the **`String`** concatenation operator (“+”) to add your objects to a **`String`**, the **`toString`** method that you wrote will be called *automatically*. For example:

   ```
   String example = “testing” + toy + “123”;
   ```

   Is functionally equivalent to:

   ```
   String example = “testing” + toy.toString() + “123”;
   ```

   The **`String`** that you build should use one line for each product (remember that you can use the escape sequence “\n” to start a new line in a String). Note: your **`toString`** method should **not** print anything out. It simply makes a String and returns it. When the **`toString`** method on your Warehouse is called, it should produce a **`String`** similar to this:

   ```
   Warehouse
   Toys
     LEGO Playset(54321), $24.99, for ages 3 to 9
     Star Wars Figure R2D2(65432), $8.99, for ages 8 to 999
     TMNT Raphael(76543), $8.99, for ages 8 to 999
   Books
     The Name of the Wind(12345), $5.99, by P Rothfuss, 800 pages
     Under the Dome(23456), $25.99, by S King, 1800 pages
     Cryptonomicon(34567), $15.99, by N Stephenson, 2800 pages
   ```

8. Create a Java class called “Square” in the package “csc171.lab06.checkers”. A Square has a color, which is either “Black” or “Red”, and may or may not be occupied by a checker (you can indicate this with a boolean that is **true** if there is a checker on the square and **false** otherwise). Write the appropriate constructor, accessors, and mutators. Write a **`toString`** method that returns a String with a pipe (“|”), the first capital letter of the color, an asterisk (“*”) if the square is occupied by a checker and a space (“ “) if it is not, and finally another pipe “|”). For example, an occupied red square would create a String like this: **`“|R*|”`**, while an unoccupied black square would create a string like this: **`“|B |”`**.

9. Create a class “Checkerboard” in the “csc171.lab06.checkers” package that has a two dimensional array of Squares as an instance variable. Create a constructor that accepts two integer parameters: one for a number of rows, and one for a number of columns. Use these parameters to initialize the two dimensional array of Squares. In the constructor use nested loops to populate your two dimensional array with **`new`** Squares. The black squares in the first three rows and the last three rows should contain checkers.

10. Implement two methods in your Checkerboard class. The first is a **`private`** method that converts a one dimensional array to a **`String`**. The signature should look like this:

    ```
    private String rowToString( Square[] row )
    ```

    Then implement a **`toString`** method that builds a **`String`** version of your checkerboard by using the **`rowToString`** method on each row of checkers. Recall that, given **`Square[][] squares`** (a two dimensional array of Squares), **`squares[i]`** will return the **`i`**th row. The output of your test method should look something like this:

    ```
    |B*||R ||B*||R ||B*||R ||B*||R |
    |R ||B*||R ||B*||R ||B*||R ||B*|
    |B*||R ||B*||R ||B*||R ||B*||R |
    |R ||B ||R ||B ||R ||B ||R ||B |
    |B ||R ||B ||R ||B ||R ||B ||R |
    |R ||B*||R ||B*||R ||B*||R ||B*|
    |B*||R ||B*||R ||B*||R ||B*||R |
    |R ||B*||R ||B*||R ||B*||R ||B*|
    ```

HAND IN

Before handing in, create two additional files in your lab directory:

1. Create a README that contains the following:
    a. Your contact information (name, class, lab session), TA name, and assignment number.
    b. A brief (one paragraph at most) description of the assignment.
    c. Instructions explaining how to run your code. While it is possible that some TAs will compile, run, and test your code in Eclipse, it is also possible that they will want to compile and run it from the command line. You should include instructions on how to compile and run each of the executable Java classes.
2. Create a file titled “SampleOutput” showing the results of running your code. (You may copy and paste from the Eclipse console). If appropriate, please include a comment above the output for each section (e.g. “#OUTPUT FOR SECTION 1”) and put a few blank lines between each section.

Hand in by uploading the compressed (i.e. “zipped”) folder containing your assignment files to Blackboard.