Instructions

Please read the instructions carefully and follow the naming conventions specified for each question. Solutions must be submitted in the Blackboard Dropbox created for Lab 2.

The deliverable should placed a package named *btp400.lab2* (This assignment has other packages nested inside it). The submission shall be in a single jar file (called *btp400lab2.jar*) which contains both source (*.java) and bytecode (*class) files. Your solution should be well documented using the JavaDoc utility to describe both your interface and your solution design.

Note that the deadline is strictly enforced. The system tracks the exact time that submissions are uploaded. **There is a 10% per day penalty for late submissions.**

Additional Notes

- You must provide appropriate test case classes with your solution. Each test class should be in the same package as the class it is testing. Document any assumptions you make about the requirements. You will have to organize your code in a way such that the JUnit test runner does not require user input.
- You will be required to present and explain your solution to the professor during the lab period.
- You may use any IDE for development but note that demonstrations and professor testing will be done exclusively on the command line.

Question 1) Define a class called *Colour* which stores a colour found in the RGB system found in most graphics formats (`https://www.w3schools.com/colors/colors_rgb.asp`). The constructor should take in the red, blue and green components as parameters. For this lab, we assume an 8-bit system where each component ranges from 0-255. Define *toString, equals and hashCode* methods for these classes. Use conventions noted in the course text (Section 4.2). Ensure that your class is to be placed in a package called *btp400.lab2.misc*.

Question 2) Define the following classes with the described properties (the instance variables used to represent them up to you)

- *Toy*, shall store a name and an age range that it is suitable for. Toys are all suitable for children.
- *Coat*, shall store a name, colour and an a size. Possible sizes are (Children's small/medium/large and Adult's small/medium/large)
- *Pump*, shall store a name, colour and a max psi (pounds per square inch). PSI pumps here do not exceed 90 psi. Pumps only come in black or white colours.

The classes should have constructors to initialize the instance variables chosen and accessor methods for the caller to retrieve them. Override *toString, equals and hashCode* methods for these classes. Use conventions noted in the course text (Section 4.2). Invalid input should be handled appropriately. The classes should be placed in a package called *btp400.lab2.items*

Question 3) Define an interface called *Taggable* which contains the following method signatures

- **void** *generateTag*(); produces a sequence of digits, based on your instance variables.
- **boolean** *isChildTag*();, identifies whether object is intended to be for children
- *String getTag*(); retrieve tag generated

Place the interface in package named *btp400.lab2.tag*

Question 4) Create concrete subclasses for each class specified in Question 2. Each subclass shall implement the *Taggable* interface. Ensure that your implementation preserves the class descriptions. Modify the *toString, equals, hashCode and clone* implementation if needed. Place the classes in package *btp400.lab2.tag*

Question 5) Write a class called *UserApp* which demonstrates the ability to store objects of varying types from Question 4 in a single array. This class is to be placed in a package called *btp400.lab2.usr*. It shall provide methods to do the following.

- retrieve the children-appropriate objects in a given array.
- retrieve the tags of all objects in a given array

Question 6) Provide an alternative implementation for your solution coded in Question 1-4. In this version, you will remove the *Taggable* interface and use an abstract class to achieve the same result. Place the classes in a new package *btp400.lab2.abscls*. **(Note that unit test cases are not required for this question)**. In a separate file (or within your JavaDoc comments), describe how your implementation changed? What are the benefits/drawback of each approach? Which approach do you think is most appropriate for this use case?

Question 7) Write a program in a class called *UserObjectTest* which demonstrates the ability query object information and invoke methods. Use a class created in Question 6. This class is to be placed in a package called *btp400.lab2.usr*. It shall do the following.

- create an object of the class selected (without using the usual "*Type t* = **new** *Type*()" style of statement)
- print the object's tag. (this must be done using an *Object* reference)