Software Dev. & Problem Solving II CSEC/SWEN-124

Abstract Classes Assignment 3.2

Goals of the Assignment

The goal of this assignment is to implement a simple class hierarchy that makes use of inheritance including abstract classes. As always, you are expected to practice good software engineering, including unit testing and practicing the Git workflow. Please read this document ***in its entirety*** before asking for help from the course staff.

Background

All ***toys*** have certain common characteristics including a ***unique product code*** that is at least 7 digits (e.g. 1000000, 2543734, etc.), a ***name*** (e.g. "Robot", "Doll", etc.), and a ***manufacturer's suggested retail price*** (***MSRP***). A child may ***play*** with any toy, but each toy exhibits unique behavior during play. There are circumstances under which a toy can no longer be played with, e.g. it is broken or its batteries are depleted.

There are several different kinds of toys, each of which has its own unique attributes and/or behavior, which are described in the table below.

Toy Type	Attributes	Behavior
All Toys	7-digit **product code** **name** **MSRP**	**play**: all toys can be played with, but every toy behaves differently during play.
Robot	**charge** (0-100%, starts at 0) **sound** (your choice). A robot always plays the same sound, but other robots may play different sounds.	The robot may be **charged** by plugging it into a power source. **play**: Walks in circles and plays its sound. Its charge is depleted by 20%. If the charge is totally depleted, the robot cannot play. If there is at least some charge, the robot will play.
Doll	**hair color** (your choice) **eye color** (your choice) **sayings** (your choice)	**play**: Features a pull string that plays one of a set of pre-recorded phrases randomly. Breaks after 10 plays.
Action Figure	**hair color** (your choice) **eye color** (your choice) **sayings** (your choice) May have **Kung-Fu Grip™**	**play**: Plays one of a set of pre-recorded *action phrases* randomly. If equipped with Kung-Fu Grip™, also brandishes a sword. Breaks after 10 plays.

Activities

1. Create a package called "`toys`" and, unless otherwise instructed, use it for the artifacts required for this assignment.

2. Use the table above to create a draft UML diagram of the classes that you think you will need. It does not have to be complete or perfect; a decent attempt will suffice. You ***must*** have at least one `abstract` class and several inheritance relationships between classes. ***Hint***: Your diagram should have *at least* 4-5 classes in it when you are done. Tool options include:
    a. Tables in a Google Slide (screenshot)
    b. Lucidchart or another UML Drawing Tool (screenshot or PDF)

   c. Hand-drawn on paper or a whiteboard ***legibly*** and photographed (though a paper drawing will be harder to update in the next assignment)

   You must ***commit*** your UML class diagram to your repository ***before*** moving to the next step. Use a comment like "initial UML class diagram" to make it easy for your grader to find it in your commit history. You do not need to revisit the diagram if you make changes to your design afterwards, but it should show obvious effort.

3. Implement the classes in your UML diagram. If you have to change the classes so that they are different from the diagram, that's OK. Don't worry about updating it. Here are some things to consider when implementing your classes:
   a. Product codes should be *guaranteed to be unique* and automatically assigned to each toy as it is created. They must be at least 7-digits. ***Hint***: 1000000 is the smallest 7-digit number.
   b. The name of a specific toy is the same as it's type, e.g. `"Doll"` for Dolls.
   c. Wherever the requirements say *"your choice"* it's up to you, but there should be several different options that are unique to the class.
   d. Every class should have a `toString()` method that can be used to print a nicely formatted string representation of the object.
   e. Consider whether or not to make fields in each class `final` - once a value has been assigned, should it ever change?
   f. Some classes may need more than one constructor!
   g. DRY - *don't repeat yourself*. If you find that two or more of your classes have duplicate code, consider whether or not one class should subclass the other, or if you may need a common parent class.

4. You should write JUnit tests for at least the toy classes (`Robot`, `Doll`, and `ActionFigure`). Write at least one test for each non-trivial method; do not worry about testing accessors, mutators, or `toString()` methods (though you may *use* them in your other tests).

Submission Instructions

You must ensure that your solution to this assignment is pushed to GitHub *before* the start of the next lecture period. See the course syllabus for the rubric that will be used to evaluate your submission.