Part II: PROGRAMMING

PROBLEM:
The Ridge Fluffy Dev (RFD) Corporation is one of the world's leading manufacturers of widgets. More precisely they make `Sonics`, `FloofyLops`, `Critters`, `SootBalls`, `DooToros` and `CatBus`. The RFD Corporation is very good at what they do, but even they have their shortcomings. Their current data management system is haywire. It consists of a ballpoint pen and a pile of faded napkins that they steal from the diner next door. The founder of RFD, Fernsby Fernsworth, has decided that it is time to get the company's act together and end the current delinquent behaviour. Fernsby Fernsworth is 103 years old and he would like to retire and concentrate on his golf game like many of his friends at his age. He, therefore, has decided passing the company to his 18 year-old grandson Cerny Fernsworth. Cerny is very tech savvy and has received an iPad for his 18th birthday. He has also decided that he wants to hire you - yes YOU - to implement his new high-tech, object oriented Ridge Fluffy Dev widget management system. Woo hoo!!!

Your task is to convert Cerny's system specifications into a clean, well-structured Java or C++ framework. Cerny's directions are as follows.

1. The most basic entity at RFD is the `Widget`. `Widgets` are not products themselves and hence, you cannot instantiate a `Widget` class directly. However, subclasses may be derived from a base `Widget` class.

   Any instance of a `Widget` class has a name (a `string`) and methods for setting and retrieving that name (which may also be possible to set when a subclass is instantiated).

   All subclass of `Widget` must provide a `calculatePrice()` method that returns a `double`, the selling price of the `widget`.

   Cerny would like you to mek it possible to print any `widget` type object by simply passing it to the `cout` in C++ or `System.out.println()` method. At the very least, this would produce output that looks like as follows:

   **** Ridge Fluffy Dev ****
   Teto Tatter: 67.0

   where "Teto Tatter" is the user-defined name of the particular widget (not the name of the class itself) and 67.0 is the price (formatting of decimal positions not so important here).

2. There are two types of `Widget` classes designed at RFD, those for children and those for adults. `Widgets` designed for children are called `DooToros`. Similar to the `Widget` class, there are no actual products called `DooToros`. Hence, `DooToros` act as a template for other types of objects.

   Since, `DooToros` are for children, a minimum age, `minAge`, is always associated with **any** type of `DooToros`. Any `DooToros` subclass must provide methods for setting and retrieving this integer value.

   Aside from this, `DooToros` classes calculate the price in quite a specific way. To be more precise, Cerny - a budding and somewhat confused mathematician - would like to compute the `DooToros` price as: $(|minAge| * 8 * 3x) \mod 100$, where

   - x is the number of characters in the `DooToros`'s user-defined name
   - $\mod$ is the modulus operatior (remainder of the division)
   - 8 is an "age factor" that Cerny thinks is useful.

   In addition, any `DooToros` object can be created in three different ways.

   (a) It can simply be given a name. The minimum age in this case must be set to a default value of 2.
   (b) It can be initialized with just a minimum age value. The name in this case will be initialized to "RDF DooToros".
   (c) It can be given both a name and a minimum age.

   Note the default minimum age value 2 and the age vactor value 8 may very well change in the future.

3. There are three types of `DooToros` products. Two of them are direct descendants of the `DooToros` class. One is called the `CatBus` and the other is `FloofyLop`.

   The `FloofyLop` are interesting because they can move by themselves. Other `widgets` have this feature too (explained in more detail below). Any kind of moveable `widget` provides a pair of methods.

This `Locomote` **interface** consists of:

```
public int howFar()
public String style()
```

The `FloofyLop` widget currently moves 5 metres and hence, the `FloofyLop howFar()` method should return 5, though this value is subject to change. The `FloofyLop style()` method should return "Roll".

`FloofyLop`s have two ways to calculate price:

(a) they calculate it just like any `DooToros` object.

(b) `calculatePrice()` accepts an integer argument that is then multiplied by `howFar()` value.

At last, but not least, when `FloofyLop` object is printed by `cout` in C++ or `System.out.println()` in Java, it should add a linke to the default `widget` printout that says "For All Ages".

4. The `CatBus` widget is also a moveable product and should have the `howFar()` and `style()` methods. Its moveable distance is a paramater to be provided by the user. Its locomotion style is "Saunter".

   Interestingly, the `CatBus` can also speak. This is a feature that is shared with other RDF widgets. Any speaking widget must implement two methos in its `Talker` **interface**:

   ```
   public String sayHi()
   public String sayBye()
   ```

   A `CatBus` says Hi by returning "Hi" and it says bye by returning "Bye Bye".

   A `CatBus` widget can be created in two ways.

   (a) It can be created with a name and a distance. If the distance is less than some minimum value - currently set to 5- then the minimum age for this widget is set to the default for `DooToros`. Otherwise, the minimum age is set to -1.

   (b) It can be created by providing a name, distance, and a minimum age. In this case, the values are recorded directly and no adjustment of the distance is required.

   When printing a `CatBus` object, the output is modified by adding a line defining the appropriate age for the widget. If the stored minimum age is -1, then we add "Age: unrestricted" to the regular `Widget` printout. Otherwise, we add "Age: `minAge` and above", where `minAge` is the stored minimum age value.

5. A `SootBall` is a special type of `CatBus` and is the third `DooToros` product. It can be created with a name and minimum age value. It has a `style()` method called "Jump".

   It also has its own `calculatePrice()` method. A `CatBus` is cheaper in price than a `SootBall`. There is an additional fixed cost that is currently 4.15 (subject to change). If the cost of the `SootBall` widget is greater than 100, then the price of a `SootBall` is equal to a default price of 9.99 (subject to change). If the calculation is less than 100, then we simply return this value as the price.

6. Everything up to now was dealing with widgets for children. Now for the adult widgets. The `Critter` is a subclass of `Widget`. Unlike `DooToros`, `Critters` are real products and can be instantiated directly. To create a `Critter` object, a name and a manufacturing cost (a `double`) is provided.

   The `calculatePrice()` method returns that manufacturing price plus a standard "mark up" value - currently set at 25.99.

   The `Critter` object also provides a `calculateSalePrice()` method that returns a sale price. It is calculated as the regular price for a `Critter` $+20\% - 10\%$.

7. A `Sonic` is a special type of `Critter`. Like `CatBus`, it is a talking widget. It says "Hi" by returning "Good day fellow traveler" and says goodbye by returning "Have a nice day".

   A `Sonic` can be created in two ways:

   (a) by providing a name and a manufacturing cost (a `double`)

   (b) by providing no parameters. In this case the `Sonic` will be called "Sonic Toy" and will be given a default manufacturing cost of 99.99.

The `Sonic` provides a `calculatePrice` method in two ways:

(a) by taking a markup value (a `double`) as an argument. This value is simply added to the standard `calculatePrice` value for a `Critter`.

(b) by taking no arguments. If the manufacturing cost is less than a pre-defined cutoff (currently defined as 25.99), then price is determined by calling the the other `calculatePrice` method of `Sonic` with a default markup of 5.55 (subject to change in the future). Otherwise, we just use the standard `calculatePrice` method of `Critter` type.

When a `Sonic` object is printed it should add a line to the regular `Widget` printout. This extra line consists of its `sayHi()` output.

****

So, this is the task that Cerny has laid out for you. **Remember that this system should be build to last.** In other words, by simply providing the right results for the calculations is not enough. You must use good design and programming practices. You will be graded on the quality of the code you write.

Good Luck!