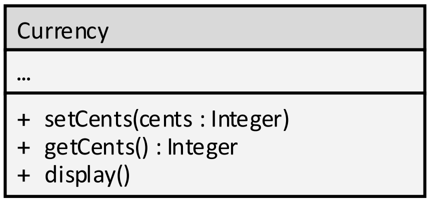
CSE 230 Problem Set 07

Problem 23.1: Currency

Consider the following scenario:

> A financial package uses a `Currency` class to manage money. This class cannot have rounding errors so, for fidelity reasons, uses an integer as the internal representation. Note that the software only works with dollars and cents, does not need to make coin change or anything like that. The software would add a dollar to a total, or find the difference between two values.

Consider the following class diagram



Classify the level of convenience for this class. Justify your answer.

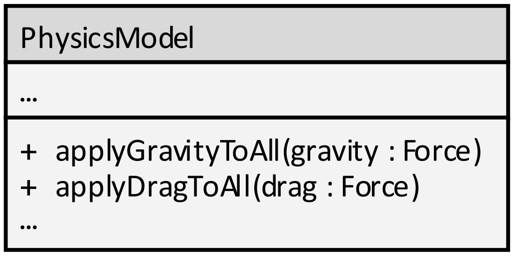
We would classify the level of convenience as convoluted. We would like to note that we interpreted the use of integers to mean that instead of representing 1 dollar as 1.00, it would be represented as 100 cents, thus an integer. However, we don't believe there is any way to verify that the input parameter is of type int because one could potentially pass a double into the method, and it be automatically converted to an integer, and you would lose data. Another issue that the class has no method that finds the difference between two values.

Problem 23.2: Physics Model

Consider the following scenario:

> A video game uses one class to manage all applications of physics. Everything in the game uses a `Force` object to monitor force: force due to gravity, force due to thrusting, force due to drag, and force due to collisions. Note that the `Force` class is used throughout the project and we can assume that it is fully flushed out and has seamless convenience.

Consider the following class diagram:


PhysicsModel
…
+ applyGravityToAll(gravity : Force)
+ applyDragToAll(drag : Force)
…

Classify the level of convenience for this class. Justify your answer.

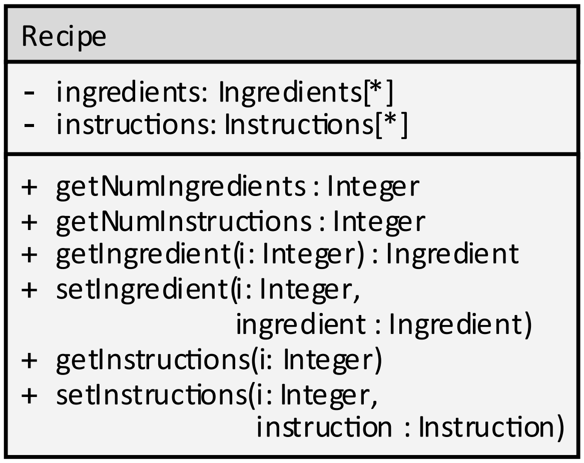
We would classify the convenience level as straightforward. We are a bit confused as to why different named parameters are being used when the Force class contains the all the forces of physics. We also believe a better way of designing the class would be to have the Physics Model have a Force object as an attribute and remove the need for parameters if Force is only to be used inside of this class which is what we concluded from the first line in the description. We also assume that all the objects that the physics are being applied to would be listed in the attributes. If not, the methods would need to take them as parameters. Overall we believe the class can be designed more efficiently and but is still able to complete its designed task.

Problem 23.3: Recipe I

Consider the following scenario:

> A recipe program has a scaling option where the user can specify the number of servings needed for a given meal. For example, you should be able to scale a taco recipe originally designed for three people to accommodate a party of seven.

Consider the following class diagram:



```
Recipe
- ingredients: Ingredients[*]
- instructions: Instructions[*]
+ getNumIngredients : Integer
+ getNumInstructions : Integer
+ getIngredient(i: Integer) : Ingredient
+ setIngredient(i: Integer,
                ingredient : Ingredient)
+ getInstructions(i: Integer)
+ setInstructions(i: Integer,
                  instruction : Instruction)
```

Classify the level of convenience for this class. Justify your answer.

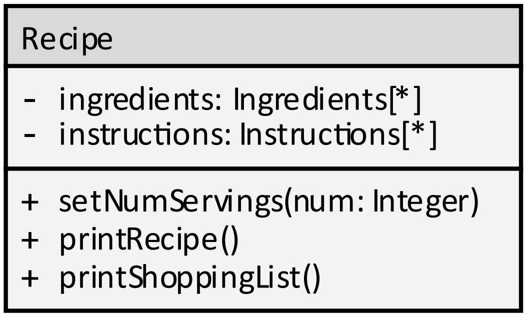
We would classify the convenience as prohibitive. We interpret the attributes as being lists of class objects. If the Ingredients or Instructions class contains an attribute that shows the amount needed for each ingredient then one could potentially use the getter and setters to manually scale the amounts for each ingredient. However, this requires far more work than if we were to completely redesign the class.

Problem 23.4: Recipe II

Consider the following scenario:

> A recipe program has a scaling option where the user can specify the number of servings needed for a given meal. For example, you should be able to scale a taco recipe originally designed for three people to accommodate a party of seven.

Consider the following class diagram:


Recipe
- ingredients: Ingredients[*]
- instructions: Instructions[*]
+ setNumServings(num: Integer)
+ printRecipe()
+ printShoppingList()

Classify the level of convenience for this class. Justify your answer.

We would classify the convenience of this class as seamless. The class contains attributes and methods related to its purpose plus additional features such as displaying the price and shopping list relative to the scaled amount.