Lab 4 - Block Factories
CS 5004 Object Oriented Design

1. Goals:

- Create and logically use an enum
- Explore the use of abstraction
- Use inheritance correctly
- Use interfaces correctly to establish polymorphism
- Use switch statements logically
- Explore prototyping
- Study existing code

2. Instructions:

One of the newest evolutions in games are idle games and resource management type games where you create resources that are then combined to form more complex items. We don't have time to create one of these completely, but we could start one. What you'll be creating this week is a set of factories. Each factory takes in a raw resource and produces blocks. We are going to use the provided prototype to test out our idea and theoretically help us design a more complete game later.

I'll give you a head start by providing a driver I wrote to test my finished concept, but it'll be up to you to implement the rest. You'll know you were successful when you can run the driver correctly. Feel free to comment sections out and get it working one section at a time. Before you get started make sure to study the driver and understand what it is doing. Studying already created code is often part of the job.

Note: you can stop an infinite loop at the command line with CTRL + C.

We are also going to stray away from test driven design this time and think more in terms of prototyping. Prototyping is when we create code to help us with the design process and facilitate the creative process. In the end, we typically throw away the code we created and start with a more formal process like test driven development. However, you are welcome to create some JUnit tests if you would like and to get the practice.

Here's a list of object descriptions and requirements. Use these, the rubric, and the already provided driver to create the finished application. Note that the driver should not be modified. You will have to design your classes using inheritance and an interface to make the driver work correctly. This process can be thought of as reverse-engineering the driver file. In another sense, the assignment is asking you to implement classes provided a set of requirements as would be typical of test-driven development.

Objective 1: Enum ResourceType

1.1 This object will define all the allowable types for your application. For this prototype use: stone and wood.

1.2 Make sure to use a stand alone file to establish your enum so that it is accessible from any file in your project.

Objective 2: Class Object Const

2.1 This class will hold constant variables you want to use throughout your application. That is its only purpose. <u>Thus, make sure it can't be inherited from.</u>

2.2 All of its variables should be accessible from outside and unchangeable.

2.3 Inside this class, create constants for the weight of a stone block and the weight of a wood block. Suggested weights to match the driver prototype are Stone 20.0 and Wood 10.0 but you may choose your own weights.

Make sure to use the correct naming convention here.

Objective 3 : Class Object Resource

3.1 Think of this class object as raw material. You can have any amount including partial amounts. Thus, this will need to be a double. We'll use weight to record this amount, but we'll ignore what unit that weight is in. You will need a constructor for this Resource that takes a weight and a type.

3.2 A resource has a weight and a resource type of stone or wood. Make sure the weight and type can't be changed by other objects, but can be retrieved if needed.

3.3 You are also going to need a method to add to an amount of an existing resource of the same type. That just means you increase the weight of that resource. You also want to be able to subtract from a resource by reducing the weight.

3.4 Code defensively by throwing appropriate exceptions with appropriate error messages if another object tries to subtract more than what's there already.

Objective 4 : Class Object Block

4.1 Block is a class that should never be instantiated. Make sure that isn't possible.

4.2 It contains a weight and a resource type of stone or wood.

4.3 The constructor for a block only accepts a resource type and weight.

4.4 Make sure other class objects can get the type and weight of a block, but not set them. Go ahead and create an appropriate toString override as well.

Objective 5 : Wood and Stone blocks

5.1 Wood and Stone blocks are children of Block.

5.2 They each use a no argument constructor. They make use of their parent's constructor that uses a weight and resource type within that no argument constructor and they set their weight based on the constants for a block of that type.

5.3 Make sure not to duplicate code from the parent class.

Objective 6 : Factory interface

This application is designed to be able to create many different factories in many different ways as a result of future development.

6.1 Thus, we need to establish a factory protocol: A factory should take a resource, produce a block, and display its inventory. How each factory does that will be up to that factory's implementation.

Objective 7 : Stone and Wood block factories

7.1 A stone or wood block factory needs to have a resource bin (a resource object) to store resources and implement the factory interface. Once it is safe to do so, add any resource sent to it to the resource bin.

7.2 Make sure to account for situations where a factory attempts to produce a block but its bin has insufficient weight. You'll have to determine how to implement this functionality using exception handling and a try-catch statement.

Objective 8 : Run the simulator

8.1 This time instead of a test file, I'm going to give you the completed driver class. Studying this should help you complete the assignment. Reverse engineering existing code is a common part of the job. Often in industry we have to work backwards from legacy code and/or work with a team to finish an assignment.

3. **Report:**

Each assignment must include a short report. The generation of this report should take you no more than 15 minutes. This gives you a chance to reflect back on what you learned and it makes grading easier on your grader. For this report, I want the following sections:

1. Reflection (*What did you learn?)*
2. How did this design incorporate future growth?

5. **Submission:**

Please read carefully. Failure to follow submission instructions can result in a reduced score.

Submit all files on Canvas under the appropriate assignment. Submit your files as a single zip file named: `"Your Name"_"Assignment".zip`

Make sure to submit the following files:

- ResourceType.java
- Const.java
- Resource.java
- Block.java
- WoodBlock.java
- StoneBlock.java
- Factory.java
- StoneBlockFactory.java
- WoodBlockFactory.java
- Main.java
- Report.pdf

Submission checklist:

- ☐ Did you include adequate comments?
- ☐ Did you include comment blocks at the top of <u>each file</u>?
- ☐ Did you name your files as requested?
- ☐ Does your code compile?
- ☐ Did you take care of any warnings presented by your IDE?