Assignment Product Factory

Object Orientation

Spring 2025

1 Learning Objectives

After making this exercise you should be able to

- Construct interfaces containing default methods and static methods,
- Construct classes implementing multiple interfaces,
- Treat different types of objects differently using
  - Downcasting in combination with `instanceof`,
  - Double dispatch/visitor pattern,
- Construct classes with many fields using static builder method and the builder pattern.

We recommend you read the whole document before starting the implementation at Section 5. A minimal version of the classes and methods you have to implement is summarized in Figure 1. Sample output for the program can be found in Section A.

2 Product Types

In this assignment, you are going to implement a factory for products (TVs, fridges, and chairs) and a warehouse storing them. As these products share some common behavior, so we are going to create some interfaces to express this shared behavior. All three products can be sold. Two products are electronic (TV, fridge). Only one product has a display (TV).

Product All products produced by the factory are sold and therefore have a price. The `Product` interface will contain a method for obtaining the price as a `double`, a method for setting it, and a default method that use the previous two methods to multiply the current price by a `double` factor. This will be useful for computing discounts.

EnergyConsumer Create an interface that contains a method for obtaining the *voltage* (V) and the *current* (I) of the device as a `double`. The interface should also contain a default method for computing the yearly power consumption in kilowatt-hour. This can be computed using the following formula[1]: $E = \frac{V \cdot I \cdot 24 \cdot 365}{1000}$.

[1]In practice it is more complicated but we will ignore this.

Display A display has a vertical resolution, horizontal resolution, refresh rate, bits per pixel, all stored as `int`. Furthermore, a display has brand, model, display technology, all stored as `String`. Additionally, a display can have an arbitrary amount of *connectors*, which are stored as a list of `String`. The interface should contain a method to obtain the amount of connectors of a certain type, e.g. HDMI.

3 Products

For all the products listed below, you will have to implement the `toString` method for producing a human-readable representation of the product.

3.1 Chair

`Chair` is a product produced by the factory that only implements the `Product` interface. A chair contains a `String` which describes its material.

3.2 Fridge

A fridge is a product produced by the factory that implements both the `Product` and the `EnergyConsumer` interfaces. Additionally, the `Fridge` class should store a `boolean` indication whether the fridge has a freezer, an `int` indicating the total volume in liter, and a `String` storing the brand of the fridge.

Fridge Builder To facilitate making common types of fridge, we can create static factory methods. Create a method `static Fridge createFreezer(String brand, int volume, double price)` that constructs a fridge that has a freezer, the specified volume, brand and price, and runs on $240V$, $2A$. Similarly, create a method `static Fridge createFridge(String brand, int volume, double price)` that does the same, but creates a fridge without a freezer. Note that that these methods have the same arguments, which is not possible for constructors. Secondly, they have names that describe their purpose, which is also not possible for constructors.

3.3 TV

A TV is a product produced by the factory that implements `Product`, `EnergyConsumer` and `Display`. We assume that TVs are only turned on 1 hour per day, so the method that returns the yearly energy consumption should return only $\frac{1}{24}$th of the default value.

TV Builder As you might have noticed, the TV class contains quite a few fields. Initializing all these fields in a single constructor can become quite unwieldy. Instead, we will create a `Builder` class which will facilitate this process, following the builder[2] pattern. The `Builder` class will be nested within the `Tv` class.

The TV class should only contain a private constructor which takes a `Builder` object. TVs can then only be obtained by calling the `Builder.build()` method, which in turn uses this private constructor. The builder class should contain methods for setting all the TV fields, including a method to add a single connector to the current TV being constructed. The constructor of the builder class should initialize the required fields, namely, the horizontal and vertical resolution. You can look at Figure 1 for inspiration.

[2]For an example, see: `https://www.digitalocean.com/community/tutorials/builder-design-pattern-in-java`

4 Warehouse

The warehouse will store a selection of products. Similar to the static builder methods created for the fridge, create a static builder method for a warehouse containing the products seen in Section A. We will implement two tasks to be performed on the warehouse called procedure 1 and 2.

4.1 Procedure 1

Write a method that reduces the price of all TVs that use LCD as their display technology by 10%, and also increases the price of all fridges that have a freezer by 20%. You can use `instanceof` to check the type of each `Product`. To access product specific methods, use *downcasting*.

4.2 Procedure 2

Extend the product interface such that it contains a method `void accept(ProductVisitor visitor)`. A `ProductVisitor`[3] is a new interface, that contains the methods `void visit(Chair chair)`, `void visit(Fridge fridge)`, `void visit(Tv tv)`.

This procedure should:

- Create a class implementing the `ProductVisitor` interface, e.g., `ProcedureOnevisitor`, which:
  - Discounts every wooden chair by 15%, while increasing the price of metal chairs by 12%.
  - Discounts every Samsung fridge by 5%.
  - Discounts every TV having 2 or more DP connectors by 30%.
- Create an object of type `ProcedureOneVisitor` called `visitor`.
- Iterate over all products and call `product.accept(visitor)`.

5 Your Tasks

1. Create the `Product` interface. For simplicity, you can keep out the `visit` method for now.
2. Create the `Chair` class implementing the `Product` interface.
3. Create the `EnergyConsumer` interface.
4. Create the `Fridge` class implementing the `Product` and `EnergyConsumer` interfaces.
5. Create the `Display` interface.
6. Create the `Tv` class implementing the `Product`, `EnergyConsumer` and `Display` interfaces.
7. Create the `Builder` class as a nested class inside of the `Tv` class.
8. Fill in the `Warehouse` class storing the products. Implement procedure 1 described in Section 4.1 and procedure 2 described in Section 4.2. At this point, you need to extend the product interface to contain the `visit(ProductVisitor)` method, and implement the `ProductVisitor` interface too.
9. The main method provided should produce an output similar to Section A. The printing of the warehouse should use the `toString` method of the warehouse, which in turn uses the `toString` function of the products.

[3]This is called the visitor pattern, see e.g. `https://www.digitalocean.com/community/tutorials/visitor-design-pattern-java`

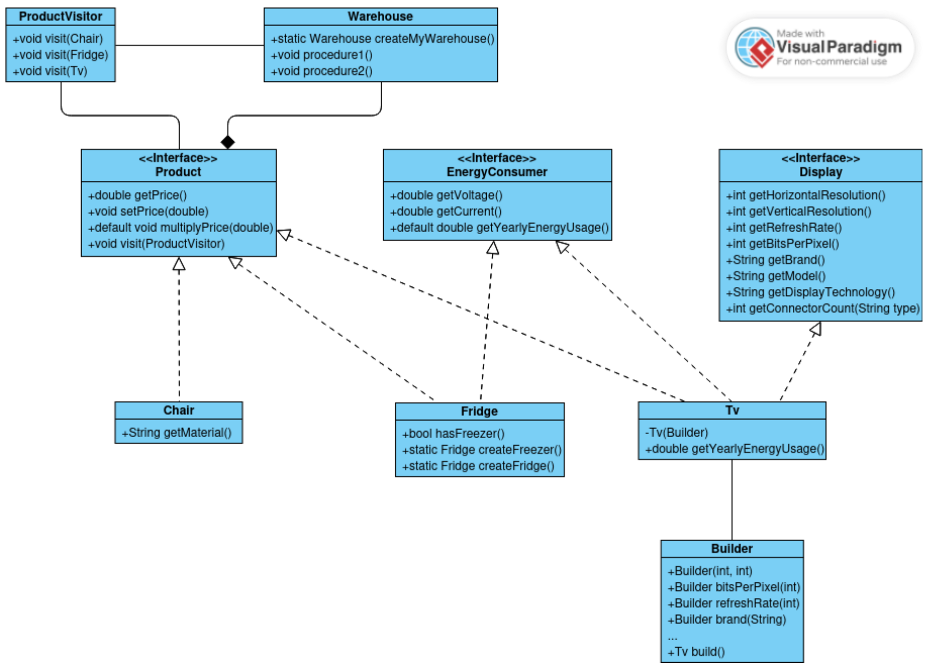
ProductVisitor
+void visit(Chair)
+void visit(Fridge)
+void visit(Tv)
Warehouse
+static Warehouse createMyWarehouse()
+void procedure1()
+void procedure2()
Made with Visual Paradigm For non-commercial use
<<Interface>> Product
+double getPrice()
+void setPrice(double)
+default void multiplyPrice(double)
+void visit(ProductVisitor)
<<Interface>> EnergyConsumer
+double getVoltage()
+double getCurrent()
+default double getYearlyEnergyUsage()
<<Interface>> Display
+int getHorizontalResolution()
+int getVerticalResolution()
+int getRefreshRate()
+int getBitsPerPixel()
+String getBrand()
+String getModel()
+String getDisplayTechnology()
+int getConnectorCount(String type)
Chair
+String getMaterial()
Fridge
+bool hasFreezer()
+static Fridge createFreezer()
+static Fridge createFridge()
Tv
-Tv(Builder)
+double getYearlyEnergyUsage()
Builder
+Builder(int, int)
+Builder bitsPerPixel(int)
+Builder refreshRate(int)
+Builder brand(String)
...
+Tv build()

Figure 1: Class Diagram

5.1 Submit Your Project

To submit your project, follow these steps.

1. Find the folder that contains your assignment. In Visual Studio Code, you can find this by going to: File → Open Folder. Add the correct folder to a zip file. At the root level, your zip file should contain only one folder, e.g. `assignment-student-start`, which contain the entire project, i.e. the `app` folder and the Gradle files. *Do not submit only the `.java` files or the `src` folder!*
2. **Submit this zip file on Brightspace**. Do not submit individual Java files. Only one person in your group has to submit it. Submit your project before the deadline, which can be found on Brightspace.
3. **Do not submit any other format.** Do not submit .rar or .tar.gz or .7z files. Only zip files are allowed.

A Example Output

```
Initial warehouse:

Warehouse 'My warehouse'
Products:
0:      $50.00  wooden chair
1:      $100.00 metal chair
2:      $300.00 LG L27LCD+, 1920x1080x24@60hz LCD display [HDMI, HDMI, DP] 240V 0.10A (8.76kWh)
3:      $300.00 LG L27LCD+, 1920x1080x24@60hz LCD display [HDMI, HDMI, DP] 240V 0.10A (8.76kWh)
4:      $500.00 Sony S33IPS++, 1920x1080x24@90hz IPS display [HDMI, HDMI, DP, DP] 240V 0.10A (8.76kWh)
5:      $400.00 Bosch 60L fridge with freezer 240V 2.00A (4204.80kWh)
6:      $600.00 Samsung 100L fridge without freezer 240V 2.00A (4204.80kWh)

After procedure 1:

Warehouse 'My warehouse'
Products:
0:      $50.00  wooden chair
1:      $100.00 metal chair
2:      $270.00 LG L27LCD+, 1920x1080x24@60hz LCD display [HDMI, HDMI, DP] 240V 0.10A (8.76kWh)
3:      $270.00 LG L27LCD+, 1920x1080x24@60hz LCD display [HDMI, HDMI, DP] 240V 0.10A (8.76kWh)
4:      $500.00 Sony S33IPS++, 1920x1080x24@90hz IPS display [HDMI, HDMI, DP, DP] 240V 0.10A (8.76kWh)
5:      $480.00 Bosch 60L fridge with freezer 240V 2.00A (4204.80kWh)
6:      $600.00 Samsung 100L fridge without freezer 240V 2.00A (4204.80kWh)

After procedure 2:

Warehouse 'My warehouse'
Products:
0:      $42.50  wooden chair
1:      $112.00 metal chair
2:      $270.00 LG L27LCD+, 1920x1080x24@60hz LCD display [HDMI, HDMI, DP] 240V 0.10A (8.76kWh)
3:      $270.00 LG L27LCD+, 1920x1080x24@60hz LCD display [HDMI, HDMI, DP] 240V 0.10A (8.76kWh)
4:      $350.00 Sony S33IPS++, 1920x1080x24@90hz IPS display [HDMI, HDMI, DP, DP] 240V 0.10A (8.76kWh)
5:      $480.00 Bosch 60L fridge with freezer 240V 2.00A (4204.80kWh)
6:      $570.00 Samsung 100L fridge without freezer 240V 2.00A (4204.80kWh)
```