Project

Scenario - OraclProduction

OraclProduction Ltd are specialists in producing production line manufacturing plants.

They could be asked to create a production plant for any type of product ranging from a simple packaging system to a variety of electronic devices.

Recently they have been asked to create a production line for multimedia devices which include music and movie players. They wish to employee you to design a template in Java for creating and recording all future production line items. For this particular production facility you will only implement a concrete class for music and movie players.

Your task is to create a flexible structure that could be used in any production line. This structure would then allow easy modification to handle different products.

~~Step 1~~

Create an interface called **Item** that will force all classes to implement the following functions.

- A constant called manufacturer that would be set to "OracleProduction".
- A method setProductionNumber that would have one integer parameter
- A method setName that would have one String parameter
- A method getName that would return a String
- A method getManufactureDate that would return a Date
- A method getSerialNumber that would return an int

~~Step 2~~

All items will have a pre-set type. Currently there are 4 types. Create an enum called **ItemType** that will store the following information.

Type	Code
Audio	AU
Visual	VI
AudioMobile	AM
VisualMobile	VM

Step 3

Create an abstract type called **Product** that will implement the Item interface. Product will implement the basic functionality that all items on a production line should have. Add the following fields to Product

- int serialNumber
- String manufacturer
- Date manufacuredOn
- String name

Add an integer class variable called currentProductionNumber. This will store the next number to be assigned to serialNumber.

Complete the methods from the interface Item.

Add a constructor that will take in the name of the product and set this to the field variable name. You will also assign a serial number from the currentProductionNumber. The currentProductionNumber should be incremented in readiness for the next instance.
Set manufacturedOn as the current date and time.

Add a toString method that will return the following: (example data shown).

```
Manufacturer  : OraclProduction
Serial Number : 1
Date          : Thu May 14 15:18:43 BST 2015
Name          : Product Name
```

Step 4

All of the items on this production line will have basic media controls. Create an interface called MultimediaControl that will define the following methods.

- public void play();
- public void stop();
- public void previous();
- public void next();

Step 5

We require a concrete class that will allow us to capture the details of an audio player. Create a class called AudioPlayer that is a subclass of Product and implements the MultimediaControl interface.

The class will have 2 fields

- String audioSpecification
- ItemType mediaType

Create a constructor that will take in 2 parameters – name and audioSpecification.
The constructor should call its parents constructor and also setup the media type.

Implement the methods from the MultimediaControl interface by simply writing the action to the console. E.g. in play System.out.println("Playing"); Normally we would have code that would instruct the media player to play, but we will simply display a message.
Create a toString method that will display the superclasses toString method, but also add rows for Audio Spec and Type.

~~Step 6~~

Create a driver class for AudioPlayer that will test to see whether we can instantiate occurrences of it, use the media controls and print out their details to the console.

~~Step 7~~

The production facility will also create portable movie players. The main difference between these and the audio players is that they contain screens.

Create an enum called MonitorType that will store

Type
LCD
LED

~~Step 8~~

Create an interface called ScreenSpec. This will define 3 methods:

- public String getResolution();
- public int getRefreshRate();
- public int getResponseTime();

~~Step 9~~

Create a class called Screen that implements ScreenSpec. Add three fields
String resolution
int refreshrate
int responsetime

Complete the methods from the ScreenSpec interface.

Add a toString method that will return the details of the 3 field in the same format as the Product Class.

Step 10

Create a Driver class for Screen that tests the functionality of the screen class

Step 11

Create a class called MoviePlayer that extends Product and implements MultimediaControl.

Add 2 fields to this class called screen and monitor type and assign appropriate types to them.

Complete the methods from the MultimediaControl interface in a similar fashion to the audio player.

Create a toString method that calls the product toString, displays the monitor and the screen details.

Step 12

Create a diver class to test the functionality of the movie player.

Step 13

The audio players and the movie players share the same control interface on the physical devices. The control interface does not care if the device is a video player or an audio player. Create a driver class that will demonstrate that any class that implements the MultimediaControl Interface would be able to be instantiated and use its methods used no matter if it was an audio or movie player.

Step 14

Add functionality to your classes that would allow them to be sorted by name with the Collections.sort method.

Step 15

You are going to store a collection of the devices as they come off the production line. Choose an appropriate Java Collection and create a driver class that demonstrates its use. You should also demonstrate the Collections.sort method.

Step 16

Create a method called print that would take your collection and list all of the contents. It should handle all of your classes

Step 17 – Bonus

This step is optional and challenging.

Create a static method called printType in Product that will iterate through your Collection and print all the classes of a particular type.
Example – print only AudioPlayer classes in the collection.
For an extra bonus you could modify it so that it would accept the Class that you want to print in the parameter list. This way we could use it against classes that we have not yet built.
Limit the collection to only use subclasses of Product.

Step 18

The program is required to create an audit trail on its tests of the production line so that it records which employee ran the test. To accomplish this you will need to create a class named EmployeeInfo that will allow the user to input their full name and then create a user id of their first initial and surname.

The class will have 2 fields

- StringBuilder name;
- String code;

The class will have the following methods defined:

- public StringBuilder getName()
- public String getCode()
- private void setName()
- private void createEmployeeCode(StringBuilder name)
- private String inputName()
- private boolean checkName(StringBuilder name)

The setName() method will be called from the constructor which will use inputName() to get a name (firstname and surname) as a single input from the user before checkName() is used to make sure that the name supplied has a space in it.

If a valid name is given then createEmployeeCode() is used to take the first initial from the first name and add it to the full surname to create the code. If there is no space then default value of guest is to be used as the value for code.

In the TestProductionLine class create an employee object using the EmployeeInfo class. Using the getCode() method display the employee code at the bottom of the product output.

Step 19

An additional piece of information is required to be produced for the auditing with the users department information being required as well. The department code is made up of four letters and two numbers. The format of the department code is the first letter must be in uppercase with the following three all being lowercase and no spaces.

The following three fields need to be added to the EmployeeInfo class:

- String deptId;

- Pattern p;
- Scanner in;

The following new methods have to be defined:

- public String getDeptId()
- private void setDeptId()
- private String getId()
- private boolean validId(String id)

As there will be multiple inputs across the class now the scanner will need to be declared and closed in the constructor. The pattern to control the format of the input will also have to be declared in the constructor. In between opening and closing the scanner, the constructor will need to not only get the name but also the deptId of the user.

setDeptId() will call getDeptId() to get the id from the user before validId() is used to check if the input matches the pattern. If the pattern matches then the given id is set to deptId otherwise a default value of None01 should be assigned.

As there are now two values to be displayed (code, deptId) create a toString() method that will override the output and allow you to simply display the value of the object to the screen.

Update the TestProductionLine class to use the toString() method to display the values to the console.

Step 20

To ensure that sensitive information is not leaked it is important that the information saved to file is encoded. To meet these regulations you need to add a method to the EmployeeInfo class that will reverse the order of the text stored for the department id. This should be done recursively using a method named reverseString().

The following new methods have to be defined:

- public String reverseString(String id)

If a valid department id is provided then reverseString() should be called before assigning the user input to the deptId field.

Step 21

OraclProduction Ltd now wants to store the results of the tests and who carried out that test to a file for archiving purposes. To achieve this create a new class named ProcessFiles that will create a directory and file structure by combining the path and file name. It will have to save both the products and the employee information to file so create two methods that override each other that accept either the employee object or products arraylist.
The directory and file should be created when the object is created.

The class will have three fields:

- private Path p;
- private Path p2;
- private Path p3;

The following new methods have to be defined:

- private void CreateDirectory()
- public void WriteFile(EmployeeInfo emp) throws IOException
- public void WriteFile(ArrayList<Product> products) throws IOException

For both the file and directory if they do not exist then create them. The directory should be created at the root of the C drive and be named LineTests(p). The file should be named TestResults.txt(p2) and should be created within the LineTests directory. The p3 field should be used to store the resolved path combining p and p2.

Update the TestProductionLine class to use the WriteFile() methods to save the product information to file followed by the employee information. This information should be appended so that no historical information is lost in this process.

Test the file output to ensure that it is formatted the same as the console output.

Step 22

Create a driver class named ViewFileInfo that will read the information from the TestResults.txt file and display the contents to the console. Use a try statement to handle situations where the file does not exist.

Step 23

Create a program that would allow a user to Add a new product and to specify how many items of that product should be created. The program will then create these items and store them in a collection.
The collection can be displayed at any time.
New Products can be added at any time.
Production statistics can be displayed – Total items produced, number of each item, the number of unique products created etc.
The operator will decide when to stop the program. The program will be menu driven and can either be a console program or with a richer graphical user interface.