Федеральное государственное автономное образовательное учреждение

высшего образования

Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Высшая школа «Киберфизические системы и управление»

**Отчет № 11**

по дисциплине «Системный подход к разработке программного обеспечения»

|  |  |  |
| --- | --- | --- |
| Выполнил:  студент гр. 3530902/90201 | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Позолотин О. В. |
|  | <*подпись*> |  |
|  |  |  |
|  |  |  |
| Руководитель:  Кандидат т.н. | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Нестеров С. А. |
|  | <*подпись*> |  |

«\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2022 г.

Санкт-Петербург

2022

**Steps to Deploying an Application (Using the Eclipse IDE)**

1. Choose File or Right click the project file folder on the project explorer window and choose Export from the menu

2. From the dialog box open the Java option and then choose runnable JAR file from the list

3. Click on the next button

4. On the screen configure the options as you want

− Launch Configuration

- This is the driver class of your application

− Export Destination

- Where you want to save the JAR file

− Library Handling

- Leave this as the default

−ANT File allows you to create documentation that will recreate the JAR file for you in the future, save this in your workspace

5. Go to your save location and double click the JAR file

6. Your program should run

**Coding a JavaFX application**

When you create a JavaFX project you combine components and action listeners to create an interactive GUI application. The following application will allow users to vote for either Java or SQL as their favourite topic.

a. Create the following instance fields above the start() method in the code. These will be used to keep the score and for the content of a label.

b. Create a label that will display the category for the vote using the following code at the start of the main method:

Label cat1Lbl = **new** Label(cat1);

cat1Lbl.setLayoutX(110);

cat1Lbl.setLayoutY(10);

cat1Lbl.setTextFill(Color.***RED***);

c. You will have to import libraries to use these components. Always make sure you import the javaFX library.

d. You will now create another label that will display the number of votes cast. This will be displayed under the previous label and centered to the position of cat1Lbl.

Label score1Lbl = **new** Label(""+cat1Score);

score1Lbl.setLayoutX(130);

score1Lbl.setLayoutY(40);

score1Lbl.setTextFill(Color.***RED***);

e. A button is needed that the user can click to cast their vote. This will be positioned under the previous two labels. You will have to import the Button library, remember to choose the fx option.

Button cat1Btn = **new** Button("Vote " + cat1);

cat1Btn.setLayoutX(90);

cat1Btn.setLayoutY(80);

f. To make a component interactive you need to add an event handler to the component. Use the drop-down list of content prompts in eclipse to create the following action listener that uses a Lambda expression:

cat1Btn.setOnAction((ActionEvent event) -> {});

g. The code inside the handle method is fired whenever the button is clicked. Add the code to increment the number of votes (stored as an instance field) and update the text in the label to show the new value.

cat1Btn.setOnAction((ActionEvent event) -> {

cat1Score++;

score1Lbl.setText(""+cat1Score);

});

h. Although you have created the components you have not yet added them to your application so if you were to run your code now you would not see them. To add the components you have to add them to an FX layout manager. Currently the application uses a BorderPane layout (you can learn more about these through the Java API) but you are going to use a Group layout that allows the placing of components using X and Y co-ordinates. Update the existing code:

i. Add each component individually to the root node using the getChildren().add method:

root.getChildren().add(cat1Lbl);

root.getChildren().add(score1Lbl);

root.getChildren().add(cat1Btn);

j. Update the size of the scene (this is what you will add the root to in order to display the components in the primary stage (Window) to have a width of 300 and a height of 150.

Scene scene = **new** Scene(root,400,150);

k. You will not be using the stylesheets in this exercise so you can remove the line that starts with scene.getStyleSheets().

l. Add the following code that sets a title on the window and sets the width and height of the window equal to the dimensions of the scene.

5. Use the following information to create a second set of labels and buttons:

a. An integer instance field named cat2Score that is initialized to 0.

b. A String instance filed named cat2 that is initialized to “SQL Team”

c. Create a cat2Lbl at position 230, 10 that displays the text in blue.

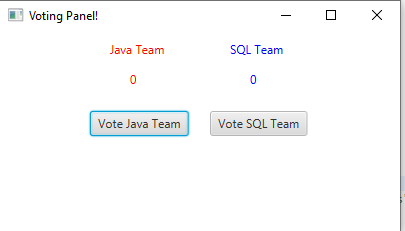
d. Create a score2Lbl at position 250, 40 that displays the score in blue.

e. Create a button named cat2Btn that displays vote plus the value of cat2. Its position should be 210, 80.

f. Add an event handler for the button that increases cat2Score by one and updates the text value.

g. Add the components to the Group layout manager.

h. Run and test that your code works.



6. You will now add images to the application by including them in a resources folder with the application.

a. In Eclipse, right click on the application package and select New, Source Folder.

b. Name the source folder img and then click Finish.

c. The img folder is displayed in your project directory.

d. Drag the image file Java\_Duke.png from your local directory into the img folder. Choose copy link and then OK to complete the process.

e. Above the try catch block in your code add the following line of code that creates a reference (jImage) to the Java\_Duke image in the img folder. You will need to import the fx image library.

f. To display the image in the application you have to place it in an ImageView object that you can then use to set the display co-ordinates. Add this

7. Use the following information to add an image for the SQL Team vote.

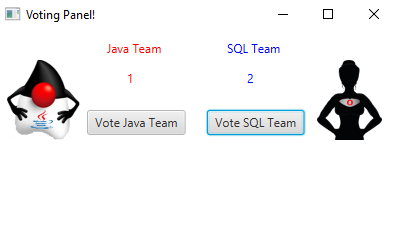
a. Drag the Datagirl.png image file into the img source folder.

b. Create a reference to the Datagirl image named sqlImage.

c. Create a sqlImgView using the sqlImage reference.

d. Place the sql image at the following co-ordinates: X- 320, Y-30. e. Add the image view to the Group layout m,anager.

f. Your application should look like this:



8. You now have a working app that allows votes to be cast, however the user experience can be improved with a few tweaks.

a. The vote count should not be displayed on screen until the vote is over.

i. Under the setTextFill() method for each score label add the following code so that the labels are not displayed to screen.

ii. Create a private method named showVotes that accepts no parameters and returns no values. This method will set the value of the scores as the text in the score labels and then display the labels to screen. Create the method above the main method at the bottom of the code.

iii. You get errors as the compiler cannot recognise the scoreLbl components. This is because you created them locally in the start method. To fix this you can create the reference to the component as an instance field and then create the component object locally. Add the following instance fields to the top of your class:

iv. Update the code in the start menu to instantiate the object referenced as an instance field instead of creating a second one. For the score labels update the code by not declaring a type at the beginning of the statement so that it does this:

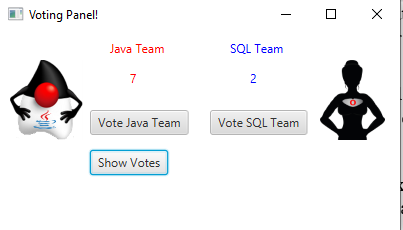
b. To view the votes cast,

i. create a new button named showBtn that will display the text “Show Votes”. Display the button at the following co-ordinates: X-90, Y-120 and add it to the Group layout manager.

ii. Call the showVotes() method from its event handler.

iii. Run and test your program

iv. Your application should look like this:



c. You can see that the size of the buttons is different and doesn’t look quite right. You can fix this by setting a preferred width for the buttons.

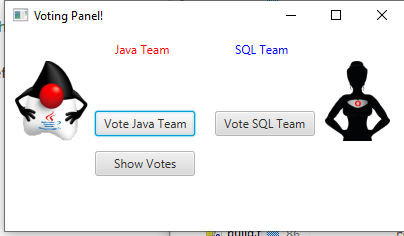
i. Create an unchangeable value at the beginning of the start method:

ii. In the properties list for all of the buttons use the setPrefWidth() method to set the width of the button equal to the BTN\_WIDTH value.

iii. Do the same for the cat1Btn and the cat2Btn buttons.

iii. Run your program

iv. Your application should look like this:



d. You will now add functionality that will reset the votes to zero so that voting can begin again.

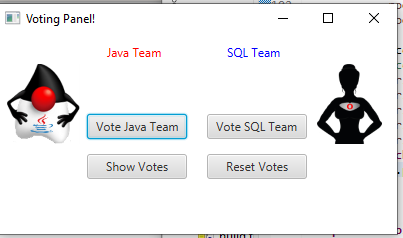
i. Create a private method that will set the values of the scores to zero and hide the scores labels.

ii. Create a button named resetBtn that displays “Reset Votes” as its text, it’s positioned at 210, 120 and uses the preferred button width.

iii. Call the resetVotes() method from its event handler.

iv. Run and test your program

v. Your application should look like this:



e. There are a few issues with the application:

i. After showing the vote values you can click the vote buttons to add to the totals.

ii. You can reset the votes at any time.

iii. The votes are not reset after they have been viewed.

f. Create a private void method named setBtnUse that accepts 4 boolean parameters that will be used to control the disabled state of the buttons. You will have to create button references as instance fields for this to work (remember to remove the Button type from the start method)

g. You can fix these issues by simply controlling access to the buttons.

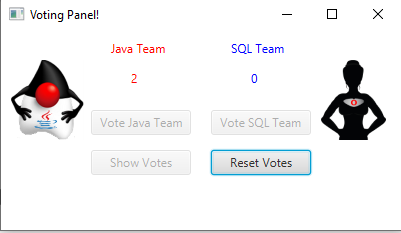
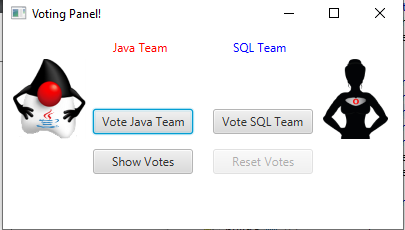
i. The reset button should be disabled until the showVote button is clicked. When the reset button is created set it to disabled:

This will disable the two vote buttons and the show button and enable the reset votes button

iii. In the event handler for the resetBtn call the setBtnUse method passing values so that the two vote buttons and the show votes button are enabled, and the reset button is disabled.

iv. Run and test your program

v. Your application should look like this:



9. To be able to run the application outside of the Eclipse IDE you need to create a runnable JAR file.

a. In Eclipse select File, Export.

**package** application;

**import** javafx.application.Application;

**import** javafx.event.ActionEvent;

**import** javafx.scene.Group;

**import** javafx.scene.Scene;

**import** javafx.scene.control.Button;

**import** javafx.scene.control.Label;

**import** javafx.scene.image.Image;

**import** javafx.scene.image.ImageView;

**import** javafx.scene.paint.Color;

**import** javafx.stage.Stage;

**public** **class** Main **extends** Application {

**int** cat1Score=0;

**int** cat2Score=0;

String cat1 = "Java Team";

String cat2 = "SQL Team";

**private** Label score1Lbl, score2Lbl;

**private** Button cat1Btn, cat2Btn, showBtn, rstBtn;

@Override

**public** **void** start(Stage primaryStage) {

**final** **double** BTN\_WIDTH = 100;

Label cat1Lbl = **new** Label(cat1);

cat1Lbl.setLayoutX(110);

cat1Lbl.setLayoutY(10);

cat1Lbl.setTextFill(Color.***RED***);

Label cat2Lbl = **new** Label(cat2);

cat2Lbl.setLayoutX(230);

cat2Lbl.setLayoutY(10);

cat2Lbl.setTextFill(Color.***BLUE***);

//create second label

score1Lbl = **new** Label(""+cat1Score);

score1Lbl.setLayoutX(130);

score1Lbl.setLayoutY(40);

score1Lbl.setTextFill(Color.***RED***);

score1Lbl.setVisible(**false**);

score2Lbl = **new** Label(""+cat2Score);

score2Lbl.setLayoutX(250);

score2Lbl.setLayoutY(40);

score2Lbl.setTextFill(Color.***BLUE***);

score2Lbl.setVisible(**false**);

cat1Btn = **new** Button("Vote " + cat1);

cat1Btn.setLayoutX(90);

cat1Btn.setLayoutY(80);

cat1Btn.setPrefWidth(BTN\_WIDTH);

cat2Btn = **new** Button("Vote " + cat2);

cat2Btn.setLayoutX(210);

cat2Btn.setLayoutY(80);

cat2Btn.setPrefWidth(BTN\_WIDTH);

cat1Btn.setOnAction((ActionEvent event) -> {

cat1Score++;

// score1Lbl.setText(""+cat1Score);

});

cat2Btn.setOnAction((ActionEvent event) -> {

cat2Score++;

// score2Lbl.setText(""+cat2Score);

});

showBtn=**new** Button("Show Votes");

showBtn.setLayoutX(90);

showBtn.setLayoutY(120);

showBtn.setPrefWidth(BTN\_WIDTH);

showBtn.setOnAction((ActionEvent event)->{

showVotes();

setBtnUse(**true**, **true**, **true**, **false**);

});

rstBtn = **new** Button("Reset Votes");

rstBtn.setLayoutX(210);

rstBtn.setLayoutY(120);

rstBtn.setPrefWidth(BTN\_WIDTH);

rstBtn.setOnAction((ActionEvent event)->{

resetVotes();

setBtnUse(**false**,**false**,**false**,**true**);

});

rstBtn.setDisable(**true**);

//add image

Image jImage = **new** Image(Main.**class**.getResourceAsStream("/Java\_Duke.png"));

ImageView jImgView = **new** ImageView(jImage);

jImgView.setX(10);

jImgView.setY(30);

Image sqlImage=**new** Image(Main.**class**.getResourceAsStream("/Datagirl.png"));

ImageView sqlImgView = **new** ImageView(sqlImage);

sqlImgView.setX(320);

sqlImgView.setY(30);

**try** {

Group root = **new** Group();

root.getChildren().add(cat1Lbl);

root.getChildren().add(score1Lbl);

root.getChildren().add(cat1Btn);

root.getChildren().add(cat2Lbl);

root.getChildren().add(score2Lbl);

root.getChildren().add(cat2Btn);

root.getChildren().add(jImgView);

root.getChildren().add(sqlImgView);

root.getChildren().add(showBtn);

root.getChildren().add(rstBtn);

//

Scene scene = **new** Scene(root,400,200);

// scene.getStylesheets().add(getClass().getResource("application.css").toExternalForm());

primaryStage.setTitle("Voting Panel!");

primaryStage.setMinWidth(scene.getWidth());

primaryStage.setMinHeight(scene.getHeight());

primaryStage.setScene(scene);

primaryStage.show();

} **catch**(Exception e) {

e.printStackTrace();

}

}

**private** **void** showVotes() {

score1Lbl.setText(""+cat1Score);

score2Lbl.setText(""+cat2Score);

score1Lbl.setVisible(**true**);

score2Lbl.setVisible(**true**);

}

**private** **void** resetVotes() {

cat1Score=0;

cat2Score=0;

score1Lbl.setVisible(**false**);

score2Lbl.setVisible(**false**);

}

**private** **void** setBtnUse(**boolean** cat1, **boolean** cat2,

**boolean** show, **boolean** reset) {

cat1Btn.setDisable(cat1);

cat2Btn.setDisable(cat2);

showBtn.setDisable(show);

rstBtn.setDisable(reset);

}

**public** **static** **void** main(String[] args) {

*launch*(args);

}

}

5-3: Deploying an Application Practice Activities

Try It/Solve It:

1. Take the following java code and place it into a package named greeting in Eclipse:

2. Create a new package called converse in the same project and then create a class called Hello. You need to use the sayHello method from your MyName class.

a) What do you need to include in your java program to get this to work?

b) Show your completed code.

**package** converse;

**import** greeting.MyName;

**public** **class** Hello {

**public** **static** **void** main(String[] args) {

MyName obj=**new** MyName();

obj.sayHello();

}

}

3. You want to be able to change the name and then display the new name from within the Hello class.

a) How could you call the setName in a program before displaying the new name (Bob) to screen without using an import statement for the package?

b) Show your completed code.

**package** converse;

//import greeting.MyName;

**public** **class** Hello {

**public** **static** **void** main(String[] args) {

greeting.MyName obj=**new** greeting.MyName();

obj.setName("Bob");

obj.sayHello();

}

}

4. Using the following example create a runnable JAR file using Eclipse. Example Steps

a) In Eclipse create a new Java Project named dice.

b) In this project create a package also named dice.

c) In this package create a class named Random.

d) Add the following code to the Random class:

**package** dice;

**import** javax.swing.JOptionPane;

**public** **class** Random {

**public** **static** **void** main(String[] args) {

**int** rollOfDice;

String output;

// generate a random number between 1 and 6 inclusive

rollOfDice = (**int**) (Math.*random*() \* 6) + 1;

//create a String message for the output window

output = "You rolled a " + rollOfDice;

// print message using a window

JOptionPane.*showMessageDialog*(**null**, output, "Random Number Demo",

JOptionPane.***INFORMATION\_MESSAGE***);

System.*exit*(0);

}

}

a) Save and run your code.

b) Now create a runnable JAR file and test that it runs outside the IDE!

c) Detail the steps taken to create the runnable JAR file

Java Programming (Sections 1 – 5) Course Project

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

**package** oraclProd;

**import** java.util.Date;

**public** **interface** Item {

**static** **final** String ***manufacturer*** = "OracleProduction";

**void** setProductionNumber(**int** productionNumber);

**void** setName(String name);

String getName();

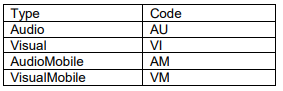
Date getManufactureDate();

**int** getSerialNumber();

}

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.



**package** oraclProd;

//step 2

**public** **enum** ItemType {

***AUDIO***("AU"),

***VISUAL***("VI"),

***AUDIO\_MOBILE***("AM"),

***VISUAL\_MOBILE***("VM");

**public** String code;

ItemType(String code) {

**this**.code = code;

}

**public** String getCode() {

**return** code;

}

}

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:

**package** oraclProd;

//step 3

**import** java.util.Date;

**public** **abstract** **class** Product **implements** Item {

**private** **int** serialNumber;

**private** String manufacturer=Item.***manufacturer***;

**private** Date manufacturedOn;

**private** String name;

**private** **static** **int** *currentProductionNumber* = 0;

**public** **void** setProductionNumber(**int** productionNumber) {

**this**.serialNumber=productionNumber;

}

**public** **void** setName(String name) {

**this**.name=name;

}

**public** String getName() {

**return** **this**.name;

}

**public** Date getManufactureDate() {

**return** **this**.manufacturedOn;

}

**public** **int** getSerialNumber() {

**return** serialNumber;

}

**public** Product(String name) {

**this**.name=name;

serialNumber = *currentProductionNumber*;

*currentProductionNumber*++;

manufacturedOn = **new** Date(System.*currentTimeMillis*());

}

@Override

**public** String toString() {

**return**

"\n" +

"Manufacturer : " + manufacturer + "\n" +

"Serial Number : " + serialNumber + "\n" +

"Date : " + manufacturedOn + "\n" +

"Name : " + 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();

**package** oraclProd;

//step 4

**public** **interface** MultimediaControl {

**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.

**package** oraclProd;

**public** **class** AudioPlayer **extends** Product **implements** MultimediaControl {

String audioSpecification;

ItemType mediaType;

**public** AudioPlayer(String name,String audioSpecification) {

**super**(name);

**this**.audioSpecification=audioSpecification;

**this**.mediaType=ItemType.***AUDIO***;

}

@Override

**public** **void** play() {

System.***out***.println("Playing");

}

@Override

**public** **void** stop() {

System.***out***.println("Stopping");

}

@Override

**public** **void** previous() {

System.***out***.println("Previous");

}

@Override

**public** **void** next() {

System.***out***.println("Next");

}

@Override

**public** String toString() {

**return** **super**.toString()+"\nAudio Specification: "+audioSpecification+"\nType "+mediaType;

}

}

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.

**package** oraclProd;

//step 7

**public** **class** AudioPlayerDriver {

**public** **static** **void** AudPlTest() {

AudioPlayer a1 = **new** AudioPlayer("Walkman", "MP3");

a1.play();

a1.stop();

a1.next();

a1.previous();

System.***out***.println(a1.toString());

}

}

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

**package** oraclProd;

//step 7

**public** **enum** MonitorType {

***LCD***,***LED***

}

Step 8 Create an interface called ScreenSpec. This will define 3 methods:

• public String getResolution();

• public int getRefreshRate();

• public int getResponseTime();

**package** oraclProd;

//step 8

**public** **interface** ScreenSpec {

**public** String getResolution();

**public** **int** getRefreshRate();

**public** **int** getResponseTime();

}

Step 9

Create a class called Screen ScreenSpec. Add three fields

String resolution

int refreshrate

int responsetime

Complete the methods from the ScreenSpec interface.

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

**package** oraclProd;

//step 9

**public** **class** Screen **implements** ScreenSpec {

**private** String resolution;

**private** **int** refreshrate;

**private** **int** responsetime;

**public** Screen(String resolution, **int** refreshRate, **int** responseTime) {

**this**.resolution = resolution;

**this**.refreshrate = refreshRate;

**this**.responsetime = responseTime;

}

@Override

**public** String getResolution() {

**return** resolution;

}

@Override

**public** **int** getRefreshRate() {

**return** refreshrate;

}

@Override

**public** **int** getResponseTime() {

**return** responsetime;

}

**public** String toString() {

**return** "Resolution : " + getResolution() + "\n" +

"Refresh rate : " + getRefreshRate() + "\n" +

"Response time : " + getResponseTime();

}

}

Step 10

Create a Driver class for Screen that tests the functionality of the screen class using the following values:

new Screen("600x400",40,22)

**package** oraclProd;

//step 10

**public** **class** ScreenDriver {

**public** **void** screenTest() {

Screen sc1=**new** Screen("600x400",40,22);

sc1.toString();

}

}

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.

**package** oraclProd;

//step 11

**public** **class** MoviePlayer **extends** Product **implements** MultimediaControl {

**public** MonitorType monitorType;

**public** Screen screen;

**public** MoviePlayer(String name, MonitorType monitorType, Screen screen) {

**super**(name);

**this**.monitorType=monitorType;

**this**.screen=screen;

}

@Override

**public** **void** play() {

System.***out***.println("Playing movie");

}

@Override

**public** **void** stop() {

System.***out***.println("Stoping movie");

}

@Override

**public** **void** previous() {

System.***out***.println("Previous movie");

}

@Override

**public** **void** next() {

System.***out***.println("Next movie");

}

**public** String toString() {

**return**

**super**.toString() + "\n" +

"Screen : " + screen + "\n" +

"Monitor Type : " + monitorType;

}

}

Step 12

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

**package** oraclProd;

//step 12

**public** **class** MoviePlayerDriver {

**public** **void** moviePlayerTest() {

MoviePlayer mp=**new** MoviePlayer("Sony",MonitorType.***LED***,**new** Screen("920x240",40,22));

System.***out***.println(mp.toString());

}

}

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.

**package** oraclProd;

//step 13

**public** **class** MediaDriver {

**public** **static** **void** MediaTest() {

MultimediaControl mc1=**new** AudioPlayer("Sony","WAVE");

MultimediaControl mc2=**new** MoviePlayer("Philips",MonitorType.***LCD***,**new** Screen("200x400",40,200));

mc1.next();

mc1.play();

mc1.previous();

mc1.stop();

mc2.next();

mc2.play();

mc2.previous();

mc2.stop();

}

}

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

**package** oraclProd;

**import** java.util.ArrayList;

**import** java.util.Collections;

**public** **class** Main {

**public** **static** **void** main(String[] args) {

ArrayList<Product> products;

products=*testCollection*();

Collections.*sort*(products);

*print*(products);

}

//step 15

**public** **static** ArrayList<Product> testCollection() {

AudioPlayer a1 = **new** AudioPlayer("iPod Mini", "MP3");

AudioPlayer a2 = **new** AudioPlayer("Walkman", "WAV ");

MoviePlayer m1 = **new** MoviePlayer("DBPOWER MK101",

MonitorType.***LCD***, **new** Screen(" 720x480", 40, 22));

MoviePlayer m2 = **new** MoviePlayer("Pyle PDV156BK", MonitorType.***LED***,**new** Screen("1366x768", 40, 22));

ArrayList<Product> products = **new** ArrayList<>();

products.add(a1);

products.add(a2);

products.add(m1);

products.add(m2);

**return** products;

}

//step 16

**public** **static** **void** print(ArrayList<Product> list) {

**for**(Product prod:list) {

System.***out***.print(prod+" ");

System.***out***.println();

}

}

}

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.

**UML Diagram**

