**Homework Assignment 8**

This homework addresses class and interface design and how different designs can support dynamic binding in an array of objects. You need to unzip file “**exampleOfClassInterfaceDesign.zip**”, and study the java source code and document “versionComparison.docx”, before start working on this homework.

Step 1: create superclass (including Interface) and subclass

Create an **Eclipse project** named JohnDoeHw8, and use the default package, i.e., no package name should be provided. Then inside this project, create a new class **Movie** as specified in the table below. Save it in file **Movie.java**. Also create **THREE child classes (subclasses)** of **Movie** as shown below. Save each class in a separate file – **Animated.java, Documentary.java,** and **Drama.java*.***

Be sure to follow the template file **CorrectSequence.java** in Moodle folder “chap 4” zip file “UML-classDiagram.zip” to code Movie and its subclasses, and this template file **includes the appropriate program header** and **comments.**

Also create Interface **Profitable**, and save it in file **Profitable.java**, and please use file **Payable.java as a refernece**.

Your class-interface design needs to support dynamic binding, so you need to follow either version 1 or version 2 in the class-interface design examples.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Movie** | **Animated**  *(child of Movie)* | **Documentary**  *(child of Movie)* | **Drama**  *(child of Movie)* | **Profitable**  *(an interface)* |
| **private instance variables (non-static variables)** | | | | **static final variables** |
| title | royalty rate | # of distributors | # of tickets sold |  |
| director | income of the related products | premium paid by each distributor | average price of each ticket |  |
| year (int type) |  |  |  |  |
| production cost |  |  |  |  |
| **public methods** | | | | **abstract methods** |
| constructor w/ and w/o parameters | constructor w/ and w/o parameters | constructor w/ and w/o parameters | constructor w/ and w/o parameters |  |
| one setter and one getter for each data member | one setter and one getter for each data member | one setter and one getter for each data member | one setter and one getter for each data member |  |
| effectors | effectors | effectors | effectors |  |
|  | get category | get category | get category | *get category* |
|  | calculate revenue | calculate revenue | calculate revenue | *calculate revenue* |
|  | calculate profit | calculate profit | calculate profit | *calculate profit* |
| toString | toString | toString | toString |  |

Here are the detail explanation of these five files:

In Interface **Profitable :**

* **NO** **data field** is needed
* **THREE abstract methods are defined**: ***get category*** return a String object*,* ***calculate revenue***and***calculate profit*** both return double type. There is **NO formal parameter in any** of these three methods.

In class **Movie**:

* **title** and **director** are of **String** type; **year** is of **int** type; **product cost** is of **double** type measured **in million dollars**. For example, a value of 200.5 means the production cost is 200.5 million dollars.
* There should be **two constructors**, one is the **default constructor** with **no parameter**, and **the other** has all **four parameters** to initialize the four private data fields, respectively.
* There should be a **public getter** and a **public setter** **for each of the four private data fields**.
* There is **one effector method toString(),** which is to generate an String variable that contains the four private data fields. This **method return a String type**. Please refer to class “Employee.java” for how to implement toString() method. You need to learn how to use String.format() method, which has the same syntax of System.out.printf() method, except that String.format() sends the fomrated output to a String object, while printf() method sends the output to the console screen.
* an example of the output String object of method toString() looks like this line below(there is an "\n" symbol output at the end of this line):

Movie *Avatar* was directed by *James Cameron* in *2009* with production cost of *200.5* millions.

In class **Animated: zootopia**

* **royalty rate** is of **double** type; **income of the related products** is measured in terms of **million dollars**, **double** type.
* There should be **two constructors**, one is the **default constructor** with no parameter, and **the other has six parameters** to initialize the four inherited data fields, plus the two private data fields of its own, respectively.
* There should be a **public getter** and a **public setter** for each **of the two private data fields**.
* There are four effector methods:
  + **get category** : it should return **String** "animated".
  + **calculate revenue** : **MULTIPLY** **income of the related products** (measured in million dollars) with **royalty** **rate**, return **double** type, measured in term of **million dollars**
  + **calculate profit** : subtract production cost (measured in million dollars) from the revenue ( also measured in million dollars), return double type
  + **toString** : it invokes the toString() in superclass first,then store parent’s toString() message in a local String variable, and then generates its own String message, which is like the line below (there is an "\n" symbol output at the end of this line):

This is an *animated* movie, and it generates *profit* (or *no profit*).

Depending on whether the profit is a positive value or not, you should insert “profit” or “no profit” into the above message.

Then concatenate its own String message after its parent’s toString() message, and return the concatenated message. Please refer to class SalesPerson, on how to implement toString() method.

in class **Documentary:**

* number of distributors is of int type; premium paid by each distributor is measured in terms of million dollars, double type.
* There should be two constructors, one is the default constructor with no parameter, and the other has six parameters to initialize the four inherited data fields, plus the two private data fields of its own, respectively.
* There should be a public getter and a public setter for each of the two private data fields.
* There are four effector methods:
  + **get category** : it should return String "documentary".
  + **calculate revenue** : multiply premium paid by each distributor (measured in million dollars) with number of distributors, return double type, measured in term of million dollars
  + **calculate profit** : subtract production cost (measured in million dollars) from the revenue ( also measured in million dollars), return double type
  + **toString** : it invokes the toString() in superclass first,then store parent’s toString() message in a local String variable, and then generates its own String message, which is like the line below (there is an "\n" symbol output at the end of this line):

This is an *documentary* movie, and it generates *profit* (or *no profit*).

Depending on whether the profit is positive value or not, you should insert “profit” or “no profit” into the above message.

Then concatenate its own String message after its parent’s toString() message, and return the concatenated message. Please refer to class SalesPerson, on how to implement toString() method.

in class **Drama:**

* number of tickets sold is of double type, measured in millions; average price of each ticket is of double type
* There should be two constructors, one is the default constructor with no parameter, and the other has six parameters to initialize the four inherited data fields, plus the two private data fields of its own, respectively.
* There should be a public getter and a public setter for each of the two private data fields.
* There are four effector methods:
  + **get category** : it should return String "drama".
  + **calculate revenue** : multiply number of tickets sold (measured in millions) with average price of each ticket, return double type, measured in term of million dollars.
  + **calculate profit** : subtract production cost (measured in million dollars) from the revenue ( also measured in million dollars), return double type
  + **toString** : it invokes the toString() in superclass first,then store parent’s toString() message in a local String variable, and then generates its own String message, which is like the line below (there is an "\n" symbol output at the end of this line):

This is an *drama* movie, and it generates *profit* (or *no profit*).

**Depending on whether the profit is positive value or not**, you should insert “profit” or “no profit” into the above message.

Then concatenate its own String message after its parent’s toString() message, and return the concatenated message. Please refer to class **SalesPerson**, on how to implement toString() method.

Precision requirements for formatting data items in class Movie and its subclasses in method toString():

* **The income, premium, revenue, profit, and production cost** in the above classes are **measured in terms of million dollars**, e.g., a result of 31.807 stands for 31.807 million dollars, and the required formatting precision is **3 digits after the decimal point**.
* The **number of tickets** sold in class **Drama** is measured in **million tickets**, and it is double type requiring **5 digitis after the decimal point**, for example, a value of 4.27618 is correct.
* The **ticket price** variable in class **Drama** is a double type and it requires **2 digits after the decimal point**.
* The **royalty rate** in class **Animated** should have **3 digits after the decimal point**.

Step 2: UML class diagram : JohnDoeMovie.jpg

You need to use Visual Pradigm to draw an UML class diagram including interface **Profitable**, class **Movie**, **Animated**, **Documentary** and **Drama**. Show the relationship between interface, superclass and subclasses. Export the UML class diagram to a file named **JohnDoeMovie.jpg**, and replace JohnDoe with your name. Please refer to the UML class diagram in file “versionComparison.docx”.

A remind here: you need to follow either version 1 or version 2 in file “exampleOfClassInterfaceDesign.zip”.

Step 3: Application file : JohnDoeHw8.java

Follow the steps below:

1. Create a new file called **JohnDoeHw8.java**. In its main method, the **first line of output** should be student’s name and class meeting time, such as **John Doe 5:30pm Wednesday**. Then do the following:
2. In the *main*() method, **create a Movie array of size 6**. **Populate this array with 2 Animated objects, and 2 Documentary objects, and 2 Drama objects**, with the order given. You need to hardcode the parameters by using the constructors with all parameters for the corresponding classe. Please refer to the main method in file “PayrollArrayDemo.java”, and see how to initialize an array of objects. No need to use loop here in this step.
3. Now create a method name **printMovieInfo**, and its **ONLY** parameter is the Movie type array. Inside this method, **use an “for-each” loop to navigate this array**. For each movie, output to the console the return value of method toString() in the movie’s subclass (Animated, Documentary, or Drama), and then output the revenue and profit of this movie to the console. **This method’s RETURN TYPE is void**.
4. Back to main method, invoke method **printMovieInfo**, and plug in the movie list as the only parameter.
5. Now create a method name **calcTotalProfit**( ), and its ONLY parameter is the Movie type array . Inside this method, **use a regular for loop to navigate this array**, and **sum the profits of all movies in the array, and return the total profit of all movies**. **The RETURN TYPE is double**, measured in terms of **million dollars**.
6. Back to main method, invoke method **calcTotalProfit**, and plug in movie array as the only parameter. In the main method, **use a local variable to received this method’s return value**, and then output this variable as the total profits of all movies together.
7. Now the main method finishes.

!!**Note** that there is no need to use Java class **ArrayList** or **Collection**. You just use a simple array of size 6 to hold the 6 movie items.

**--------------------------------------How to submit this homework: -----------------------------------**

After finishing the homework, copy file ***JohnDoeMovie.jpg***  into the folder where all the java source code locates, then generate the zip file ***JohnDoeHw8.zip***, which includes **ONLY** the following files:

* ***Profitable.java***( this is the interface )
* ***Movie.java***
* ***Animated.java***
* ***Documentary.java***
* ***Drama.java***
* ***JohnDoeHw8.java*** (this is the application file with the main method)
* ***JohnDoeMovie.jpg*** (this is the exported image file of the UML class diagram for class Movie and its subclasses, plus the interface Profitable)

Finally, you can submit the zip file to Moodle homework 8 drop box. After submission, be sure to follow the **VERIFICATION** process: download your .zip file to a local folder in your computer, unzip the zip file and then create a new Elcipse project, and include the extracted java files into the project, and compile then run the Eclipse project. If the verification process works correctly, then you have submitted the .zip file successfully in Moodle; if not, you need to fix the problem and re-submit. **DO NOT zip any eclipse project folder into your submission zip file**. If the submitted java files program does not compile or does not run, the score is zero.

For how to locate the java source code files, and how to zip multiple files into one zip file, please refer to “Homework7.docx” for instructions.

This assignment has **20 points. Grading components is listed in the table below:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Movie class & its subclasses, interface** | **main method implemented correctly** | **Correct UMLclass diagram** | **Code indent & align** | **program header &**  **comment** | **Java naming convention** | **Submit file format** | **Program logic** |
| **7** | **4** | **4** | **1** | **1** | **1** | **1** | **1** |

The grading components are, but not limited to: code alignment and indentation, variable/method/class naming convention, programmer header, suitable comments, submitted file format, overall program logic.

For Eclipse java file, if there is still any red circle check mark in java source code, you will receive zero point, because red circle check mark in Eclipse IDE means the java source code still has compilation error. You have to fix this compilation error first, before you proceed to the next phase of running java program. Even in some rare scenarios, you can run the program in Eclipse with compilation error, but a compilation error in java source code will result in zero point for the homework.

----------------------------------------**Several hints for this homework**-------------------------------------

***Hint 1: about the hardcoded data for movies in main method.***

You need to hardcode the data for the 6 objects in the movie array. The movie name, director name, and year must be real world data, and the classification must also be correct (being an animated, documentary, or drama movie). For the other data such as production cost, plus all the instance variables in the three child classes, you can make up your own data, and they do not have to be the real world statistics of the movie. If there are multiple directors for a single movie, pick ONLY one director. Of each movie category, manipulate your data, so that one movie generates profit, and the other does not.

***Hint 2: about invoking overridden method of the parent class.***

Think about this question: inside the child class’s method toString() , how to invoke the toString() method of the parent class Movie?

The answer is to use the keyword super as below:

Assume we are coding the toString() method inside the Animated class (a child class):

public String toString() // this method is inside the child class

{

String fromParent = **super.**toString(); // invoke the overridden method in the parent class

String myOwnMessage = String.format(……); // put this class’s own message here

return fromParent + myOwnMessage ; // return the concatenated messages

}

On how to use method *String.format()*, it is the same language syntax of method *System.out.printf();*

---------------------------------Summary of this homework----------------------------------

This homework covers materials in Moodle folder chapter 4, 6, 7, and 8.

Please treat this homework as **a learning opportunity** to review these four lessons. The main check points of this homework include:

chap 4: OOP principle of encapsulation, creation of OOP class with private data and public method

chap 6: array declaration and initialization; navigate array using loop

chap 7: abstract class, superclass and subclass , constructors in inherited class

chap 8: interface and abstract class design; overriding, dynamic binding, and parent reference variable refers to child objects (part of the polymorphism context)