Project: Marketing Campaign – Part 1

Deliverables

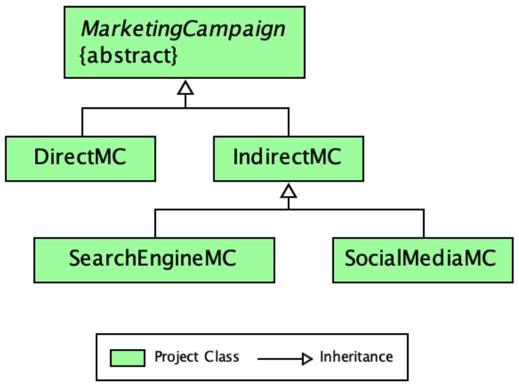
Your project files should be submitted to Web-CAT by the due date and time specified. Note that there is also an optional Skeleton Code assignment which will indicate level of coverage your tests have achieved (there is no late penalty since the skeleton code assignment is ungraded for this project). The files you submit to skeleton code assignment may be incomplete in the sense that method bodies have at least a return statement if applicable or they may be essentially completed files. In order to avoid a late penalty for the project, you must submit your completed code files to Web-CAT no later than 11:59 PM on the due date for the completed code assignment. If you are unable to submit via Web-CAT, you should e-mail your project Java files in a zip file to your TA before the deadline. The grades for the Completed Code submission will be determined by the tests that you pass or fail in your test files and by the level of coverage attained in your source files as well as usual correctness tests in Web-CAT.

Files to submit to Web-CAT:

- MarketingCampaign.java
- DirectMC.java, DirectMCTest.java
- IndirectMC.java, IndirectMCTest.java
- SearchEngineMC.java, SearchEngineMCTest.java
- SocialMediaMC.java, SocialMediaMCTest.java

Specifications

Overview: This project is the first of three that will involve the cost and reporting for marketing campaigns. You will develop Java classes that represent categories of marketing campaign (MC) including direct marketing campaign and indirect marketing campaign (both search engine and social media). As you develop each class, you should create the associated JUnit test file with the required test methods to ensure the classes and methods meet the specifications. You should create a jGRASP project upfront and then add the source and test files as they are created. All of your files should be in a single folder. Below is the UML class diagram for the required classes which shows the inheritance relationships.



You should read through the remainder of this assignment before you start coding.

- **MarketingCampaign.java**

  Requirements: Create an *abstract* MarketingCampaign class that describes marketing campaign data and provides methods to access the data.

  Design: The MarketingCampaign class has fields, a constructor, and methods as outlined below.

  (1) **Fields**:
  instance variables (*protected*) for: (1) name of type String and (2) revenue of type double.
  class variable (*protected static*) for the count of MarketingCampaign objects that have been created; set to zero when declared and then incremented in the constructor.
  These are the only fields that this class should have.

  (2) **Constructor**: The MarketingCampaign class must contain a constructor that accepts two parameters representing the instance variables (name and revenue) and then assigns them as appropriate. Since this class is abstract, the constructor will be called from the subclasses of MarketingCampaign using *super* and the parameter list. The count field should be incremented in the constructor.

  (3) **Methods**: Usually a class provides methods to access (or read) and modify each of its instance variables (known as get and set methods) along with any other required methods. At minimum you will need the following methods.
  - `getName`: Accepts no parameters and returns a String representing the name.
  - `setName`: Accepts a String representing the name, sets the field, and returns nothing.
  - `getRevenue`: Accepts no parameters and returns a double representing revenue for the marketing campaign.
  - `setRevenue`: Accepts a double representing revenue, sets the field, and returns nothing.
  - `getCount`: Accepts no parameters and returns an int representing the count. Since count is *static*, this method should be *static* as well.
  - `resetCount`: Accepts no parameters, resets count to zero, and returns nothing. Since count is *static*, this method should be *static* as well.
  - `calcROI`: Accepts no parameters and returns ROI calculated as follows:
    (revenue - campaignCost()) / campaignCost()
  - `toString`: Returns a String, which does not begin with \n, describing the MarketingCampaign object. This method may be invoked by a subclass as super.toString(). See the DirectMC class and IndirectMC class below. Note that you can get the class name for an instance c by calling c.getClass() [or if inside the class, this.getClass()]. For formatting patterns, use `"$#,##0.00"` for the *monetary values* and use `"0.##%"` for the ROI value. For IndirectMC mc0, the following should be produced by this method.

```
Ad Mailing (class DirectMC)
Revenue: $10,000.00   Campaign Cost: $7,000.00   ROI: 42.86%
```

- `campaignCost`: An *abstract* method that accepts no parameters and returns a double representing the cost of the marketing campaign.

Code and Test: Since the MarketingCampaign class is abstract you cannot create instances of MarketingCampaign upon which to call the methods. However, these methods will be inherited by the subclasses of MarketingCampaign. You should consider first writing skeleton code for the methods in order to compile MarketingCampaign so that you can create the first subclass described below, DirectMC. At this point you can begin completing the methods in MarketingCampaign and writing the JUnit test methods for DirectMCTest that tests the methods in MarketingCampaign and DirectMC.

- **<u>DirectMC.java</u>**

Requirements: Derive the class DirectMC.java from MarketingCampaign.

Design: The DirectMC class has fields, a constructor, and methods as outlined below.

(1) **Fields**:
instance variables (private) for (1) cost per mail piece of type double and (2) number of mail pieces of type int.
class constant (*public static final*) for BASE_COST of type double set to 1000, which should be referenced as DirectMC.BASE_COST outside the class. Note that a public field requires a Javadoc comment.
<u>These are the only fields that should be declared in this class</u>.

(2) **Constructor**: The DirectMC class must contain a constructor that accepts four parameters representing the two instance fields in the MarketingCampaign class (name and revenue) and the two instance fields (costPerMailPiece and numberOfMailPieces) declared in DirectMC. Since this class is a subclass of MarketingCampaign, the super constructor should be called with field values for MarketingCampaign. The instance variables for costPerMailPiece and numberOfMailPieces should be set with the last two parameters. Below is an example of how the constructor could be used to create a DirectMC object:

```
DirectMC mc0 = new DirectMC("Ad Mailing", 10000.00, 3.00, 2000);
```

(3) **Methods**: Usually a class provides methods to access (or read) and modify each of its instance variables (known as get and set methods) along with any other required methods. At minimum you will need the following methods.

- `getCostPerMailPiece`: Accepts no parameters and returns a double representing costPerMailPiece.
- `setCostPerMailPiece`: Accepts a double representing the costPerMailPiece, sets the field, and returns nothing.
- `getNumberOfMailPieces`: Accepts no parameters and returns an int representing numberOfMailPieces.
- `setNumberOfMailPieces`: Accepts an int representing the numberOfMailPieces, sets the field, and returns nothing.

- `campaignCost`: Accepts no parameters and returns a double representing the campaign cost for the DirectMC calculated as BASE_COST + (costPerMailPiece * numberOfMailPieces).
- `toString`: Returns a String, which does not begin with \n, describing the DirectMC object by calling parent's toString method, super.toString() and then appending the lines for base cost and mail cost. Below is an example of the toString result for DirectMC mc0 as it is declared above.

```
Ad Mailing (class DirectMC)
Revenue: $10,000.00   Campaign Cost: $7,000.00   ROI: 42.86%
   Base Cost: $1,000.00
   Mail Cost: $6,000.00 = $3.00 per mail piece * 2000 mail pieces
```

***Details*:** The first two lines above should be returned by the toString method in MarketingCampaign (see notes in that class). The amounts on the Revenue line are separated by three spaces (not tabs). For the formatting, use `"$#,##0.00"` for the *monetary values* and use `"0.##%"` for the ROI value. In this toString method, after calling super.toString(), the next two lines are appended with three leading spaces (not tabs).

Code and Test: As you implement the DirectMC class, you should compile and test it as methods are created. Although you could use interactions, it should be more efficient to test by creating appropriate JUnit test methods in DirectMCTest. You can now continue developing the methods in MarketingCampaign (parent class of DirectMC). The test methods in DirectMCTest should be used to test the methods in both MarketingCampaign and DirectMC. Remember, a DirectMC object *is-a* MarketingCampaign object which means DirectMC inherited the instance methods defined in MarketingCampaign. Therefore, you can create instances of DirectMC in order to test methods of the MarketingCampaign class.

- <u>**IndirectMC.java**</u>

Requirements: Derive the class IndirectMC from MarketingCampaign.

Design: The IndirectMC class has fields, a constructor, and methods as outlined below.

(1) **Fields:**
instance variables (*protected*): (1) cost per ad of type double and (2) number of ads of type int. These variables should be declared with the *protected* access modifier.
class constant (*public static final*) BASE_COST of type double set to 1500.0, which may be referenced as IndirectMC.BASE_COST outside the class. Note that a public field requires a Javadoc comment.
<u>These are the only fields that should be declared in this class</u>.

(2) **Constructor**: The IndirectMC class must contain a constructor that accepts four parameters representing the two instance fields in the MarketingCampaign class (name and revenue) and the two instance fields (cost per ad and number of ads) declared in IndirectMC. Since this class is a subclass of MarketingCampaign, the super constructor should be called with field values for MarketingCampaign. The instance variables for cost per ad and number of ads

should be set with the last two parameters. Below is an example of how the constructor could be used to create a IndirectMC object:

```
IndirectMC mc1 = new IndirectMC("Web Ads 1", 15000.00, 2.0, 3500);
```

(3) **Methods**: Usually a class provides methods to access (or read) and modify each of its instance variables (known as get and set methods) along with any other required methods. At minimum you will need the following methods.

- o `getCostPerAd`: Accepts no parameters and returns a double representing cost per ad.
- o `setCostPerAd`: Accepts a double representing the cost per ad, sets the field, and returns nothing.
- o `getNumberOfAds`: Accepts no parameters and returns an int representing the number of ads.
- o `setNumberOfAds`: Accepts an int representing the number of ads, sets the field, and returns nothing.
- o `getBaseCost`: Accepts no parameters and returns a double representing the constant BASE_COST. This is an instance method (i.e., it should not be static) so that we can access the BASE_COST associated with this class.
- o `campaignCost`: Accepts no parameters and returns a double representing the campaign cost for the DirectMC calculated as getBaseCost() + (costPerAd * numberOfAds).
- o `toString`: Returns a String, which does not begin with \n, describing the IndirectMC object by calling parent's toString method, super.toString() and then appending the lines for base cost and ad cost. Be sure to call the getBaseCost method for the BASE_COST value. Below is an example of the toString results for IndirectMC mc1 as it is declared above. The *monetary values should use* `"$#,##0.00"` for the formatting pattern. See the previous class for details for the first two lines below from the toString in MarketingCampaign.

```
Web Ads 1 (class IndirectMC)
Revenue: $15,000.00   Campaign Cost: $8,500.00   ROI: 76.47%
   Base Cost: $1,500.00
   Ad Cost: $7,000.00 = $2.00 per ad * 3500 ads
```

Code and Test: As you implement this class, you should compile and test it as methods are created. For example, as soon you have implemented and successfully compiled the constructor, you should create an instance of this class in a JUnit test method in the corresponding test class and then run the test file. If you want to view your objects in the Canvas, set a breakpoint in your test method the run *Debug* on the test file. When it stops at the breakpoint, step until the object is created. Then open a canvas window using the canvas button at the top of the Debug tab. After you drag the instance onto the canvas, you can examine it for correctness. If you change the viewer to "toString" view, you can see the formatted toString value. You can also enter the object variable name in interactions and press ENTER to see the toString value. *Hint: If you use the same variable names for objects in the test methods, you can use the menu button on the viewer in the canvas to set "Scope Test" to "None". This will allow you to use the same canvas with multiple test methods.*

- **SearchEngineMC.java**

  Requirements: Derive the class SearchEngineMC from IndirectMC.

  Design: The SearchEngineMC class has a field, a constructor, and methods as outlined below.

  (1) **Field**: **constant (*public static final*)** BASE_COST of type double set to 2000.0, which can be referenced as SearchEngineMC. BASE_COST outside of the class. Note that a public field requires a Javadoc comment.
  This is the only fields that should be declared in this class.

  (2) **Constructor**: The SearchEngineMC class must contain a constructor that accepts four parameters representing the two instance fields in the MarketingCampaign class (name and revenue) and the two instance fields (cost per ad and number of ads) declared in IndirectMC. Since this class is a subclass of IndirectMC, the super constructor should be called with the four parameters. Below is an example of how the constructor could be used to create a SearchEngineMC object:

```
SearchEngineMC mc2 = new SearchEngineMC("Web Ads 2", 27500.00, 2.50, 5000);
```

  (3) **Methods**: Usually a class provides methods to access (or read) and modify each of its instance variables (known as get and set methods) along with any other required methods. At minimum you will need the following methods.
  - `getBaseCost`: Accepts no parameters and returns a double representing the BASE_COST for this class. Although this method is returning a constant, it should not be *static* to ensure BASE_COST for this class is returned of when called on an object of this class.
  - **There is no `campaignCost` method in this class**: When `campaignCost` is invoked on an instance of SearchEngineMC, the `campaignCost` method inherited from IndirectMC is called.
  - **There is no `toString` method in this class**. When toString is invoked on an instance of SearchEngineMC, the toString method inherited from IndirectMC is called. Below is an example of the toString result for SearchEngineMC mc2 as it is declared above.

```
Web Ads 2 (class SearchEngineMC)
Revenue: $27,500.00   Campaign Cost: $14,500.00   ROI: 89.66%
   Base Cost: $2,000.00
   Ad Cost: $12,500.00 = $2.50 per ad * 5000 ads
```

  Code and Test: As you implement the SearchEngineMC class, you should compile and test it as methods are created. For details, see **Code and Test** above for the IndirectMC class.

- **SocialMediaMC.java**

  Requirements: Derive the class SocialMediaMC from class IndirectMC.

  Design: The SocialMediaMC class has a field, a constructor, and methods as outlined below.

  (1) **Field**: **constant (*public static final*)** BASE_COST of type double set to 3000.0, which can be referenced as SocialMediaMC.BASE_COST outside the class. Note that a public field requires a Javadoc comment.
  This is the only field that should be declared in this class.

  (2) **Constructor**: The SocialMediaMC class must contain a constructor that accepts four parameters representing the two instance fields in the MarketingCampaign class (name and revenue) and the two instance fields (cost per ad and number of ads) declared in IndirectMC. Since this class is a subclass of IndirectMC, the super constructor should be called with the four parameters. Below is an example of how the constructor could be used to create a SocialMediaMC object:

```
SocialMediaMC mc3 = new SocialMediaMC("Web Ads 3", 35000.00, 3.00, 8000);
```

  (3) **Methods**: Usually a class provides methods to access (or read) and modify each of its instance variables (known as get and set methods) along with any other required methods. At minimum you will need the following methods.
  - `getBaseCost`: Accepts no parameters and returns a double representing the cost factor. Although this method is returning a constant, it should not be *static* to ensure the BASE_COST for this class is returned of when called on an object of this class.
  - **There is no `toString` method in this class**. When toString is invoked on an instance of SocialMediaMC, the toString method inherited from IndirectMC is called. Below is an example of the toString result for SocialMediaMC mc3 as it is declared above.

```
Web Ads 3 (class SocialMediaMC)
Revenue: $35,000.00   Campaign Cost: $27,000.00   ROI: 29.63%
   Base Cost: $3,000.00
   Ad Cost: $24,000.00 = $3.00 per ad * 8000 ads
```

  Code and Test: As you implement the SocialMediaMC class, you should compile and test it as methods are created. For details, see **Code and Test** above for the IndirectMC class.