


## Deliverables

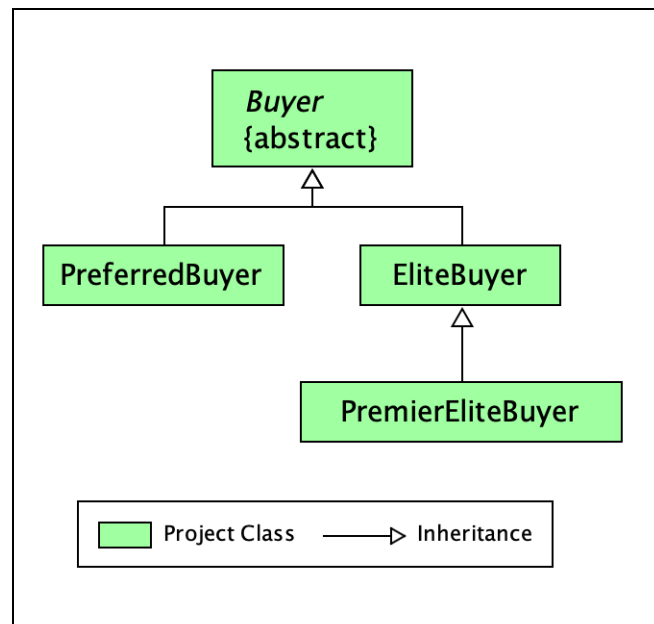
Your project files should be submitted to Web-CAT by the due date and time specified. Note that there is also an optional Skeleton Code assignment which will indicate level of coverage your tests have achieved (there is no late penalty since the skeleton code assignment is ungraded for this project). 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. 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. The level of coverage required is method, statement, and condition.

Files to submit to Web-CAT:

- Buyer.java, *no test file (abstract class)* - methods should be tested in PreferredBuyerTest.java
- PreferredBuyer.java, PreferredBuyerTest.java
- EliteBuyer.java, EliteBuyerTest.java
- PremierEliteBuyer.java, PremierEliteBuyerTest.java

## Specifications – Use arrays in this project; ArrayLists are not allowed!

**Overview:** This project is the first of three to process the monthly purchases made by members of the “Buyers” online store. Part 1 of the project focuses on the implementation of the classes of buyers. You will develop Java classes that represent categories of buyers, including Buyer (an abstract class), PreferredBuyer, EliteBuyer, and PremierEliteBuyer. Note that there is no requirement for a class with a main method in this project. You will need to create a JUnit test file for the indicated classes and write one or more test methods to ensure the classes and methods meet the specifications. You should create a jGRASP project and add the source and test files as they are created. All your files should be in a single folder. The UML class diagram at right provides a visual overview of how the classes in the project relate to one another. As you add your classes to the jGRASP project, you should generate the UML class diagram by double-clicking  <UML> for the project in the Open Projects window. Once generated, you can use the mouse to select/drag a class to arrange the diagram similar to the one shown. Alternatively, select Buyer, right-click on it and select Layout > Tree Down, then select Dependents of Selected.



---

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

- **Buyer.java**

**Requirements:** The Buyer class is an *abstract* class from which all to Buyer classes are derived. It contains fields and methods that will be inherited by the classes derived from Buyer.

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

- (1) **Fields:** protected instance variables for category, acctNumber, name (all String objects), and an array of type double representing purchases (note: the double array contains a list of dollar amounts of purchases);  
and a public constant SALES\_TAX\_RATE set to 0.08 (a double).
- (2) **Constructor:** The constructor accepts character strings for the Buyer's acctNumber and name as and assigns the fields accordingly. The constructor should also create the purchases array of type double with initial length of 0.
- (3) **Methods:** Usually a class provides methods to access and modify each of its instance variables (i.e., getters and setters) along with any other required methods. The methods for Buyer are described below.
  - `getAcctNumber` and `setAcctNumber` are the getter and setter methods for the acctNumber fields. The getter accepts no parameters and returns a String. The set method accepts a String and does not return a value.
  - `getName` and `setName` are the getter and setter methods for the name field. The getter accepts no parameters and returns a String. The set method accepts a String and does not return a value.
  - `getPurchases` and `setPurchases` are the getter and setter methods for the purchases array field. The set method accepts a double[] and does not return a value. The getter accepts no parameters and returns a double[].
  - `addPurchases` has no return value and accepts a variable length parameter list of type double and appends them to the end of the current purchases array. Before values are added/appended, the purchases array should be copied and replaced with a new array of appropriate length so that the length of the array reflects the number of values in the array. Note that `java.util.Arrays` contains a static method called `Arrays.copyOf` which takes two parameters (double[] original, int newLength) and returns the copy of the array with increased or decreased length. For example, the following statement makes a copy of an array called purchases and with the length increased by 1, and then replaces the previous array with the new one.

```
purchases = Arrays.copyOf(purchases, purchases.length + 1);
```
  - `deletePurchases` has no return value and accepts a variable length parameter list of type double and deletes them from the purchases array. A value is deleted from the purchases array by locating the value to be removed and then copying each element to the right one place to the left, thus, overwriting the element to be removed. The purchases array should be copied and replaced with a new array which has a length one less than the previous array so that the length of the array reflects the number of values in the array. If a value to be deleted is not found in the purchases array, the array is left unchanged. You may want to create a

private method that deletes a single purchase and then call it from the deletePurchases method described here as it loops through the array of purchases to be deleted.

- calcSubtotal returns a double representing the sum of all elements in the purchase array before tax.
- calcTotal returns a double representing subtotal plus the sales tax calculated on subtotal.
- toString returns a String describing a Buyer (see example output below for PreferredBuyer pb in Code and Test). This method should use the DecimalFormat class with the pattern ("\$.##0.00") to format the values representing dollars as shown in the example return value for a PreferredBuyer.  
Preferred Buyer  
AcctNo/Name: 10001 Smith, Sam  
Subtotal: \$548.00  
Tax: \$43.84  
Total: \$591.84  
Award Points: 548
- calcAwardPoints is an *abstract* method that returns an int representing the award points earned on the subtotal for the purchases array.

**Code and Test:** Since the Buyer class is abstract you cannot create instances of Buyer upon which to call the methods. However, these methods will be inherited by the subclasses of Buyer. You should consider first writing skeleton code for the methods to compile Buyer so that you can create the first subclass described below. At this point you can begin completing the methods in Buyer and writing the JUnit test methods for your subclass that tests the methods in Buyer. The toString example above is based on pb created as follows (see Notes on last page for an alternative):

```
PreferredBuyer pb = new PreferredBuyer("10001", "Smith, Sam");  
pb.addPurchases(34.5, 100.0, 63.50, 350.0);
```

- **PreferredBuyer.java**

**Requirements:** The PreferredBuyer class is derived from the Buyer class. Objects of this class will receive one award point per dollar spent.

**Design:** The Preferred class has fields, constructor, and methods as outlined below.

- (1) **Fields:** No fields are added, but category, acctNumber, name, purchases array, and SALES\_TAX\_RATE are inherited from Buyer.
- (2) **Constructor:** The constructor accepts character strings for the Buyer's acctNumber and name and passes them to the super class's constructor. It should also set the category field to "Preferred Buyer".
- (3) **Methods:** The methods in Buyer are inherited; the following methods need to be added:
  - calcAwardPoints overrides the abstract method inherited from the Buyer and returns an int calculated at one point per dollar of the subtotal.

**Code and Test:** Since the parent of this class was abstract, the inherited methods, as well as the method created in this class, should be tested in the JUnit test file for this class. You will need to complete any inherited methods that have not been completed. It is common to complete the methods in the source files as you develop the associated test methods in the JUnit test file. You also may want to create a driver class with a main method that creates instances of the class and invokes the toString method and perhaps others to get you started.

- **EliteBuyer.java**

**Requirements:** The EliteBuyer class is derived from the Buyer class. An Object of this class will receive 10 award points per dollar spent as well as a 5% discount on the subtotal.

**Design:** The EliteBuyer class has fields, constructor, and methods as outlined below.

- (1) **Fields:** Adds a field for discountRate set to .05, a double. Other fields are inherited from Buyer. *This instance variable should be protected rather than private.*
- (2) **Constructor:** The constructor accepts character strings for the Buyer's acctNumber and name and passes them to the super class's constructor. It should also set the category field to "Elite Buyer".
- (3) **Methods:** The methods in Buyer are inherited; the following methods need to be added:
  - getDiscountRate and setDiscountRate are the getter and setter methods for the discountRate field. The getter accepts no parameters and returns a double. The set method accepts a double and does not return a value.
  - calcAwardPoints overrides the abstract method inherited from the Buyer and returns an int calculated at 10 points per dollar of the subtotal.
  - calcSubtotal overrides the inherited calcSubtotal method to include the discount indicated by the discountRate field.
  - toString overrides the inherited toString to return a String that includes a statement that a discount was applied to subtotal (see example return value below for EliteBuyer eb in Code and Test).

```
Elite Buyer
AcctNo/Name: 10002 Jones, Pat
Subtotal: $473.10
Tax: $37.85
Total: $510.95
Award Points: 4730
(includes 0.05 discount rate applied to Subtotal)
```

**Code and Test:** As this class and its methods are implemented, the methods should be tested in the JUnit test file for this class. You also may want to create a driver class (e.g., BuyerApp) with a main method that creates instances of the class and invokes the toString method and perhaps others to get you started. The toString example above is based on eb created as follows (see Notes on last page for an alternative):

```
EliteBuyer eb = new EliteBuyer("10002", "Jones, Pat");
eb.addPurchases(34.5, 100.0, 63.50, 300.0);
```

- **PremierEliteBuyer.java**

**Requirements:** The PremierEliteBuyer class is derived from the EliteBuyer class. An Object of this class will receive 20 award points per dollar spent as well as a 15% discount on the subtotal. Also, if the subtotal exceeds \$500.00, a bonus of 1000 award points is earned.

**Design:** The PremierEliteBuyer class has fields, constructor, and methods as outlined below.

- (1) **Fields:** Adds a field for bonusAwardPoints set to 1000, an int. Other fields are inherited from Buyer and EliteBuyer. *This instance variable should be private.*
- (2) **Constructor:** The constructor accepts character strings for the Buyer's acctNumber and name and passes them to the super class's constructor. It also sets the category field to "Premier Elite Buyer" and sets the discountRate field to 0.15 (this field was inherited from EliteBuyer).
- (3) **Methods:** The methods in Buyer are inherited; the following methods need to be added:
  - `getBonusAwardPoints` and `setBonusAwardPoints` are the getter and setter methods for the bonusAwardPoints field. The getter accepts no parameters and returns an int. The set method accepts an int and does not return a value.
  - `calcAwardPoints` overrides the inherited method and returns an int calculated at 20 points per dollar of the subtotal. If the subtotal is greater than \$500.00, the value of bonusAwardPoints is also added.
  - `toString` overrides the inherited toString to returns a String that includes the toString of the superclass and if the subtotal is greater than \$500, a statement indicating that bonus points were added to the award points (see two example return values below).


```
Premier Elite Buyer
AcctNo/Name: 10003 King, Kelly
Subtotal: $423.30
Tax: $33.86
Total: $457.16
Award Points: 8460
(includes 0.15 discount rate applied to Subtotal)
```

```
Premier Elite Buyer
AcctNo/Name: 10004 Jenkins, Jordan
Subtotal: $508.30
Tax: $40.66
Total: $548.96
Award Points: 11160
(includes 0.15 discount rate applied to Subtotal)
(includes 1000 bonus points added to Award Points)
```

**Code and Test:** As this class and its methods are implemented, the methods should be tested in the JUnit test file for this class. You also may want to create a driver class with a main method that creates instances of the class and invokes the toString method and perhaps others to get you started. The toString example above is based peb1 and peb2 created as follows (see Notes on last page for alternatives):

```
PremierEliteBuyer peb1 = new PremierEliteBuyer("10003", "King, Kelly");
peb1.addPurchases(34.5, 100.0, 63.50, 300.0);

PremierEliteBuyer peb2 = new PremierEliteBuyer("10004", "Jenkins, Jordan");
peb2.addPurchases(34.5, 100.0, 63.50, 300.0, 100.0);
```

**UML Class Diagram:** If you have not already done so, add your classes to the jGRASP project, then generate the UML class diagram by double-clicking  <UML> for the project in the Open Projects window. Once generated, you can use the mouse to select/drag a class to arrange the diagram like the one on page 1. Alternatively, select Buyer, right-click on it and select Layout > Tree Down, then select Dependents of Selected.

## Notes

Here's an alternative way to create the example objects above.

```
PreferredBuyer pb = new PreferredBuyer("10001", "Smith, Sam");
double[] pbPurchases = {34.5, 100.0, 63.50, 350.0};
pb.setPurchases(pbPurchases);

EliteBuyer eb = new EliteBuyer("10002", "Jones, Pat");
double[] ebPurchases = {34.5, 100.0, 63.50, 300.0};
eb.setPurchases(ebPurchases);

PremierEliteBuyer peb1 = new PremierEliteBuyer("10003", "King, Kelly");
double[] peb1Purchases = {34.5, 100.0, 63.50, 300.0};
peb1.setPurchases(peb1Purchases);

PremierEliteBuyer peb2 = new PremierEliteBuyer("10004", "Jenkins, Jordan");
double[] peb2Purchases = {34.5, 100.0, 63.50, 300.0, 100.0};
peb2.setPurchases(peb2Purchases);
```