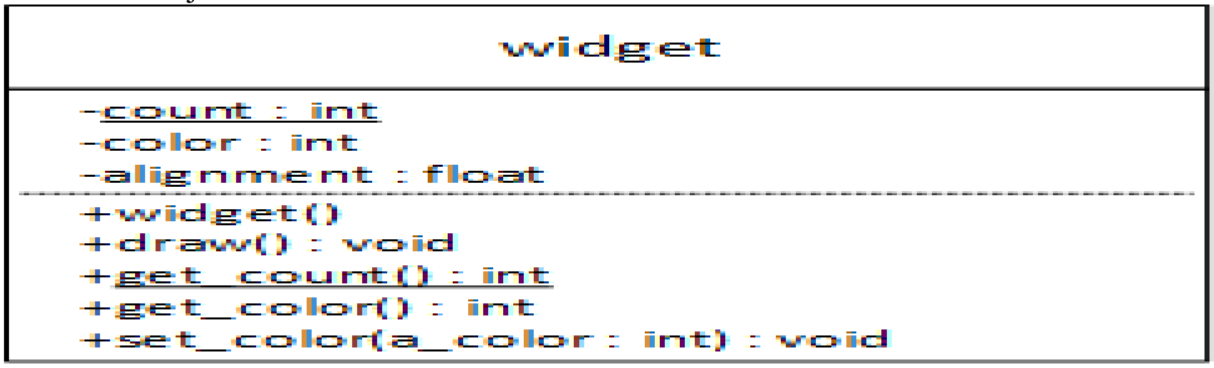
Advanced Programming (Java)

Week 3 Work

(To be completed during the week)

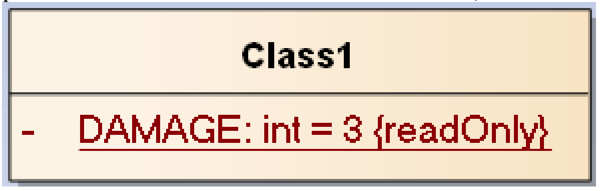
Example 1 - static And UML Diagrams

Non-static class features (attributes or operations) belong to individual instances of the class (i.e., to an object). These features are bound to the object with the "this" pointer. On the other hand, static features belong to the class as a whole. Unless otherwise specified, features in a UML class diagram are non-static. The UML denotes static features by underlining the feature in the class diagram. The underlining is translated into the static keyword when the UML class diagram is translated into java code.



Denoting static features in a UML class diagram. The UML denotes static features by underlining them. The static keyword may modify attributes and operations alike, and is independent of other modifiers such as public, private, or later) protected.

private static final int DAMAGE = 3; in a uml diagram.



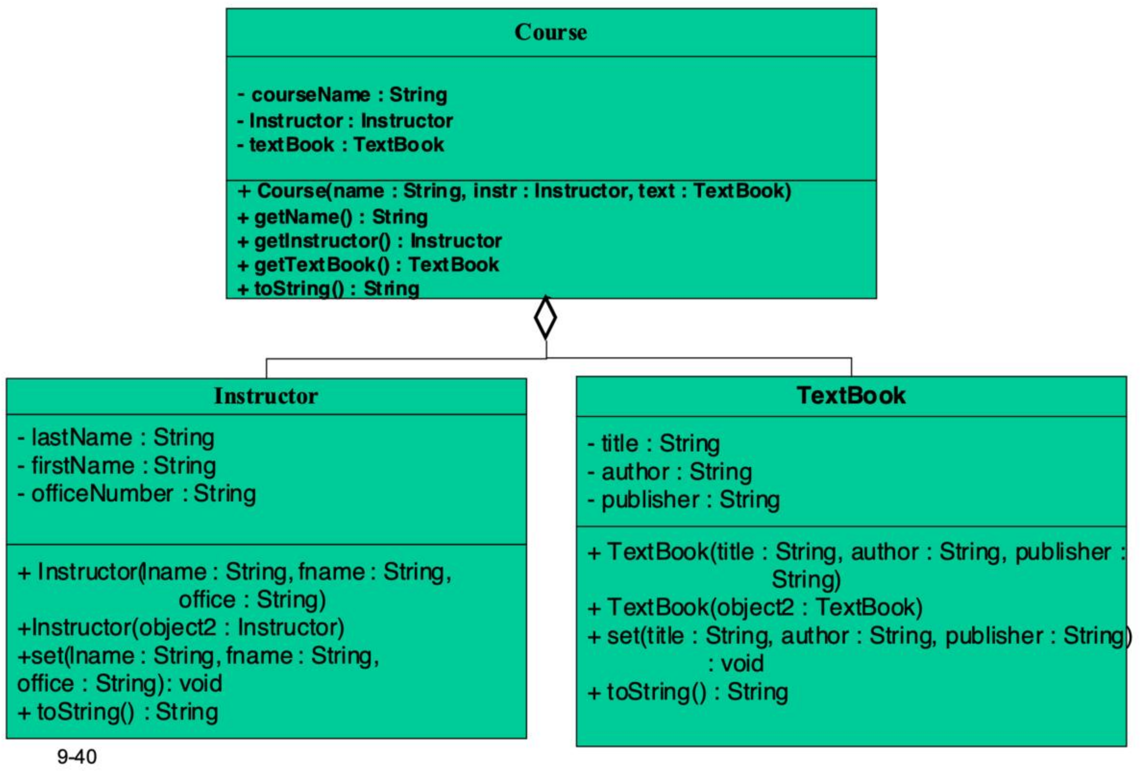
Private is symbolized by a minus. The static attribute is shown by an underline. The initial value is shown by = <value>. Since, we learned that final denotes a constant, which is shown as {readOnly}.

Example 2 – Method Overloading

A program calculates and displays bonus amounts to pay various types of employees. There are 3 separate departments, numbered 1, 2, and 3. Department 1 employees are paid a bonus based on their sales: If their sales amount is over $5000 they get 5% of those sales, otherwise they get nothing. Department 2 employees are paid a bonus based on the number of units they sell: They get $20 per unit sold, and an extra $10 per unit if they sell 25 units or more; if they sell no units, they get nothing. Department 3 employees assemble parts in the plant and are paid a bonus of 10 cents per part if they reach a certain level: Part-time employees must assemble more than 250 parts to get the 10-cent-per-part bonus, and full-time employees must assemble more than 700. Write a set of 3 overloaded methods called getBonus() that works with the program below, according to the specifications described above.

```
public class Bonus{

public final static int UNITS_PT = 250;
public final static int UNITS_FT = 700;
public final static double SALES_BONUS = 5000.0;
public final static double SALES_BONUS_RATE = 0.05;
public final static double SALES_UNIT_REG = 20.0;
public final static double SALES_UNIT_EXTRA = 10.0;
public final static int SALES_UNIT_BONUS = 25;
public final static double PARTS_BONUS = 0.1;

  public static void main(String[] args) {

  Scanner keysIn = new Scanner(System.in);
  System.out.println("Enter department: ");
  int dept = keysIn.nextInt();
  double bonus = 0;

  switch (dept)   {
    case 1:
      System.out.print("Enter sales: ");
      double sales = keysIn.nextDouble();
      bonus = getBonus(sales);
      break;
    case 2:
      System.out.print("Enter number of units sold: ");
      int numUnits = keysIn.nextInt();
      bonus = getBonus(numUnits);
      break;
    case 3:
      System.out.print("Enter # of pieces completed: ");
      int pieces = keysIn.nextInt();
      System.out.print("Full-time (1) or Part-Time (2)? ");
      int empType = keysIn.nextInt();
      int bonusLimit = (empType == 1) ? UNITS_FT : UNITS_PT;
      bonus = getBonus(pieces, bonusLimit);
      break;
    default:
      System.out.print("Error!  ");
   }
   System.out.printf("Bonus Amount: $%.2f%n", bonus);
}

//add your over loaded methods here
}
```
Example 3 – Aggregation\Composition

Aggregation in UML Diagrams

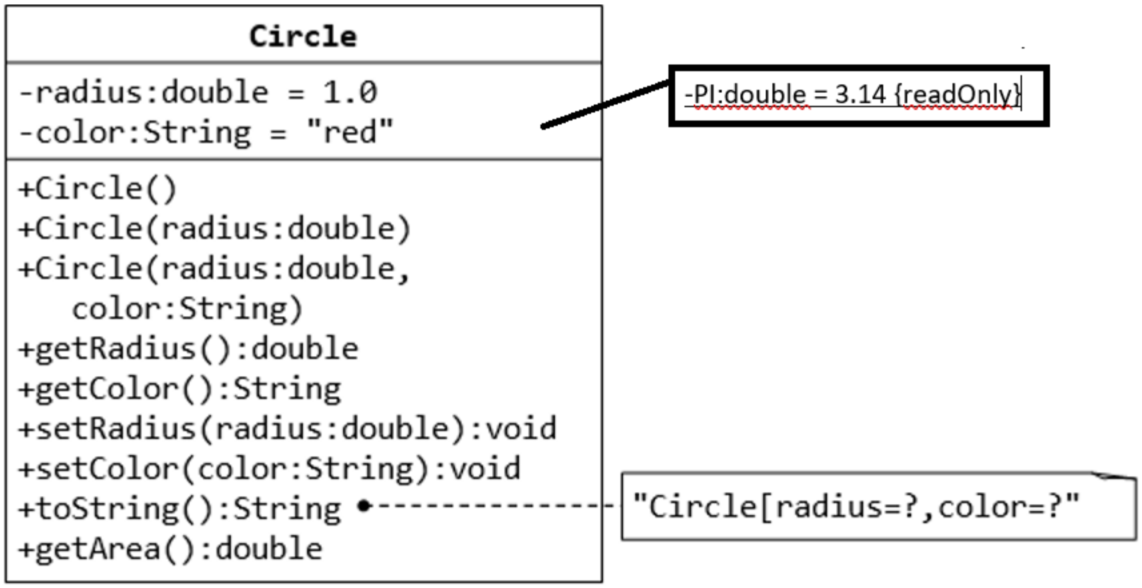

Course
- courseName : String
- Instructor : Instructor
- textBook : TextBook
+ Course(name : String, instr : Instructor, text : TextBook)
+ getName() : String
+ getInstructor() : Instructor
+ getTextBook() : TextBook
+ toString() : String
Instructor
- lastName : String
- firstName : String
- officeNumber : String
+ Instructor(lname : String, fname : String, office : String)
+Instructor(object2 : Instructor)
+set(lname : String, fname : String, office : String): void
+ toString() : String
TextBook
- title : String
- author : String
- publisher : String
+ TextBook(title : String, author : String, publisher : String)
+ TextBook(object2 : TextBook)
+ set(title : String, author : String, publisher : String) : void
+ toString() : String
9-40

<u>Daily Task</u>

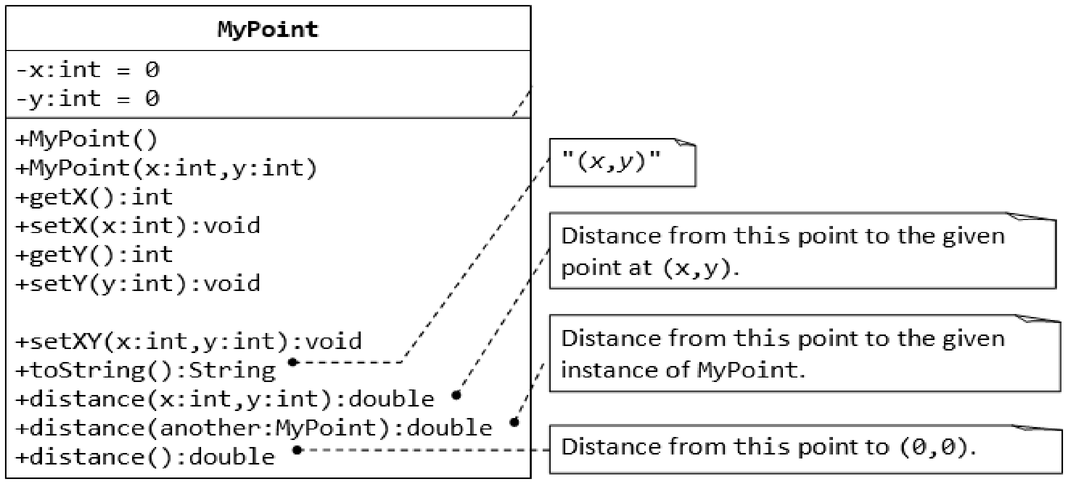
Task 1:

Remember Circle class from Week 2. Open the same Circle class, Add a static final variable PI = 3.14. Modify the getArea() method to use this final variable instead. Modify the main method to print the area of 2 circle objects at least.

Try to assign PI a new value in CircleDemo's main method (Circle.PI = 22/7) and test what happens.



Task 2(a): The MyPoint Class


MyPoint
-x:int = 0
-y:int = 0
+MyPoint()
+MyPoint(x:int,y:int)
+getX():int
+setX(x:int):void
+getY():int
+setY(y:int):void
+setXY(x:int,y:int):void
+toString():String
+distance(x:int,y:int):double
+distance(another:MyPoint):double
+distance():double
"(x,y)"
Distance from this point to the given point at (x,y).
Distance from this point to the given instance of MyPoint.
Distance from this point to (0,0).

A class called MyPoint, which models a 2D point with x and y coordinates, is designed as shown in the class diagram. It contains:

- Two instance variables x (int) and y (int).
- A default (or "no-argument" or "no-arg") constructor that construct a point at the default location of (0, 0).
- A overloaded constructor that constructs a point with the given x and y coordinates.
- Getter and setter for the instance variables x and y.
- A method setXY() to set both x and y.
- A toString() method that returns a string description of the instance in the format "(*x*, *y*)".
- A method called distance(int x, int y) that returns the distance from *this* point to another point at the given (x, y) coordinates, e.g.,
- MyPoint p1 = new MyPoint(3, 4);

```
System.out.println(p1.distance(5, 6));
```

- An overloaded distance(MyPoint another) that returns the distance from *this* point to the given MyPoint instance (called another), e.g.,

```
MyPoint p1 = new MyPoint(3, 4);
MyPoint p2 = new MyPoint(5, 6);
System.out.println(p1.distance(p2));
```

- Another overloaded distance() method that returns the distance from this point to the origin (0,0), e.g.,

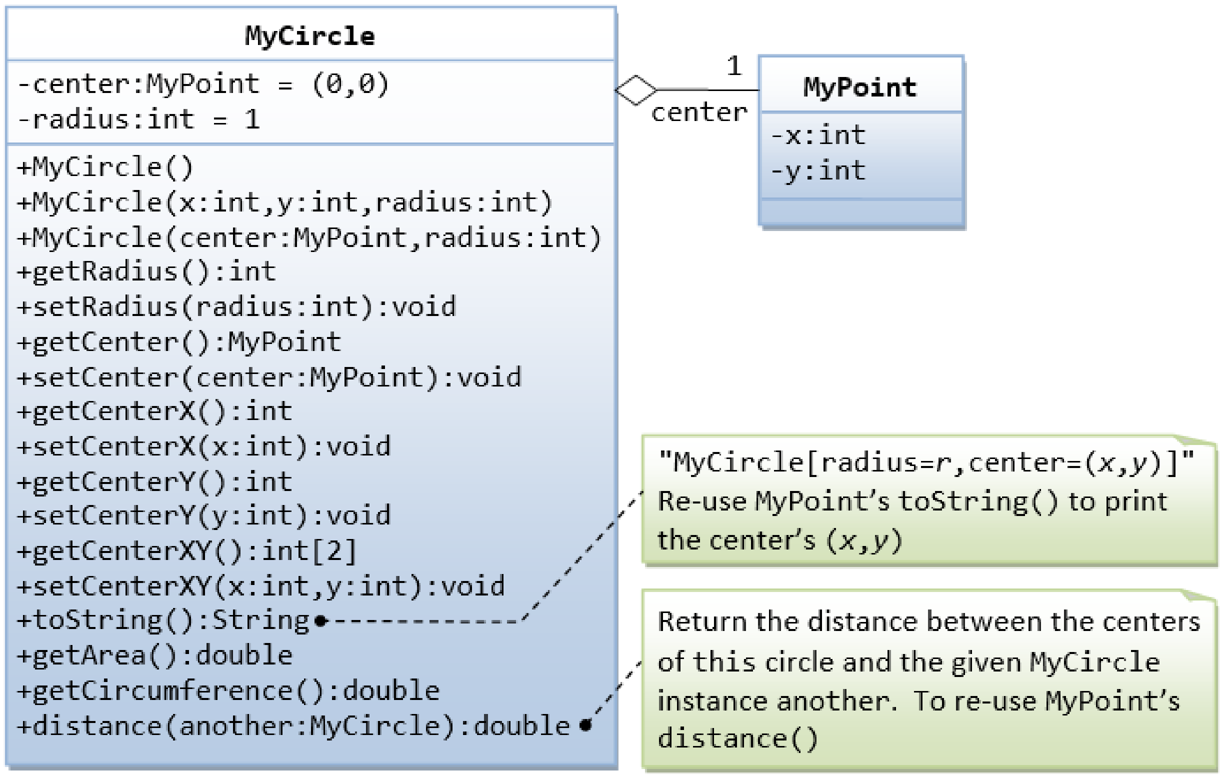
```
MyPoint p1 = new MyPoint(3, 4);
System.out.println(p1.distance());
```

You are required to:

1. Write the code for the class MyPoint. Also write a test program (called TestMyPoint) to test all the methods defined in the class.
   Hints:

```
// Overloading method distance()
// This version takes two ints as arguments
public double distance(int x, int y) {
  int xDiff = this.x – x;
  int yDiff = ......
  return Math.sqrt(xDiff*xDiff + yDiff*yDiff);
}

// This version takes a MyPoint instance as argument
public double distance(MyPoint another) {
  int xDiff = this.x – another.x;
  .......
}
// Test program to test all constructors and public methods
MyPoint p1 = new MyPoint();  // Test constructor
System.out.println(p1);      // Test toString()
p1.setX(8);   // Test setters
p1.setY(6);
System.out.println("x is: " + p1.getX());  // Test getters
System.out.println("y is: " + p1.getY());
p1.setXY(3, 0);   // Test setXY()
System.out.println(p1);

MyPoint p2 = new MyPoint(0, 4);  // Test another constructor
System.out.println(p2);
// Testing the overloaded methods distance()
System.out.println(p1.distance(p2));    // which version?
System.out.println(p2.distance(p1));    // which version?
System.out.println(p1.distance(5, 6));  // which version?
System.out.println(p1.distance());      // which version?
```

A class called `MyCircle`, which models a circle with a center and a radius, is designed as shown in the class diagram. The `MyCircle` class uses a `MyPoint` instance (written in the earlier exercise) as its center.



MyCircle
-center:MyPoint = (0,0)
-radius:int = 1
+MyCircle()
+MyCircle(x:int,y:int,radius:int)
+MyCircle(center:MyPoint,radius:int)
+getRadius():int
+setRadius(radius:int):void
+getCenter():MyPoint
+setCenter(center:MyPoint):void
+getCenterX():int
+setCenterX(x:int):void
+getCenterY():int
+setCenterY(y:int):void
+getCenterXY():int[2]
+setCenterXY(x:int,y:int):void
+toString():String
+getArea():double
+getCircumference():double
+distance(another:MyCircle):double
1
center
MyPoint
-x:int
-y:int
"MyCircle[radius=r,center=(x,y)]"
Re-use MyPoint's toString() to print the center's (x,y)
Return the distance between the centers of this circle and the given MyCircle instance another. To re-use MyPoint's distance()

The class contains:

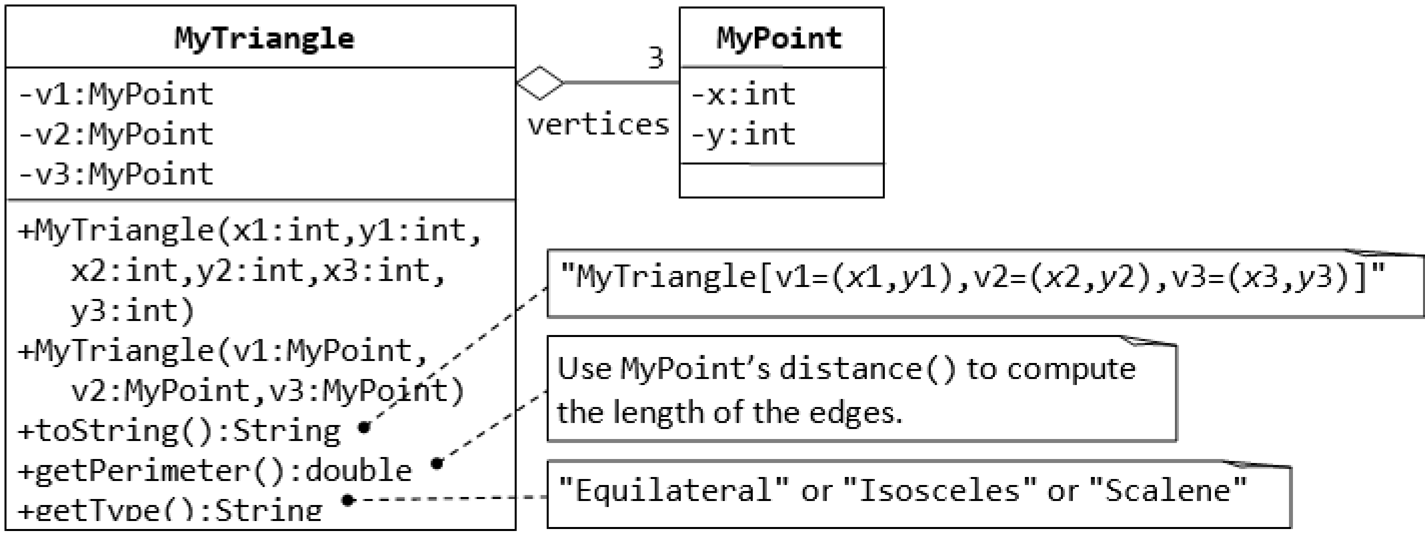
- Two `private` instance variables: `center` (an instance of `MyPoint`) and `radius` (`int`).
- A constructor that constructs a circle with the given center's (x, y) and `radius`.
- An overloaded constructor that constructs a `MyCircle` given a `MyPoint` instance as `center`, and `radius`.
- A *default* constructor that construct a circle with center at `(0,0)` and radius of `1`.
- Various getters and setters.
- A `toString()` method that returns a string description of this instance in the format "`MyCircle[radius=`*r*`,center=(`*x*`,`*y*`)]`". You shall reuse the `toString()` of `MyPoint`.
- `getArea()` and `getCircumference()` methods that return the area and circumference of `this` circle in `double`.
- A `distance(MyCircle another)` method that returns the distance of the centers from `this` instance and the given `MyCircle` instance. You should use `MyPoint`'s `distance()` method to compute this distance.

Write the `MyCircle` class. Also write a test driver (called `TestMyCircle`) to test all the public methods defined in the class.

Hints:

```
// Constructors
public MyCircle(int x, int y, int radius) {
   // Need to construct an instance of MyPoint for the variable center
   center = new MyPoint(x, y);
   this.radius = radius;
}
public MyCircle(MyPoint center, int radius) {
   // An instance of MyPoint already constructed by caller; simply assign.
   this.center = center;
   ......
}
public MyCircle() {
   center = new MyPoint(.....);  // construct MyPoint instance
   this.radius = ......
}

// Returns the x-coordinate of the center of this MyCircle
public int getCenterX() {
   return center.getX();   // cannot use center.x and x is private in MyPoint
}

// Returns the distance of the center for this MyCircle and another MyCircle
public double distance(MyCircle another) {
   return center.distance(another.center); // use distance() of MyPoint
}
```

Task 2(b): The MyTriangle and MyPoint Classes


MyTriangle
-v1:MyPoint
-v2:MyPoint
-v3:MyPoint
+MyTriangle(x1:int,y1:int, x2:int,y2:int,x3:int, y3:int)
+MyTriangle(v1:MyPoint, v2:MyPoint,v3:MyPoint)
+toString():String
+getPerimeter():double
+getType():String
3
vertices
MyPoint
-x:int
-y:int
"MyTriangle[v1=(x1,y1),v2=(x2,y2),v3=(x3,y3)]"
Use MyPoint's distance() to compute the length of the edges.
"Equilateral" or "Isosceles" or "Scalene"

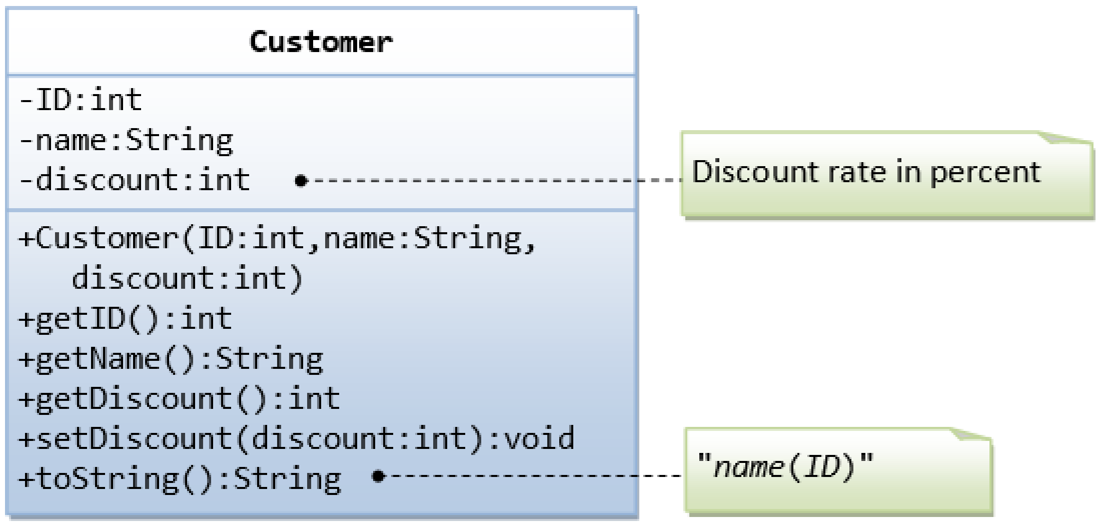
A class called MyTriangle, which models a triangle with 3 vertices, is designed as shown. The MyTriangle class uses three MyPoint instances (created in the earlier exercise) as its three vertices.
It contains:

- Three private instance variables v1, v2, v3 (instances of MyPoint), for the three vertices.
- A constructor that constructs a MyTriangle with three set of coordinates, v1=(x1, y1), v2=(x2, y2), v3=(x3, y3).
- An overloaded constructor that constructs a MyTriangle given three instances of MyPoint.

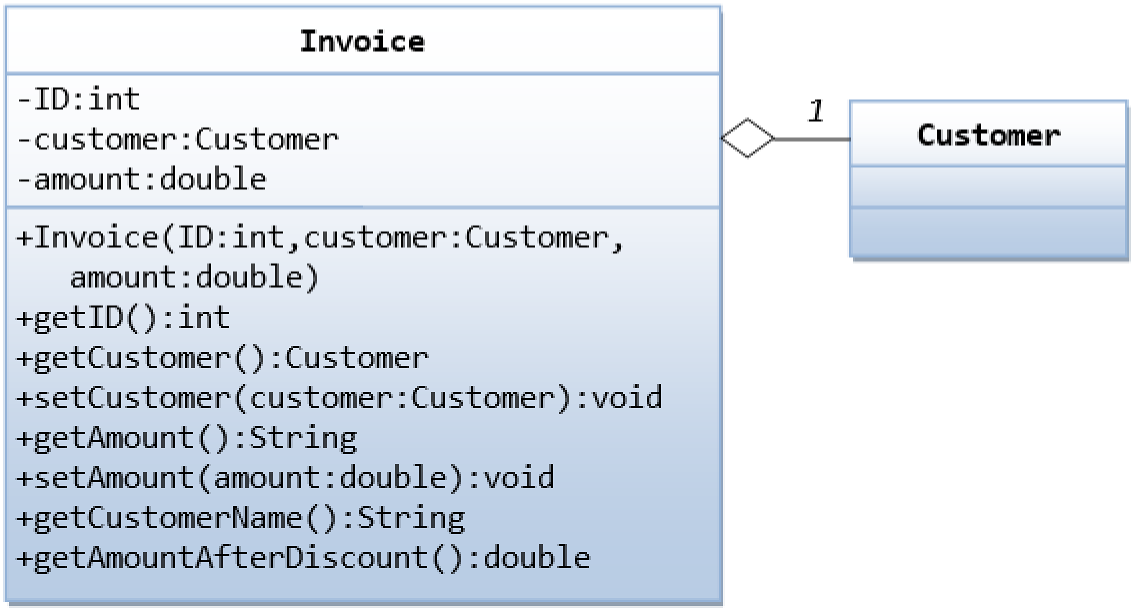
- A toString() method that returns a string description of the instance in the format "MyTriangle[v1=(*x*1,*y*1),v2=(*x*2,*y*2),v3=(*x*3,*y*3)]".
- A getPerimeter() method that returns the length of the perimeter in double. You should use the distance() method of MyPoint to compute the perimeter.
- A method printType(), which prints "equilateral" if all the three sides are equal, "isosceles" if any two of the three sides are equal, or "scalene" if the three sides are different.

Write the MyTriangle class. Also write a test driver (called TestMyTriangle) to test all the public methods defined in the class.

Task 3 (a): The Customer and Invoice classes

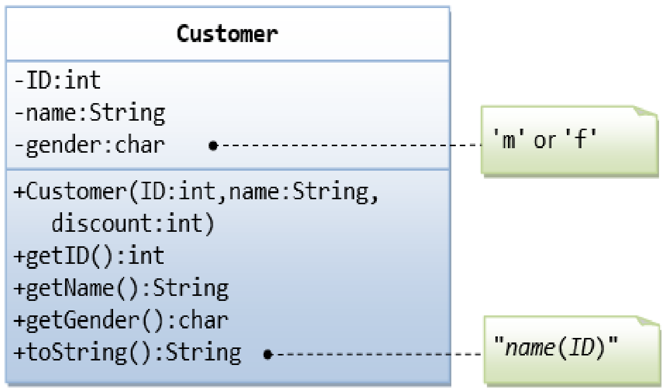


The Customer class models a customer is design as shown in the class diagram. Write the codes for the Customer class and a test driver to test all the public methods.

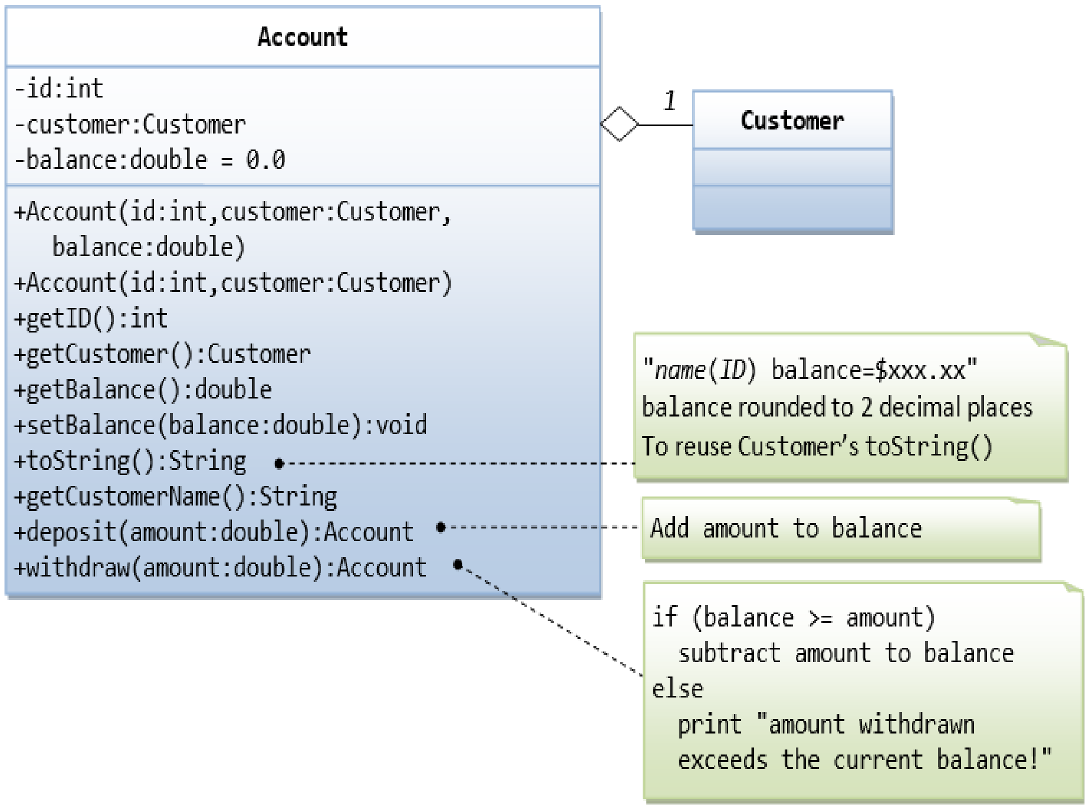

Invoice
-ID:int
-customer:Customer
-amount:double
+Invoice(ID:int,customer:Customer, amount:double)
+getID():int
+getCustomer():Customer
+setCustomer(customer:Customer):void
+getAmount():String
+setAmount(amount:double):void
+getCustomerName():String
+getAmountAfterDiscount():double
1
Customer

The Invoice class, design as shown in the class diagram, composes a Customer instance (written earlier) as its member. Write the codes for the Invoice class and a test driver to test all the public methods.

Task 3 (b): The Customer and Account classes



The Customer class models a customer is design as shown in the class diagram. Write the codes for the Customer class and a test driver to test all the public methods.


Account
-id:int
-customer:Customer
-balance:double = 0.0
+Account(id:int,customer:Customer, balance:double)
+Account(id:int,customer:Customer)
+getID():int
+getCustomer():Customer
+getBalance():double
+setBalance(balance:double):void
+toString():String
+getCustomerName():String
+deposit(amount:double):Account
+withdraw(amount:double):Account
1
Customer
"name(ID) balance=$xxx.xx"
balance rounded to 2 decimal places
To reuse Customer's toString()
Add amount to balance
if (balance >= amount)
subtract amount to balance
else
print "amount withdrawn exceeds the current balance!"

The Account class models a bank account, design as shown in the class diagram, composes a Customer instance (written earlier) as its member. Write the codes for the Account class and a test driver to test all the public methods.