

# TUTORIAL 3

---

Class Methods & Inheritance

# Q1

- Design a program to implement a vending machine for buying drinks. Write a class `VendingMachine` that has the following instance variables and methods:



# Q1

- Design a program to implement a vending machine for buying drinks. Write a class `VendingMachine` that has the following instance variables and methods:

```
public class VendingMachine {  
    public VendingMachine() {} // constructor  
    // select a drink, and return the cost of the drink  
    public double selectDrink() {...}  
    // insert the coins and returns the amount inserted  
    public double insertCoins(double drinkCost) {...}  
    // check the change and print it on screen  
    public void checkChange(double amount, double drinkCost) {...}  
    // print the receipt and collect the drink  
    public void printReceipt(){...}  
}
```

# Q1

- The UML class diagram for the `VendingMachine` class is given below:

VendingMachine
+ VendingMachine() + selectDrink(): double + insertCoins(drinkCost: double): double + checkChange(amount: double, drinkCost: double): void + printReceipt(): void

# Q1

- Write an application class `VendingMachineApp` to test the class `VendingMachine`. The program allows users to select the drink to buy, and accept coins inserted by the user to pay for the drink. The program will also print the receipt for user to collect the drink.

# Q1. Sample output

```
===== Vending Machine =====
|1. Buy Beer ($3.00) |
|2. Buy Coke ($1.00) |
|3. Buy Green Tea ($5.00) |
|=====
Please enter selection:
1
Please insert coins:
===== Coins Input =====
|Enter 'Q' for ten cents input |
|Enter 'T' for twenty cents input|
|Enter 'F' for fifty cents input |
|Enter 'N' for a dollar input |
=====
```

```
Q
Coins inserted: 0.10
T
Coins inserted: 0.30
F
Coins inserted: 0.80
N
Coins inserted: 1.80
N
Coins inserted: 2.80
N
Coins inserted: 3.80
Change: $ 0.80
Please collect your drink.
Thank You !!
```

# Constructor + VendingMachine()

```
import java.util.Scanner;  
public class VendingMachine {  
    public VendingMachine() { }
```

## + selectDrink(): double

```
public double selectDrink() {
    Scanner sc = new Scanner(System.in);
    int drinkSelection;
    double drinkCost=0;
    System.out.println("== Vending Machine ==\n" +
        "|1. Buy Beer ($3.00)|\n" +
        "|2. Buy Coke ($1.00)|\n" +
        "|3. Buy Green Tea ($5.00)|\n" +
        "|=====|\n");

    do {
        System.out.println("Please enter selection: ");
        drinkSelection = sc.nextInt();
    } while (drinkSelection < 1 || drinkSelection > 3);
    if (drinkSelection == 1)
        drinkCost = 3.00;
    else if (drinkSelection == 2)
        drinkCost = 1.00;
    else if (drinkSelection == 3)
        drinkCost = 5.00;
    return drinkCost;
}
```



## insertCoin(drinkCost: double): double

```
public double insertCoins (double drinkCost) {
    double amount=0.0;
    Scanner sc = new Scanner(System.in);
    System.out.println("Please insert coins:\n" +
        "==== Coins Input ==== \n" +
        "|Enter 'Q' for ten cents input|\n" +
        "|Enter 'T' for twenty cents input|\n" +
        "|Enter 'F' for fifty cents input |\n" +
        "|Enter 'N' for a dollar input    |\n" +
        "===== \n");
    do {
        char coin = sc.next().charAt(0);
        switch (coin) {
            case 'Q': case 'q': amount += 0.10; break;
            case 'T': case 't': amount += 0.20; break;
            case 'F': case 'f': amount += 0.50; break;
            case 'N': case 'n': amount += 1.00; break; }
        System.out.printf("Coins inserted: %.2f \n", amount);
    } while (amount < drinkCost);
    return amount;
}
```

+ checkChange(amount: double, drinkCost: double): void

```
public void checkChange(double amt, double drCost) {  
    double change=0.0;  
    if (amt > drCost) {  
        change = amt - drCost;  
        System.out.printf("Change: $ %.2f \n", change);  
    }  
}
```

+ printReceipt(): void

```
public void printReceipt() {  
    System.out.println("Please collect your drink\n" +  
        "Thank You !!");  
}
```

# Application class: VendingMachineApp

```
public class VendingMachineApp {  
    public static void main(String[] args) {  
        double drCost=0, amtInserted = 0.0;  
        VendingMachine vM = new VendingMachine();  
        drCost = vM.selectDrink();  
        amtInserted = vM.insertCoins(drCost);  
        vM.checkChange(amtInserted, drCost);  
        vM.printReceipt();  
    }  
}
```

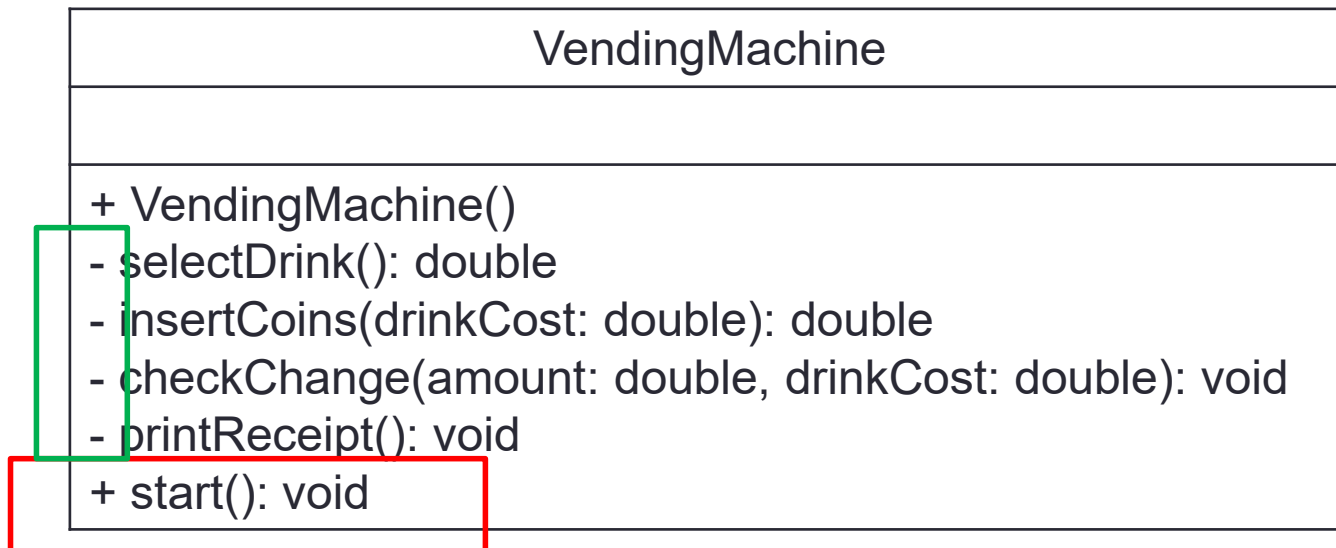
# Q1

- a) Discuss the design of the `VendingMachine` class and how it can be improved.



# A1

1. A start method in VendingMachine to get it going – VM should know how it operates
2. With start as public, the rest may just be declared private => info hiding



+ start(): void

```
public void start() {  
    boolean end = false ;  
    do {  
        drinkCost = selectDrink() ;  
        double amt = insertCoins(drinkCost) ;  
        checkChange(amt, drinkCost) ;  
        printReceipt() ;  
    }while(!end) ;  
}
```

Application class: VendingMachineApp

```
public class VendingMachineApp {  
    public static void main(String[] args) {  
        VendingMachine vm = new VendingMachine() ;  
        vm.start() ;  
    }  
}
```

- a) What will be a relevant class to relate to the `VendingMachine` class?
- b) Suggest how the application can be re-designed to involve the class in (b)?
- c) [Optional] Implement your design.





```
public double selectDrink() {
    Scanner sc = new Scanner(System.in);
    int drinkSelection;
    double drinkCost=0;
    System.out.println("== Vending Machine ===\n" +
        "|1. Buy Beer ($3.00)|\n" +
        "|2. Buy Coke ($1.00)|\n" +
        "|3. Buy Green Tea ($5.00)|\n" +
        "|=====|\n");

    do {
        System.out.println("Please enter selection: ");
        drinkSelection = sc.nextInt();
    } while (drinkSelection < 1 || drinkSelection > 3);
    if (drinkSelection == 1)
        drinkCost = 3.00;
    else if (drinkSelection == 2)
        drinkCost = 1.00;
    else if (drinkSelection == 3)
        drinkCost = 5.00;
    return drinkCost;
}
```



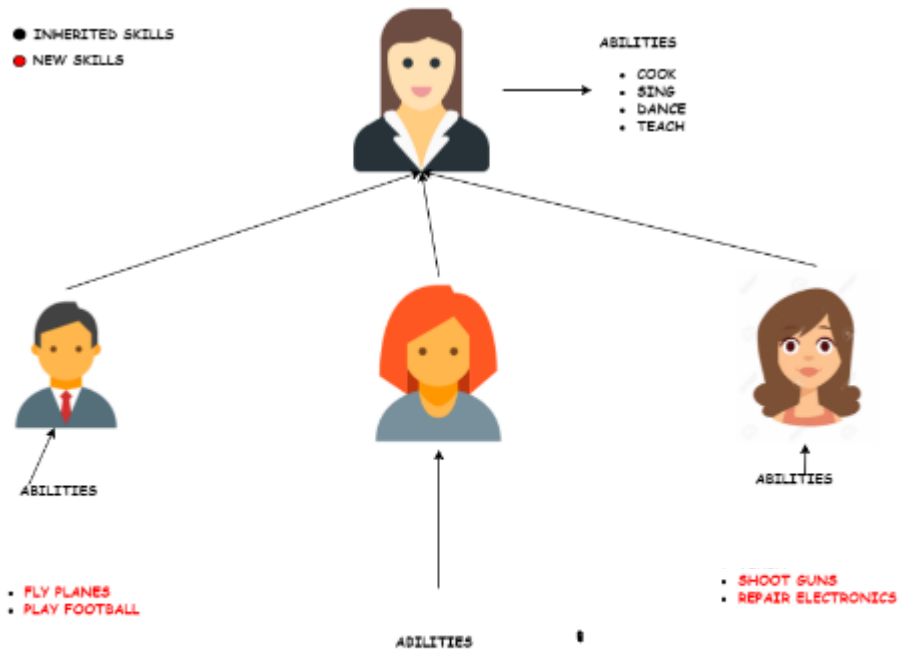
# A1

Introduce a Drink  
class to hold  
drink details



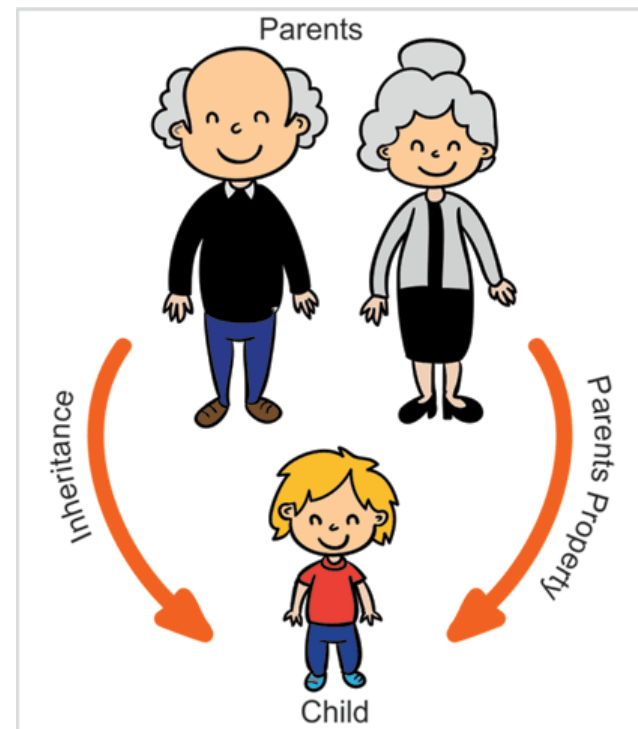
```
public class Drink {  
    private String name;  
    private double cost;  
    public Drink(String n, double c) {  
        name = n ;  
        cost = c;  
    }  
    public String getName() { return name;}  
    public double getCost() { return cost;}  
}
```





BIOSE NONSO EMMANUEL

CODE

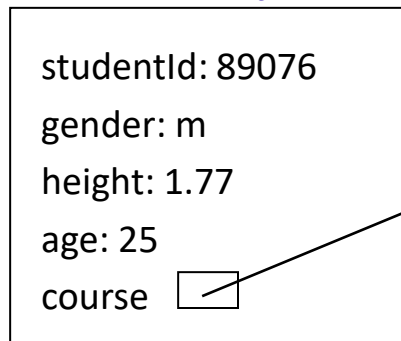


# Object Composition

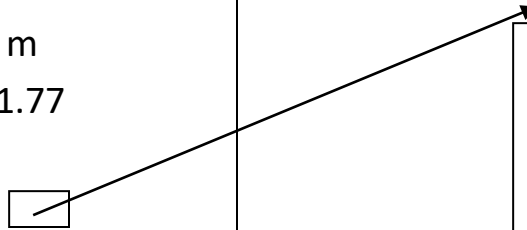
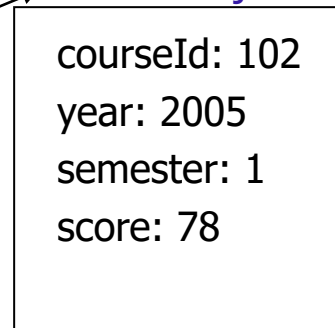
- An object can include other objects as its **data** member. This is called **object composition**.
- A class contains **object references** from other classes as its **instance variables**.
- Object references are of reference data types.
- In OO, this is called the “**has-a**” relationship.
- Example: a student *has a* course.

```
public class Student {  
    private int    studentId;  
    private char   gender;  
    private double height;  
    private int    age;  
    private Course course;  
}
```

Student object



Course object



# A1

Relationship between  
classes? Is-a or has a



## Object Composition and dynamic load Drink details

### VendingMachineImprove

- drinks: Drink[ ]

+ VendingMachine(drinks: Drink[ ])

- selectDrink(): double

- insertCoins(drinkCost: double): double

- checkChange(amount: double, drinkCost: double): void

- printReceipt(): void

+ start(): void

## Attribute + VendingMachineImprove()

```
private Drink[] drinks ; // object composition - relevant
```

## Constructor + VendingMachineImprove()

```
public VendingMachineImprove(Drink[] drinks) {  
    this.drinks = drinks;  
}
```

# Application class: VendingMachineAppImprove

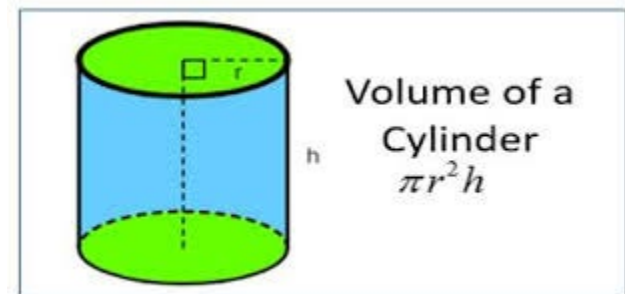
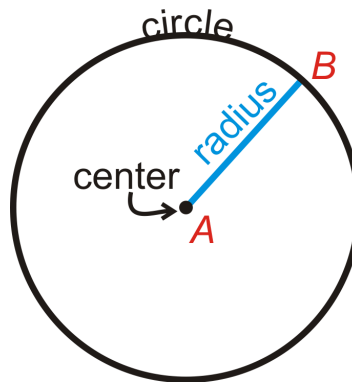
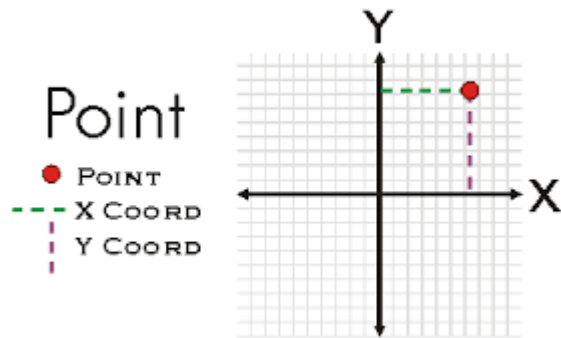
```
import java.util.Scanner;
public class VendingMachineAppImprove {
    public static void main(String[] args)
    {
        Scanner sc = new Scanner(System.in);
        System.out.println("Settings....");
        // future improvement can just read from file or DB
        System.out.println("Please enter number of drink types: ");
        int drinkTypes = sc.nextInt();
        Drink[] drinks = new Drink[drinkTypes];
        for (int i =0; i<drinkTypes; i++){
            System.out.println("Please enter drink name: ");
            String drinkName = sc.next();
            System.out.println("Please enter drink price: ");
            double drinkPrice = sc.nextDouble();
            drinks[i] = new Drink(drinkName,drinkPrice) ;
        }
        VendingMachineImprove vm = new VendingMachineImprove(drinks) ;
        vm.start() ;
    }
}
```

## + selectDrink(): double

```
private double selectDrink() {
    int drinkSelection = 0;
    double drinkCost = 0;
    Scanner sc = new Scanner(System.in);
    System.out.println("\n\n\n==== Vending Machine ====");
    for (int i = 0 ; i < drinks.length ; i++)
        System.out.println("|" + (i+1) + " Buy " + drinks[i].getName() +
            String.format(" %.2f ", drinks[i].getCost()) + " |");
    System.out.println("|=====");
    do {
        System.out.println("Please enter selection: ");
        drinkSelection = sc.nextInt();
    } while (drinkSelection < 1 || drinkSelection > drinks.length );
    drinkCost = drinks[drinkSelection-1].getCost() ;
    return drinkCost;
}
```

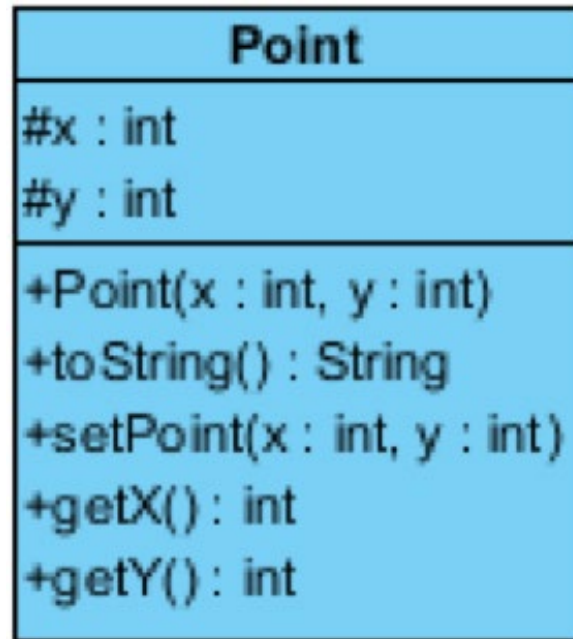


# Q2



## Q2

- You are given the class diagram for the `Point` class :



- The `toString()` method will return the `x` and `y` value in the format “[ `x`, `y` ]”.
- Write the code in Java.

# Visibility for Java (Chapter 3 Topic 4)

	If <b>Visibility</b> is (Access Level)	Visible			
		Within <b>Class</b> ?	Within <b>Package</b> ?	to a <b>Subclass outside</b> the package?	to the <b>World</b> ? (outside Classes)
+	<i>public</i>	Y	Y	Y	Y
#	<i>protected</i>	Y	Y	Y	N
	<i>Not defined (default)</i>	Y	Y	N	N
-	<i>private</i>	Y	N	N	N



Point
#x : int
#y : int
+Point(x : int, y : int)
+toString() : String
+setPoint(x : int, y : int)
+getX() : int
+getY() : int

```

public class Point {
    protected int x, y;

    public Point() { x = 0; y = 0; }
    public Point(int x, int y) {
        this.x = x; this.y = y;
    }

    public void setPoint(int x, int y) {
        this.x = x; this.y = y;
    }

    public String toString() {
        return "[" + x + "," + y + "]";
    }

    public int getX() {return x; }
    public int getY() {return y; }
}

```

```
public class MyThisTest {  
    private int a;  
  
    public MyThisTest(int a) {  
        a = a;  
    }
```

```
public void test() {  
    int a = 1;  
  
    System.out.println(a);  
    System.out.println(a);  
}
```

```
public void testField() {  
    System.out.println(a);  
}
```

```
public class DemoThis {  
  
    public static void main(String[] args) {  
        MyThisTest t = new MyThisTest(5);  
        t.test();  
        t.testField();  
    }  
}
```

slido



**What is the output of previous code?**

① Start presenting to display the poll results on this slide.

```
public class MyThisTest {  
    private int a;
```

```
    public void test(int a) {  
        a = a;  
    }
```

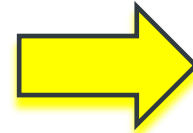
**WARNING**

The assignment to variable a has no effect

```
    public void test() {  
        int a = 1;
```

```
        System.out.println(a);  
        System.out.println(a);  
    }
```

```
    public void testField() {  
        System.out.println(a);  
    }
```




1  
1  
0

```
public class DemoThis {
```

```
    public static void main(String[] args) {  
        MyThisTest t = new MyThisTest(5);  
        t.test();  
        t.testField();  
    }  
}
```

## 1) clarify that you are talking about a field, when there's also something else with the same name as a field

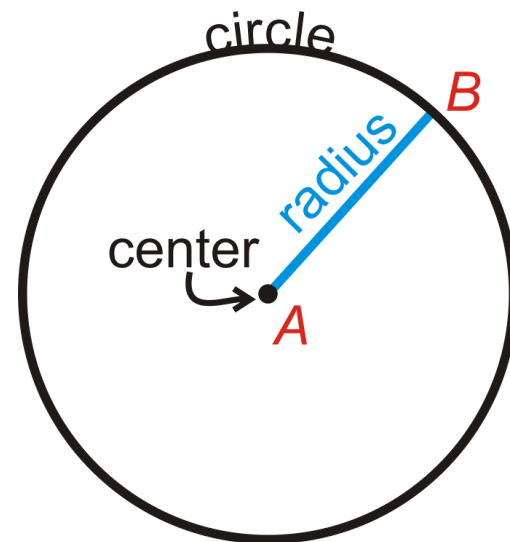
```
public class MyThisTest {  
    private int a;  
  
    public MyThisTest(int a) {  
        this.a = a; // assigns the value of the parameter a to the field of the same name  
    }  
  
    public void test() {  
        int a = 1;  
  
        System.out.println(a); // refers to the local variable a  
        System.out.println(this.a); // refers to the field a  
    }  
  
    public void testField() {  
        System.out.println(a); // refers to the field a  
    }  
}
```





## Q2

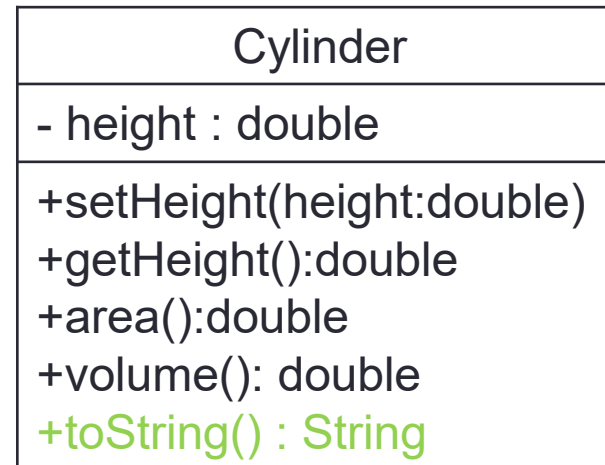
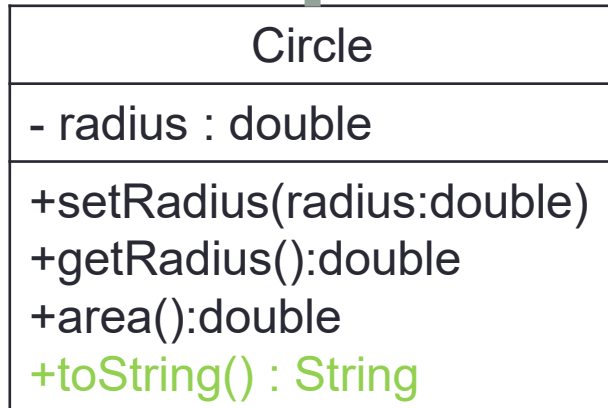
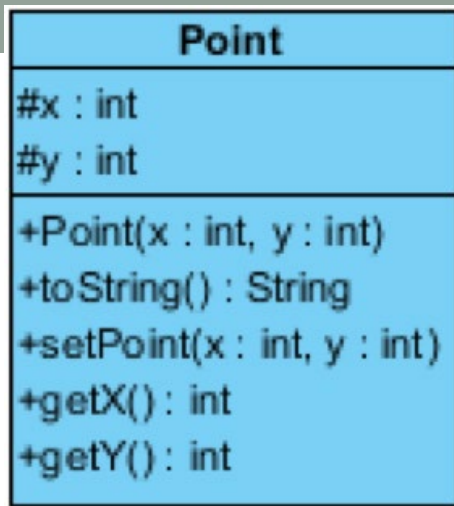
- Create a class `Circle` to extend from the `Point` class. The `Circle` class is to have the following methods:
  - `setRadius`
  - `getRadius`
  - `area`
- Reuse whatever you can from the `Point` class.




## Q2

- Create a class `Cylinder` to extend from the any of the classes above. The `Cylinder` class is to have the following methods:
  - `setHeight`
  - `getHeight`
  - `area`
  - `volume`

```
class Point {  
    protected double x;  
    protected double y;  
    :  
}  
  
class Circle extends Point {  
    protected double radius;  
    :  
}  
  
class Cylinder extends Circle {  
    protected double height;  
    :  
}
```



```
class Animal{
public Animal(){
System.out.print("Base Constructor");
}
}
class Cat extends Animal{
public Cat(){
System.out.print("Derived Constructor");
}
}
public class Program {
public static void main(String[] args) {
Cat c = new Cat();
}
}
```





**What is the output of the following java program?**

① Start presenting to display the poll results on this slide.

```
class Animal{
public Animal(){
System.out.print("Base Constructor");
}
}
class Cat extends Animal{
public Cat(){
System.out.print("Derived Constructor");
}
}
public class Program {
public static void main(String[] args) {
Cat c = new Cat();
}
}
```

Base Constructor	Derived Constructor
------------------	---------------------

```
public class Animal {  
    private String name;  
    public Animal(){  
        System.out.print("Base-con");  
    }  
    public Animal(String name){  
        this.name = name;  
        System.out.print("Base-con name");  
    }  
}  
  
class Cat extends Animal{  
    public Cat(){  
        System.out.print("sub-con");  
    }  
    public Cat(String name){  
        System.out.print("sub-con name");  
    }  
}}
```

```
public class Test {  
    public static void  
    main(String[] args) {  
        Cat c = new Cat("snow");  
    }  
}
```





**What is the output of the following java program?**

① Start presenting to display the poll results on this slide.

```
public class Animal {  
    private String name;  
    public Animal(){  
        System.out.print("Base-con");  
    }  
    public Animal(String name){  
        this.name = name;  
        System.out.print("Base-con name");  
    }  
}  
  
class Cat extends Animal{  
    public Cat(){  
        System.out.print("sub-con");  
    }  
    public Cat(String name){  
        System.out.print("sub-con name");  
    }  
}}
```

Base-consub-con name

```
public class Test {  
    public static void  
    main(String[] args) {  
        Cat c = new Cat("snow");  
    }  
}
```

```
public class Animal {  
    private String name;  
  
    public Animal(){  
        System.out.print("Base-con");  
    }  
  
    public Animal(String name){  
        this.name = name;  
        System.out.print("Base-con name");  
    }  
    class Cat extends Animal{  
        public Cat(){  
            System.out.print("sub-con");  
        }  
  
        public Cat(String name){  
            Animal(name);  
            System.out.print("sub-con name");  
        }  
    }  
}
```



# Subclass Definition

- Subclass is defined using the `extends` keyword.
- `super`:
  - Can be used in subclass constructor to call the superclass's `constructor`. The call to `super()` must be the **first statement** in the constructor for the class.
  - Can also be used in method definition to call the superclass's `method`, i.e., `super.print();`.

Point
#x : int
#y : int
+Point(x : int, y : int)
+toString() : String
+setPoint(x : int, y : int)
+getX() : int
+getY() : int

```

public class Point {
    protected int x, y;

    public Point() { x = 0; y = 0; }
    public Point(int x, int y) {
        this.x = x; this.y = y;
    }

    public void setPoint(int x, int y) {
        this.x = x; this.y = y;
    }

    public String toString() {
        return "[" + x + "," + y + "]";
    }

    public int getX() {return x; }
    public int getY() {return y; }
}

```

```
public class Circle extends Point {
    private double radius;
    public Circle() { radius = 1; }
    public Circle(double radius) { this.radius = radius; }
    public Circle(double radius, int a, int b) {
        super(a,b); this.radius = radius;
    }
    public double getRadius() { return radius; }
    public void setRadius(double radius) { this.radius = radius; }
    public double area() {
        return Math.PI * Math.pow(radius,2);
    }
    public String toString() {
        return "Circle of radius "+
            radius+" at point ["+x+", "+y+"]";
    }
}
```

```
public class Cylinder extends Circle {
    private double height;
    public Cylinder() { height = 1; }
    public Cylinder(double h) { height = h; }
    public Cylinder(double h, double r) {
        super(r) ; height = h; }
    public Cylinder(double h, double r, int a, int b) {
        super(r,a,b); height = h; }
    public double getHeight() { return height;}
    public void setHeight(double height){
        this.height = height; }
    public double area() {
        return 2 * (super.area() +
            vMath.PI*super.getRadius() * height);}
    public double volume() {
        return super.area() * height;}
    public String toString() {
        return "Cylinder of height " +
            height + ", radius " + getRadius() +
            " at point [" + x + ", " + y + "]";
    }
}
```

Q2 Create and use instances of a circle and a cylinder to test classes you have created.

```
public class Test{  
    public static void main(String args[]) {  
        Circle testCircle1 = new Circle();  
        Circle testCircle2 = new Circle(35, 2, 4);  
        Cylinder testCylinder1 = new Cylinder();  
        Cylinder testCylinder2 = new Cylinder(8, 20, 3, 3);  
        System.out.println("Area of Circle 1 = " +  
            testCircle1.area());  
        System.out.println("Area of Circle 2 = " +  
            testCircle2.area());  
        System.out.println("Surface Area of Cylinder 1 = " +  
            testCylinder1.area());  
        System.out.println("Surface Area of Cylinder 2 = " +  
            testCylinder2.area());  
        System.out.println("Volume of Cylinder 1 = " +  
            testCylinder1.volume());  
        System.out.println("Volume of Cylinder 2 = " +  
            testCylinder2.volume());  
    }  
}
```



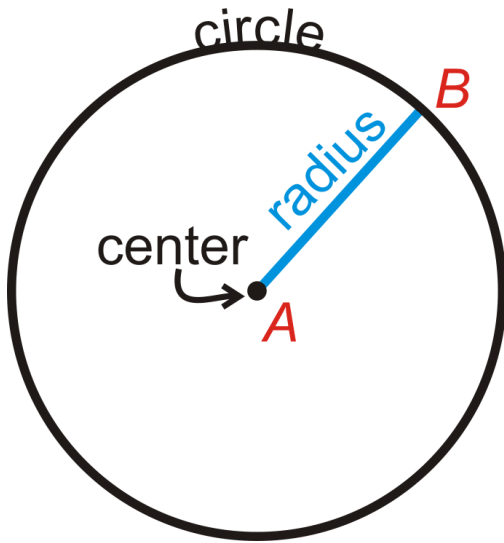
# Better option?

```
class Point {  
    protected double x;  
    protected double y;  
    :  
}  
  
class Circle extends Point {  
    protected double radius;  
    :  
}  
  
class Cylinder extends Circle {  
    protected double height;  
    :  
}
```

```
class Point {  
    double x;  
    double y;  
    :  
}  
  
class Circle {  
    Point center;  
    double radius;  
    :  
}  
  
class Cylinder {  
    Circle base;  
    double height;  
    :  
}
```

Composition allows us to build more complex classes from simpler ones, and is usually favored over inheritance unless necessary

Q2: Do you think that it was a good choice to use Point as the base class? Suggest alternatives.



A class called Circle, which models a circle with a center (x,y) and a radius,

Alternative will be to use Object Composition (has-a relationship) rather than inheritance (is-a relationship).

Circle
- radius : double # center: Point
+setRadius(radius:double) +getRadius():double +area():double +toString() : String

```
public class Circle {  
    protected Point centre = new Point();  
    protected double radius;  
    public Circle() { radius = 1; }  
    public Circle(double radius) { this.radius = radius;}  
    // note the use of super  
    public Circle(double radius, int a, int b) {  
        centre = new Point(a,b);  
        this.radius = radius;  
    }  
    public double getRadius() { return radius; }  
    public void setRadius(double radius) { this.radius = radius;}  
    public double area() {  
        return Math.PI * Math.pow(radius,2);  
    }  
    public String toString(){  
        return "Circle of radius " + radius + " at point [" + centre.getX() +  
            "," + centre.getY() + "];"  
    }  
}
```

# Tutorial 6: Different type of class relationships

- **Has a** relationship

- Dependency

- Association

- Aggregation

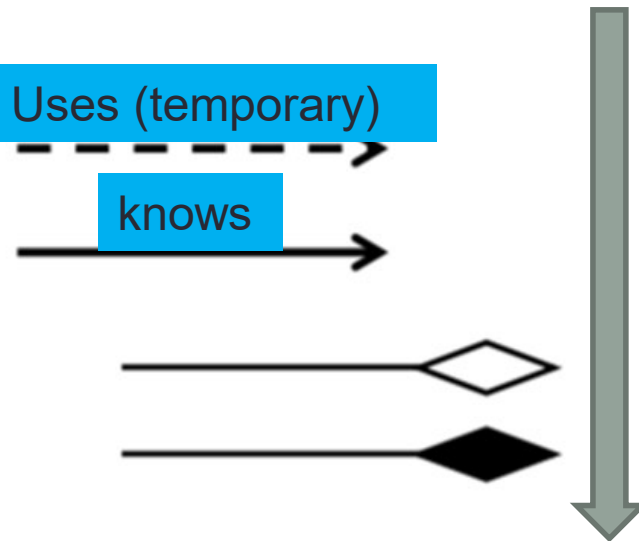
- Composition

Uses (temporary)

knows

tighter

B is  
composed  
of A  
Whole/part  
relationship



## You will be revisiting object composition again in Lab 3: Seating reservation application program .

PlaneSeat
<ul style="list-style-type: none"><li>- seatId: int</li><li>- assigned: boolean</li><li>- customerId: int</li></ul>
<ul style="list-style-type: none"><li>+ PlaneSeat(seat_id: int)</li><li>+ getSeatID(): int</li><li>+ getCustomerID(): int</li><li>+ isOccupied(): boolean</li><li>+ assign(cust_id: int): void</li><li>+ unAssign(): void</li></ul>

Plane
<ul style="list-style-type: none"><li>- seat: PlaneSeat[ ]</li><li>- numEmptySeat: int</li></ul>
<ul style="list-style-type: none"><li>+ Plane()</li><li>- sortSeats() : PlaneSeat[ ]</li><li>+ showNumEmptySeats(): void</li><li>+ showEmptySeats(): void</li><li>+ showAssignedSeats(bySeatId : boolean): void</li><li>+ assignSeat(seatId : int, cust_id : int): void</li><li>+ unAssignSeat(seatId : int): void</li></ul>

Write an application class *PlaneApp* that implements the seating reservation program.

Next week: E-Learning week