Object Oriented Programming (OOP)

Mohamed Ezz

Lecture 4

Review

- Complete the constructor of the following classes:
  - Person
  - Time
  - Student
- Example for object Ref.

  Point P1=new Point(1,2); Point P2=new Point(1,2);

  Point p3=p1;

  P1.x=3,p2.x=4;p3.x=5;

  P1.y=6;p2.y=7;p3.y=8;

  Draw object reference using UML define state of each object
- How to use classes AA,BB,CC, DD by class Test:

package m.n;
Class AA{
}
	package f;
	Class BB{
	}
		package k;
		Class CC{
		}
			Class DD{
			}
				Package t;
				Class Test{
				public void main (String a[]){
				} }

Review

Draw the UML of the following classes:

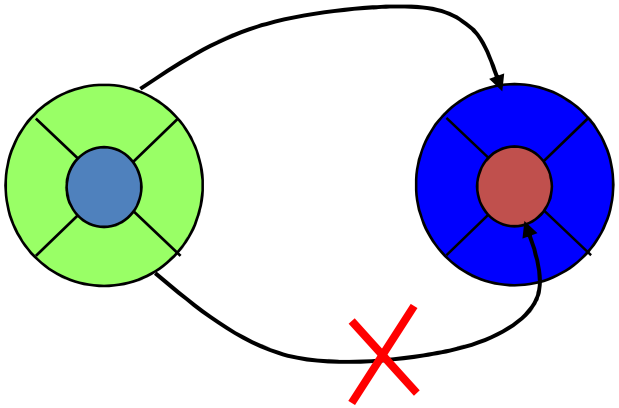
- Computer
- Mobile
- Time

Lecture Objectives

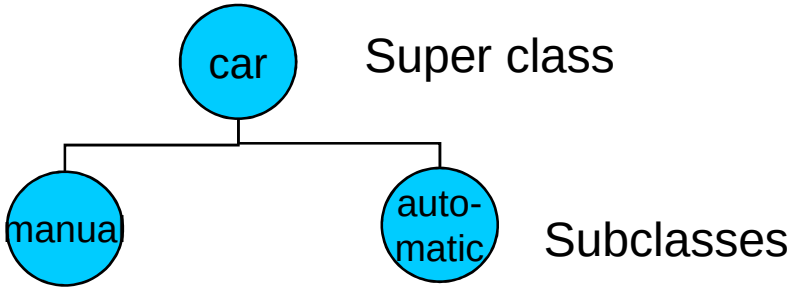
- Differentiate between Class Instance variables & Method local variables
- Understand how reuse method name by overloading
- Understand static method of the Class & when should used
- Differentiate between static & instance method
- Understand composition (has-a relationship)
- Understand Access Specifier
- Understand uses of keyword (this)

The three principles of OOP

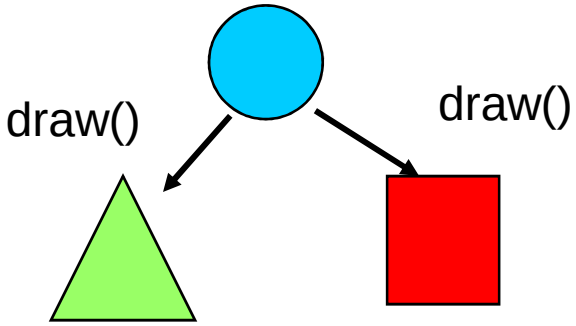
Encapsulation:
Objects hide their functions (methods) and data (instance variables)



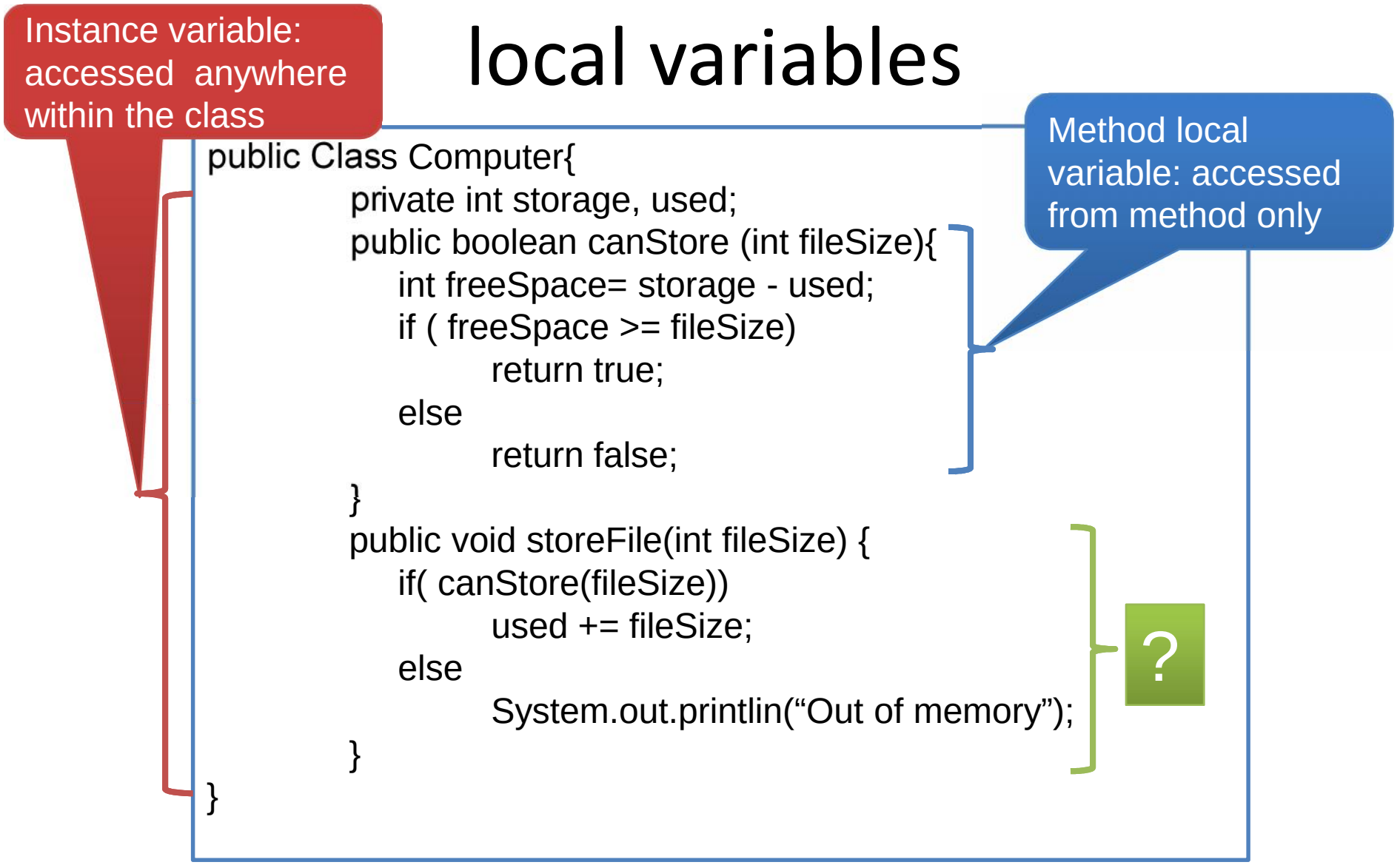
Inheritance
Each subclass inherits all variables of its superclass



Polymorphism
Interface same despite different data types



Class Instance variables & Method local variables



Method Overloading

- Multiple methods share the same method name, but each of them have a different parameters set (different method signature)

Examples:

int ***method***()

int ***method***(int a)

String ***method***(int a, String b)

void ***method***(int a, int b)

void ***method***(String a, int b)

String **method**(String a, int b) **// are this correct?**

- java println:

System.out.println(5);

System.out.println(2.5);

System.out.println("Hi All");

Method Overloading

Rectangle.java

```
class Rectangle{
  public float l; w;
}
```

Square.java

```
class Square{
 public float l;
}
```

Circle.java

```
class Circle{
 public float radius;
}
```

Method Overloading

TestShape.java

```
class TestShape{
  public static void main(String arg[]){
      Rectangle r1= new Rectangle();
      r1.l=2;
      r1.w=3;

     Square s1= new Square();
     s1.l=5;

     Circle c1= new Circle();
     c1. radius =5;

     System.out.prinln("Recarea:" + getArea(r1));
     System.out.prinln("Square area:" + getArea (s1));
     System.out.prinln("Circle area:" + getArea (c1));

  }
```

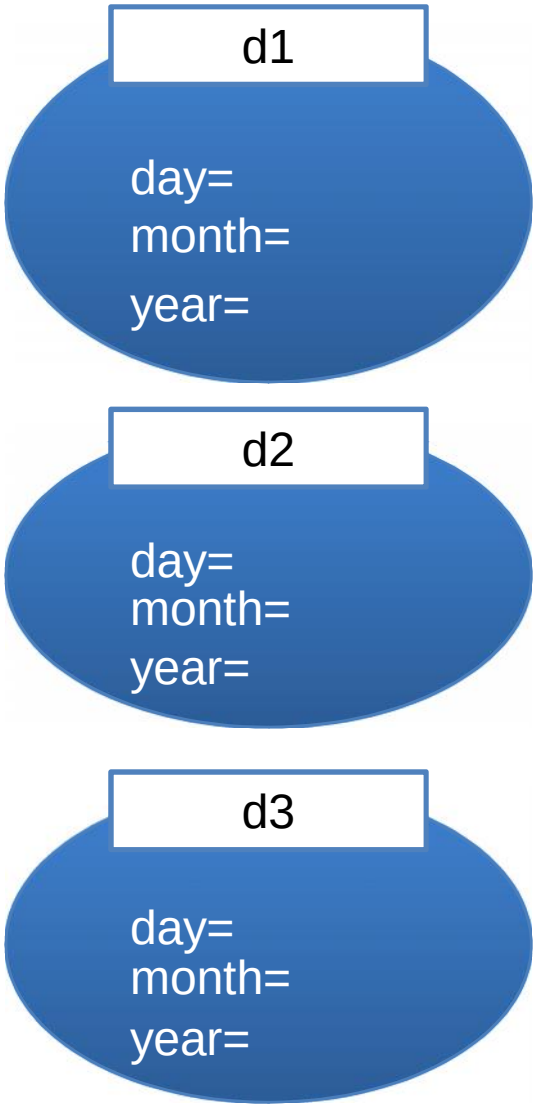
```
public static float getArea (Rectangle r) {
       float a = r.l  * r.w;
       return a;
  }
 public static float getArea (Square x) {
       float a = x.l  * x.l;
       return a;
  }

 public static float getArea (Circle c) {
      float a = c.radious * c.radius * 3.14;
       return a;
  }
}
```

Constructor Overloading

- Like methods, constructors can be overloaded
- This offers greater flexibility and convenience of creating objects

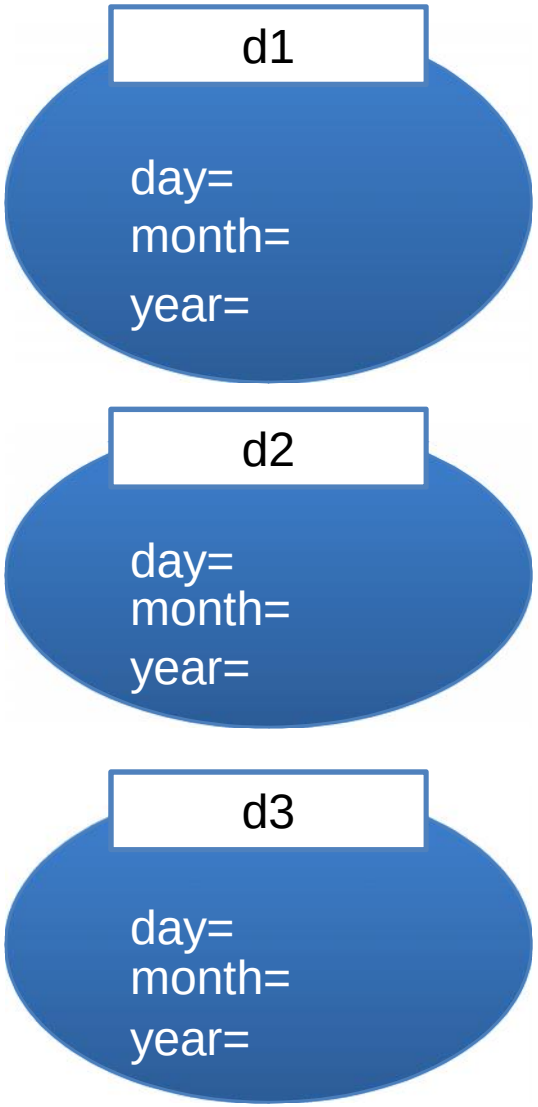
Constructor Overloading

```
public Class TestDate {

        pubic static void main(String arg[]){
                Date d1= new Date();
                Date d2= new Date(12,5,2008);
                Date d3 = new Date(d1);
                Date d4 = d3;

                System.out.println(d1);
                System.out.println(d2);
                System.out.println(d3);

        }
}
```



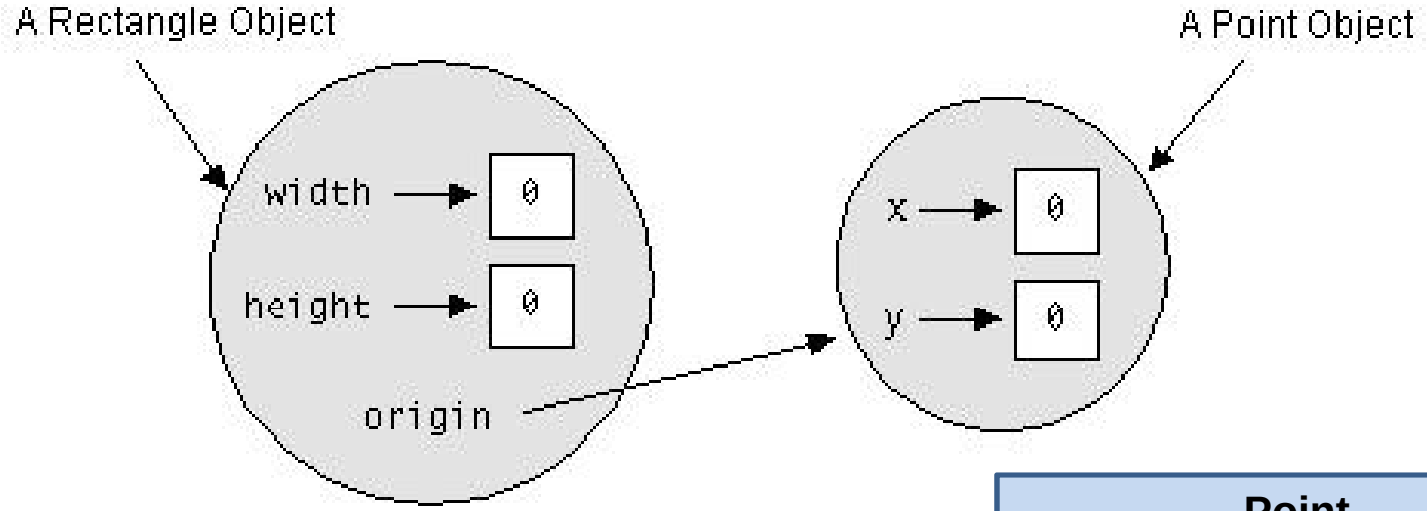
Constructor Overloading

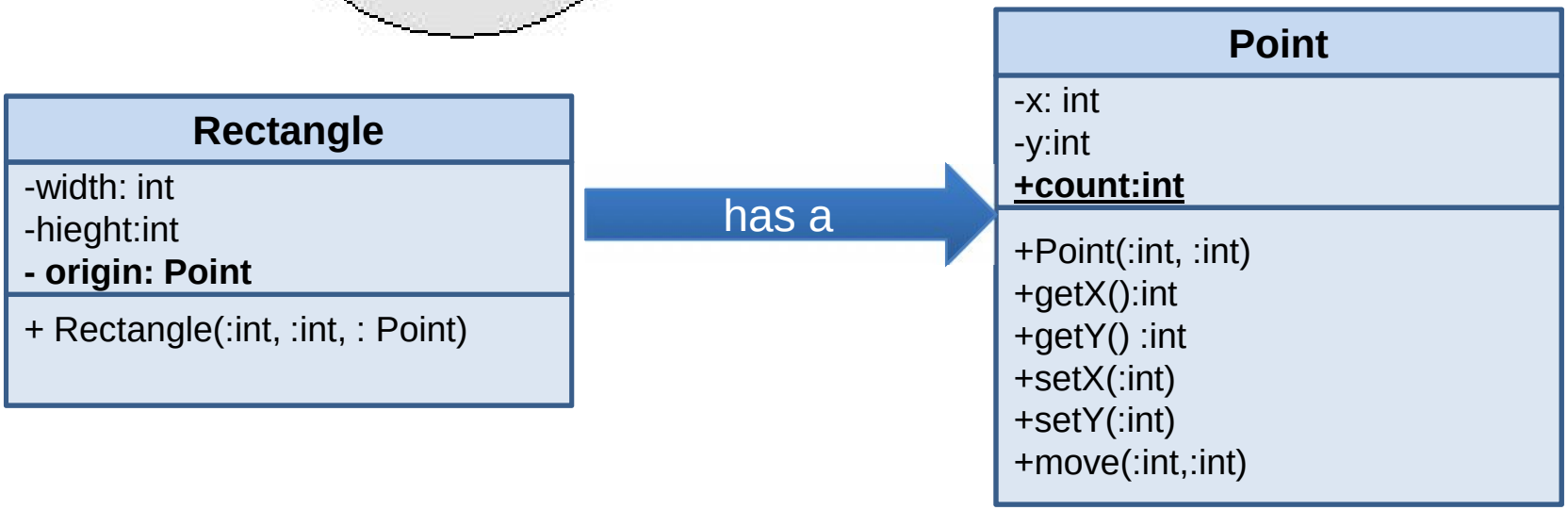
```
public class Date {
        public int day, month, year;
                Date (){
                        day = 1;
                        month= 1;
                        year= 2013;
                }
                Date (int d, int m, int y){
                         day = d;
                        month= m;
                        year= y
                }
                Date (Date  a){
                         day = a.day;
                        month= a.month;
                        year= a.year;
                }
}
```



Composition (has-a relationship)

- Employee has Date of birth
- Rectangle has Point

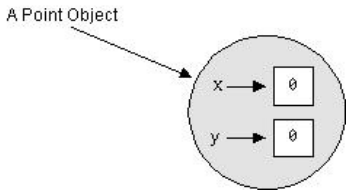


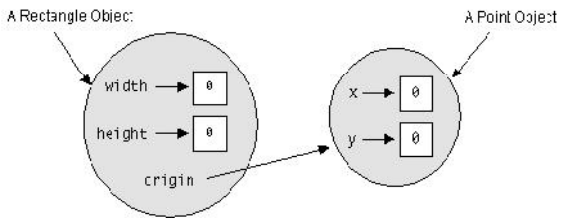


Rectangle Has-a Point

```
public class Point {
    // same as point class which contain x,y
}
```

```
public class Rectangle {
    private int width = 0;
    private int height = 0;
    private Point origin = new Point();
    public Rectangle(int w, int h, Point p){
        width =w;
        height=h;
        origin = p;
    }
}
```



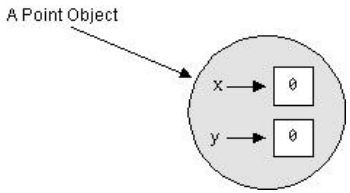


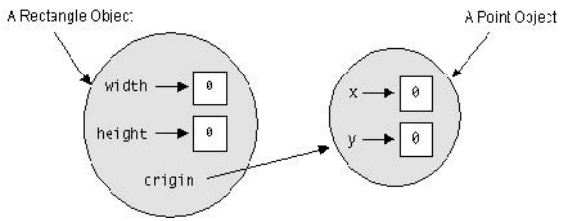

```
public class Test {
        public static void main(String arg[]){
                Point p1 = new Point (2,2);
                Rectangle rec1 = new Rectangle (2,3,p1);
                Rectangle rec2 = new Rectangle (5,6,p1);
                Point p2 = new Point (0,0);
                Rectangle rec3 = new Rectangle (1,4,p2);
        }
}
```

What Difference in Rectangle?

```
public class Point {
    // same as point class which contain x,y
}
```

```
public class Rectangle {
    private int width = 0;
    private int height = 0;
    private Point origin = new Point();
    public Rectangle(int w, int h, Point p){
        width =w;
        height=h;
        origin.setX(p.getX();
        origin.setY(p.getY();
    }
}
```

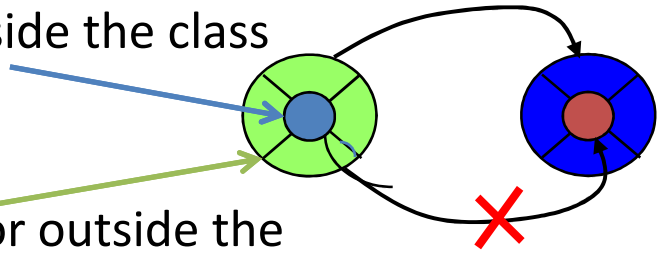





```
public class Test {
        public static void main(String arg[]){
                Point p1 = new Point (2,2);
                Rectangle rec1 = new Rectangle (2,3,p1);
                Rectangle rec2 = new Rectangle (5,6,p1);
                Point p2 = new Point (0,0);
                Rectangle rec3 = new Rectangle (1,4,p2);
        }
}
```

Access Specifier (Visibility)

- **private**
  - ➢The variable or method accessed only from inside the class
    - ✓From inside member method only



public

- ➢The variable or method accessed from inside or outside the class or other forging packages
  - ✓From inside member method
  - ✓From inside Non-member method belong to the class in same or forging package
  - ✓From inside main function

- **Default (if method or variable defined without specifier)**
  - ➢The variable or method accessed from inside the class and the sister classes inside same package
    - ✓From inside member method
    - ✓From inside main method
    - ✓From inside Non-member method belong to classes in the same package

Access Specifier Syntax

- Variable
  - **Access_specifier** type variable_name;

    private int x;

    public int y;

    int z; // without specify mean default
- Method
  - **Access_specifier** return method_name(paramater);

    private int getX();

    public int setY(int yy);

    int getZ(); // without specify mean default

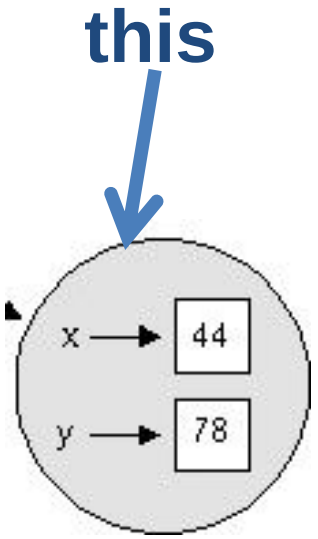
Example of Access Specifier

package lib1;	**package lib1;**	**package lib2;**

Class A{
  private int x;
  int y;
  public int z;
  public void test(){
  x=5;
  y=3;
  z=2;
  }
}
Which assignment correct?
	Class B{
	  public void test(){
	  A a = new A();
	  a.x=5;
	  a.y=3;
	  a.z=2;
	  }
	}
	Which assignment correct?
		Class C{
		  public void test(){
		  A a = new A();
		  a.x=5;
		  a.y=3;
		  a.z=2;
		  }
		}
		Which assignment correct?

Itself Object reference (this)

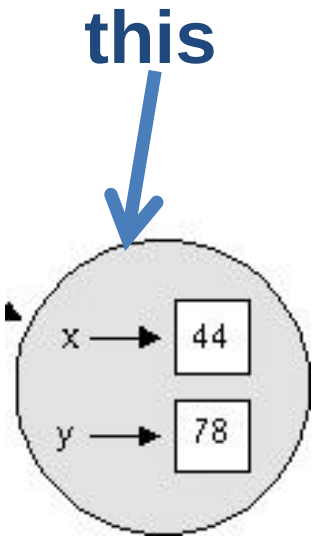
```
public class Point {
   int x,y;
   public Point(int x, int y){
        x=x; // assigned to itself - which incorrect
        y=y; // assigned to itself - which incorrect

   }

   public Point move(int dx, int dy){
        x+= dx;
        y+=dy;
        return this;
   }
}
```




```
public class Test {
        public static void main(String arg[]){
                Point p1 = new Point (2,2);
                Point p2;
                if( (p1.move(1,3)).x==0 )
                        System.out.println("point on x axis");


        }
}
```

Itself Object reference (this)

```
public class Point {
   int x,y;
   public Point(int x, int y){
        this.x=x; // correct
        this.y=y; // correct

   }

   public Point move(int dx, int dy){
        x+= dx;
        y+=dy;
        return this;
   }
}
```



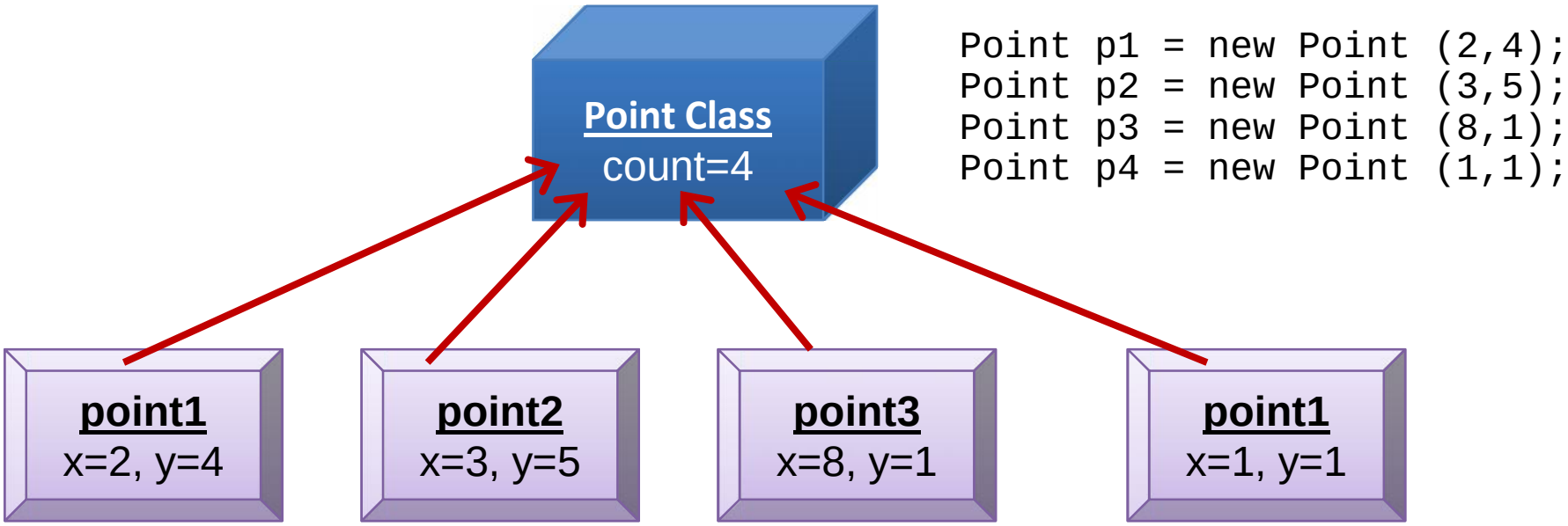

```
public class Test {
        public static void main(String arg[]){
                Point p1 = new Point (2,2);
                Point p2;
                if( (p1.move(1,3)).x==0 )
                        System.out.println("point on x axis");


        }
}
```

Static field of the class

- How you can know No. of object created from the class?

Example:
- How many points object created from the class Point?

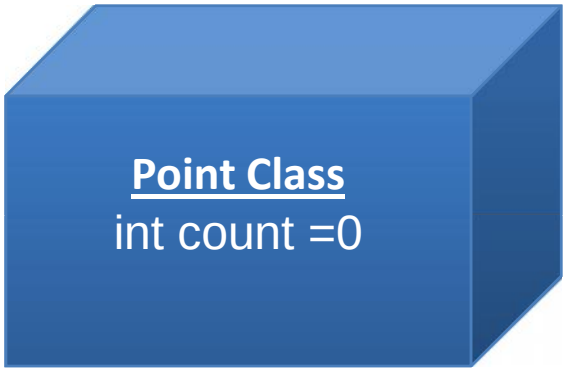
- We requires a variable associated with Class not objects




```
Point p1 = new Point (2,4);
Point p2 = new Point (3,5);
Point p3 = new Point (8,1);
Point p4 = new Point (1,1);
```

Static field example

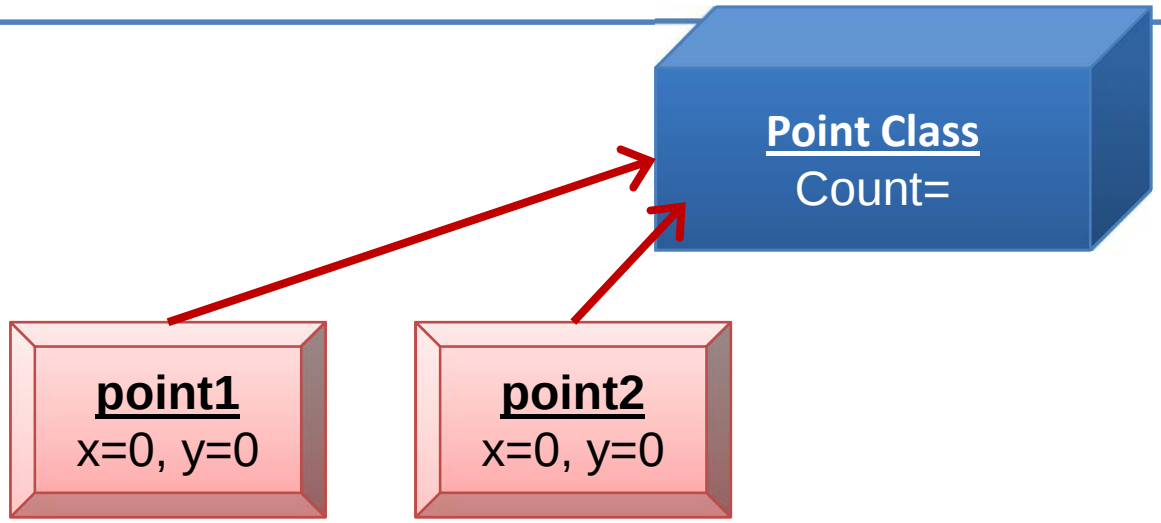
```
public class Point
{
        public static int count=0;
        private int x, int y;


        public Point() {
                count ++;
        }
}
```



- Static variable defined inside class using **static** keyword
- Can be initialized

Static field example

```
public class TestPoint
{
        public static void main (String arg[]){
        Point p1,p2,p3, p4;
        p1= new Point();
        p2= new Point();
        System.out.println("No. of points created="+ Point.count);
        System.out.println("No. of points created="+ p2.count);
}
```

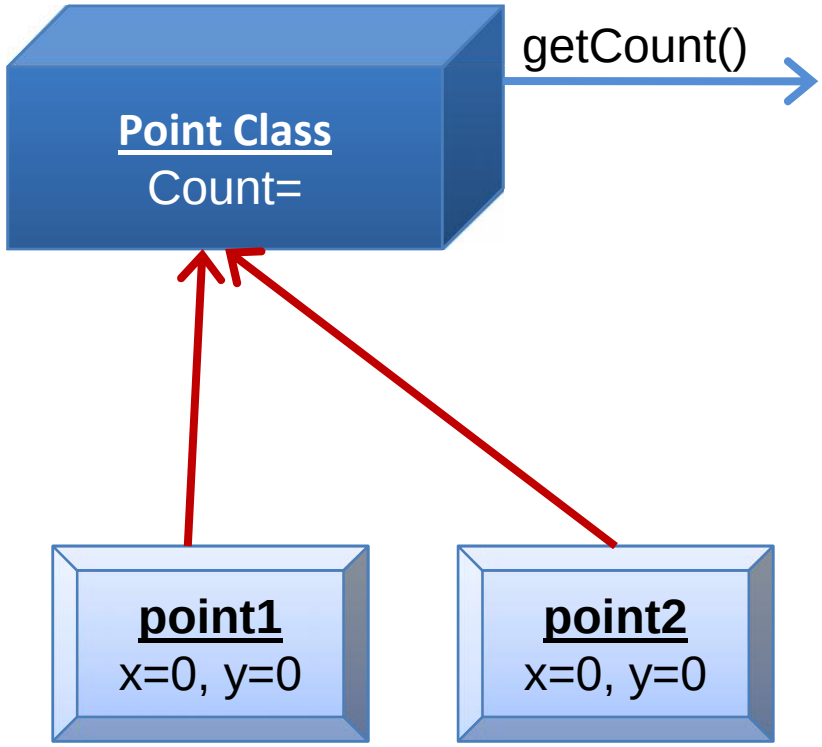


Static Variables and Methods

- Static variables are shared by all the instances of the class.
- To declare static variables, and methods, use the **static** modifier.
- Static methods are not tied to a specific object.
- Constants variables is static with **final** keyword which shared by all the instances of the class.
- Preferred to access static variable through static method

Static field example

```
public class Point
{
        private static int count=0;
        private static final int max=100;
        private int x, int y;

        public Point() {
                count ++;
        }
        public static int getCount(){
                return count;
        }
}
```



Class Variable

What is the difference between member(instant) variable & class variable (static)?

	Instant Variable	**Class Variable**
Declaration	Inside class int x, int y;	Inside class with **static** keyword int static count;
Access	Using object p1.x;	Using object or Class p1. count; Point. count;
Change Value	Effected in each object	Effected for all class instances Initialized in the class
Memory allocation	Separate location for each object	Shared location for all class objects & the class

can we use a class variable without existing of any object?

Class/Instance Variable/Method

Complete the following?

	Instant Variable (such as x)	**Class Variable (such as count)**
Instance method (such as move)	-Can instance method access instance variable directly? -	-Can instance method access class variable directly? -
Class Method (such as getCount)	-Can class method access instance variable directly? -	-Can class access class variable directly? -

The “main” Method

Following OOP guidelines, the use of the “main” method should be deemphasized

- The “main” method is merely a starting point of the application
- Make it simple, clear, few statements in the main method
- Using objects and methods effectively

Ideally, the following should be inside the main method:

- Creating objects
- Calling class/object methods