Overloading

- Two or more methods (including constructors) with the same name can be allowed when they have different parameter types.

```
class Rectangle3 {
    private int leftTopX, leftTopY ;
    private int rightBottomX, rightBottomY ;

    public Rectangle3(int x1, int y1, int x2, int y2) {
        leftTopX = x1 ; leftTopY = y1 ;
        rightBottomX = x2 ; rightBottomY = y2 ;
    }
    public void moveBy(int deltaX, int deltaY) {
        leftTopX += deltaX ; leftTopY += deltaY ; rightBottomX += deltaX ; rightBottomY += deltaY ;
    }
    public void moveBy(int delta) { moveBy(delta, delta) ; }
    public void print() {
        System.out.printf("(%6d,%6d), (%6d,%6d)%n", leftTopX, leftTopY, rightBottomX, rightBottomY) ;
    }
}
```

```
Rectangle3 r1 = new Rectangle3(0, 0, 50, 50) ;
r1.print() ;

r1.moveBy(10, 20) ; r1.print() ;

r1.moveBy(10) ; r1.print() ;
```

IMPORTANT METHODS: EQUALS(), HASHCODE()

Important methods: toString() and equals()

```
class Rectangle4 {
   private int leftTopX, leftTopY ;
   private int rightBottomX, rightBottomY ;

   public Rectangle4(int x1, int y1, int x2, int y2) {
     leftTopX = x1 ; leftTopY = y1 ; rightBottomX = x2 ; rightBottomY = y2 ;
   }
   public boolean equals(Object otherRectangle) {
     if ( ! ( otherRectangle instanceof Rectangle4 )  ) return false ;
     var r = (Rectangle4) otherRectangle ; // casting from Object to Rectangle4
     return leftTopX == r.leftTopX && leftTopY == r.leftTopY &&
       rightBottomX == r.rightBottomX && rightBottomY == r.rightBottomY ;
   }
   public String toString() {
     return String.format("(%6d,%6d), (%6d,%6d)",
       leftTopX, leftTopY, rightBottomX, rightBottomY) ;
   }
}
```

Important methods: toString() and equals()

```
public class UsefulMethods {

   public static void main(String[] args) {

      var r1 = new Rectangle4(0, 0, 10, 10) ;
      var r2 = new Rectangle4(0, 0, 10, 20) ;

      System.out.println("R1: " + r1) ;
      System.out.println("R2: " + r2) ;

      var msg =  r1.equals(r2) ? "They are the same." : "They are not the same." ;
      System.out.println(msg) ;
   }
}
```

Every object is converted into a String whenever necessary !

```
R1: (      0,      0), (     10,     10)
R2: (      0,      0), (     10,     20)
They are not the same.
```

equals(): Example

```
public class Employee {
   private String name;
   private double salary;
   private LocalDate hireDay;

   ...
}
```

```
public boolean equals(Object otherObject) {
   if (this == otherObject) return true;
   if (otherObject == null) return false;
   if (getClass() != otherObject.getClass()) return false;
   Employee other = (Employee) otherObject;
   return name.equals(other.name)
      && salary == other.salary
      && hireDay.equals(other.hireDay);
}
```

what if name is null?

```
public boolean equals(Object otherObject) {
   if (this == otherObject) return true;
   if (otherObject == null) return false;
   if (getClass() != otherObject.getClass()) return false;
   Employee other = (Employee) otherObject;
   return Objects.equals(name, other.name)
      && salary == other.salary
      && Objects.equals(hireDay, other.hireDay);
}
```

Objects.equals(object1, object2)
Arrays.equals(array1, array2)

Important methods: hashCode()

- A hash code is an integer that is derived from an object.
- if x and y are two distinct objects, there should be a high probability that x.hashCode() and y.hashCode() are different

```
public class StringHashCode {
   public static void main(String[] args) {
      var string1 = "Hello"; var string2 = "hello";
      System.out.println(getHash(string1) + ":" + getHash(string2));
      // 69609650:99162322
      System.out.println(string1.hashCode() + ":" + string2.hashCode());
      // 69609650:99162322
   }
   private static int getHash(String string) {
      var hash = 0;
      for (int i = 0; i < string.length(); i++)
        hash = 31 * hash + string.charAt(i);
      return hash;
   }
}
```

Important methods: hashCode()

- You must override hashCode() in every class that overrides equals()

- Your definitions of equals and hashCode must be compatible
  - If x.equals(y) is true, then x.hashCode() must return the same value as y.hashCode().
  - For example, if you define Employee.equals to compare employee IDs, then the hashCode method needs to hash the IDs, not employee names or memory addresses.

hashCode()

```
public class Employee {
   private String name;
   private double salary;
   private LocalDate hireDay;
   public Employee(String name, double salary, LocalDate hireDay) {
      this.name = name;
      this.salary = salary;
      this.hireDay = hireDay;
   }
   public int hashCode1() {     // version 1, but does not consider when name is null
      return 7 * name.hashCode()
         + 11 * Double.valueOf(salary).hashCode()
         + 13 * hireDay.hashCode();
   }
   public int hashCode2() {     // version 2 == version 1, but consider when name is null
      return 7 * Objects.hashCode(name)
         + 11 * Double.hashCode(salary)
         + 13 * Objects.hashCode(hireDay);
   }
   public int hashCode3() {   // version 3: Eclipse default
      return Objects.hash(name, salary, hireDay);
   }
   ... // version 4: Alt + Insert ( Generate equals() and hashCode() ) in IntelliJ
}
```

hashCode()

```
  public static void main(String[] args) {
    var e1 = new Employee("Kim", 200, LocalDate.of(2019, 9, 15));
    var e2 = new Employee("Kim", 200, LocalDate.of(2019, 9, 15));
    var e3 = new Employee("kim", 200, LocalDate.of(2019, 9, 15));

    System.out.println(e1.hashCode()+":"+ e1.hash1()+":"+ e1.hash2() +":"+ e1.hash3());
    System.out.println(e2.hashCode()+":"+ e2.hash1()+":"+ e2.hash2() +":"+ e2.hash3());
    System.out.println(e3.hashCode()+":"+ e3.hash1()+":"+ e3.hash2() +":"+ e3.hash3());
  }
}
```

31168322:-943758132:-943758132:-783759971
17225372:-943758132:-943758132:-783759971
5433634:-943542868:-943542868:-754207299

Methods for Hashing: Summary

java.lang.Object

int hashCode()	returns a hash code for this object. The default is derived from the object's memory address

java.lang.(Integer|Long|Short|Byte|Double|Float|Character|Boolean)

static int hashCode(xxx value)	returns the hash code of the given value

java.util.Objects

static int hashCode(Object a)	returns 0 if a is null or a.hashCode() otherwise
static int hash(Object... objects)	returns a hash code that is combined from the hash codes of all supplied objects.

java.util.Arrays

static int hashCode(xxx[] a)	computes the hash code of the array a. The component type xxx of the array can be Object, int, long, short, char, byte, boolean, float, or double

Parameter Passing

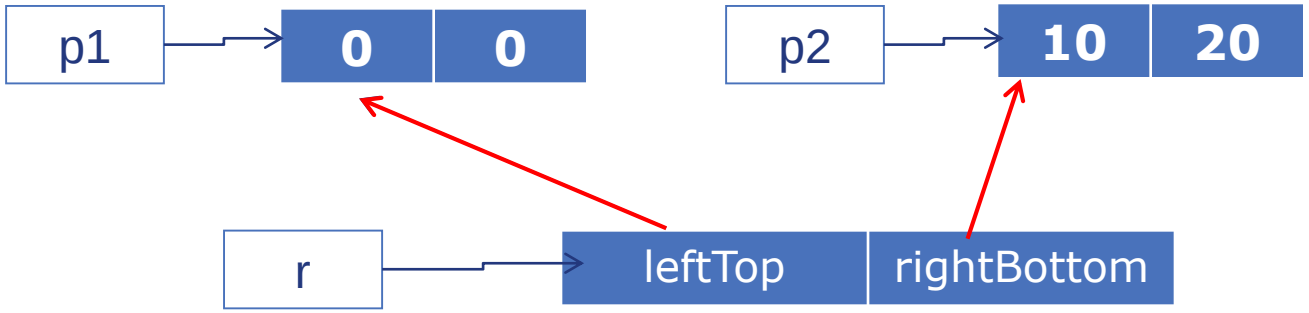
- **Everything in Java is pass-by-value**
- in Java Language Specification (8.4.1)
  - "When the method or constructor is invoked (§15.12), **the values of the actual argument expressions initialize newly created parameter variables**, each of the declared type, before execution of the body of the method or constructor. The Identifier that appears in the FormalParameter may be used as a simple name in the body of the method or constructor to refer to the formal parameter."
- **Do not use "primitives are passed by value, objects are passed by reference" in Java**

"Java is Pass-by-Value, Dammit!" http://www.javadude.com/articles/passbyvalue.htm

```
class Point {
    private int x, y ;
    public Point(int x, int y) { set(x, y) ; }
    public void set(int x, int y) { this.x = x ; this.y = y ; }
    public String toString() { return String.format("(%d, %d)", x, y) ; }
    public boolean equals(Object otherPoint) {
       var p = (Point) otherPoint ;
       return x == p.x && y == p.y ;
    }
}

class Rectangle5 {
    private Point leftTop ;
    private Point rightBottom ;

    public Rectangle5(Point p1, Point p2) { leftTop = p1 ; rightBottom = p2 ; }
    public boolean equals(Object otherRectangle) {
       var r = (Rectangle5) otherRectangle ;
       return leftTop.equals(r.leftTop) && rightBottom.equals(r.rightBottom) ;
    }
    public String toString() { return leftTop + "," + rightBottom ; }
}
```

Each parameter of class variable is reference type and passed by value.
Thus, leftTop and p1 refer to the same Point !

```
public class ParameterPassing {
    public static void main(String[] args) {
        var p1 = new Point(0, 0) ;
        var p2 = new Point(10, 20) ;

        var r = new Rectangle5(p1, p2) ;
        System.out.println(r) ; // (0, 0),(10, 20)

        p2.set(100, 200) ;
        System.out.println(r) ; // (0, 0),(100, 200), not (0, 0),(10, 20)
    }
}
```



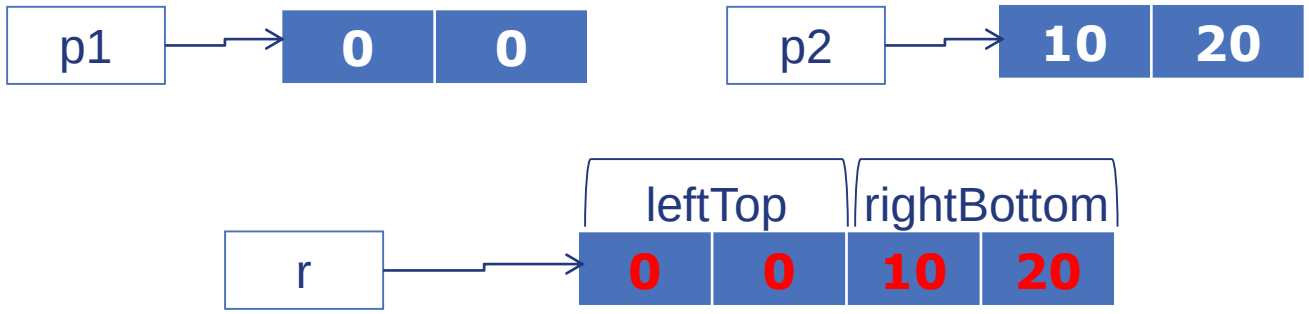
In the constructor of class Rectangle, **the references are only copied!**

```
    public Rectangle5(Point p1, Point p2) {
        leftTop = p1 ; rightBottom = p2 ;
    }
```

Deep Copy

- We need to copy the object itself, not the reference!

- What we want is as follows !



- Let's change the constructor of class Rectangle like this !

```
public Rectangle5(Point p1, Point p2) {
  // leftTop = p1 ; rightBottom = p2 ;
  leftTop = new Point(p1.getX(), p1.getY()) ;
  rightBottom = new Point(p2.getX(), p2.getY()) ;
}
```

```
class Rectangle6 {
    private Point leftTop ;
    private Point rightBottom ;

    public Rectangle6(Point p1, Point p2) {
       leftTop = new Point(p1.getX(), p1.getY()) ;
       rightBottom = new Point(p2.getX(), p2.getY()) ;
    }
    public boolean equals(Object otherRectangle) {
       var r = (Rectangle6) otherRectangle ;
       return leftTop.equals(r.leftTop) && rightBottom.equals(r.rightBottom) ;
    }
    public String toString() { return leftTop + "," + rightBottom ; }
}

public class DeepCopy {
    public static void main(String[] args) {
       var p1 = new Point(0, 0) ;
       var p2 = new Point(10, 20) ;

       var r = new Rectangle6(p1, p2) ;
       System.out.println(r) ; // (0, 0),(10, 20)

       p2.set(100, 200) ;
       System.out.println(r) ; // (0, 0),(10, 20), not (0, 0),(100, 200)
    }
}
```