

Call by Value: **Re-Assigning** **Primitive** Parameter

```
public class Util {  
    void reassignInt(int j) {  
        j = j + 1; }  
    void reassignRef(Point q) {  
        Point np = new Point(6, 8);  
        q = np; }  
    void changeViaRef(Point q) {  
        q.moveHorizontally(3);  
        q.moveVertically(4); } }  
}
```

```
1  @Test  
2  public void testCallByVal() {  
3      Util u = new Util();  
4      int i = 10;  
5      assertTrue(i == 10);  
6      u.reassignInt(i);  
7      assertTrue(i == 10);  
8  }
```

Call by Value: Re-Assigning Reference Parameter

```
public class Util {  
    void reassignInt(int j) {  
        j = j + 1; }  
    void reassignRef(Point q) {  
        Point np = new Point(6, 8);  
        q = np; }  
    void changeViaRef(Point q) {  
        q.moveHorizontally(3);  
        q.moveVertically(4); } }
```

```
1 @Test  
2 public void testCallByRef_1() {  
3     Util u = new Util();  
4     Point p = new Point(3, 4);  
5     Point refOfPBefore = p;  
6     u.reassignRef(p);  
7     assertTrue(p == refOfPBefore);  
8     assertTrue(p.getX() == 3);  
9     assertTrue(p.getY() == 4);  
10 }
```

```
public class Point {  
    private int x;  
    private int y;  
    public Point(int x, int y) {  
        this.x = x;  
        this.y = y;  
    }  
    public int getX() { return this.x; }  
    public int getY() { return this.y; }  
    public void moveVertically(int y) { this.y += y; }  
    public void moveHorizontally(int x) { this.x += x; }  
}
```

Call by Value: **Calling Mutator** on **Reference** Parameter

```
public class Util {  
    void reassignInt(int j) {  
        j = j + 1; }  
    void reassignRef(Point q) {  
        Point np = new Point(6, 8);  
        q = np; }  
    void changeViaRef(Point q) {  
        q.moveHorizontally(3);  
        q.moveVertically(4); } }
```

```
1 @Test  
2 public void testCallByRef_2() {  
3     Util u = new Util();  
4     Point p = new Point(3, 4);  
5     Point refOfPBefore = p;  
6     u.changeViaRef(p);  
7     assertTrue(p == refOfPBefore);  
8     assertTrue(p.getX() == 6);  
9     assertTrue(p.getY() == 8);  
10 }
```

```
public class Point {  
    private int x;  
    private int y;  
    public Point(int x, int y) {  
        this.x = x;  
        this.y = y;  
    }  
    public int getX() { return this.x; }  
    public int getY() { return this.y; }  
    public void moveVertically(int y) { this.y += y; }  
    public void moveHorizontally(int x) { this.x += x; }  
}
```

Object Equality: First Example

```
class PointV1 {  
    private int x;  
    private int y;  
    public PointV1(int x, int y) {  
        this.x = x;  
        this.y = y;  
    }  
}
```

Q1:

Does PointTester compile, given that **equals** is not declared in PointV1?

Q2:

Where does **equals** come from?

```
class PointTester {  
    ... main(...) {  
        PointV1 p1 = new PointV1(3, 4);  
        PointV1 p2 = new PointV1(3, 4);  
        System.out.println(p1.equals(p2));  
    }  
}
```

The equals Method: To Override or Not?

```
public class Object {  
    ...  
    public boolean equals(Object obj) {  
        return this == obj;  
    }  
}
```

extends

extends

```
public class PointV1 {  
    private double x;  
    private double y;  
    public PointV1 (double x, double y) {  
        this.x = x;  
        this.y = y;  
    }  
}
```

```
public class PointV2 {  
    private int x; private int y;  
    public PointV2 (int x, int y) { ... }  
    public boolean equals(Object obj) {  
        if(this == obj) { return true; }  
        if(obj == null) { return false; }  
        if(this.getClass() != obj.getClass()) { return false }  
        PointV2 other = (PointV2) obj;  
        return this.x == other.x  
            && this.y == other.y;  
    }  
}
```