Lecture – 5

Passing Objects

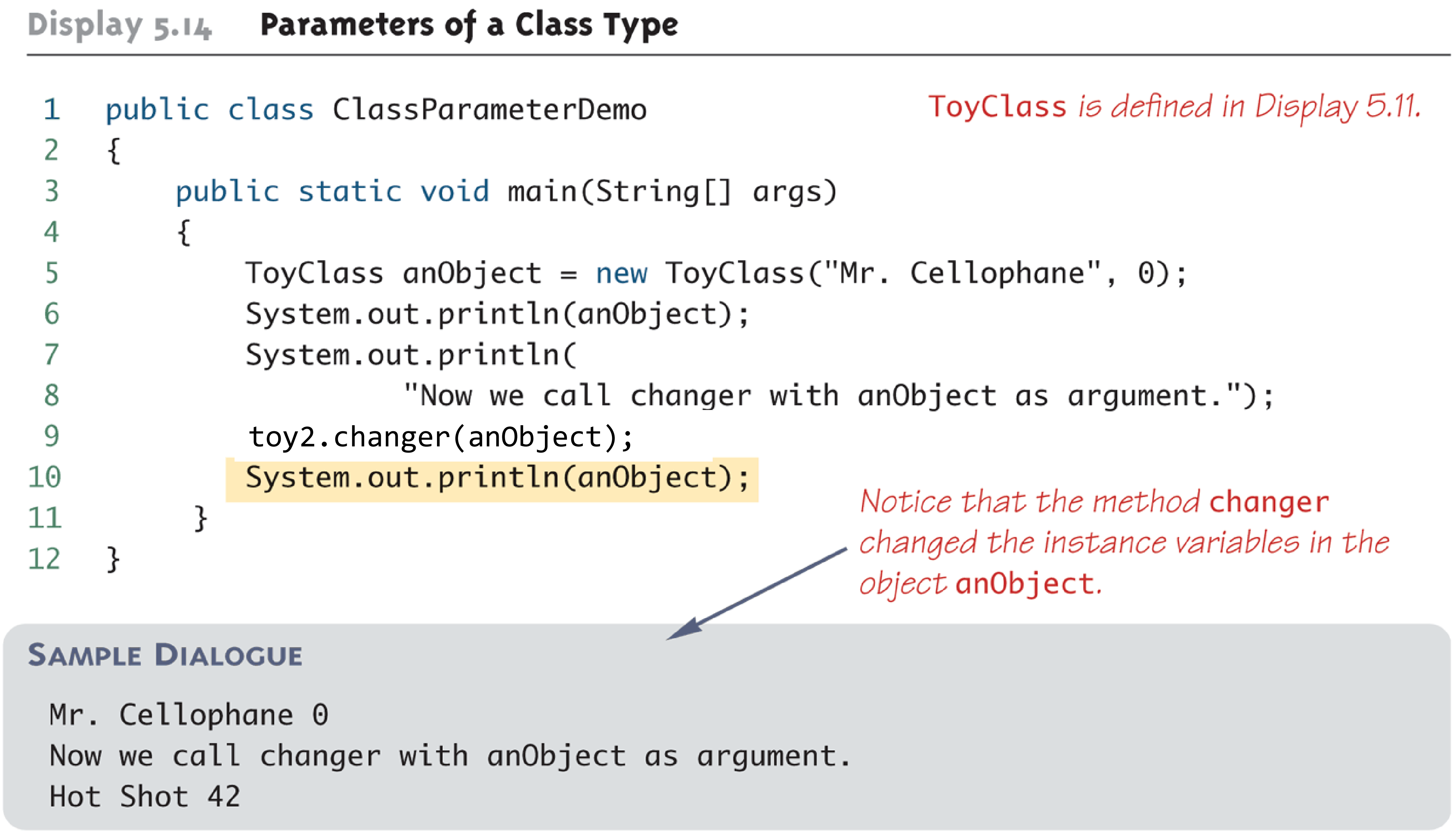
Primitive Parameters

- Primitive types: boolean, byte, char, short, int, long, float, double
- In Java, all primitives are passed by value. This means a copy of the value is passed into the method
- Modifying the primitive parameter in the method does NOT change its value outside the method

Object Parameters

- Objects can be passed natively, just like primitives
- It is often misstated that Object parameters are passed by Reference.
- While it is true that the parameter is a reference to an Object, the reference itself is passed by Value.
- What we pass in method is a *handle of an object*, and in the c*alled method* a **new handle created** and **pointed to the same object**.
- Now when more than one handles tied to the same object, it is known as **aliasing**.

Object Parameters



Display 5.14 **Parameters of a Class Type**

```
public class ClassParameterDemo
{
    public static void main(String[] args)
    {
        ToyClass anObject = new ToyClass("Mr. Cellophane", 0);
        System.out.println(anObject);
        System.out.println(
                 "Now we call changer with anObject as argument.");
        toy2.changer(anObject);
        System.out.println(anObject);
     }
}
```

`ToyClass` is defined in Display 5.11.

Notice that the method `changer` changed the instance variables in the object `anObject`.

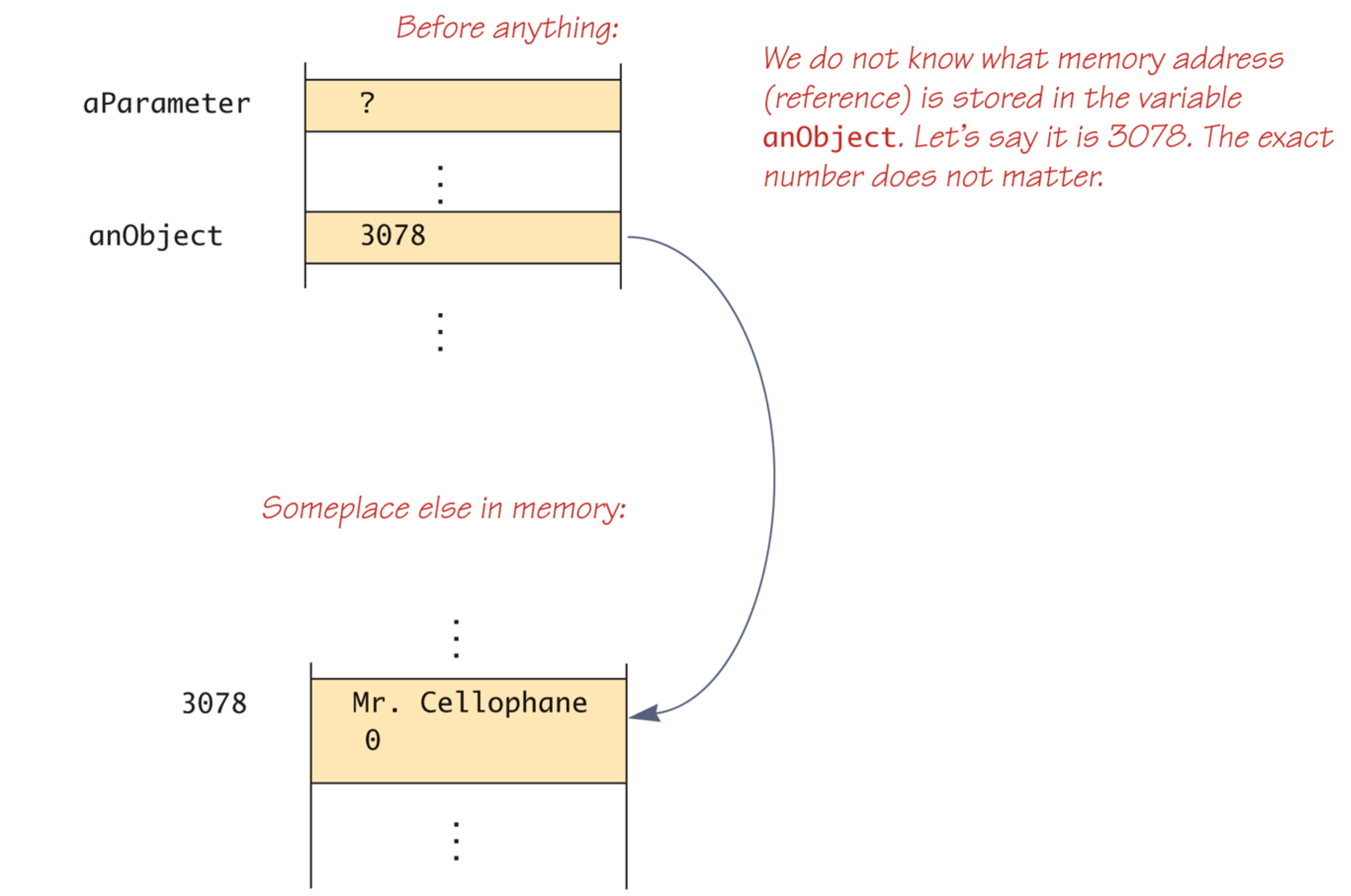
Sample Dialogue

```
Mr. Cellophane 0
Now we call changer with anObject as argument.
Hot Shot 42
```

```java
class ToyClass{
    private String name;
    private int number;
    public ToyClass(String initialName, int initialNumber){
        name = initialName;
        number = initialNumber;
    }
    public ToyClass(){
        name = "No name yet.";
        number = 0;
    }
    public static void changer(ToyClass aParameter){
        aParameter.name = "Hot Shot";
        aParameter.number = 42;
    }
    public void tryToMakeEqual(int aNumber){
        aNumber = number;
    }
    public boolean equals(ToyClass otherObject){
        return ((name.equals(otherObject.name)) && (number == otherObject.number) );
    }
    public String toString(){
        return (name + " " + number);
    }
}
```

Object Parameters - Memory Picture(Part 1 of 3)

Display 5.15 **Memory Picture for Display 5.14**

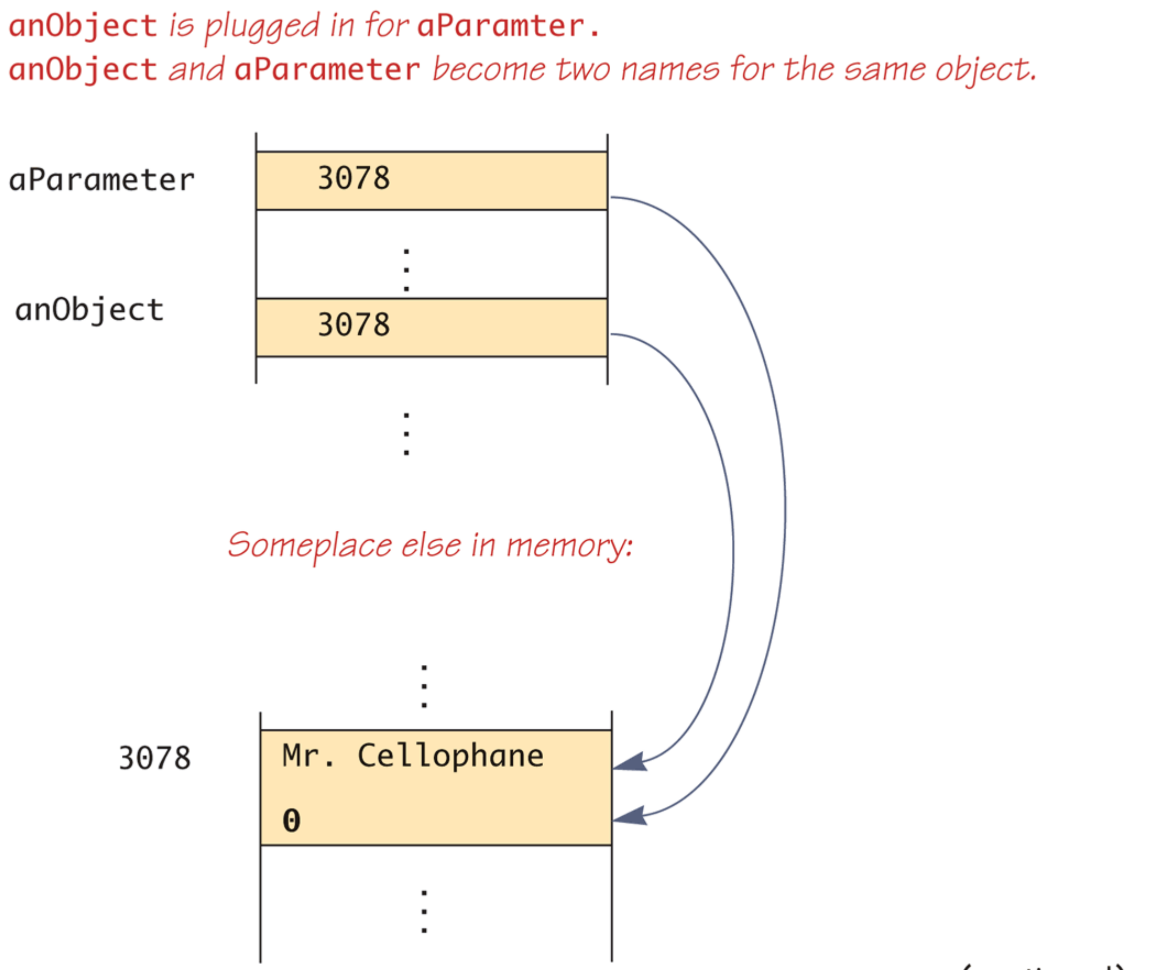


Before anything:
aParameter
?
anObject
3078
We do not know what memory address (reference) is stored in the variable `anObject`*. Let's say it is 3078. The exact number does not matter.*
Someplace else in memory:
3078
Mr. Cellophane
0

(continued)

Memory Picture for Display 5.14 (Part 2 of 3)

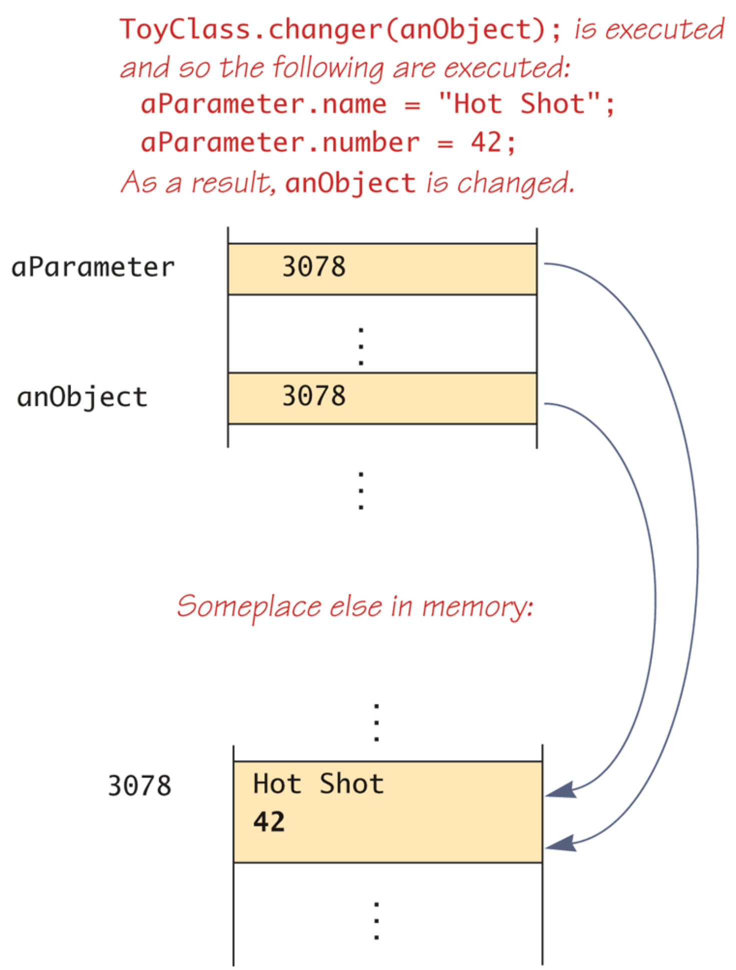
Display 5.15 **Memory Picture for Display 5.14**


anObject is plugged in for aParamter.
anObject and aParameter become two names for the same object.
aParameter
3078
anObject
3078
Someplace else in memory:
3078
Mr. Cellophane
0

(continued)

Memory Picture for Display 5.14 (Part 3 of 3)

Display 5.15 **Memory Picture for Display 5.14**


ToyClass.changer(anObject); is executed
and so the following are executed:
aParameter.name = "Hot Shot";
aParameter.number = 42;
As a result, anObject is changed.
aParameter
3078
anObject
3078
Someplace else in memory:
3078
Hot Shot
42

Return Objects From Methods

```
public class Complex{
    private double real;
    private double img;
    //Default Constructor
    public Complex(){
        real = 0.0;
        img  = 0.0;
    }
    //Overloaded Constructor
    public Complex(double r, double im){
        real = r;
        img  = im;
    }
    //Adding Two Complex objects and return Complex object
    public Complex addComplex(Complex b){
        double r = real + b.real;
        double i = img + b.img;
        //Create a temporary Complex to return it
        Complex temp = new Complex(r , i);
        return temp;
        //Or return new Complex(r , i);
    }
    //toString Method to display object values in instance variables

    public String toString(){
        return(real+" "+img);
    }
}
```

Main class

```
Complex c1 = new Complex(11 , 2.3);
Complex c2 = new Complex(9 , 2.7);
System.out.println("Complex-1: "+c1);
System.out.println("Complex-2: "+c2);

Complex c3 = c1.addComplex(c2);

System.out.println("Complex-3: "+c3);
```

```
Complex-1: 11.0 2.3
Complex-2: 9.0 2.7
Complex-3: 20.0 5.0
```

Objects can be passed natively, just like primitives

```java
public class Point{
    public int x;
    public int y;
    public Point(int a, int b){
        x = a;
        y = b;
    }
    public Point(){}
    public void tricky(Point pa , Point pb){
        Point temp = new Point();
        temp = pa;
        pa   = pb;
        pb   = temp;
        System.out.println("pa.X: "+pa.x + " pa.Y: "+pa.y);
        System.out.println("pb.X: "+pb.x + " pb.Y: "+pb.y);
    }

}
```

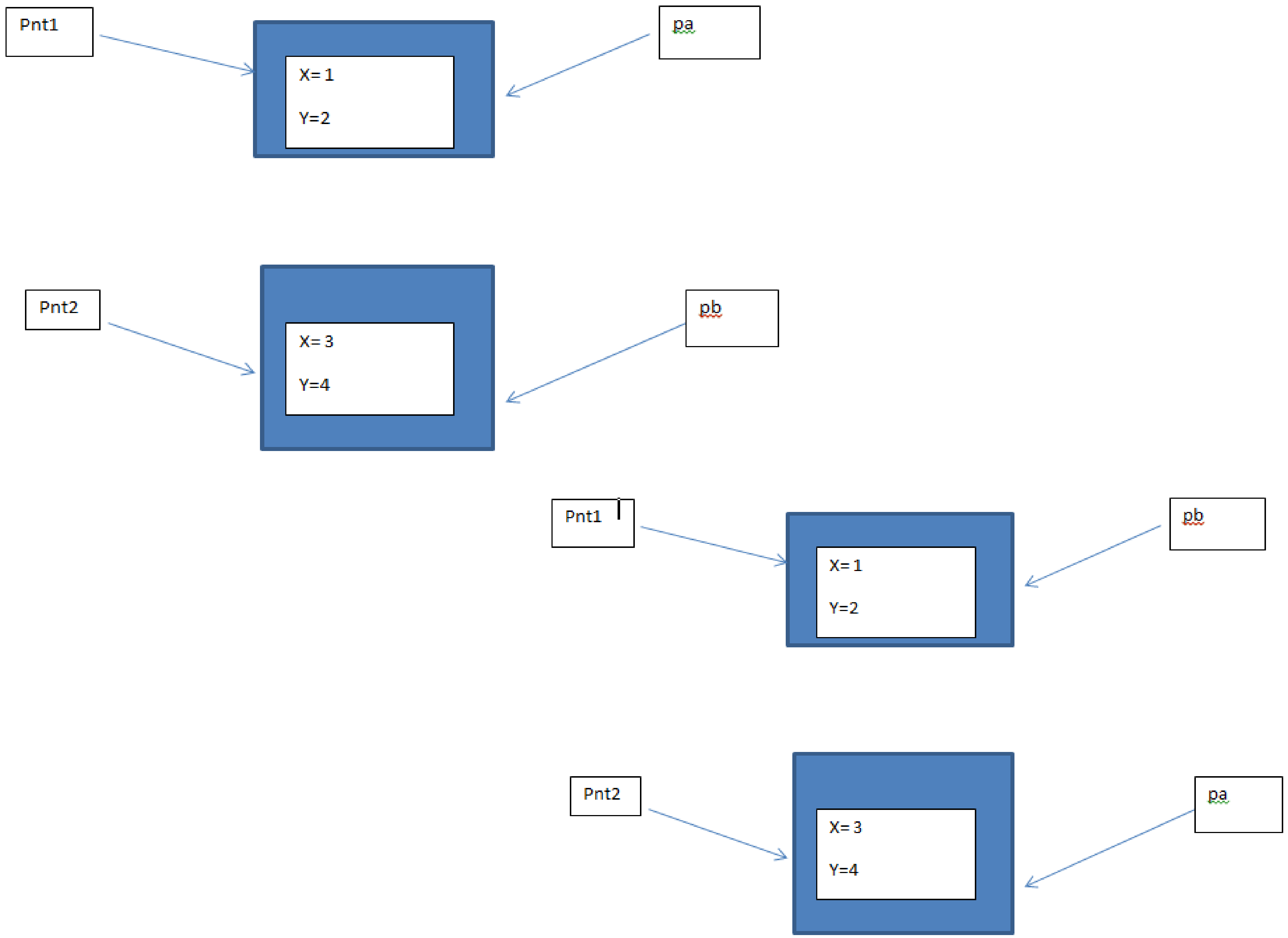
The method “tricky” is not performing swapping of object passed by main(), it swaps the objects in the function “tricky”

Main Method

```java
    Point pnt1 = new Point(1,2);
    Point pnt2 = new Point(3,4);
    System.out.println("pnt1.X: "+pnt1.x + " pnt2.Y: "+pnt1.y);
    System.out.println("pnt2.X: "+pnt2.x + " pnt2.Y: "+pnt2.y);

    pnt1.tricky(pnt1 , pnt2);
    System.out.println("pnt1.X: "+pnt1.x + " pnt1.Y: "+pnt1.y);
    System.out.println("pnt2.X: "+pnt2.x + " pnt2.Y: "+pnt2.y);
```

```
pnt1.X: 1 pnt2.Y: 2
pnt2.X: 3 pnt2.Y: 4
pa.X: 3 pa.Y: 4
pb.X: 1 pb.Y: 2
pnt1.X: 1 pnt1.Y: 2
pnt2.X: 3 pnt2.Y: 4
```


Pnt1
X= 1
Y=2
pa
Pnt2
X= 3
Y=4
pb
Pnt1
X= 1
Y=2
pb
Pnt2
X= 3
Y=4
pa

The Constant **`null`**

- **`null`** is a special constant that may be assigned to a variable of any class type

```
    YourClass yourObject = null;
```

- It is used to indicate that the variable has no "real value"
  - It is often used in constructors to initialize class type instance variables when there is no obvious object to use
- **`null`** is not an object: It is, rather, a kind of "placeholder" for a reference that does not name any memory location
  - Because it is like a memory address, use **==** or **!=** (instead of **`equals`**) to test if a class variable contains null

```
    if (yourObject == null)
       System.out.println("No real object here.");
```

Pitfall: Null Pointer Exception

- Even though a class variable can be initialized to **`null`**, this does not mean that **`null`** is an object
  - **`null`** is only a placeholder for an object

- Any attempt to do this will result in a "Null Pointer Exception" error message

```
ToyClass2 aVariable = null ;
String representation = aVariable.toString();
```

Anonymous Objects

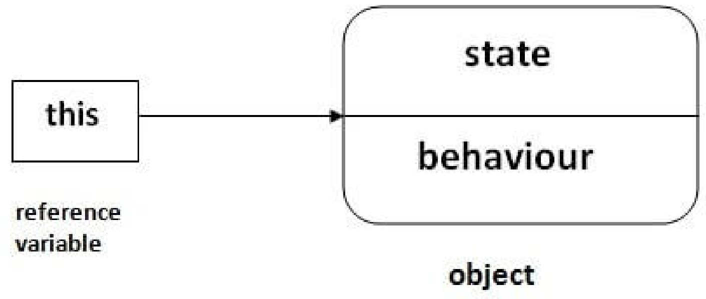
- Sometimes the object created is used as an argument to a method, and never used again
  - In this case, the object need not be assigned to a variable, i.e., given a name
- An object whose reference is not assigned to a variable is called an **`anonymous object`**

```
if (variable1.equals( new ToyClass("JOE", 42)))
    System.out.println("Equal");
else
    System.out.println("Not equal");
```

=

```
ToyClass temp = new ToyClass("JOE", 42);
if (variable1.equals(temp))
    System.out.println("Equal");
else
    System.out.println("Not equal");
```

this Pointer



- `this` is a **reference variable** that refers to the current object
- `this` can be used to refer current class instance variable
- `this` can be used to invoke current class method
- `this()` can be used to invoke current class constructor
- `this` can be passed as an argument in the method call
- `this` can be passed as argument in the constructor call
- `this` can be used to return the current class instance from the method

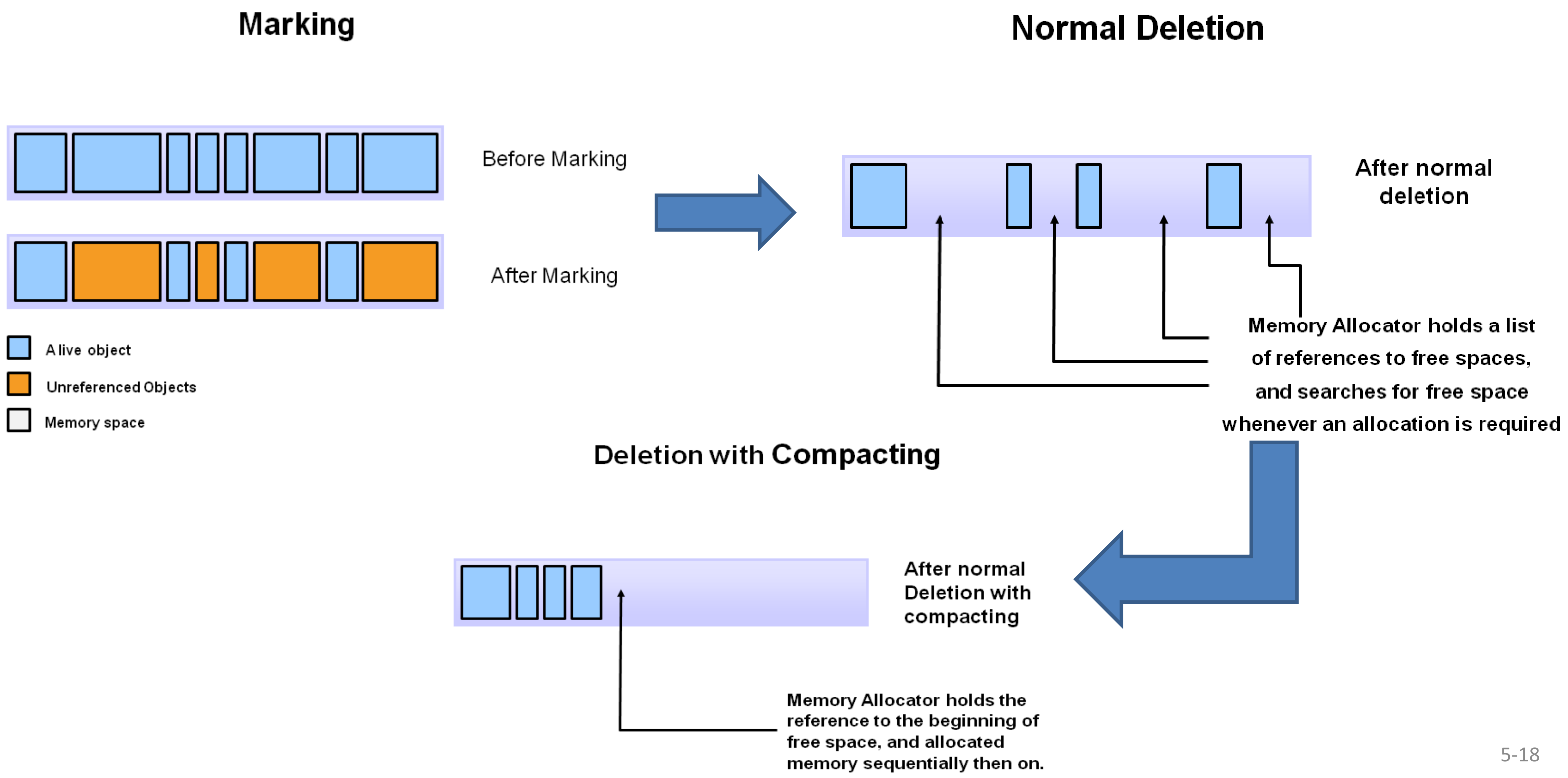
Important Points

- If another object is required for the operation of a method , we need to pass it through the argument.
- Class name is a user defined type.
- Class references can be used as function argument
- Class references can be returned from Functions
- Object is a composite entity
- Do not apply any arithmetic and logical operation on object name directly.

Java Garbage Collection

- In java, garbage means unreferenced objects.
- Garbage Collection is process of reclaiming the runtime unused memory automatically.
- Advantages
  - It makes java **memory efficient** because garbage collector removes the unreferenced objects from heap memory.
  - It is **automatically done** by the garbage collector(a part of JVM) so we don't need to make extra efforts.

Garbage Collection - Basic Process



Memory Allocator holds a list of references to free spaces, and searches for free space whenever an allocation is required

Memory Allocator holds the reference to the beginning of free space, and allocated memory sequentially then on.

How can an object be unreferenced?

- By nulling a reference:

```
Employee e=new Employee();
e=null;
```

- By assigning a reference to another:

```
Employee e1=new Employee();
Employee e2=new Employee();
e1=e2;//now the first object referred by e1 is available for garbage collection
```

- By anonymous object:

```
new Employee();
```

Java Object finalize() Method

- Finalize() is the method of Object class
- Called just before an object is garbage collected

```
protected void finalize(){}
```