Declaring/Creating Objects in a Single Step

```
ClassName objectRefVar = new ClassName();
```

Example:

Assign object reference

Create an object

```
Circle myCircle = new Circle();
```

Accessing Object's Members

- Referencing the object's data:

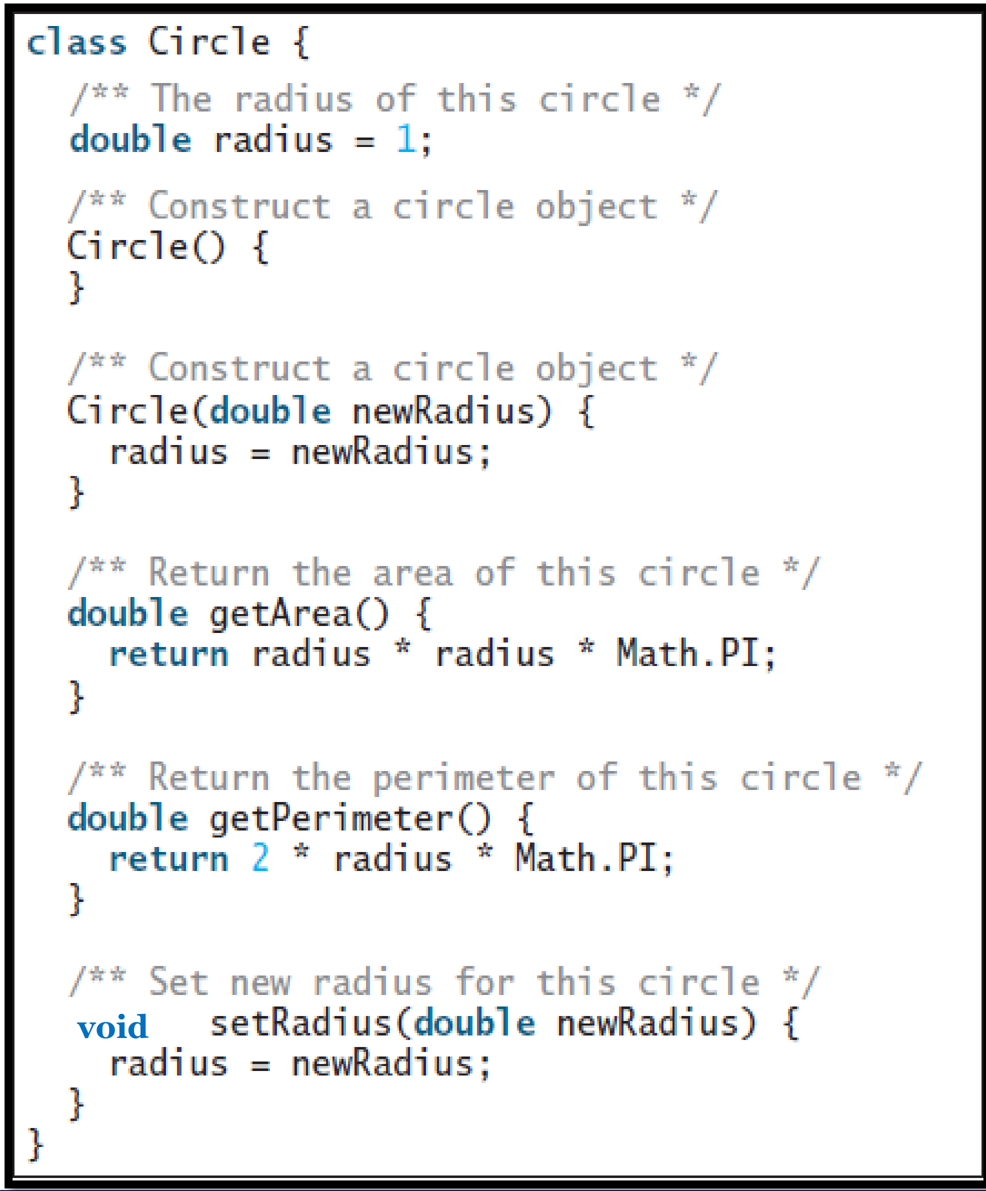
  `objectRefVar.data`

  e.g., `myCircle.radius`

- Invoking the object's method:

  `objectRefVar.methodName(arguments)`

  e.g., `myCircle.getArea()`



```
class Circle {
  /** The radius of this circle */
  double radius = 1;

  /** Construct a circle object */
  Circle() {
  }

  /** Construct a circle object */
  Circle(double newRadius) {
    radius = newRadius;
  }

  /** Return the area of this circle */
  double getArea() {
    return radius * radius * Math.PI;
  }

  /** Return the perimeter of this circle */
  double getPerimeter() {
    return 2 * radius * Math.PI;
  }

  /** Set new radius for this circle */
  void    setRadius(double newRadius) {
    radius = newRadius;
  }
}
```

Check Point

1) Which of the following reserved words in Java is used to create an instance of a class?

[✓ A) new]

B) public or private, either could be used

C) import

D) class

E) public

Check Point

2) A class constructor usually defines
- A) the number of methods in the class
- B) how an object is interfaced
- C) the number of instance data in the class
- [✓ D) how an object is initialized]
- E) if the instance data are accessible outside of the object directly

3) Having multiple class methods of the **same name** where each method has a **different number of or type of parameters** is known as
- A) information hiding
- [✓ B) method overloading]
- C) importing
- D) encapsulation
- E) tokenizing

Check Point

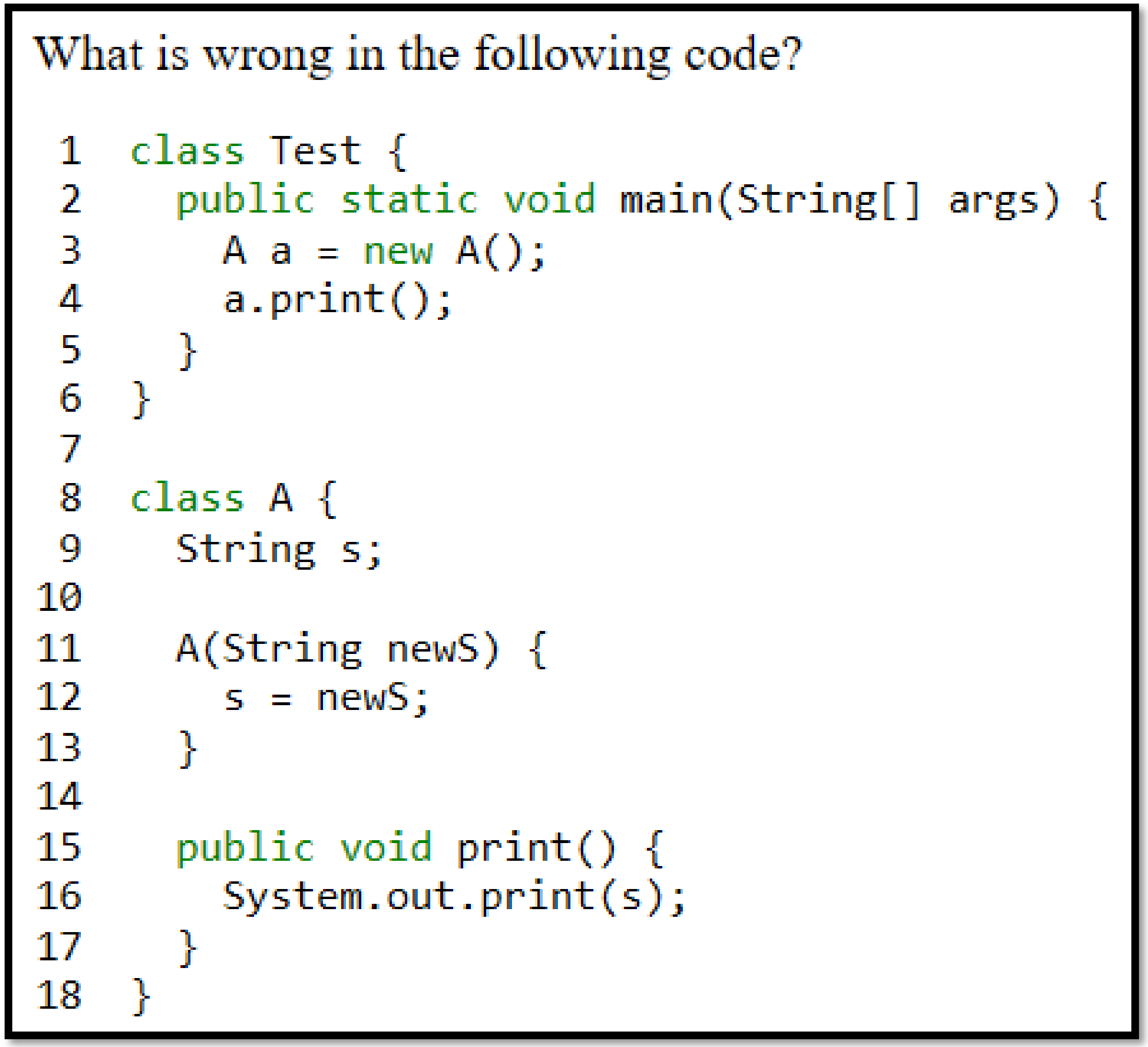
What are the differences between constructors and methods?

Constructors are special kinds of methods that are called when creating an object using the **new operator**. Constructors **do not have a return type-not even void**.

When will a class have a default constructor?

A class has a default constructor only if the class does not define any constructor.

Check Point



What is wrong in the following code?

```
class Test {
  public static void main(String[] args) {
    A a = new A();
    a.print();
  }
}

class A {
  String s;

  A(String newS) {
    s = newS;
  }

  public void print() {
    System.out.print(s);
  }
}
```

The program does not compile because new A() is used in class Test, but class A **does not have a default constructor**.

Reference Data Fields

The data fields can be of reference types. For example, the following Student class contains a data field name of the String type.

```
public class Student {
  String name; // name has default value null
  int age; // age has default value 0
  boolean isScienceMajor; // isScienceMajor has default value false
  char gender; // c has default value '\u0000'
}
```

The null Value

If a data field of a reference type does not reference any object, the data field holds a special literal value, null.

Default Value for a Data Field

The default value of a data field is

- null for a reference type
- 0 for a numeric type
- false for a boolean type
- '\u0000' for a char type.

Java assigns no default value to a **local variable inside a method.**

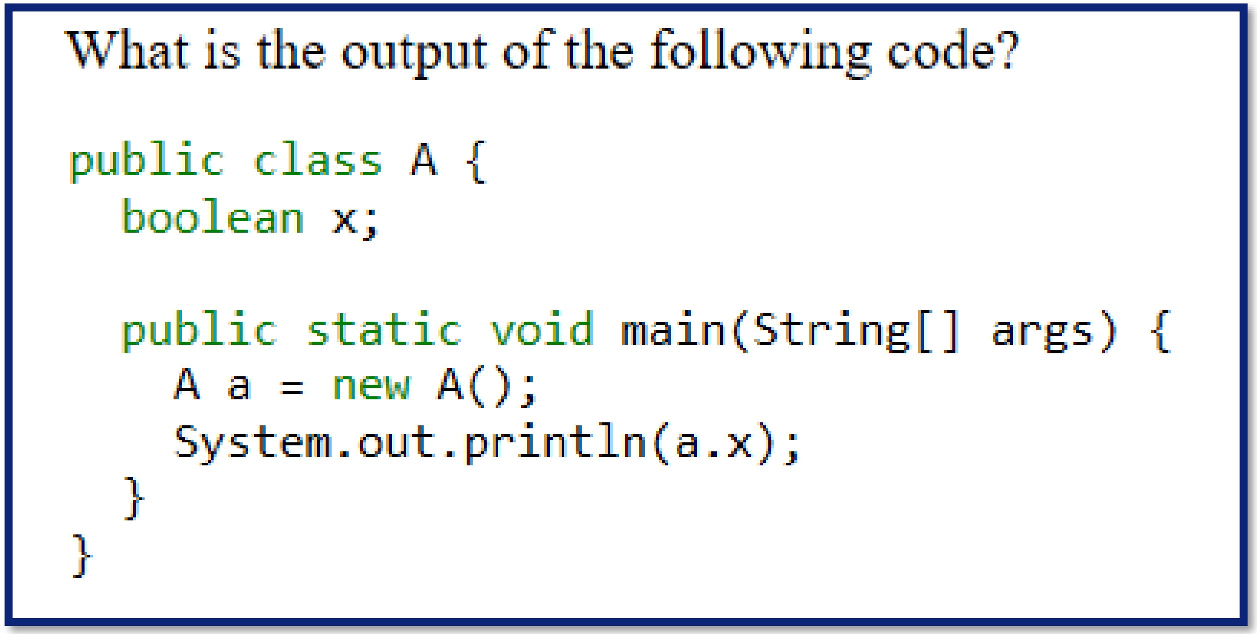
Example

Java assigns no default value to **a local variable inside a method.**

```
public class Test {
  public static void main(String[] args) {
    int x; // x has no default value
    String y; // y has no default value
    System.out.println("x is " + x);
    System.out.println("y is " + y);
  }
}
```

Compile error: variable not initialized

Check Point

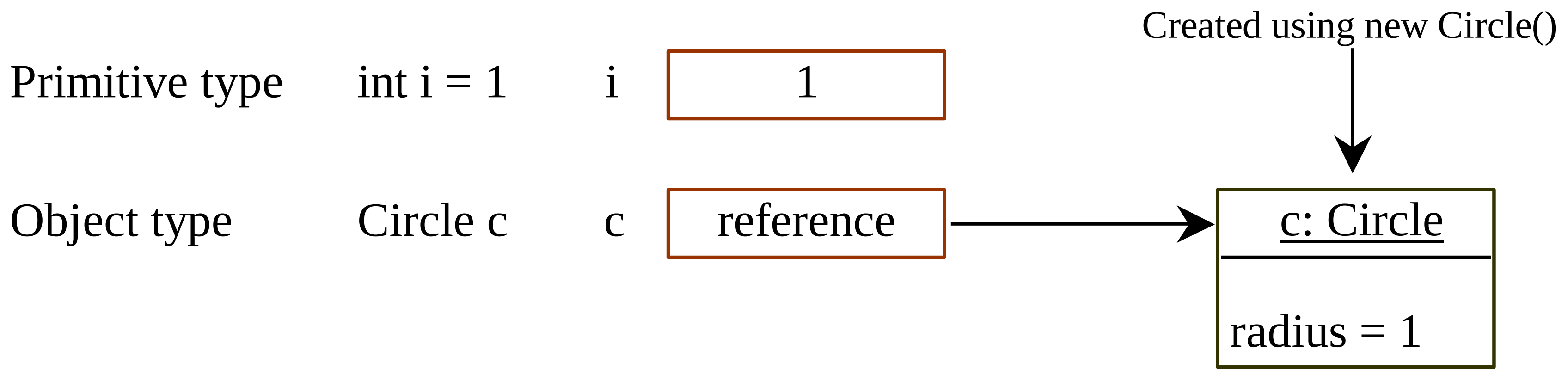


What is the output of the following code?

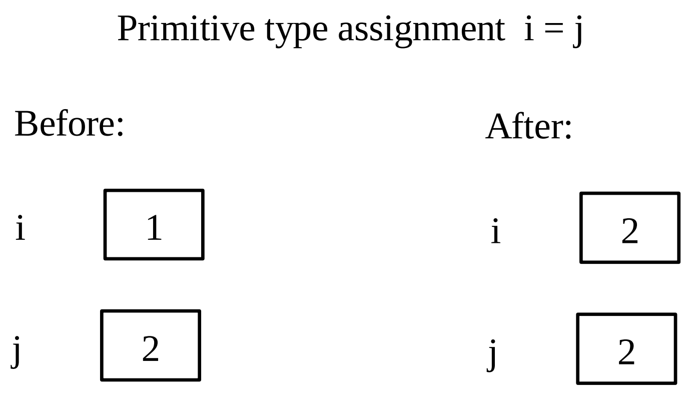
```
public class A {
  boolean x;

  public static void main(String[] args) {
    A a = new A();
    System.out.println(a.x);
  }
}
```

false

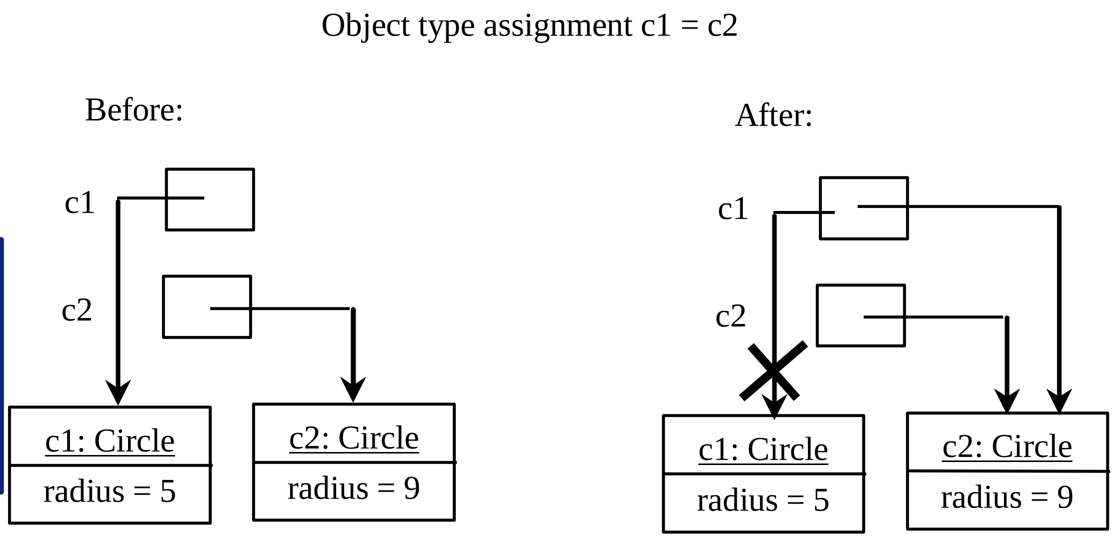
Differences between Variables of Primitive Data Types and Object Types



Copying Variables of Primitive Data Types and Object Types



```
int i=1;
int j=2;
i=j;
```

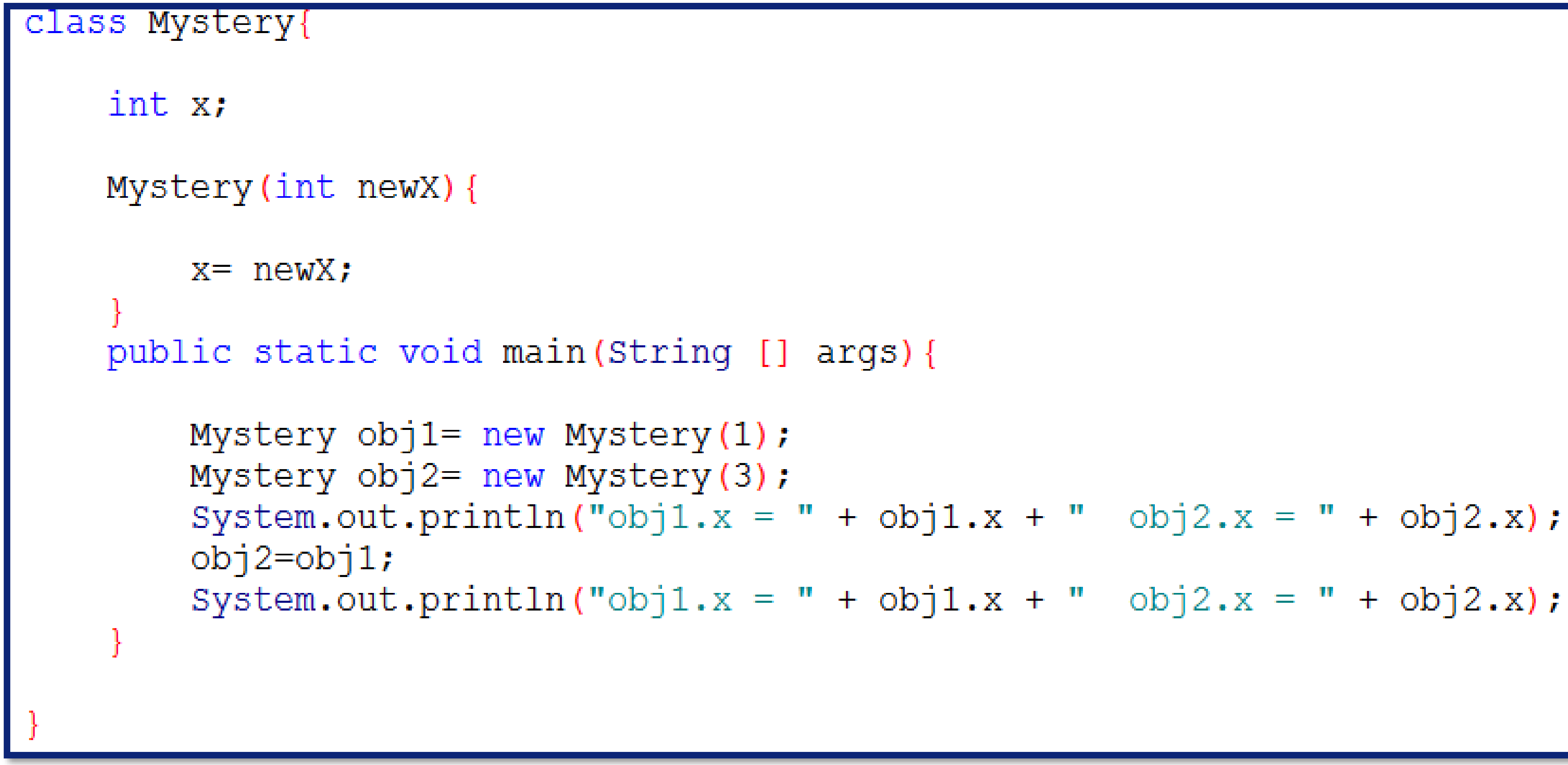


```
Circle c1= new Circle (5);
Circle c2= new Circle (9);
 c1=c2;
```

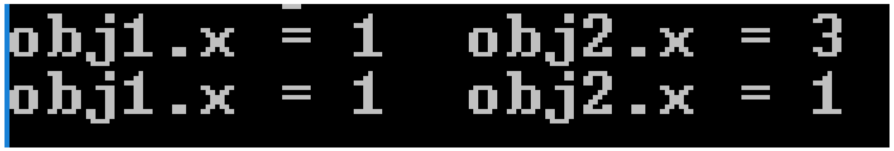
Garbage Collection

As shown in the previous figure, after the assignment statement c1 = c2, c1 points to the same object referenced by c2. The object previously referenced by c1 is no longer referenced. This object is known as garbage. Garbage is automatically collected by JVM.

Example: Output



```
class Mystery{

    int x;

    Mystery(int newX){

        x= newX;
    }
    public static void main(String [] args){

        Mystery obj1= new Mystery(1);
        Mystery obj2= new Mystery(3);
        System.out.println("obj1.x = " + obj1.x + "  obj2.x = " + obj2.x);
        obj2=obj1;
        System.out.println("obj1.x = " + obj1.x + "  obj2.x = " + obj2.x);
    }

}
```

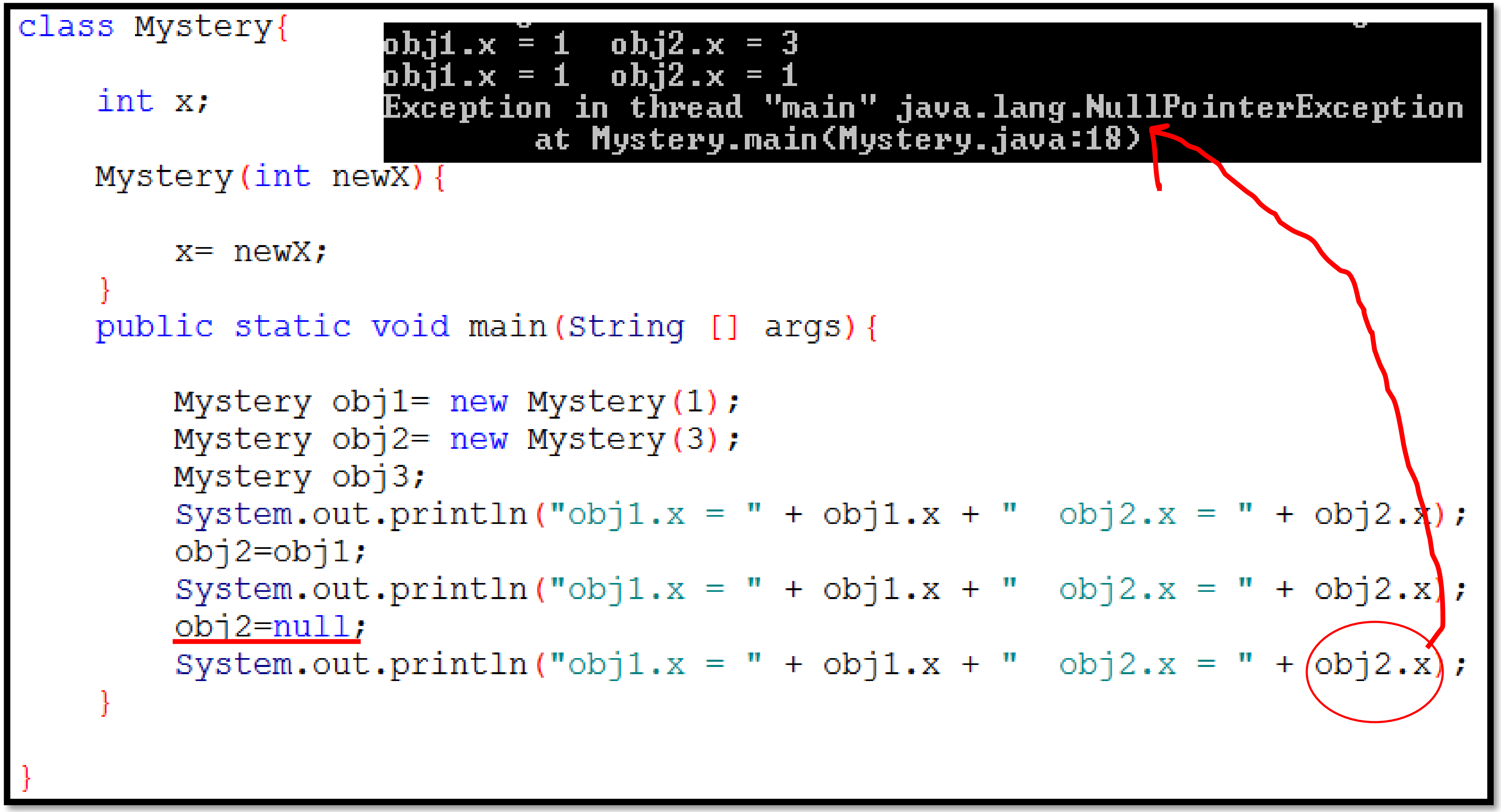


```
obj1.x = 1  obj2.x = 3
obj1.x = 1  obj2.x = 1
```

Garbage Collection, cont

TIP: If you know that an object is no longer needed, you can explicitly assign null to a reference variable for the object. The JVM will automatically collect the space if the object is not referenced by any variable .

Example: Output



```
obj1.x = 1   obj2.x = 3
obj1.x = 1   obj2.x = 1
Exception in thread "main" java.lang.NullPointerException
        at Mystery.main(Mystery.java:18)
```

```java
class Mystery{

    int x;

    Mystery(int newX){

        x= newX;
    }
    public static void main(String [] args){

        Mystery obj1= new Mystery(1);
        Mystery obj2= new Mystery(3);
        Mystery obj3;
        System.out.println("obj1.x = " + obj1.x + "  obj2.x = " + obj2.x);
        obj2=obj1;
        System.out.println("obj1.x = " + obj1.x + "  obj2.x = " + obj2.x);
        obj2=null;
        System.out.println("obj1.x = " + obj1.x + "  obj2.x = " + obj2.x);
    }

}
```

Instance Variables, and Methods

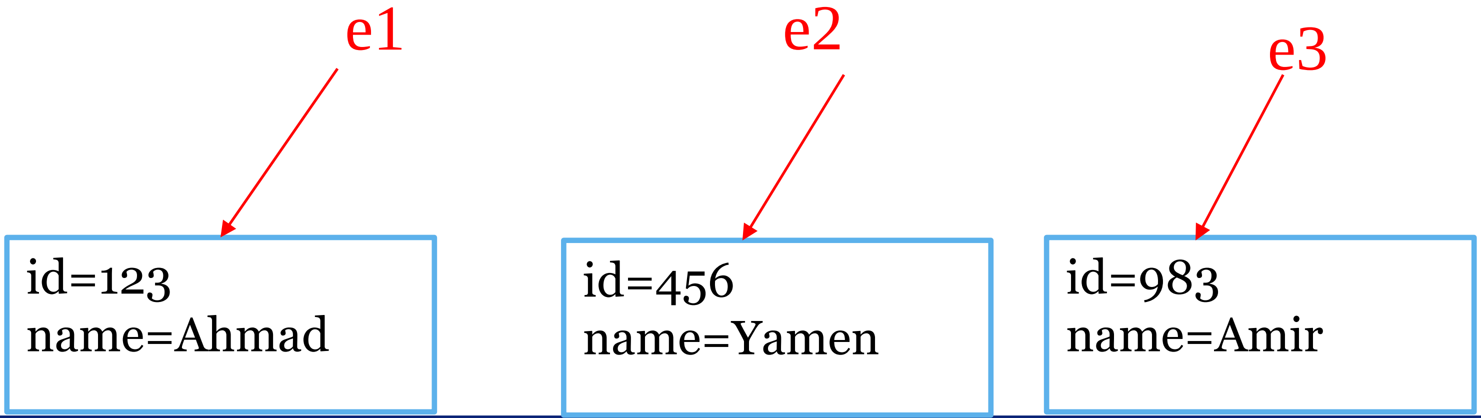
Instance variables(**non-static variables**) belong to a specific instance.

Instance methods(**non-static methods**) are invoked by an instance of the class.

Example

```
class Employee{

    int id;
    String name;

    Employee (int newId, String newName){
            id=newId;
            name=newName;
    }
   public static void main (String [] args){

         Employee e1= new Employee(123,"Ahamd");
         Employee e2= new Employee(456,"Yamen");
         Employee e3= new Employee(983,"Amir");

   }

}
```

Instance variables



Static Variables, Constants, and Methods

Static variables are **shared by all the instances of the class.**

Static methods are **not tied to a specific object**.

Static constants are **final variables shared by all the instances of the class.**