innovate achieve lead

BITS Pilani
Pilani Campus

Access Modifiers, Constructors, and Reference Types

Access Modifiers – public variables

```
public class A {
        public int i, j;
        public void show(){
                System.out.println("i = " + i + " and j = " + j );
        }
}
```

```
public class Test {
    public static void main(String[] args) {
        A a = new A();
        a.i = 5;
        a.j = 10;
        a.show();
    }
}
```

```
/* public variables can be accessed
outside of class definition */
```

Access Modifiers – private variables

```
public class A {
        private int i, j;
        public void show(){
                System.out.println("i = " + i + " and j = " + j );
        }
}
```

```
public class Test {
    public static void main(String[] args) {
        A a = new A();
        a.i = 5;
        a.j = 10;
        a.show();
    }
}
```

```
Test.java:11: i has private access in A
        a.i = 5;
         ^
Test.java:12: j has private access in A
        a.j = 10;
         ^
2 errors
```

Access Modifiers – public methods

```
public class A {
        public int i, j;
        public void show(){
                System.out.println("i = " + i + " and j = " + j );
        }
}
```

```
public class Test {
    public static void main(String[] args) {
        A a = new A();
        a.i = 5;
        a.j = 10;
        a.show();
    }
}
```

```
/* public methods of a class can be invoked from outside of the class definition */
```

Access Modifiers – private methods

```
public class A {
        private int i = 5, j = 10;
        private void show(){
                System.out.println("i = " + i + " and j = " + j );
        }
}
```

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

```
/* The method show() from the type A is
not visible */
```

Access Modifiers –

```
public class A {
        private int i, j;
        private void show(){
                System.out.println("i = " + i + " and j = " + j );
        }
        public void display(){
            i = 5;
            j = 10;
            show();
        }
}
```

```
/* public or private methods of a class
can refer to both public or private fields
and methods of the same class*/
```

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

Constructors –

```
public class A {
        private int i;
        public A(){
                i = 5;
        }
        public void show(){
                System.out.println(" Value of i = " + i)
        }
}
```

```
/* Constructor is called to instantiate
a class and initialize the fields */
```

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

```
/* A call to the constructor is the
first thing that happens when you try
to instantiate any class */
```

Overloading Constructors –

```
public class A {

    private int i;

    public A(){
       i = 5;
    }

    public A(int x){
       i = x;
    }

    public void show(){
       System.out.println(" Value of i = " + i)
    }
}
```

```
/* Overloaded constructors are used to instantiate and initialize the fields with the values supplied via the constructor */
```

Overloading Constructors (Contd.)

```
public class Student {

   private int id;
   private String name;
   private double cgpa;

   public Student(){
        id = 0;
        name = null;
        cgpa = 0.0;
   }

   public Student(int i,
        String nam,
        double cgp){
        id = i;
        name = nam;
        cgpa = cgp;
   }
}
```

```
public class TestStudent {
public static void main(String[] args){
    Student s1 = new Student();
    Student s2 = new Student(1, "Avi",9.0);
    Student s3 = new Student(2, "Dol",9.0);
  }
}
```

```
/* When a class is instantiated the JVM
first looks for the no argument constructor
defined for this class, if it is not found
then the default constructor is called. If
you have defined an overloaded constructor
for this class then the class should always
be instantiated with all the arguments
specified in their correct order as
specified in the constructor declaration.
This is because, any constructor that you
explicitly define hides the default
constructor */
```

Pass by Value

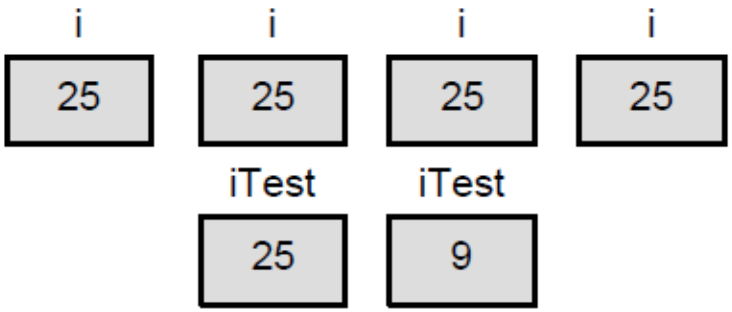
1. Most methods are passed arguments when they are called. An argument may be a constant or a variable.
   - For example, in the expression `Math.sqrt(33)`, the constant 33 is passed to the sqrt() method of the Math class. In the expression `Math.sqrt(x)`, the variable x is passed.

2. This is simple enough, however there is an important but simple principle at work here. If a variable is passed, the method receives a copy of the variable's value. The value of the original variable cannot be changed within the method. This seems reasonable because the method only has a copy of the value; it does not have access to the original variable. This process is called pass by value.

Pass by Value - Example

```
public class DemoPassByValue {
        public static void main(String[] args) {
                int i = 25;
                System.out.println(i);
                iMethod(i);
                System.out.println(i);
        }

        public static void iMethod(int iTest)  {
                iTest = 9; // change it
                System.out.println(iTest);
        }
}
```



Pass by Reference

1. See Java Memory Management Slides before reading this section.
2. If the variable passed as an object, then the effect is different.
3. We often say things like,
   - This method returns an object
   - This method is passed an object as an argument
4. This not quite true, more precisely, we should say,
   - This method returns a reference to an object
   - This method is passed a reference to an object as an argument
5. In fact, objects, per se, are never passed to methods or returned by methods. It is always "a reference to an object" that is passed or returned.
6. The term "variable" also deserves clarification. There are two types of variables in Java:
   - Those that hold the value of a primitive data type and those that hold a reference to an object.
   - A variable that holds a reference to an object is an "object variable" – the prefix "object" is often dropped for brevity.

Pass by Reference vs Pass by Value

1. Pass by value refers to passing a constant or a variable holding a primitive data type to a method, and pass by reference refers to passing an object variable to a method.
2. In both cases a copy of the variable is passed to the method. It is a copy of the "value" for a primitive data type variable; it is a copy of the "reference" for an object variable. So, a method receiving an object variable as an argument receives a copy of the reference to the original object.
3. Here's the catch: If the method uses that reference to make changes to the object, then the original object is changed. This is reasonable because both the original reference and the copy of the reference "refer to" to same thing – the original object.
4. There is one exception: Strings. Since String objects are immutable in Java, a method that is passed a reference to a String object cannot change the original object.

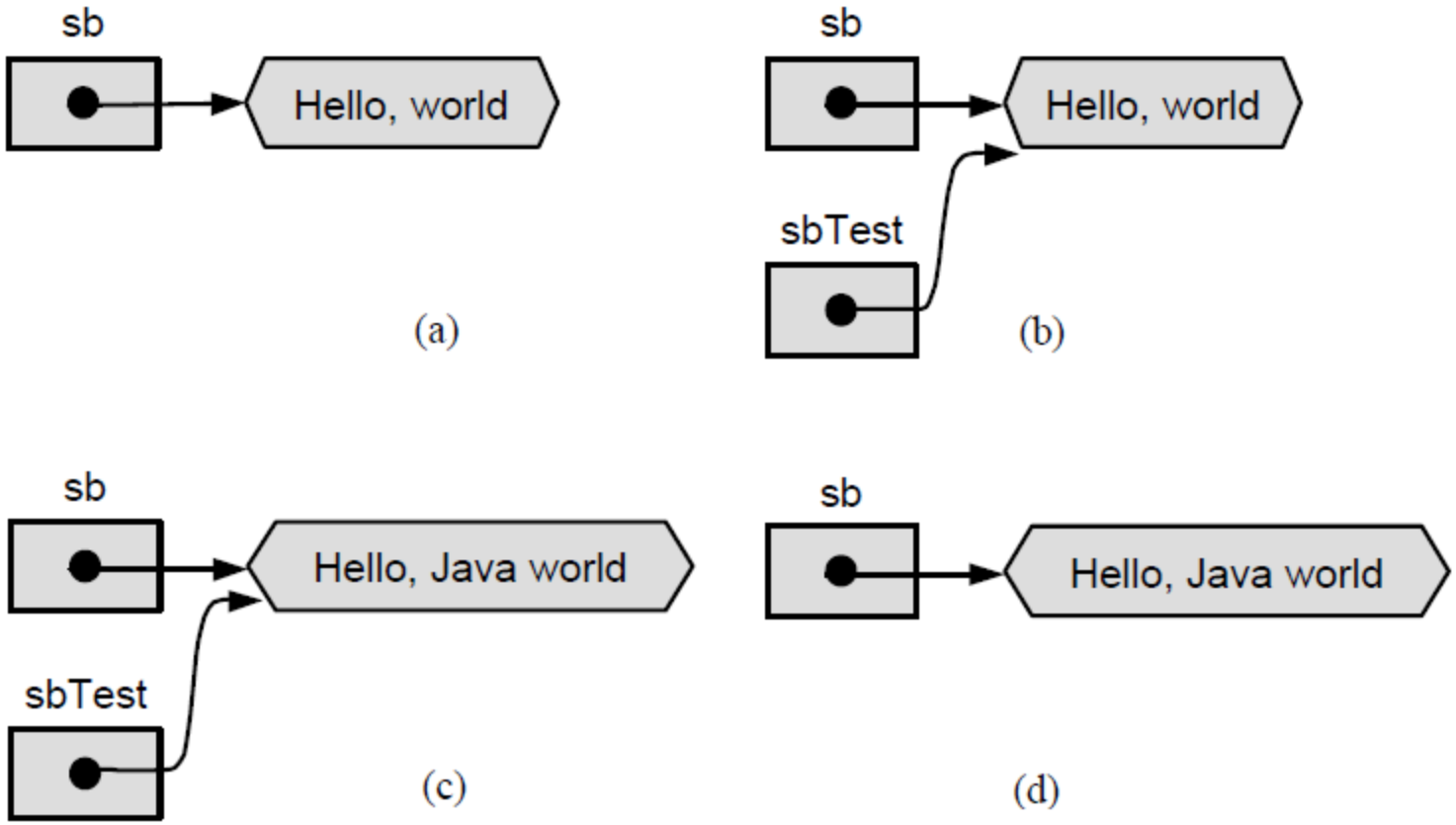
Pass by Reference – Example (StringBuffer)

```
public class DemoPassByReference1 {

        public static void main(String[] args) {
            StringBuffer sb = new StringBuffer("Hello, world");
            System.out.println(sb);
            sbMethod(sb);
            System.out.println(sb);
        }

        public static void sbMethod(StringBuffer sbTest) {
            sbTest = sbTest.insert(7, "Java ");
            System.out.println(sbTest);
        }
}
```

Pass by Reference – Example (StringBuffer)



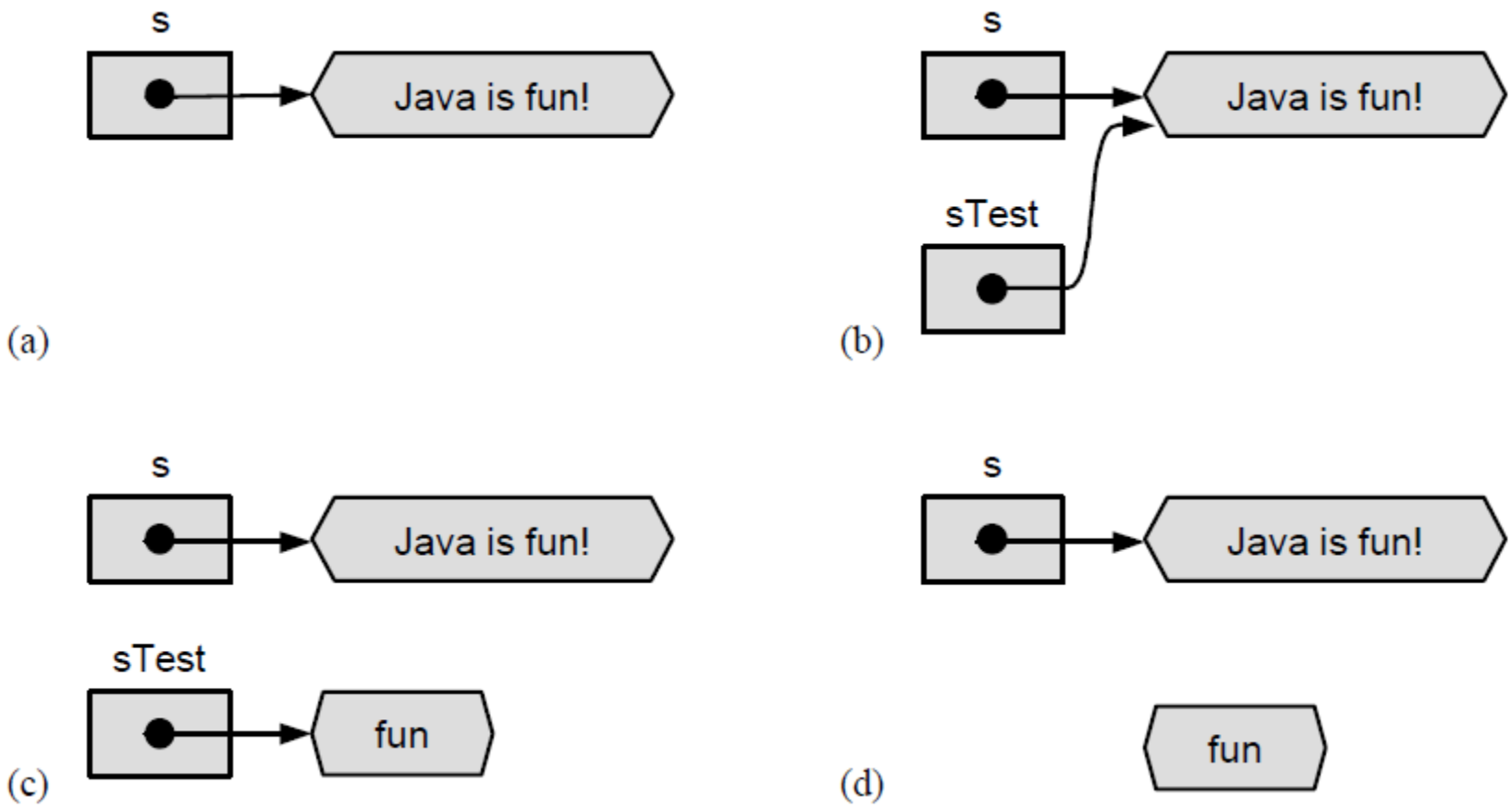
Pass by Reference – Example (String)

```
public class DemoPassByReference2 {

        public static void main(String[] args) {
                String s = "Java is fun!";
                System.out.println(s);
                sMethod(s);
                System.out.println(s);
        }

        public static void sMethod(String sTest){
                sTest = sTest.substring(8, 11);
                System.out.println(sTest);
        }
}
```

Pass by Reference – Example (String)



THANK YOU