Object-Oriented Programming (CS F213)

Module I: Object-Oriented and Java Basics

CS F213 RL 6.1: Objects as Parameters

BITS Pilani

Dr. Pankaj Vyas
Department of Computer Science, BITS-Pilani, Pilani Campus

CS F213 RL 6.1 : Topics

innovate achieve lead

- Passing Objects as Parameters

Objects as Parameters to Methods

- Objects are always 'passed by reference' to the called Methods. Changes/Updates/Modifications made to the state of the object in the called method using the passed object-reference variable will be reflected in the calling method also provided the called method does not change the value of object-reference.
- Primitive Types (byte, short, int etc.) are always 'passed by value' to the called method. Changes/Updates/Modifications made to the primitive type variable in the called method will not be reflected in the calling method.

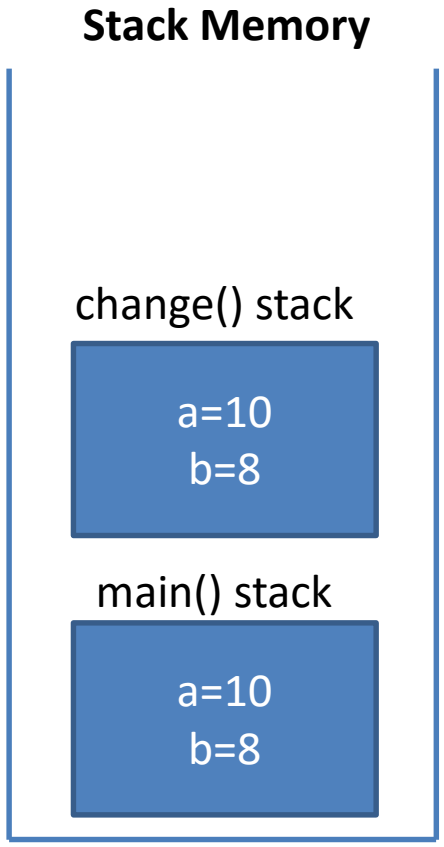
Primitive Types as Parameters

- Primitive Type Variables/Values are always passed by value to called Methods
- Example

```
// File Name: Demo.java
class Test
{
        public      static      void      change(int a, int b)
        {
                    a = 20;
                    b = 40;
        }
        public      static      void      main(String args[])
        {
                    int a = 10, b=8;
                    System.out.println("Before Calling change");
                    System.out.println("a=" +a + "b=" + b);

                    change(a , b);


        }// End of Method
}// End of class
```



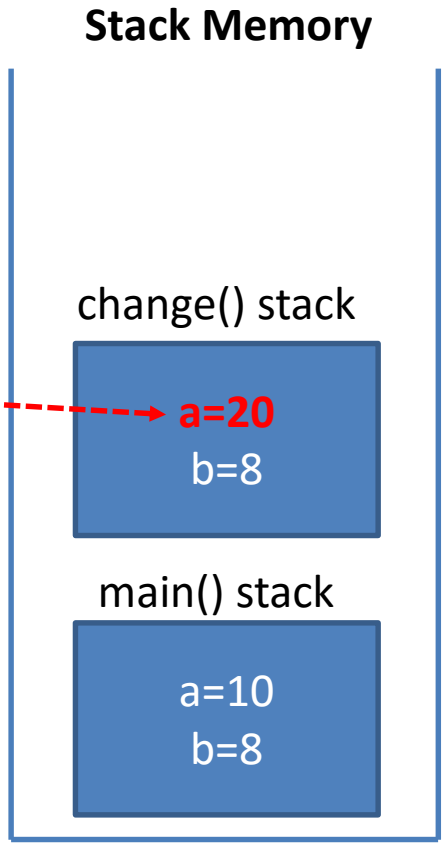
Primitive Types as Parameters

- Primitive Type Variables/Values are always passed by value to called Methods
- Example

```
// File Name: Demo.java
class Test
{
        public      static      void      change(int a, int b)
        {
                a = 20;
                b = 40;
        }
        public      static      void      main(String args[])
        {
                int a = 10, b=8;
                System.out.println("Before Calling change");
                System.out.println("a=" +a + "b=" + b);

                change(a , b);


        }// End of Method
}// End of class
```



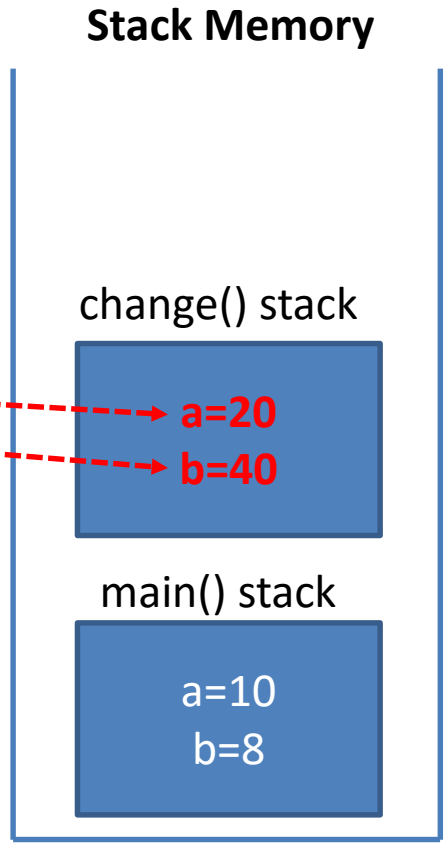
Primitive Types as Parameters

- Primitive Type Variables/Values are always passed by value to called Methods
- Example

```
// File Name: Demo.java
class Test
{
        public      static      void      change(int a, int b)
        {
                    a = 20;
                    b = 40;
        }
        public      static      void      main(String args[])
        {
                    int a = 10, b=8;
                    System.out.println("Before Calling change");
                    System.out.println("a=" +a + "b=" + b);

                    change(a , b);


        }// End of Method
}// End of class
```



Primitive Types as Parameters

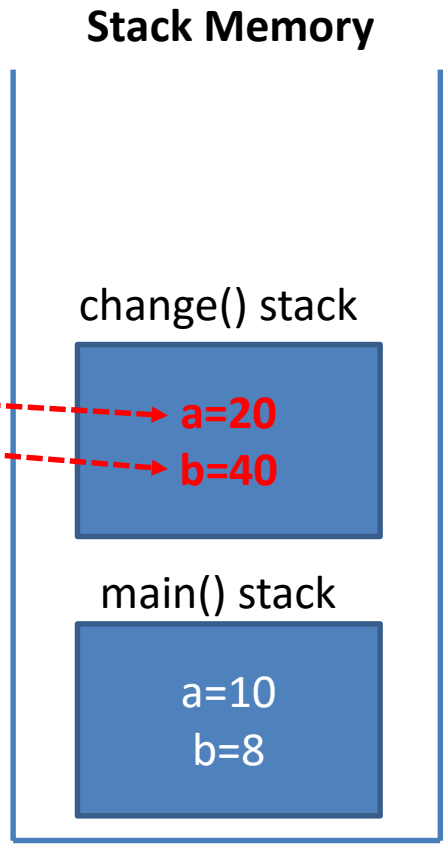
- Primitive Type Variables/Values are always passed by value to called Methods
- Example

```
// File Name: Demo.java
class Test
{
        public      static      void      change(int a, int b)
        {
                    a = 20;
                    b = 40;
        }
        public      static      void      main(String args[])
        {
                    int a = 10, b=8;
                    System.out.println("Before Calling change");
                    System.out.println("a=" +a + "b=" + b);

                    change(a , b);

                    System.out.println("After Calling change");
                    System.out.println("a=" +a + "b=" + b);

        }// End of Method
}// End of class
```



Object Memory Allocation

innovate achieve lead

- Objects are always created dynamically using ‘new’ operator
- Objects are stored in a memory area known as ‘Heap’
- Objects stored in ‘Heap’ memory area can be shared by various methods of the class
- Objects are always passed by reference to called methods

Objects as Parameters : Example 1

```
// File Name : Demo.java
class AB
{
        private     int         a;          // Instance-Field 'a'
        private     int         b;          // Instance-Field 'b'

        // Constructor Method
        AB(int a, int b) { this.a = a; this.b = b; } // End of Constructor

        // Accessor Method for 'a'
        public      int         getA()      { return this.a; }
        // Accessor Method for 'b'
        public      int         getB()      { return this.b; }

        // Mutator Method for 'a'
        public      void        setA(int a) { this.a = a; }
        // Mutator Method for 'b'
        public      void        setB(int b) { this.b = b; }

        // Method to display the values for 'a' and 'b'
        public      void        display()
        {
                System.out.println("a= " + this.a + "b= "+this.b);
        }// End of Method
}// End of class AB
```

Instance-Fields

Constructor

Accessor Methods

Mutator Methods

Method to Display Values of Attributes

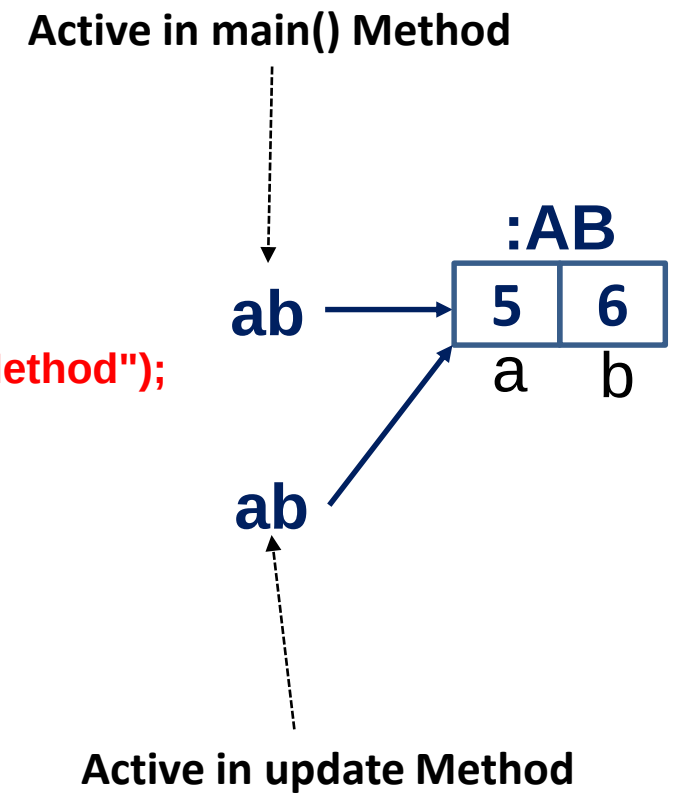
Objects as Parameters : Example 1 ….

```
// Driver class
class Test
{
  public     static     void     update(AB ab)
  {
    ab.setA(56);
    ab.setB(45);
  } // End of Method

  public static          void main(String args[])
  {
    AB     ab     =          new     AB(5,6);
    System.out.println("Before Calling Update Method");
    ab.display();

    update(ab);


  }// End of Method
}// End of class Test
```



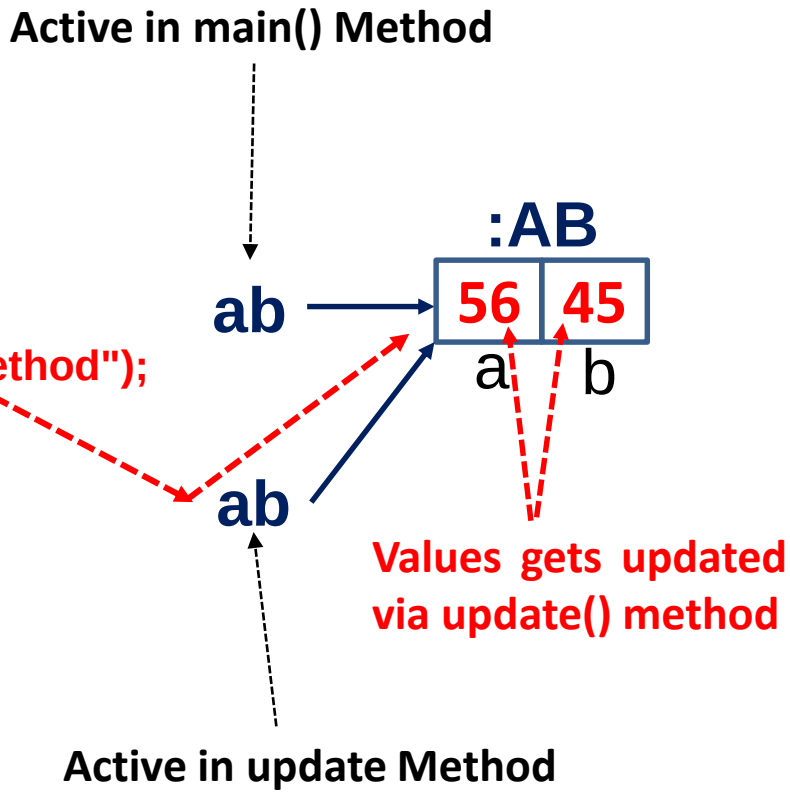
Objects as Parameters : Example 1 ….

```
// Driver class
class Test
{
  public     static     void     update(AB ab)
  {
    ab.setA(56);
    ab.setB(45);
  } // End of Method

  public static          void main(String args[])
  {
    AB     ab     =          new     AB(5,6);
    System.out.println("Before Calling Update Method");
    ab.display();

    update(ab);


  }// End of Method
}// End of class Test
```



Objects as Parameters : Example 1 ….

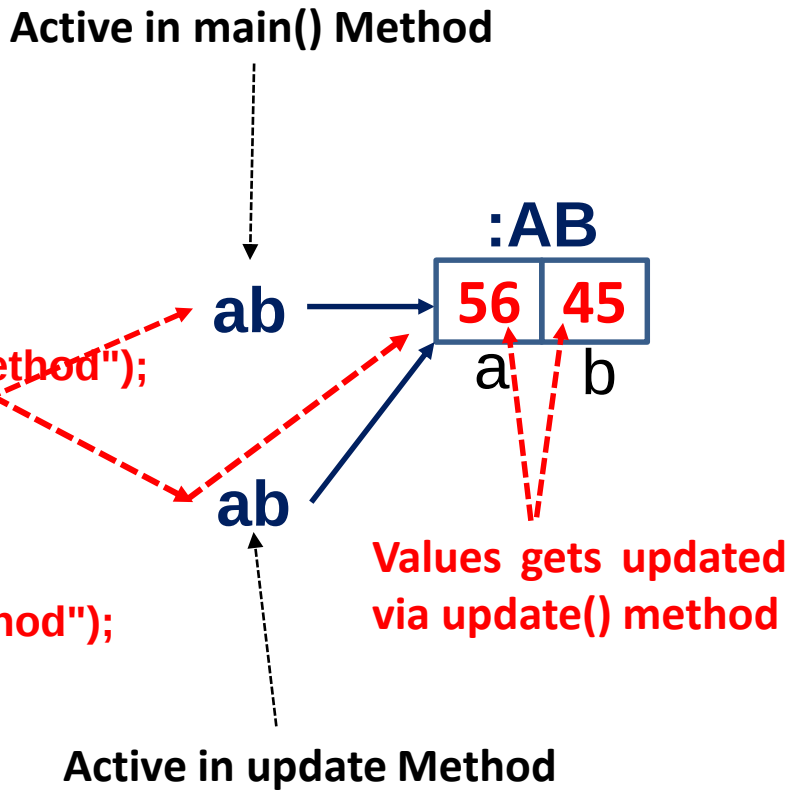
```
// Driver class
class Test
{
  public     static     void     update(AB ab)
  {
    ab.setA(56);
    ab.setB(45);
  } // End of Method

  public static          void main(String args[])
  {
    AB     ab     =     new     AB(5,6);
    System.out.println("Before Calling Update Method");
    ab.display();

    update(ab);

    System.out.println("After Calling Update Method");
    ab.display();

  }// End of Method
}// End of class Test
```



Objects as Parameters : Example 1 ….

innovate achieve lead

Output of Example 1 Program

```
F:\>java Test
Before Calling Update Method
a= 5b= 6
After Calling Update Method
a= 56b= 45
```

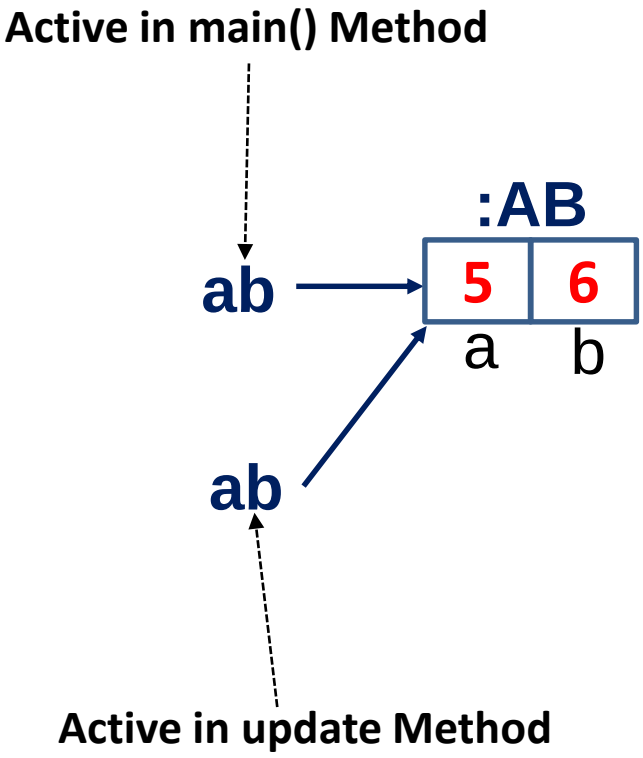
Objects as Parameters : Example 2

- **Suppose the Update Method is changed as follows**

```
// Driver class
class Test
{
  public      static      void      update(AB ab)
  {
              ab      =      new      AB(10,20);
              ab.setA(56);
              ab.setB(45);
  } // End of Method

  public static          void main(String args[])
  {
    AB      ab      =      new      AB(5,6);
    System.out.println("Before Calling Update Method");
    ab.display();
    update(ab);

  }// End of Method
}// End of class Test
```



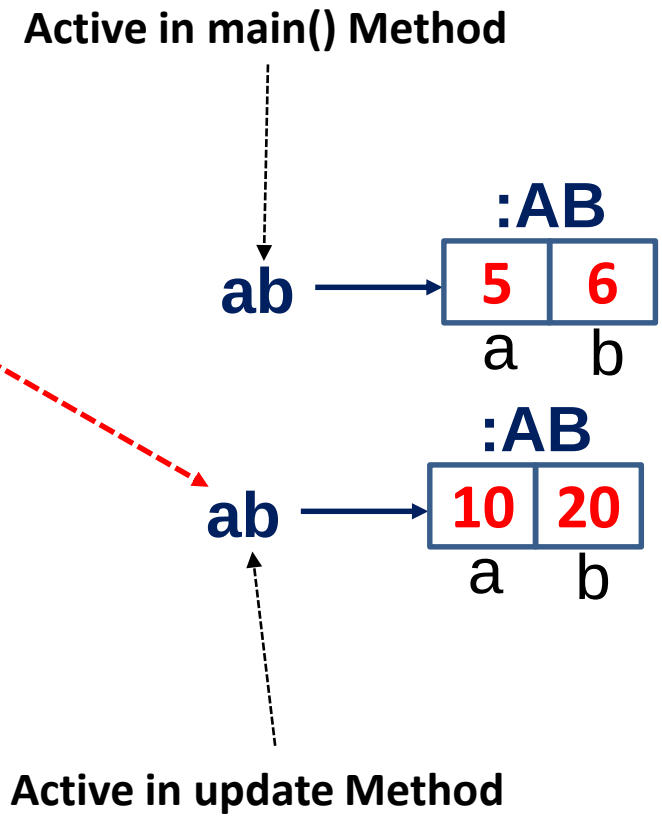
Objects as Parameters : Example 2 ….

- **Suppose the Update Method is changed as follows**

```
// Driver class
class Test
{
  public      static      void      update(AB ab)
  {
          ab      =      new      AB(10,20);
          ab.setA(56);
          ab.setB(45);
  } // End of Method

  public static          void main(String args[])
  {
    AB      ab      =          new          AB(5,6);
    System.out.println("Before Calling Update Method");
    ab.display();
    update(ab);

  }// End of Method
}// End of class Test
```



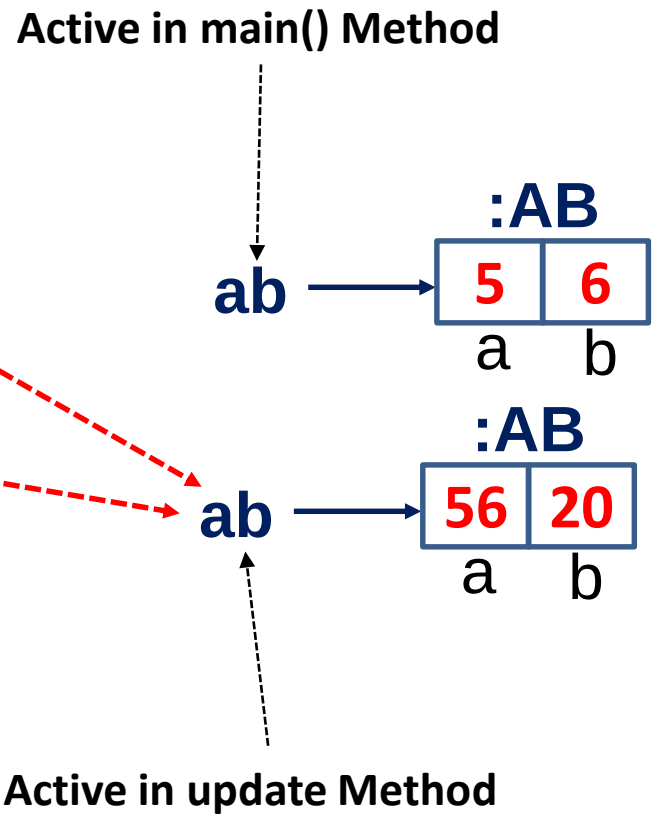
Objects as Parameters : Example 2 ….

- **Suppose the Update Method is changed as follows**

```
// Driver class
class Test
{
  public      static      void      update(AB ab)
  {
          ab      =      new      AB(10,20);
          ab.setA(56);
          ab.setB(45);
  } // End of Method

  public static          void main(String args[])
  {
    AB      ab      =          new          AB(5,6);
    System.out.println("Before Calling Update Method");
    ab.display();
    update(ab);

  }// End of Method
}// End of class Test
```



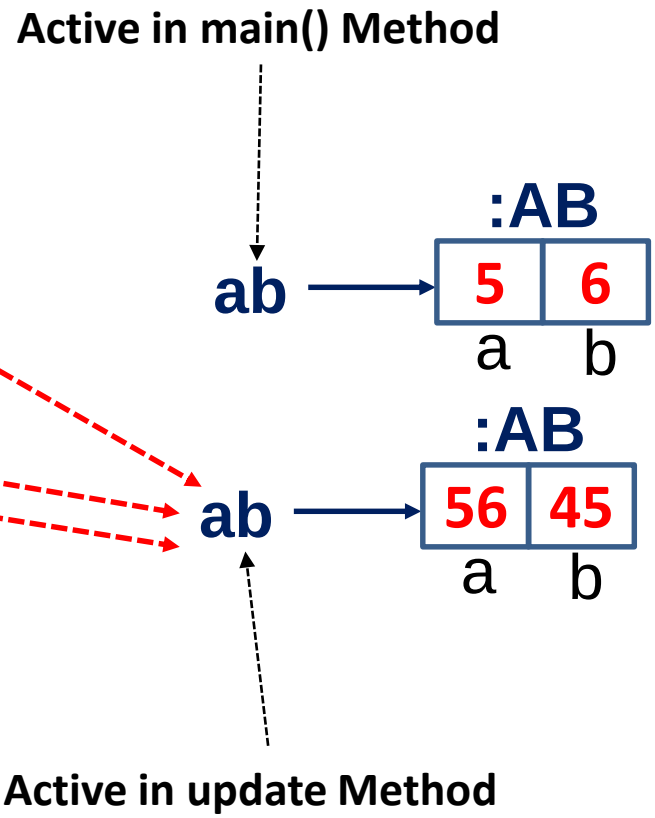
Objects as Parameters : Example 2 ….

- **Suppose the Update Method is changed as follows**

```
// Driver class
class Test
{
  public      static      void      update(AB ab)
  {
          ab      =      new      AB(10,20);
          ab.setA(56);
          ab.setB(45);
  } // End of Method

  public static          void main(String args[])
  {
    AB      ab      =          new          AB(5,6);
    System.out.println("Before Calling Update Method");
    ab.display();
    update(ab);

  }// End of Method
}// End of class Test
```



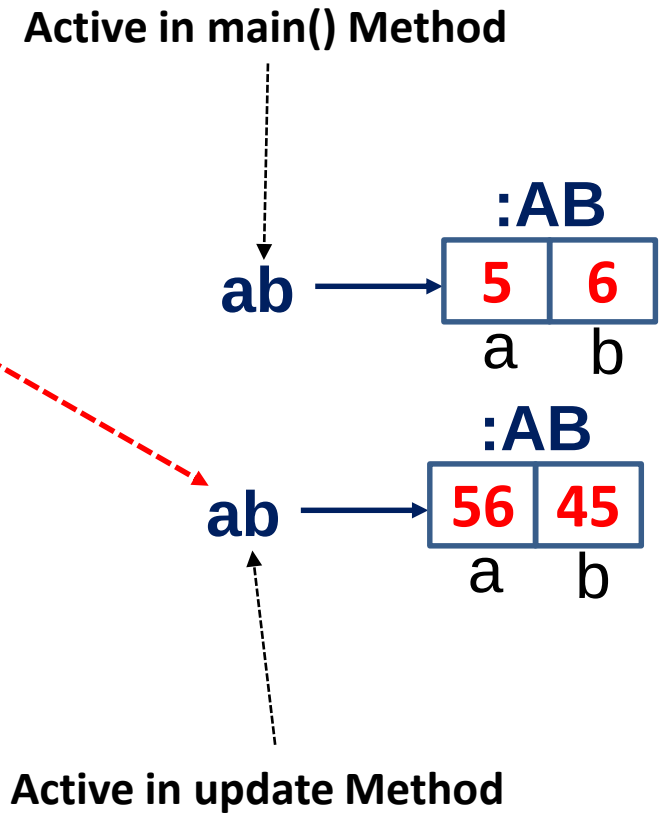
Objects as Parameters : Example 2 ….

- **Suppose the Update Method is changed as follows**

```
// Driver class
class Test
{
  public      static      void      update(AB ab)
  {
          ab      =      new      AB(10,20);
          ab.setA(56);
          ab.setB(45);
  } // End of Method

  public static          void main(String args[])
  {
    AB      ab      =          new          AB(5,6);
    System.out.println("Before Calling Update Method");
    ab.display();
    update(ab);
    System.out.println("After Calling Update Method");
    ab.display();

  }// End of Method
}// End of class Test
```



In this case updates of update() Method are not reflected in main() Method

Preventing the Called Method from Changing the Object Reference

- Declare the method argument as final

final Object Reference variable can not point to any other object-reference of same type

```
public static void update(final AB ab)
{
        ab = new AB(10,20);
} // End of Method
```

Compile-Time Error

Thank You