Java Parameter Variable is Pass by Value – Built-in Type Variables

```java
public class Tester {
    public static void main(String[] args) {
        int x=10, y=20;
        //x" value is copied to the "x" parameter and "y" value is
        // copied to "y" parameter
        int s1=swap(x, y);
        System.out.println("x: "+x+" y: "+y+" x+y: "+s1);
    }
}
```

Java Parameter Variable is Pass by Value – Primitive Array Variables

```java
public class Tester {
    public static int[] swapArray(int[] a) {
        //Since "a" is a reference value, the changes made in "a" are visible outside the function.
        int temp; temp=a[0];
        a[0]=a[1]; a[1]=temp;
        //return local array is ok as a new copy of the value is returned
        return a;
    }
}
```

Java Objects

```java
public class Tester {
    public static void main(String[] args) {
        int[] numbers=new int[2];
        numbers[0]=10;
        numbers[1]=20;
        System.out.println("array[0]: "+numbers[0]+" array[1]: "+numbers[1]);
    }
}
```

```java
public class Tester {
    public static void main(String[] args) {
        int[] numbers=new int[2];
        int[] numbers2=numbers;
        numbers[0]=10;
        numbers[1]=20;
        System.out.println("array[0]: "+numbers2[0]+" array[1]: "+numbers2[1]);
    }
}
```

Java compiler manipulates object references, instead of object itself for computation.

•This is more efficient.
•Assignment operator (=) - copy the reference value;
•Method passed the object argument by a copy of the object reference;
•When an object is returned from a method, a copy of the object reference is returned.
Java Parameter Variable is Pass by Value – ArrayList Object Variables

```java
public class Tester {
    public static ArrayList<int[]> swapArrayList(ArrayList<int[]>)
    {
        //Since "a" is a reference value, the changes made in "a" are visible
        //outside the function.
        int temp; temp=a.get(0); a.set(0,a.get(1)); a.set(1,temp);
        //return local variable is ok as a copy of the ArrayList reference is returned
        return localV;
    }
}
```

Java Parameter Variable is Pass by Value – Class Object Variables

```java
public class Tester {
    public static void main(String[] args)
    {
        String pairFirst="a"; pairSecond="b";
        returnPair=swapObject(pair); 
        System.out.print("Pair.first:");
        System.out.print(pairFirst);
        System.out.print("Pair.second:");
        System.out.print(pairSecond);
        System.out.print("returnPair.first:");
        System.out.print(returnPair.getFirst());
        System.out.print("returnPair.second:");
        System.out.print(returnPair.getSecond());
        System.out.println("\nreturn local object is ok as a copy of
//the object reference is returned
return localV; 
    }
}
```

C++ Parameter Variable is also Pass by Value – Built-in Type Variables

```cpp
int swap (int a, int b)
{
    //Since "a" and "b" are values, the changes made on "a" and "b" are not visible outside the function.
    int temp; temp=a; a=b; b=temp; 
    //return local variable is ok as a new //copy of the value is returned return temp; 
}
int main()
{
    int x=5; int y=10;
    //"x" value is copied to the "a"
    // parameter and "y" value is // copied to "b" parameter
    int s1=swap(x,y);
    cout << x << y << s1;
    return 0;
}
```

C++ Parameter Variable is Pass by Value – Object Variables

```cpp
Point swapObject(Point a) 
{
    //is an object value, the changes made are NOT visible
    //outside the function.
    int temp; temp=a.my_x;
    a.set_x(a.my_y); a.set_y(temp);
    Point localV(100,101);
    //return local object is ok as a copy of
    //the object reference is returned return localV;
}
int main()
{
    Point p(10,20);
    std::cout << p.my_x << ",", " << p.my_y << std::endl;
    //object "p" value is copied to the "a" parameter variable
    Point returnPoint=swapObject(p);
    std::cout << returnPoint.my_x << ",", " << returnPoint.my_y << std::endl;
    return 0;
}
```

The same applies to template class objects, such as vector<T> objects
C++ Objects

```cpp
int main()
{
  Point p1(10,10);
  Point p2 = p1;
  std::cout << p2.my_x() << " , " << p2.my_y();
  p2.set_x(100);
  p2.set_y(100);
  std::cout << p2.my_x() << " , " << p2.my_y();
  std::cout << p1.my_x() << " , " << p1.my_y();
}
```

C++ Pass by reference using &

```cpp
Point swapObject(Point& a)
{
  // is an object pointer, the
  // changes made are visible
  // outside the function.
  int temp;
  temp = a->my_x();
  a->set_x(temp);
  Point localV(100,101);
  return localV;
}
```

C Primitive array

- Primitive array is implemented as pointer (reference).
- Passing array as an argument to a function is passing a **reference value**. Thus, changes made on the array parameter inside a function are **visible** outside the function.
- Returning a local array variable (which is a reference) gives unreliable results (similar to that show on the previous slide).

STL Container Iterator

- Every STL container class has an **iterator type member**.
- For vector container, we can use iterator, instead of index, to access its values **sequentially**.

```cpp
for(vector<Book>::size_type i=0; i < books.size(); ++i)
  cout << books.at(i).name << endl;
for(vector<Book>::iterator iter = books.begin(); iter != books.end(); ++iter)
  cout << (*iter).name << endl;
```
STL Container Iterator related member functions

- `books.begin()`: points to the first element of the `books` vector object.
- `books.end()`: points to the one past the last element of the book vector object.
- To test for the end of a sequence, only operator `!` is needed (and not operator `>`, which is not defined for all container classes).
- `(*iter).name`: dereference `iter` point and get the name field of the object.
- `iter->name`: the same as the above.