Lab Manual

CL2001 – Data Structures

Fall-2023

National University of Computer and Emerging Sciences-FAST
Karachi Campus

Data Structures Lab#2
Course: Data Structures (CL2001) **Semester:** Fall 2023
Instructor: Shafique Rehman **T.A:** N/A

Note:

- Maintain discipline during the lab.
- Listen and follow the instructions as they are given.
- Just raise hand if you have any problem.
- Completing all tasks of each lab is compulsory.
- Get your lab checked at the end of the session**.**

Content:

- Wrapping and Unwrapping
- Shallow Copy, deep copy and Object Cloning
- Bubble Sort.
- Exercise Questions on:
    - 2D array
    - Jagged Array
    - Bubble Sort.

Wrapping and Unwrapping

The wrapper class in Java provides the mechanism to convert primitive into object and object into primitive.

Since J2SE 5.0, autoboxing and unboxing feature convert primitives into objects and objects into primitives automatically. The automatic conversion of primitive into an object is known as autoboxing and vice-versa unboxing.

Use of Wrapper classes in Java

Java is an object-oriented programming language, so we need to deal with objects many times like in Collections, Serialization, Synchronization, etc. Let us see the different scenarios, where we need to use the wrapper classes.

Change the value in Method: Java supports only call by value. So, if we pass a primitive value, it will not change the original value. But, if we convert the primitive value in an object, it will change the original value.

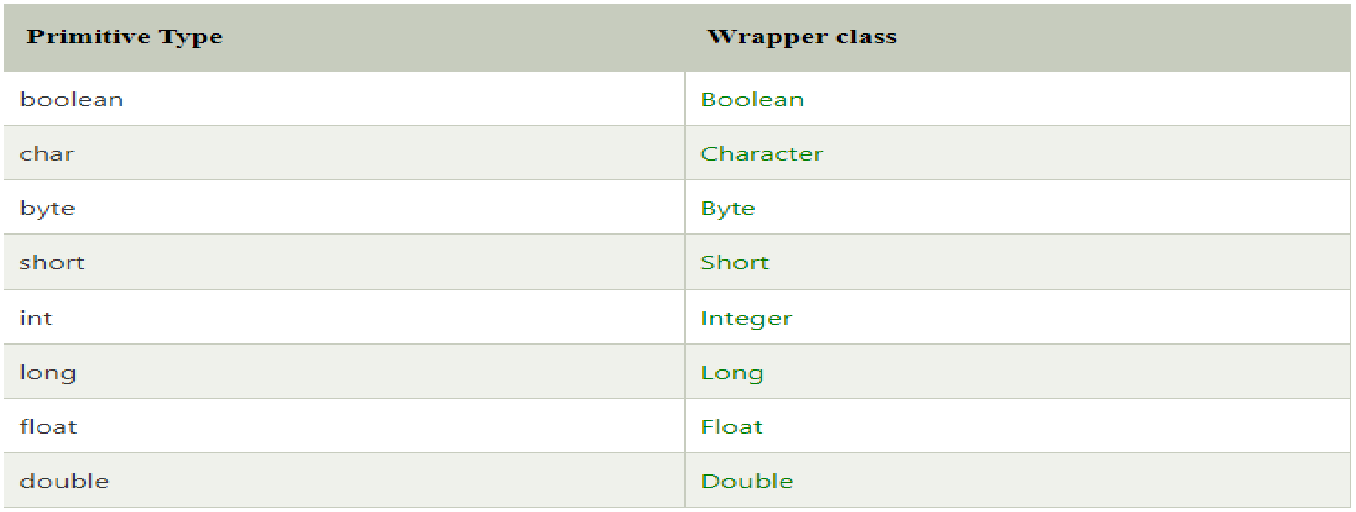
Serialization: We need to convert the objects into streams to perform the serialization. If we have a primitive value, we can convert it in objects through the wrapper classes.

Synchronization: Java synchronization works with objects in Multithreading.

java.util package: The java.util package provides the utility classes to deal with objects.

Collection Framework: Java collection framework works with objects only. All classes of the collection framework (ArrayList, LinkedList, Vector, HashSet, LinkedHashSet, TreeSet, PriorityQueue, ArrayDeque, etc.) deal with objects only.

The eight classes of the java.lang package are known as wrapper classes in Java. The list of eight wrapper classes are given below:



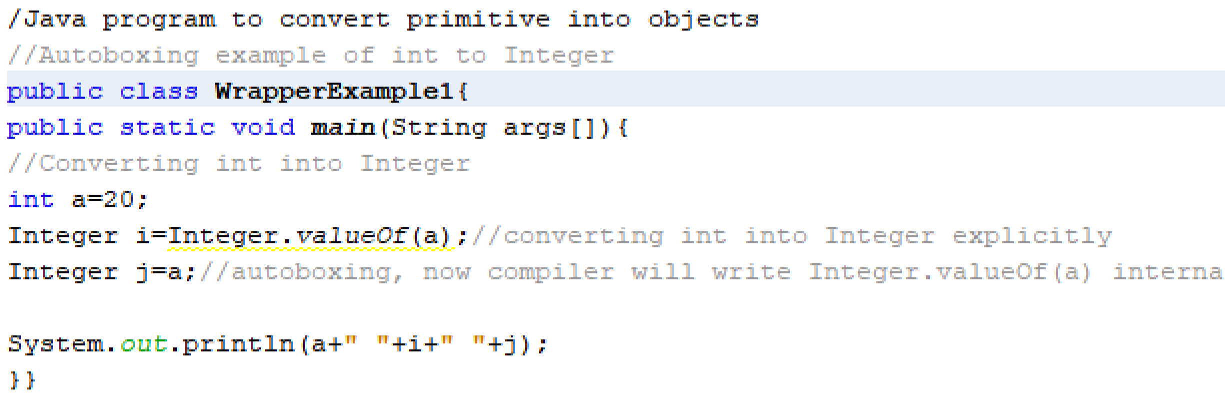
Primitive Type	Wrapper class
boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double

Autoboxing

The automatic conversion of primitive data type into its corresponding wrapper class is known as autoboxing, for example, byte to Byte, char to Character, int to Integer, long to Long, float to Float, boolean to Boolean, double to Double, and short to Short.

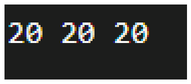
Since Java 5, we do not need to use the valueOf() method of wrapper classes to convert the primitive into objects.

Wrapper class Examples: **Primitive to Wrapper**
Example #1:



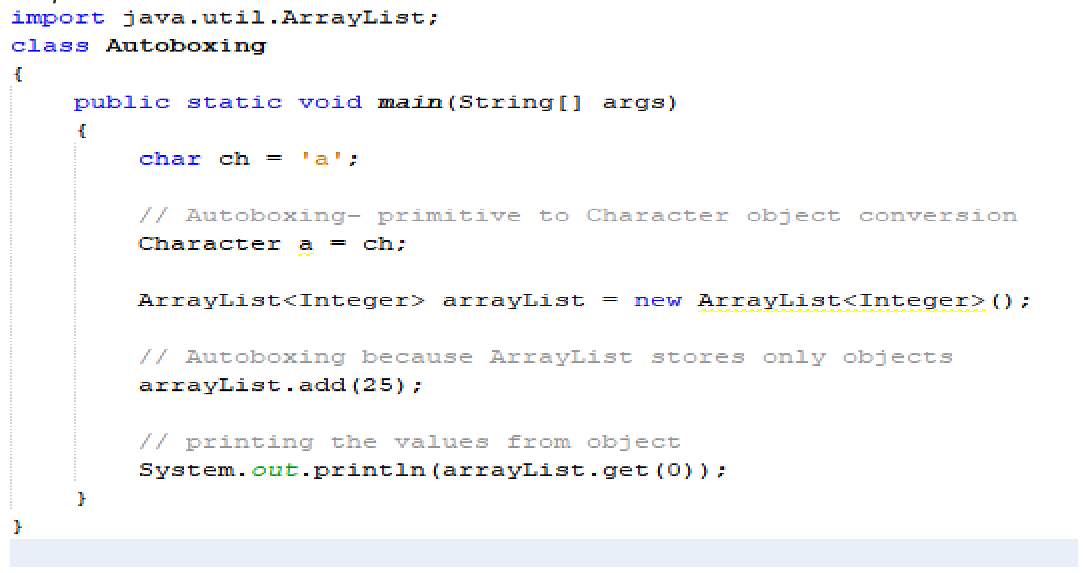
```
/Java program to convert primitive into objects
//Autoboxing example of int to Integer
public class WrapperExample1{
public static void main(String args[]){
//Converting int into Integer
int a=20;
Integer i=Integer.valueOf(a);//converting int into Integer explicitly
Integer j=a;//autoboxing, now compiler will write Integer.valueOf(a) interna

System.out.println(a+" "+i+" "+j);
}}
```

Output:



```
20 20 20
```

Example#1.1:

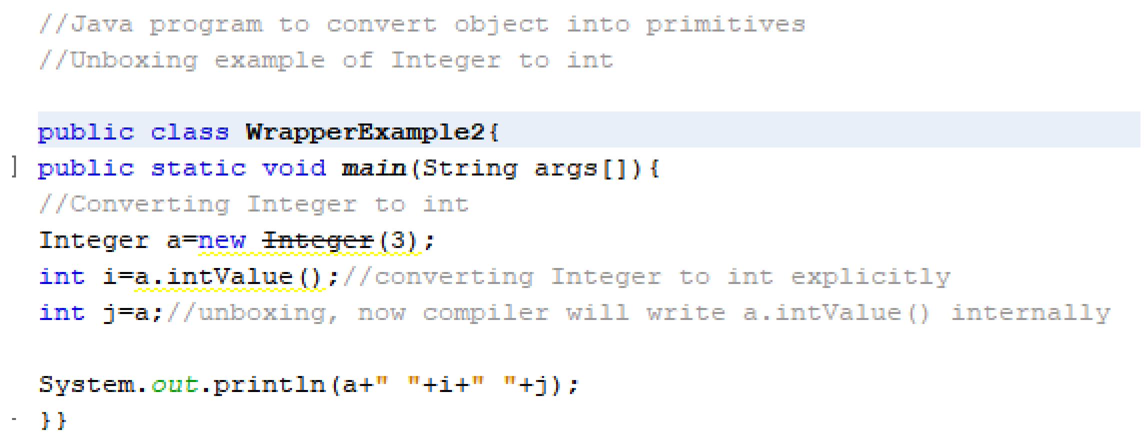


```
import java.util.ArrayList;
class Autoboxing
{
    public static void main(String[] args)
    {
        char ch = 'a';

        // Autoboxing- primitive to Character object conversion
        Character a = ch;

        ArrayList<Integer> arrayList = new ArrayList<Integer>();

        // Autoboxing because ArrayList stores only objects
        arrayList.add(25);

        // printing the values from object
        System.out.println(arrayList.get(0));
    }
}
```

Unboxing

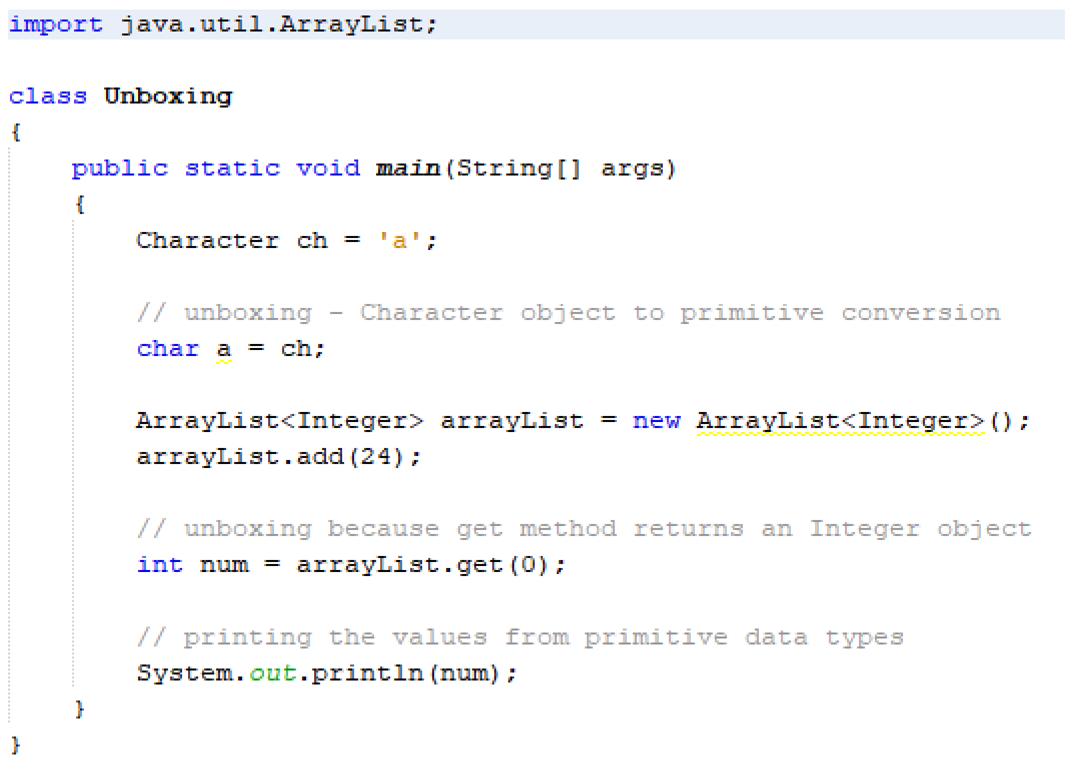
The automatic conversion of wrapper type into its corresponding primitive type is known as unboxing. It is the reverse process of autoboxing. Since Java 5, we do not need to use the intValue() method of wrapper classes to convert the wrapper type into primitives.

Wrapper class Example: Wrapper to Primitive
Example#2



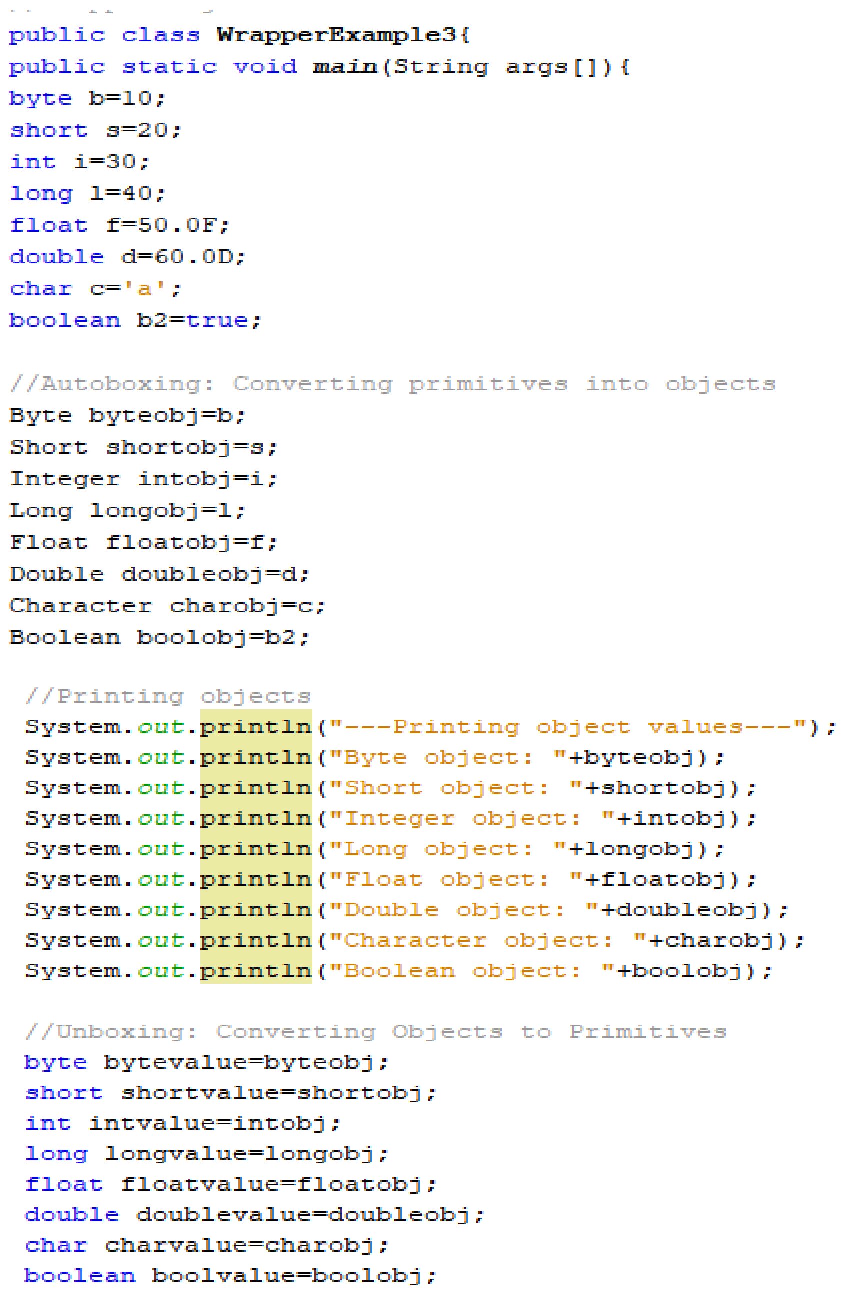
```
//Java program to convert object into primitives
//Unboxing example of Integer to int

public class WrapperExample2{
public static void main(String args[]){
//Converting Integer to int
Integer a=new ~~Integer~~(3);
int i=a.intValue();//converting Integer to int explicitly
int j=a;//unboxing, now compiler will write a.intValue() internally

System.out.println(a+" "+i+" "+j);
}}
```

Example#2.1:

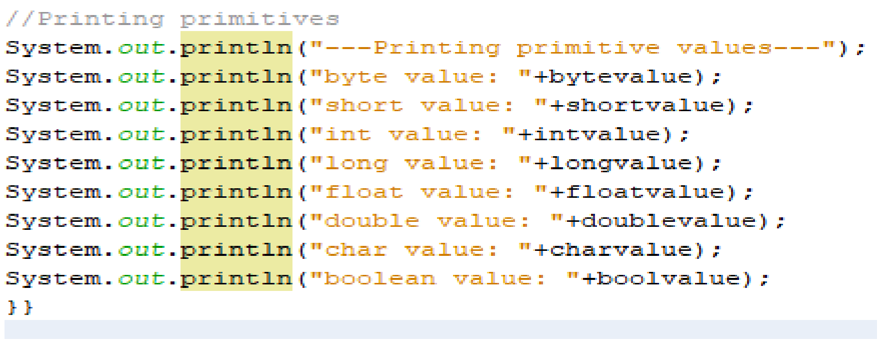


```
import java.util.ArrayList;

class Unboxing
{
    public static void main(String[] args)
    {
        Character ch = 'a';

        // unboxing - Character object to primitive conversion
        char a = ch;

        ArrayList<Integer> arrayList = new ArrayList<Integer>();
        arrayList.add(24);

        // unboxing because get method returns an Integer object
        int num = arrayList.get(0);

        // printing the values from primitive data types
        System.out.println(num);
    }
}
```

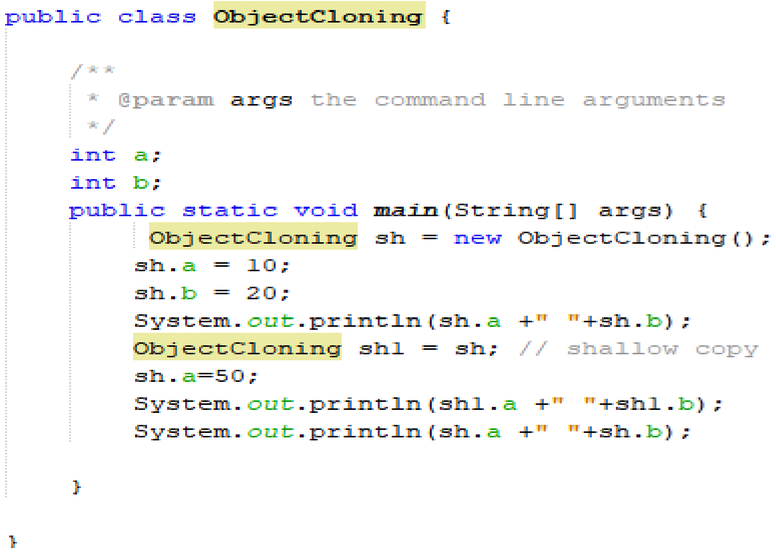
Wrapping and Unwrapping Combine Example



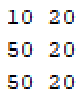
```java
public class WrapperExample3{
public static void main(String args[]){
byte b=10;
short s=20;
int i=30;
long l=40;
float f=50.0F;
double d=60.0D;
char c='a';
boolean b2=true;

//Autoboxing: Converting primitives into objects
Byte byteobj=b;
Short shortobj=s;
Integer intobj=i;
Long longobj=l;
Float floatobj=f;
Double doubleobj=d;
Character charobj=c;
Boolean boolobj=b2;

 //Printing objects
 System.out.println("---Printing object values---");
 System.out.println("Byte object: "+byteobj);
 System.out.println("Short object: "+shortobj);
 System.out.println("Integer object: "+intobj);
 System.out.println("Long object: "+longobj);
 System.out.println("Float object: "+floatobj);
 System.out.println("Double object: "+doubleobj);
 System.out.println("Character object: "+charobj);
 System.out.println("Boolean object: "+boolobj);

 //Unboxing: Converting Objects to Primitives
 byte bytevalue=byteobj;
 short shortvalue=shortobj;
 int intvalue=intobj;
 long longvalue=longobj;
 float floatvalue=floatobj;
 double doublevalue=doubleobj;
 char charvalue=charobj;
 boolean boolvalue=boolobj;
```


```
//Printing primitives
System.out.println("---Printing primitive values---");
System.out.println("byte value: "+bytevalue);
System.out.println("short value: "+shortvalue);
System.out.println("int value: "+intvalue);
System.out.println("long value: "+longvalue);
System.out.println("float value: "+floatvalue);
System.out.println("double value: "+doublevalue);
System.out.println("char value: "+charvalue);
System.out.println("boolean value: "+boolvalue);
}}
```

Shallow Copy:
When we do a copy of some entity to create two or more than two entities such that changes in one entity are reflected in the other entities as well, then we can say we have done a shallow copy. In shallow copy, new memory allocation never happens for the other entities, and the only reference is copied to the other entities. The following example demonstrates the same.

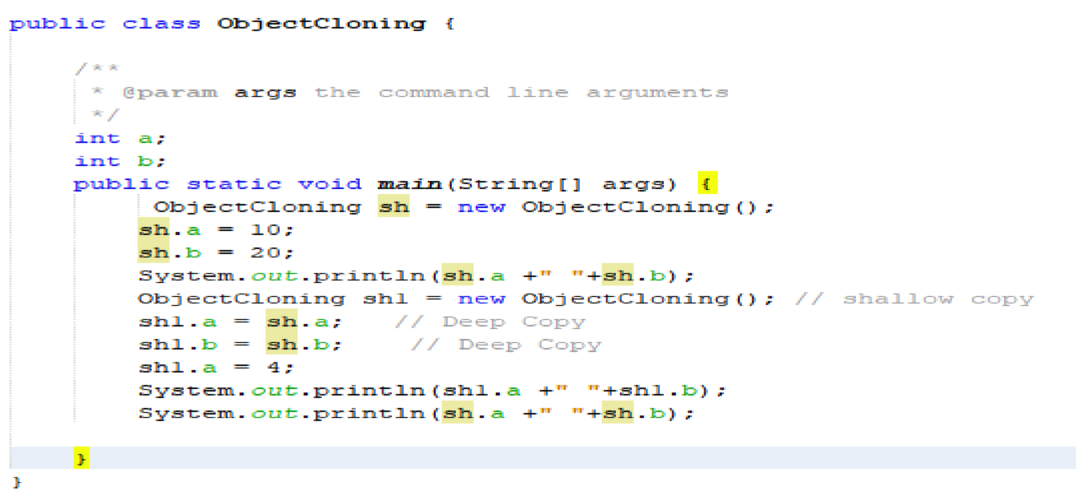

```
public class ObjectCloning {

    /**
     * @param args the command line arguments
     */
    int a;
    int b;
    public static void main(String[] args) {
         ObjectCloning sh = new ObjectCloning();
        sh.a = 10;
        sh.b = 20;
        System.out.println(sh.a +" "+sh.b);
        ObjectCloning sh1 = sh; // shallow copy
        sh.a=50;
        System.out.println(sh1.a +" "+sh1.b);
        System.out.println(sh.a +" "+sh.b);

    }

}
```

Output:


```
10 20
50 20
50 20
```

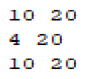
Deep Copy

When we do a copy of some entity to create two or more than two entities such that changes in one entity are not reflected in the other entities, then we can say we have done a deep copy. In the deep copy, a new memory allocation happens for the other entities, and reference is not copied to the other entities. Each entity has its own independent reference. The following example demonstrates the same.



```
public class ObjectCloning {

    /**
     * @param args the command line arguments
     */
    int a;
    int b;
    public static void main(String[] args) {
         ObjectCloning sh = new ObjectCloning();
        sh.a = 10;
        sh.b = 20;
        System.out.println(sh.a +" "+sh.b);
        ObjectCloning sh1 = new ObjectCloning(); // shallow copy
        sh1.a = sh.a;   // Deep Copy
        sh1.b = sh.b;    // Deep Copy
        sh1.a = 4;
        System.out.println(sh1.a +" "+sh1.b);
        System.out.println(sh.a +" "+sh.b);

    }
}
```

Output:



```
10 20
4 20
10 20
```

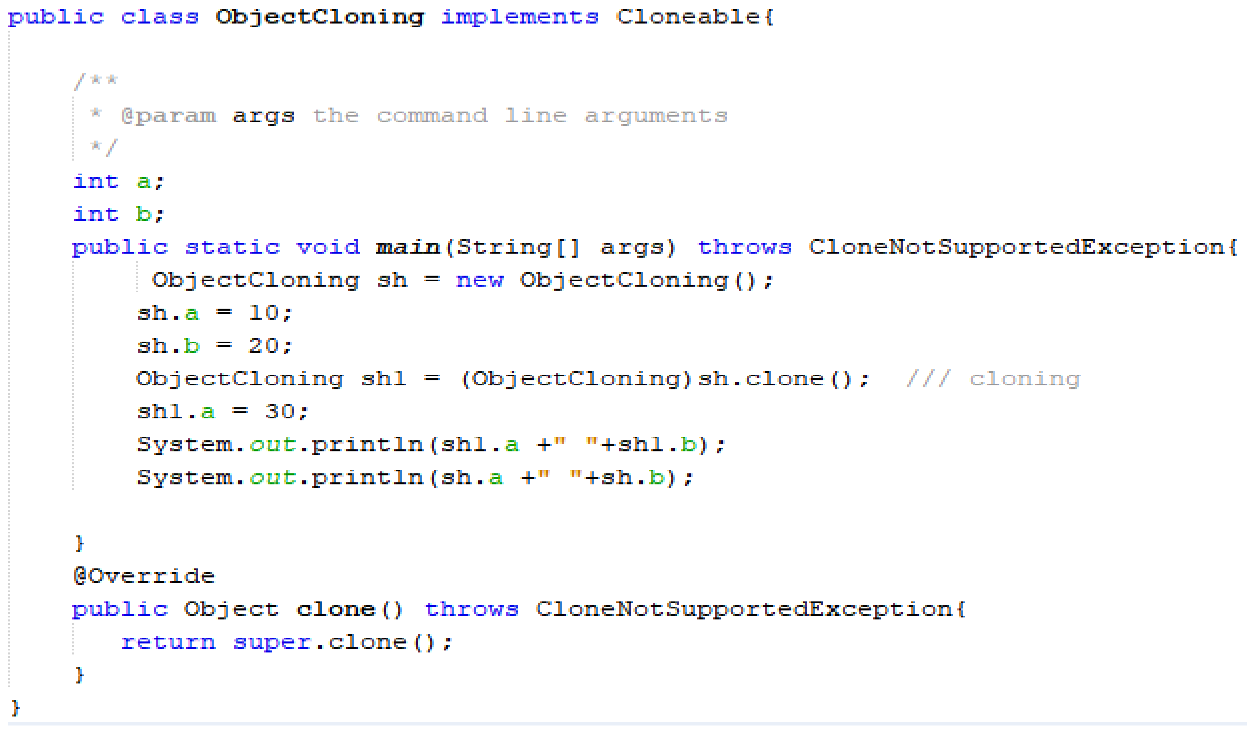
Creating a copy using the clone() method

The class whose object's copy is to be made must have a public clone method in it or in one of its parent class.

Every class that implements clone() should call super.clone() to obtain the cloned object reference. The class must also implement java.lang.Cloneable interface whose object clone we want to create otherwise it will throw CloneNotSupportedException when clone method is called on that class's object.
Syntax:

protected Object clone() throws CloneNotSupportedException

Example:



```
public class ObjectCloning implements Cloneable{

    /**
     * @param args the command line arguments
     */
    int a;
    int b;
    public static void main(String[] args) throws CloneNotSupportedException{
         ObjectCloning sh = new ObjectCloning();
        sh.a = 10;
        sh.b = 20;
        ObjectCloning sh1 = (ObjectCloning)sh.clone();  /// cloning
        sh1.a = 30;
        System.out.println(sh1.a +" "+sh1.b);
        System.out.println(sh.a +" "+sh.b);

    }
    @Override
    public Object clone() throws CloneNotSupportedException{
       return super.clone();
    }
}
```

Output:

Advantages of clone method:

If we use the assignment operator to assign an object reference to another reference variable then it will point to the same address location of the old object and no new copy of the object will be created. Due to this any changes in the reference variable will be reflected in the original object.
If we use a copy constructor, then we have to copy all the data over explicitly i.e. we have to reassign all the fields of the class in the constructor explicitly. But in the clone method, this work of creating a new copy is done by the method itself. So to avoid extra processing we use object cloning.

Java Scanner Class

Java Scanner classallows the user to take input from the console. It belongs to java.util package. It is used to read the input of primitive types like int, double, long, short, float, and byte. It is the easiest way to read input in Java program.

Syntax

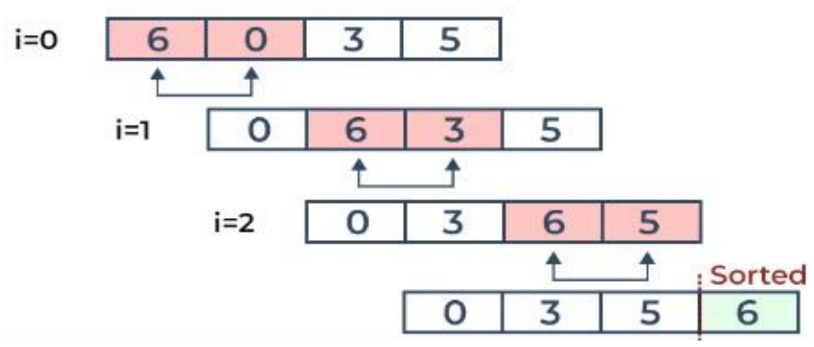
Scanner sc=new Scanner(System.in);

The above statement creates a constructor of the Scanner class having System. inM as an argument. It means it is going to read from the standard input stream of the program. The java.util packageshould beimport while using Scanner class.

Bubble Sort:
Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in the wrong order. This algorithm is not suitable for large data sets as its average and worst-case time complexity is quite high.

Bubble Sort Algorithm:

1. Traverse from left and compare adjacent elements and the higher one is placed at right side.
2. In this way, the largest element is moved to the rightmost end at first.
3. This process is then continued to find the second largest and place it and so on until the data is sorted.


i=0
6 0 3 5
i=1
0 6 3 5
i=2
0 3 6 5
Sorted
0 3 5 6