1. Understanding Object in Java

Java is a successor to a number of languages including lisp, Simula67, CLU and SmallTalk.
Java is similar to C and C++ because it syntax is borrowed from them , However a deeper level it very different.

- Objects in Java contain :
  - STATE and
  - OPERATIONS or methods.
  - Programs interact with objects by invoking methods.
  - Methods provide access to manipulate objects.

- Java programs consists of classes
  - Classes :
    - To define collections of procedures [methods]
    - To define new data types

Example: To define collections of procedures

- Sort an array
- Search

Example of Search Class

```
// search upwards
found = false;
for (int i = 0; i < a.length; i++)
    if (a[i] == e) {
        z = i;
        found = true;
    }

// search downwards
found = false;
for (int i = a.length-1; i >= 0; i--)
    if (a[i] == e) {
        z = i;
        found = true;
    }
```

2. Packages

- Packages: a group of classes and interfaces
- Purposes:
  1. Encapsulation mechanism
     - Provide a way to share info within the package while preventing its use on the outside.
     - How? By using visibility→ public, private, etc?? where public can be excess by same package and from another package.
  2. Naming purpose
     - No name conflicts between classes and interfaces defined in different packages
- It is possible to have same names in other packages

3. Object and Variables

All data are accessed by means of *variables or attributes*.
Local variables such as those declared within methods, reside on the runtime stack; space is allocated for them when the method is called and deallocated when a method returns.

- Every variable has to be declared and indicates its type.
- Primitive types→ int, boolean, char. Eg?
- Other object types→ defined by others. Eg: String and array contains *references* to objects
- Primitive types
  - Discuss example… int i = 6;

- Other object types(including arrays, String)
  - Contain references to objects
  - Created by the use of *new* operator
  - E.g.: int [ ] a = new int [3]; array type contain three integer elements

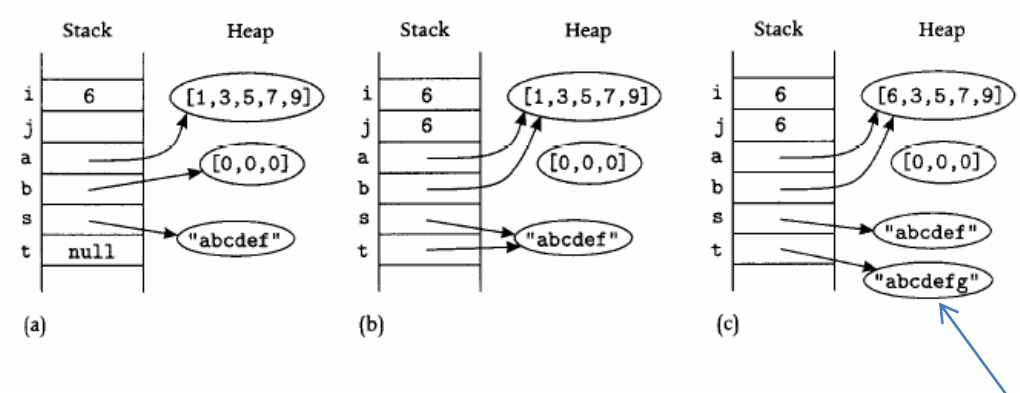
Space for a new array of int object will be allocated on the *heap* and a reference to the object will be stored in *a [i.e. in a stack memory]*.

Local Variables :

- Must be initialized when they are declared. Such as : int =6;

- Declaration: indicating type of a variable.

Example:-




```
int i=6;
int j; // uninitialized
int [] a={1,3,5,7,9};
int [] b = new int[3];
String s="abcdef";
String t=null;
```

This case discuss later

a. For figure (b)

```
j=i;
b=a;
t=s;
```

- Compare two int: j= =i?
- Is s= = null?
- Whether two variables refer to the same object: a= = b?

Note that in the case of the ***string and arrays*** variables,

We now have two variables pointing to the same object; thus, *assignment involving references causes variables to share objects*.

The = = operator can be used to determine whether two variables contain the same value. this operator is used primarily for primitive types-
for example, to compare two ints, such as int j==i;, or to determine whether a variable that might refer to an object instead contains null;
such as t== null; it can also be used to determine whether two variables refer to the same object ; in the situation in figure (a), for example
a==b will be not true, whereas in the situation in figure (b) a==b is true.

Important Note

Objects in the heap continue to exit as long as they are reachable from some variable on the stack either directly or via a path through other objects. When an object is longer reachable, its storage becomes available for *reclamation* (تسترجع) by a garbage collector. For example the array (b) is no longer reachable and is therefore available for reclamation by the **garbage collector**.

Example

```
public class Equality {

	/**
	 * @param args
	 */
	public static void main(String[] args) {
		// TODO Auto-generated method stub
		int i=9;
		int j=9 ;
		////////// Check The contain of two values  ///////
		if (i==j){ System.out.println("True1"); }
				else
				{ System.out.println("False1");}

			String Str1="Hello";
			String Str2= "Hello";

//////// Check the Contains of two objects//////////
				if (Str1==Str2) {
System.out.println("True2"); }
				else
				{ System.out.println("False2"); }

//			//////Check the Contains of two objects//////////
					if (Str1.equals(Str2)){
System.out.println("True3"); }
					else
```

```
                                                {
System.out.println("False3"); }

            String Str3=new String ("Hello");
            String Str4= new String ("Hello");

            ///// Check Equality of the same Location ////////////
            if (Str3.equals(Str4)){ System.out.println("True4"); }
            else
                  { System.out.println("False4"); }

            ////////////////////New Object //////////
                  if (Str3==Str4){ System.out.println("True5"); }
                  else
                        { System.out.println("False5"); }
      }

}
```

1. Java Mutable Example

Normally, it provides a method to modify the field value, and the object can be extended.

MutableExample.java

```
package com.mkyong;

public class MutableExample {

        private String name;

        MutableClass(String name) {

                this.name = name;
        }

        public String getName() {
                return name;
        }

        // this setter can modify the name
        public void setName(String name) {
                this.name = name;
        }

        public static void main(String[] args) {

                MutableExample obj = new MutableExample("Hello");
                System.out.println(obj.getName());

                // update the name, this object is mutable
                obj.setName("Hello2");
```

```
                System.out.println(obj.getName());

        }
}
Copy
```

Output

```
mkyong
new mkyong
Copy
```

2. Java Immutable Example

To create an Immutable object, make the class final, and don't provide any methods to modify the fields.

ImmutableExample.java

```
package com.mkyong;

// make this class final, no one can extend this class
public final class ImmutableExample {

        private String name;

        ImmutableExample (String name) {
                this.name = name;
        }

        public String getName() {
                return name;
        }

        //no setter

        public static void main(String[] args) {

                ImmutableExample obj = new ImmutableExample("mkyong");
                System.out.println(obj.getName());

                // there is no way to update the name after the object is
created.
                // obj.setName("new mkyong");
                // System.out.println(obj.getName());

        }
}
```


جامعة بابل
UNIVERSITY OF BABYLON