Object Creation And Modifiers

Object Creation:

To create an object, we require a class.

A obj=new A();

obj.show();

Here **obj** is not an object it's a reference.

new keyword is used to create an object.

And the we need to specify how much memory we need, and that's
why we us **constructors.**

Now this object in Java is called as **instance.**

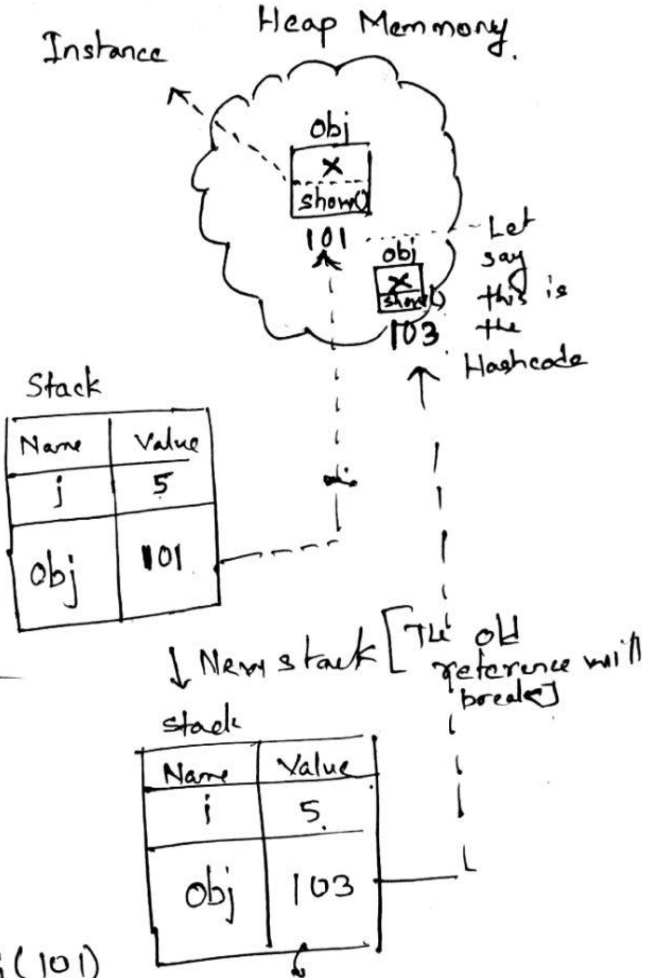
Obj gets instantiated inside the **Heap Memory.**

All the **Variables**[in this case reference variable **obj**] belongs to **stack memory**

```
Class A{
 int x;
public void show()
 {
 System.out.println(" Hello");
 }
}.
```

Reference   Constructor.

```
A obj = new A();
  Obj.show()
 int i=5;
```


Heap Memmory.
Instance
obj
x
show()
101
obj
x
show()
103
Let say this is the Hashcode
Stack
New stack [The old reference will break]
stack

Name	Value
j	5
obj	101

Name	Value
i	5.
obj	103

Now, let say. obj = new A();

so once again a new objet is created.

null.

So, the old reference obj(101) will break and it will be ready for garbage Collection.

Another, way to break the object obj=null. [Then also the link breaks]

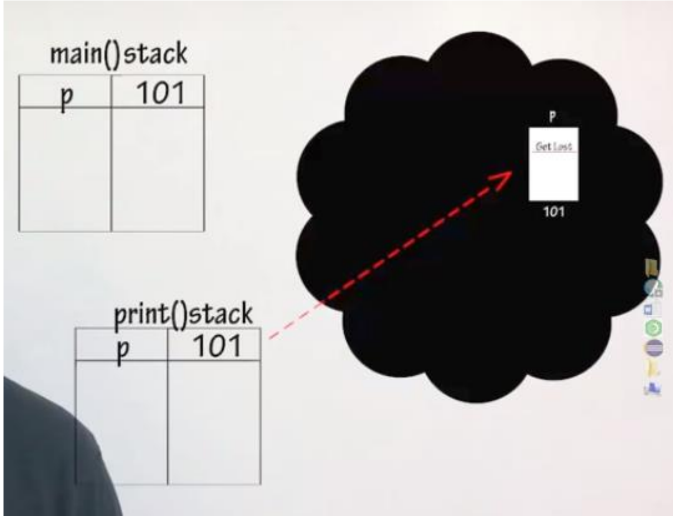
Object Passing in Java

There is a concept **Call by Value** and **Call by Reference.**
In java everything is Call by Value, even if you pass an object, it will go in a format of Value.
So whenever you pass any object as parameter, it sends the hash code of it.

```
class Paper{
      String text;
      public String getText() {
            return text;
      }
      public void setText(String text) {
            this.text = text;
      }
   }
class Printer{
      public void print(Paper p) {
            p.setText("Get Lost");
      }
   }

public class ObjectPassingInJava {
      public static void main(String[] args) {
            Paper p=new Paper();
            p.setText("Hello World");
            System.out.println(p.getText());

            Printer pr=new Printer();
            pr.print(p);
            System.out.println(p.getText());
      }
   }
```



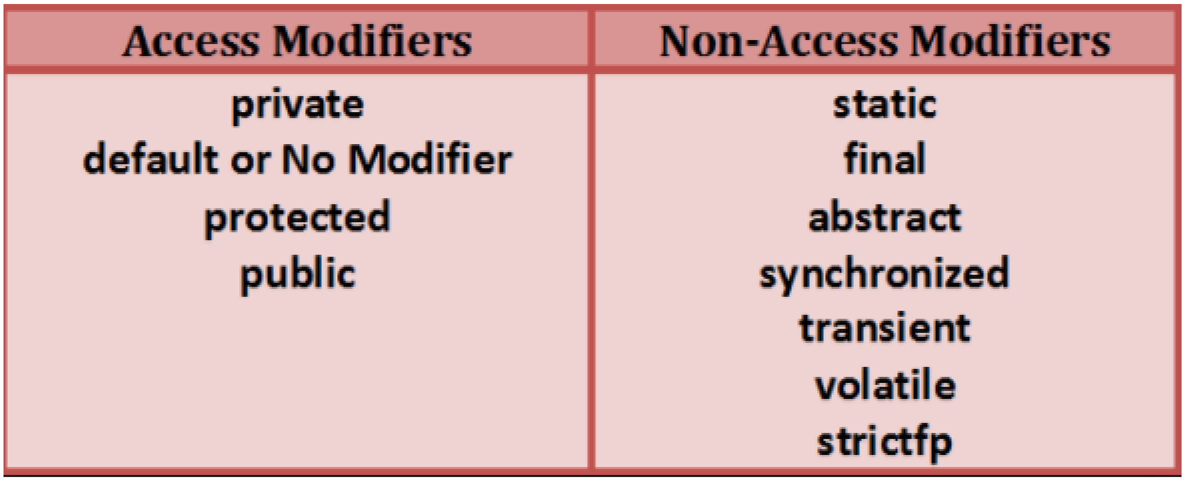
Output:

```
Hello World
Get Lost
```

Explanation:
Initially an object[P] will be created on the Heap.
And in the main() Stack it will contain reference value p and the Hashcode[101] of the P object.
So, initially when we set the p.getText() will return **Hello World.**

We will be one more Stack for the Method
Now once we pass the object p to the print method, we are actually creating a new reference to the P object. And the HashCode will be same as the object[101].
So when we print it will give, `Get Lost`

That means whenever we pass an object, we are achieving something like call by reference. But it is not actually Call By Reference but it is Call By Value, since we are passing not the object but the HashCode



Access Modifiers	Non-Access Modifiers
private default or No Modifier protected public	static final abstract synchronized transient volatile strictfp

Access Modifiers:

Non- Access Modifiers:

Classes Name can be same if they are in different Packages.

Access Modifiers in Java:

Total We have,

- Final
- Abstract
- Private
- Public
- Protected
- Default

Final Keyword:

We can use this for **variables** , **methods** and **class**

With Variable

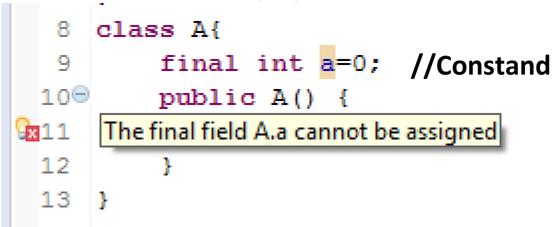
```
Class A{
	int a=0;
	public A() {
		a=10;
	}
}

public class FinalDemo {
	public static void main(String[] args) {
		A obj=new A();
		System.out.println("Value of a: "+obj.a);
	}
}
```

Output:

```
Value of a: 10
```

Here the value of a will of **i** is changing from 0 to 10, and there is no issues with that.
Now when you are changing the modifier of a to **final.**



```
class A{
    final int a=0;   //Constand
    public A() {
The final field A.a cannot be assigned
    }
}
```

It will immediately throw error that final field cannot be assigned.
So, once we make a variable final we can't change the value of it.
But,

```
class A{
      final int a;//we have not initialized the final variable
      public A() {
            a=10;//So we can initialize it anywhere
            a=5;//This will again throw compilation error as already initialized
      }
}
```

So, if the value of final variable not initialized, then we can initialize it anywhere.
But in the class, itself we must initialize the variable

With Class

A Final Class cannot be extended(i.e. it will become non-inheritable).
It is done so that if somebody else extend your class, and overrides the inherited methods and tell that it is there method.

With Method

A Final Method Cannot be override. It is mainly done to protect a user's from modifying the class

Abstract Keyword:

With class and method.

- If a Class is Abstract we cannot create a object of it.
- If a Method is abstract we do not have to write the body of the Method, we can just declare it.
- Whenever we have some abstract method in a class then the class must be abstract.
- If you extend an Abstract Class, it is compulsory to implement the abstract methods of it.

Example:

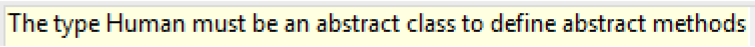
```
public class AbstractDemo {
      public static void main(String[] args) {
            Human he= new Human(); //It will throw Compilation error,
                                    //Cannot instantiate type Human
      }
}
abstract class Human{
      public void eat() {
      }
      public void walk() {
      }
}
```

Whenever a method does not have a body it is called as abstract method.

```
abstract class Human{
	abstract public void eat() ;
	abstract public void walk();
}
```

Now if we remove the abstract keyword from the class then,

```
 class Human{ //It will throw compilation error
	abstract public void eat() ;
	abstract public void walk();
}
```



The type Human must be an abstract class to define abstract methods

Then Whenever we have some abstract method in a class then the class must be abstract.

How to use an abstract class:

```
abstract class Human{ // Abstract Class

	abstract public void eat() ;

	abstract public void walk();

}
class Man extends Human{ //Concrete Class
	@Override
	public void eat() {
	}
	@Override
	public void walk() {
	}
}

public class AbstractDemo {
	public static void main(String[] args) {
		Human obj= new Man(); //Object has been created
	}
}
```

In Class Level,

Final, abstract, public.

If innercalss then we can use **private**.

- Default class can be used with the package only
- If we want to access a class outside a package make sure that the class is public

In Method Level,

Final , abstract , default , public , private, protected

- If you don’t specify anything, then by default it is **default** specifier accessible within the same package
- Private specifier methods and variables are accessible within the class.

Final	Variable: Final Variable value cannot be changed	Method: Final Method Cannot be overridden	Class: Final Class cannot be extended
Abstract		Method:	Class:
Default[If you do not specify any modifier]	Variable: within same package it is accessible/outside package not accessible(**Specific Package scope**)	Method: **Specific Package scope**	Class: **Specific Package scope**
public	Everywhere	Everywhere	Everywhere
private	Variable: will be accessible with Class[**Specific Class Scope**]	Method: **Specific Class Scope**	NA
protected	Variable: **Subsiding Class** Public within same package and outside package by inheritance	Method: **Subsiding Class**	NA

Constructor can be protected.