final variable

1. When you want to initialize a variable, and then later you don’t want to allow changes in the variable, then make the variable as final
2. A final variable can be static or non static
3. Final variable has to be initialled at the time of declaration, if it a member variable of any class, then it can be initialized inside constructor

```
public class MyClass {
        private int id;
        final String c;
        public MyClass() {
                super();
                c="Hello";
        }
        public MyClass(int id) {
                super();
                this.c="Welcome";
        }


}
```

```
public class MyClass {
        private int id;
        final String c=”test”;
        public MyClass() {
                super();

        }
        public MyClass(int id) {
                super();

        }


}
```

final function

1. If you write a function, and if the function should not be overridden by child class, then make the function as final

```
public final int myfunction(){

  //code goes here

}
```

final class

If you want the class should not be extended by any other class, then make the class final

```
final class MyClass{}
```

Modifiers

1. Private -→ all private members are accessible only within a class
2. Public-→ all public members are accessible within class, within package, and outside package also
3. Protected-→ all protected members are accessible within package, within a class, and all child classes.
4. Default--→ all default members are accessible with class, and within package.

class Parent{
    public void m1(){
     s.o.p(“in parent me method”);
   }
}
	class Parent{
	    protected void m1(){
	     s.o.p(“in parent me method”);
	   }
	}
		class Parent{
		    public void m1(){
		     s.o.p(“in parent me method”);
		   }
		}

class Child extends Parent{
    public void m1(){
     s.o.p(“in parent me method”);
   }
	class Child extends Parent{
	@Override
	    public void m1(){
	     s.o.p(“in parent me method”);
	   }
		class Child extends Parent{
		    protected void m1(){
		     s.o.p(“in parent me method”);
		   }
		
		Error

@Override, @suppressWarning, @FunctionalInterface

It gives extra information to compiler, that the function is overridden, hence the signature of the function should be same. If it is not same, then such errors can be detected at early stage, at compile time.

Interfaces

3 types of interfaces

Marker Interface	If the interface donot contain any method, it mean it is a empty interface, and called as marker interface Example: Serializable
Functional Interface	If the interface contains only one abstract method, then it is called as functional interface
Interface	It is a contract between the interface and class, that the class will override all abstract method

What is interface

1. It is a contract between the interface and class, that the class will override all abstract method
2. All the method in interface are by default public and abstract
3. In interface if you want to add method implementation, then the method has to be default (1.8 onward)
4. Interfaces can store static methods also (1.8 onward)
5. Interfaces can store private methods also(1.9 onward)
6. All the variable declared inside interface are by default public static and final
7. In interfaces we cannot write constructors
8. In interfaces we cannot override method of Object class.
9. One class can extend only one class, but can implement any number of interfaces
10. One interface can extend any number of interfaces

Abstract class vs Interfaces

Abstract classes	Interfaces
One class can extend only one class, but can implement any number of interfaces	interface can extend any number of interfaces
Extends	implements
Constructor can be written	Constructor is not available
The variables can be instance variables or static variables or final variables	Variables are always public static and final
We can override Object class methods	We cannot override Object class methods
It can have member methods	By default all methods are public and abstract If the implementation is added then the method should be default
It represents ISA relationship	It does not represent ISA relationship, It's a contract between class and interface

Object class methods

toString, equals, hashCode, getClass, wait(),wait(X),wait(x,y),notify, notifyAll

upcasting is implicit and downcasting is explicit

```
Student s=new MasterStudent();   ///upcasting

Person p=new MasterStudent();  //upcasting

Object o=new MasterStudent(); //upcasting

MasterStudent m=( MasterStudent)s;     ///downcasting
```