Interfaces

Multiple inheritance is not available in java.

(Same functions in 2 classes it will skip that hence no multiple inheritance)

Instead we have java interfaces. they have abstract functions (no body of functions)

Interface is like class but not completely. it is like an abstract class.

By default functions are public and abstract in interface.

variables are final and static by default in interface.

Interfaces specify only what the class is doing, not how it is doing it.

The problem with MULTIPLE INHERITANCE is that two classes may define different ways of doing the same thing, and the subclass can't choose which one to pick.

Key difference between a class and an interface: a class can maintain state information

(especially through the use of instance variables), but an interface cannot.

Using interface, you can specify a set of methods that can be implemented by one or more classes.

Although they are similar to abstract classes, interfaces have an additional capability:

A class can implement more than one interface. By contrast, a class can only

inherit a single superclass

(abstract or otherwise).

Using the keyword interface, you can fully abstract a class' interface from its implementation.

That is, using interface, you can specify what a class must do, but not how it does it.

Interfaces are syntactically similar to classes, but they lack instance variables, and, as a general rule, their methods are declared without any body.

By providing the interface keyword, Java allows you to fully utilize the "one interface, multiple methods" aspect of polymorphism.

NOTE: Interfaces are designed to support dynamic method resolution at run time.

Normally, in order for a method to be called from one class to another, both classes need to be present at compile time so the Java compiler can check to ensure that the method signatures are compatible. This requirement by itself makes for a static and nonextensible classing environment. Inevitably in a system like this, functionality gets pushed up higher and higher in the class hierarchy so that the mechanisms will be available to more and more subclasses. Interfaces are designed to avoid this problem. They disconnect the definition of a method or set of methods from the inheritance hierarchy. Since interfaces are in a different hierarchy from classes, it is possible for classes that are unrelated in terms of the class hierarchy to implement the same interface. This is where the real power of interfaces is realized.

Beginning with JDK 8, it is possible to add a default implementation to an interface method.

Thus, it is now possible for interface to specify some behavior.However, default methods constitute what is, in essence,

a special-use feature, and the original intent behind interface still remains.

Variables can be declared inside of interface declarations.

NOTE: They are implicitly final and static, meaning they cannot be changed by the implementing class.

They must also be initialized. All methods and variables are implicitly public.

NOTE: The methods that implement an interface must be declared public. Also, the type signature of the implementing

method must match exactly the type signature specified in the interface definition.

It is both permissible and common for classes that implement interfaces to define additional members of their own.

NOTE:

You can declare variables as object references that use an interface rather than a class type.

This process is similar to using a superclass reference to access a subclass object.

Any instance of any class that implements the declared interface can be referred to by such a variable.

When you call a method through one of these references, the correct version will be called based on the actual instance of the interface being referred to. Called at run time by the type of object it refers to.

The method to be executed is looked up dynamically at run time, allowing classes to be created later than the code which calls methods on them.

The calling code can dispatch through an interface without having to know anything about the “callee.”

CAUTION: Because dynamic lookup of a method at run time incurs a significant overhead when compared with the normal method invocation in Java, you should be careful not to use interfaces casually in performance-critical code.

Nested Interfaces:

An interface can be declared a member of a class or another interface. Such an interface is called a member interface or a nested interface. A nested interface can be declared as public, private, or protected.

This differs from a top-level interface, which must either be declared as public or use the default access level.

```
// This class contains a member interface.
class A {
    // this is a nested interface
    public interface NestedIF {
```

```
        boolean isNotNegative(int x);
    }
}
// B implements the nested interface.
class B implements A.NestedIF {
    public boolean isNotNegative(int x) {
        return x < 0 ? false: true;
    }
}
class NestedIFDemo {
    public static void main(String args[]) {
        // use a nested interface reference
        A.NestedIF nif = new B();
        if(nif.isNotNegative(10))
            System.out.println("10 is not negative");
        if(nif.isNotNegative(-12))
            System.out.println("this won't be displayed");
    }
}
```

Interfaces Can Be Extended:

One interface can inherit another by use of the keyword extends. The syntax is the same as for inheriting classes.

Any class that implements an interface must implement all methods required by that interface, including any that are

inherited from other interfaces.

Default Interface Methods (aka extension method) :

A primary motivation for the default method was to provide a means by which interfaces could be expanded without breaking existing code.

i.e. suppose you add another method without body in an interface. Then you will have to provide the body of that method

in all the classes that implement that interface.

Ex:

```
default String getString() {
    return "Default String";
}
```

For example, you might have a class that implements two interfaces.

If each of these interfaces provides default methods, then some behavior is inherited from both.

In all cases, a class implementation takes priority over an interface default implementation.

In cases in which a class implements two interfaces that both have the same default method, but the class does not

override that method, then an error will result.

In cases in which one interface inherits another, with both defining a common default method, the inheriting

interface’s version of the method takes precedence.

NOTE: static interface methods are not inherited by either an implementing class or a subinterface.

i.e. static interface methods should have a body! They cannot be abstract.

NOTE : when overriding methods, the access modifier should be same or better i.e. if in Parent Class it was protected, then then overridden should be either protected or public.