Java Interfaces

Interface is a concept which is used to achieve abstraction in Java. This is the only way by which we can achieve full abstraction. Interfaces are syntactically similar to classes, but you cannot create instance of an Interface and their methods are declared without any body. When you create an interface, it defines what a class can do without saying anything about how the class will do it.

It can have only abstract methods and static fields. However, from Java 8, interface can have default and static methods and from Java 9, it can have private methods as well.

When an interface inherits another interface extends keyword is used whereas class use implements keyword to inherit an interface.

- **Advantages of Interface**

It Support multiple inheritance

It helps to achieve abstraction

- **Syntax:**

```
interface interface_name {
        // fields
        // abstract/private/default methods
}
```

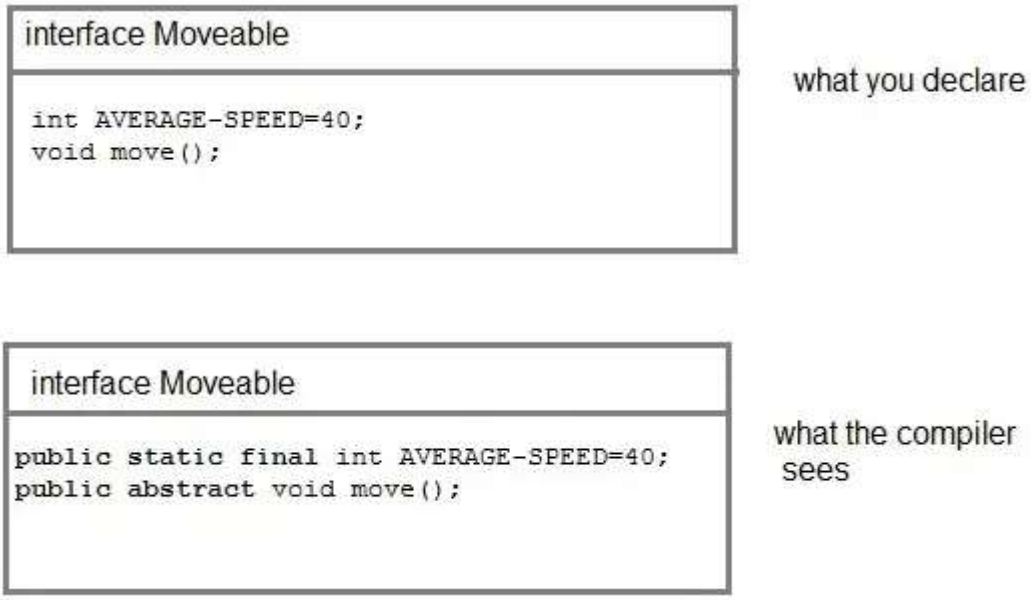
Interface Key Points

- Methods inside interface must not be static, final, native.
- All variables declared inside interface are implicitly public, static and final.
- All methods declared inside interfaces are implicitly public and abstract, even if you don't use public or abstract keyword.

- Interface can extend one or more other interface.
- Interface cannot implement a class.
- Interface can be nested inside another interface.

Let's take a simple code example and understand what interfaces are:

```
interface Moveable
{
        int AVERAGE-SPEED = 40;
        void move();
}
```



NOTE: Compiler automatically converts methods of Interface as public and abstract, and the data members as public, static and final by default.

- **Example of Interface implementation**

In this example, we created an interface and implemented using a class. lets see how to implement the interface.

```
interface Moveable
{
	int AVG_SPEED = 40;
	void move();
}
class Vehicle implements Moveable
{
	public void move()
	{
		System.out.println("Average speed is"+AVG_SPEED);
	}
	public static void main (String[] arg)
	{
		Vehicle vc = new Vehicle();
		vc.move();
	}
}
```

- **Interfaces support Multiple Inheritance**

Though classes in Java doesn't support multiple inheritance, but a class can implement more than one interfaces.

In this example, two interfaces are implemented by a class that show implementation of multiple inheritance.

```
interface Moveable
{
	boolean isMoveable();
}
interface Rollable
{
	boolean isRollable();
}
class Tyre implements Moveable, Rollable
{
	int width;
	public boolean isMoveable()
	{
		return true;
	}
	public boolean isRollable()
	{
		return true;
	}
	public static void main(String args[])
```

```
	{
		Tyre tr = new Tyre();
		System.out.println(tr.isMoveable());
		System.out.println(tr.isRollable());
	}
}
```

- **Interface extends other Interface**

Interface can inherit to another interface by using extends keyword. But in this case, interface just inherit, does not provide implementation. Implementation can be provided by a class only.

```
interface NewsPaper
{
	news();
}
interface Magazine extends NewsPaper
{
	colorful();
}
```

- **Difference between an interface and an abstract class?**

Interface and abstract class both are used to implement abstraction but have some differences as well. Some of differences are listed below.

Abstract class	Interface
Abstract class is a class which contain one or more abstract methods, which has to be implemented by its sub classes.	Interface is a Java Object containing method declaration but no implementation. The classes which implement the Interfaces must provide the method definition for all the methods.
Abstract class is a Class prefix with an abstract keyword followed by Class definition.	Interface is a pure abstract class which starts with interface keyword.
Abstract class can also contain concrete methods.	Whereas, Interface contains all abstract methods and final variable declarations.
Abstract classes are useful in a situation that some general methods should be implemented and specialization behaviour should be implemented by child classes.	Interfaces are useful in a situation that all properties should be implemented.