Interfaces:

#. The definition provided by the Sun Micro System about interface is that, Interfaces are the contract between a Java programmer and a Java programming language.

Real World Example:

Interfaces looks like a class because it also generates .class file, also has data member, member function, etcetera but Interfaces are not a class, they are a blue print of a class.

Another definition is that Interfaces are used to define standard and interfaces are one of the way to achieve the abstraction.

Interfaces are implicitly abstract, they cannot be instantiated that means we can not create the object of interfaces. Interfaces never represents an object.

In JDK7, all the interface functions were public and abstract by default, but since Java 8 If the member functions of an Interface are not static and default then they are public and abstract by default.

Since Java 8, interface can have default (non-static) and static methods.

Since JDK 9, we are allowed to define private methods in interface. We can have private static and non-static methods in interface from JDK9 onwards and by default all the methods in interface are public. Private Methods are not accessible from outside so we can only execute inside interfaces.

Interface fields are public, static and final by default, and methods are public and abstract.

Since Java 8, we can have method body in interface. But we need to make it default method

Rules1: Java Interface represents IS-A relationship. To make relationship we use "implements" keyword.

Why "implements" keyword? Because we do implementation of a contract.

Rule 2: A Child class has to override abstract method of an Interface. Abstract method must be public without any body.

Why? External package issue, if the register class is inside external package.

Rule 3: A Child class can implement more than one interfaces simultaneously (at same time as a group).

Rule 4: If a child class is getting a same method from more than one interfaces then, it has to override that method only once.

Rule 5: If we want to give the body of any non-static method in interface then we have to qualify that method with modifiers "default" and this default keyword can only be used within an interface. Java allowed to make method body inside interface for re-engineering purpose.

Rule 6: If a child class getting a same default method from more than one interfaces then it has to override that method otherwise it is a compilation error.

Rule 7: If one Interface is getting a same default method from more than one interfaces then child class has to override that method.

Rule 8: Static methods of an Interface are never inherited in a child class. So we can use it by interfaceName.functionName. We don't need follow rules like we do in case of other methods.

Rule 9: From JDK 8, we can make any interface executable by keeping a main method in it.

Rule 10: By default all the data member of an interface are final and static. Data members of an interface are required to initialized while declaration. Data members can inherit in case of interface but they can't override because by default they're final and static. To use interface data members we don't need to implement, we can use them by "interfaceName.fieldNam".

Rule 11: In Java all the interfaces are having all the non-final method of an object class.

Mine: That's why if we create that method in our child class and call it through putting child reference inside interface variable then it won't create error and run successfully. Instead if we create user define and call through interface variable then it will create error.

Rule 12: A class can inherit class and also implement interfaces at a time.

We can inherit method body from other class and abstract method from other interface. It requires to inherit first then Implement. Why?

Because if we implement interface first then it will look for overridden method for it's abstract method and that will come after inheriting the class so it will create compilation error that's why it requires to inherit first then implement.

Real world example: Every product gets body first then it gets cover.

Interface Program:

```
// Interface One
interface Computer{
	abstract public void GUI();
	abstract public void ALU();
}
// Interface two
interface CPUProcessor{
	abstract public void ALU(int x);
}
// Registration class
class ComputerReg{
	public void reg(Computer c){
		c.GUI();
		c.ALU();
	}
	public void CPUReg(CPUProcessor cp){
		cp.ALU(10);
	}
}

//Making Windows Computer GUI, Implementin the interfaces
class Windows implements Computer, CPUProcessor{
	static int x = 10;
	public static void main(String...a){
		Windows w = new Windows();
```

```
        ComputerReg cr = new ComputerReg();
        cr.reg(w);
        cr.CPUReg(w);
    }
    // Overriding the abstract methods
    public void GUI(){
        //Millions Code of Windows GUI
        System.out.println("Windows GUI");
    }
    public void ALU(){
        System.out.println("WIndows CPU Processor");
    }
    public void ALU(int x){
        System.out.println( x + " X WIndows CPU Processor");
    }
}
}
```

#. An interface that have no member is known as marker or tagged interface. For example: Serializable, Cloneable, Remote etc. They are used to provide some essential information to the JVM so that JVM may perform some useful operation.

Interface inheritance:

One interface can extend another interfaces. One interfaces can extends more than one interfaces simultaneously and this is the situation where we can say that Java supports Multiple-inheritance in case of interfaces.

A class implements interface with "**implements**" keyword but one interface extends another interface with "**extends**" keyword.

// Interface Inheritance

```
interface GrandParentInterface{
        public abstract void display();
}
interface ParentInterface extends GrandParentInterface{
        public abstract void show();
}
interface ChildInterface extends ParentInterface, GrandParentInterface{
        public abstract void draw();
        public abstract void show();  // again overwriting the show function
}
```

Data Member Inheritance:

```
interface InterfaceOne{
        int x = 50; // by default final and static
        int y = 100; // by default final and static
        //int z; // error. required to initialized
}
interface InterfaceTwo{
        int x = 900; // by default final and static
}
class Child implements InterfaceOne{
        int y = 300;
/* Can't override becasue x is final. Also it will create new variable.*/
        // static int x = 600;
        public static void main(String...a){
```

```
        System.out.println(InterfaceOne.x);    // working. Prints x value from interface
        System.out.println(x);              // working. Prints x value from interface
        System.out.println(InterfaceOne.y);    // working
        //System.out.println(y);   // error because y is non-static
        InterfaceOne io = new Child();
        System.out.println(io.y);  // working
// again working. without implementing InterfaceTwo
        System.out.println(InterfaceTwo.x);
        // now it is non-static new varaiable
        Child c = new Child();
        System.out.println(c.y);
    }
}
```

Programs:

#. If one Interface is getting a same default method from more than one interfaces then child class has to override that method.

```
interface InterfaceOne{
    default void show(){
        System.out.println("InterfaceOne Show");
    }
}
interface InterfaceTwo{
    default void show(){
        System.out.println("InterfaceTwo Show");
```

```
    }
}
```

/ /here inheriting interface from InterfaceOne and InterfaceTwo that have same defaul function show.

//in this case it requires to override that same default method

```
interface InterfaceThree extends InterfaceOne, InterfaceTwo{
    // overridding
    default void show(){
        System.out.println("InterfaceThree Show");
    }
}
```

#. Interface with static method with body {}

```
interface InterfaceOne{
    // static methods from interfaces never inherits in child class
    static void show(){
        System.out.println("Interface Static Function");
    }
    static void display(){
        System.out.println("Interface Display Function");
    }
}
interface InterfaceTwo{
    static void display(){
        System.out.println("Interface Display Function");
    }
}
```

#. Here inheriting interface from InterfaceOne and InterfaceTwo that have same static function display.

// In this case it is not require to override that same default method like we do for default method

```
interface InterfaceThree extends InterfaceOne, InterfaceTwo{
	// we don't need to overrride becasue display is static method.
	//Static don't override, they do function hidding
}
class Child implements InterfaceThree{
	static void show(){
		System.out.println("Child Static Function");
	}
	public static void main(String... a){
		// static methods from interfaces never inherits in child class
		// So we don't need to register
		InterfaceOne.show();
		show();
	}
}
```

#. We can also extends the method body from other calss and abstract method from other

```
interface InterfaceTwo{
	void show(); // by default public and abstract
}
class BodyClass{
	public void show(){
```

```
        System.out.println("Show From BodyClass");
    }
}
```

//We can also extends and implements togeher.

```
class ChildThree extends BodyClass implements InterfaceTwo{
    public static void main(String...a){
        InterfaceTwo it = new ChildThree();
        it.show();
    }
}
```

#. If we want to give the body of any non static method in interface then we have to qualify that method with modifiers "default" and this default keyword can only be used within an interface. Java allowed to make method body inside interface for re-engineering purpose.

```
class Child implements InterfaceOne{
    // Non-Static Method Without "default" keyword
    public void display(){
        System.out.println("Display");
    }
    // Non-Static Method Without "default" keyword
    public String toString(){
        return "hello";
    }
    public static void main(String...a){
        InterfaceOne io = new Child();
        //io.display(); //error
```

// Works. toString() is inside Object Class, that's why it is running. Above we are doing overriding with toString(). that's why it runs without default keyword.

```
            String s = io.toString();

            System.out.println(s);

      }

}
```

IQ: Can we make non-static variable/field in interface?

Ans: No, we can't make non-static variable in interface because by default all fields of interface are publi static and final.

IQ: Can we declare interface as final?

No. Compile time error will come. Error: Illegal modifier for the interface Sample; only public & abstract are permitted.

IQ: Can we declare constructor inside an interface?

No. Interfaces does not allow constructors.

IQ: 10.How can we access same variables defined in two interfaces implemented by a class?

Ans: By using corresponding interface.variable_name we can access variables of corresponding interfaces.

IQ: We can't make object of an interface. Interviewer trick like this.

```
      interface My{

            // No Method here called "Marker Interface"

      }

      class OuterClass{

            public static void main(String...a){

                  My m = new My(){};    // This is making an anonymous
class.
```

```
        }
    }
```

Don't consider it is making object of interface. That is syntax of anonymous class, where **My m** is holding reference of anonymous class. As we know it is required to have parent class or parent interface, here My is interface. This interface has no method that's why it is not demanding to do method overriding.

#. While creating anonymous class, why there is () after "new My()" class name or interface. Because new operator demands constructor syntax to create memory in heap area.

Difference between Abstraction and Interfaces:

When we want that the more than one parent can perform the same task, then it can only be achieved via Interfaces.

D1: All the data members of an interface class are static and final but abstract class does not.

D2: While giving a body of any method in an interface, we have to make it "**default**" but this rule does not go with abstract class.

D3: By default all the methods of an interface are public but all the methods of an abstract class are not public.

D4: Static methods of interface never inherited but abstract do static method Inheritance.

D5: If we want object + abstraction then go for abstract class. If we want only abstraction then we go for interface.

D6: If you want reusability + abstraction then up for abstract class. If we want only abstraction then go for interface.

D7: Interfaces are light-weight and faster than abstract class.

D8: We can implement more than one interface but we can’t extend more than one abstract class.

Mine:

To make standard, we need to use abstraction. We won't execute any method of interface but we will only override interface method and provide our work.