```java
/*

Interface :

--> Interface is a java feature, it will allow only abstract methods.
--> For interfaces, we area ble to create only reference variables, we are unable to create objects.
--> In the case of interfaces, by default, all the variables are
        " public static final "
--> In the case of interfaces, by default, all the methdos are
        " public and abstract"

--> In java applications, constructors are possible in classes and abstract classes
    but constructors are not possible in interfaces.

--> Intefaces provide more sharabiluty in java applications when compared with classes and abstract classes.

--> In java applications, if we declare any interface with abstract methods then it is covention to declare an
    implementation class for the interface and it is conventin to provide implementation for all the
    abstract methods in implementation class.

 */

package Interface;

public interface Interface1 {

    void m1();

    void m2();

    void m3();
}

class A implements Interface1 {

    public void m1() {
        System.out.println("m1 - A");
    }

    public void m2() {
        System.out.println("m2 - A");
    }

    public void m3() {
        System.out.println("m3 - A");
    }

    public void m4() {
        System.out.println("m4 - A");
    }
```

```
51  }
52
53  class Main1 {
54      public static void main(String[] args) {
55          Interface1 i1 = new A();
56          i1.m1();
57          i1.m2();
58          i1.m3();
59          // i1.m4(); // C.E: The method m4() is undefined for the type
            Interface1
60
61          A a = new A();
62          a.m1();
63          a.m2();
64          a.m3();
65          a.m4();
66      }
67  }
68
```

```java
/*
        Use of interface and abstract class together

--> In Java applications, if we declare an interface with abstract methods then
    it is mandatory to provide implementation for all the abstract methods in the respective implementation class.
--> In this context if we provide implementation for some of the abstract methods at the respective implementation class
    then compiler will rise an error,
--> where to come out from the compilation error we have to declare the respective implementation class as an abstract class
--> and we have to provide implementation for the remaining abstract methods by taking a sub class for the abstract class.

 */


package Interface;

public interface Interface2 {
    void m1();

    void m2();

    void m3();
}

abstract class A1 implements Interface2 {
    public void m1() {
        System.out.println("m1 - A1");
    }

    void m4() {
        System.out.println("m4 - A1");
    }
}

class B1 extends A1 {
    public void m2() {
        System.out.println("m2 - B1");
    }

    public void m3() {
        System.out.println("m3 - B1");
    }
}

class Main2 {
    public static void main(String[] args) {
        Interface2 i2 = new B1();
        i2.m1();
        i2.m2();
        i2.m3();
        // i2.m4(); // C.E: The method m4() is undefined for the type
```

```
            Interface2

            B1 b1 = new B1();
            b1.m1();
            b1.m2();
            b1.m3();
            b1.m4();
        }
    }
```

```java
/*

Multiple Inheritance using Interface

--> In Java applications, it is not possible to extend more than one class to a single class
    but it is possible to extend more than one interface to a single interface.

 */

package Interface;

public interface Interface3 {
    void m1();
}

interface Interface31 {
    void m2();
}

interface Interface32 extends Interface3, Interface31 {
    void m3();
}


class A3 implements Interface32 {

    public void m1() {
        System.out.println("m1 - A3 method");
    }

    public void m2() {
        System.out.println("m2 - A3 method");
    }

    public void m3() {
        System.out.println("m3 - A3 method");
    }
}

class Main3 {
    public static void main(String[] args) {

        Interface3 i3 = new A3();

        i3.m1();
        // i3.m2(); // C.E.
        // i3.m3(); // C.E.

        Interface31 i31 = new A3();

        // i31.m1(); // C.E.
        i31.m2();
        // i31.m3(); // C.E.
```

```
54
55              Interface32 i32 = new A3();
56
57              i32.m1();
58              i32.m2();
59              i32.m3();
60          }
61      }
62
```

```java
/*

--> In Java applications, it is possible to implement more than one interface into a single implementation class.

 */

package Interface;

public interface Interface4 {
    void m1();
}

interface Interface41 {
    void m2();
}

interface Interface42{
    void m3();
}

class A4 implements Interface4, Interface41, Interface42 {

    public void m1() {
        System.out.println("m1 - A4");
    }

    public void m2() {
        System.out.println("m2 - A4 method");
    }

    public void m3() {
        System.out.println("m3 - A4 method");
    }

}

class Main4 {
    public static void main(String[] args) {

        Interface4 i4 = new A4();
        i4.m1();
        // i4.m2(); // C.E.
        // i4.m3(); // C.E.

        Interface41 i41 = new A4();
        // i41.m1(); // C.E.
        i41.m2();
        // i41.m3(); // C.E.

        Interface42 i42 = new A4();
        // i42.m1(); // C.E.
        // i42.m2(); // C.E.
        i42.m3();

```

```
55          }
56      }
57
```

```java
/*

--> In general, if we declare abstract methods in an interface then we have to implement all that interface methods
    in more number of classes with variable implementation part.
--> In the above context, if we require any method implementation common to every implementation class with fixed implementation
    then we have to implement that method in the interface as default method.
--> To declare default methods in interfaces we have to use "default" keyword in method syntax like access modifier.
--> If we need to add new function in existing interface, then with the help of "default" implementation in interface,
    we can provide this functionality to all child classes.
--> In interface, method can be private but we have to write body also in same class.


 */

package Interface;

public interface Interface5 {
    default void m1() {
        System.out.println("m1 - default");
    }

    // It is possible to provide more than one default methods within a single interface.

    default void m2() {
        System.out.println("m2 - default");
    }

    private void m3() {
        System.out.println("m3 - private");
    }
}

class A5 implements Interface5 {

}

class A51 implements Interface5 {

}

class Main5 {
    public static void main(String[] args) {
        Interface5 i5 = new A5();
        i5.m1();

        Interface5 i51 = new A51();
        i51.m2();
    }
```

```
49    }
```

```java
/*

--> After Java8, it is possible to override default methods in the implementation classes.

 */

package Interface;

public interface Interface6 {
    default void m1() {
        System.out.println("m1 - default");
    }
}

class A6 implements Interface6 {
    public void m1() {
        System.out.println("m1 - A6");
    }
}

class A61 implements Interface6 {

}

class Main6 {
    public static void main(String[] args) {
        Interface6 i6 = new A6();
        i6.m1();

        Interface6 i61 = new A61();
        i61.m1();
    }
}
```

```
/*

Static Methods in Interfaces:

--> Up to JAVA7 version, static methods are not possible in interfaces
--> but from JAVA8 version static methods are possible in interfaces in order to improve sharability.
--> If we declare static methods in the interfaces then it is not required to declare any implementation class to access that static method,
    we can use directly interface name to access static method.
--> Static method can not be overridden in the child class.

    NOTE:

    -> If we declare static methods in an interface then they will not be available to the respective implementation classes,
    -> we have to access static methods by using only interface names not even by using interface reference variable
    -> After JAVA8 version, interfaces will allow concrete methods along with either "static" keyword or "default" keyword.

 */


package Interface;

public interface Interface7 {
    static void m1() {
        System.out.println("m1 - static");
    }
}

class A7 implements Interface7 {
    void m1(){
        System.out.println("m1 - A7");
    }
}



class Main7 {
    public static void main(String[] args) {
        Interface7.m1();
        Interface7 i7 = new A7();
        // i7.m1(); // C.E.: This static method of interface Interface7 can only be accessed as Interface7.m1
        A7 a7 = new A7();
        a7.m1();
    }
}

```