Q) What is an interface in Java? Give some real world coding example.

- way to achieve abstraction in Java.
- blueprint of a class
- interface cannot be instantiated

Theater → interface
↓ blueprint of a class
Theater name seats → class
↓ blueprint of object
pvr cinepolis → Object

Q) What is the default state of variables in an interface?

↳ instance variables ✗

↳ public, static, final

- public ★
- static ★ ↳ class property
- final ★ ↳ constants

Q) What is the default state of methods in an interface?

↳ public, abstract (100% by default abstraction)

- public ★

```java
interface ITheater {
    String industry = "Bollywood";
    // public, static, final

    // 100% abstraction

    void viewShow(); // public, abstract

    String bookShow();
}

class Theater implements ITheater {
    String name;

    public void viewShow() {
        System.out.println(x: "Only View Access");
    }

    public String bookShow() {
        System.out.println(x: "Book Access");
        return "ticket";
    }
}
```

```java
class BookMyShow {
    @SuppressWarnings("all")
    Run | Debug
    public static void main(String[] args) {
        Theater pvr = new Theater();
        pvr.name = "PVR Cinemas";

        pvr.viewShow();

        System.out.println(ITheater.industry);
        System.out.println(Theater.industry);

        Theater cinepolis = new Theater();
        cinepolis.name = "Cinepolis Experience";

        cinepolis.bookShow();

        System.out.println(pvr.industry);
        System.out.println(cinepolis.industry);

    }
}
```

```
Only View Access
Bollywood
Bollywood
Book Access
Bollywood
Bollywood
```

Q) Can there be concrete methods in interface?

↳ by default, you cannot have concrete methods in Java ( version < 8)

exception

↳ private method, static method, default method

↳ concrete ↳ concrete ↳ default implementation

Q) Can an interface be related to another interface. If yes, how?

↳ yes : one interface extends another interface!

```java
interface MyInterface {
    // Default Method: Object's Method
    default void defaultFun() {
        System.out.println(x: "Default fun: have a body");
        privateFun();
    }

    // private method: Within Interface
    private void privateFun() {
        System.out.println(x: "Private fun: have a body");
    }

    // Interface's Method
    public static void staticFun() {
        System.out.println(x: "Static fun: have a body");
    }
}

class MyClass implements MyInterface {
}

class Driver {
    Run | Debug
    public static void main(String[] args) {
        MyInterface.staticFun();

        MyClass obj = new MyClass();
        obj.defaultFun();
    }
}
```

```
Static fun: have a body
Default fun: have a body
Private fun: have a body
```

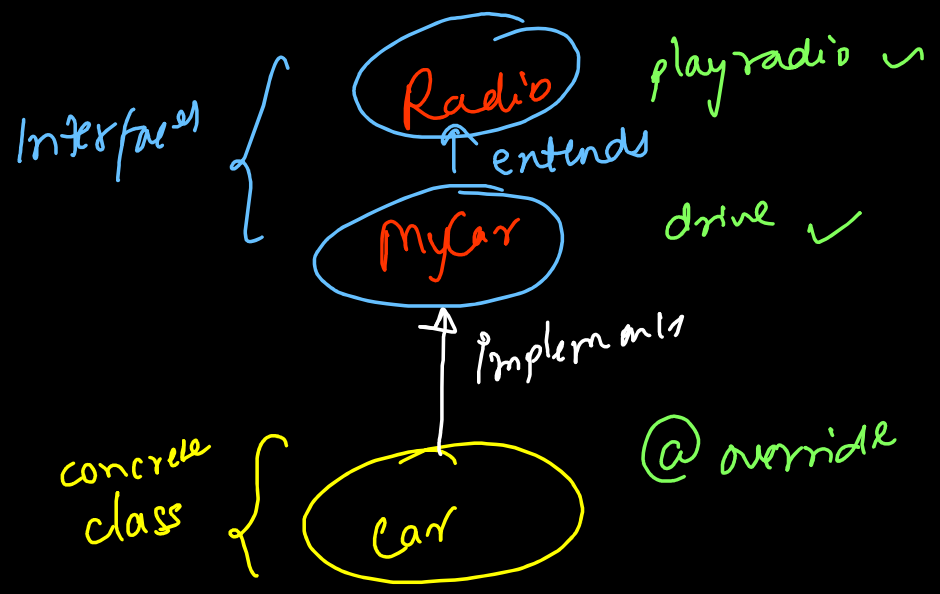
```java
interface Radio {
    void playRadio();
}

interface MyCar extends Radio {
    void drive();
}

class Car implements MyCar {
    public void playRadio() {
        System.out.println(x: "Radio Starts");
    }

    public void drive() {
        System.out.println(x: "Car Starts");
    }
}

class Driver {
    Run | Debug
    public static void main(String[] args) {
        Car i10 = new Car();
        i10.drive();
        i10.playRadio();
    }
}
```



There is no object of interface ⇒ there is no this keyword

Q) Is there (a) this → no (b) constructor → no defined in an interface?

→ 100% abstraction

variables → static & final

instance variables ∝ → constructor (initializat^n)

→ interfacey do not take part in class heirarchy

super() → child → Object ✓ / Interface ∝

Q) Can class implement more than 1 interfaces? will it not cause diamond problem?

↳ Class can implement 1 or more than 1 interfaces

{multiples interfaces can be implemented by a single class}

① no instance variable & ambiguity &

② diamond problem &

interface 1

abstract

data &

fun();

interface 2

abstract

data &

fun();

implements

implements

@ override

fun() { }

class

⟹ overriding only once

⟹ diamond problem

```
You, 1 second ago | 1 author (You)
interface Radio2 {
    void start();
}

You, 1 second ago | 1 author (You)
interface MyCar2 {
    void start();
}

You, 1 second ago | 1 author (You)
class Car2 implements Radio2, MyCar2 {
    @Override
    public void start() {
        System.out.println(x: "Radio & Car
        Starts Together");
    }
}
```

abstract ambiguity 2

→ No diamond problem

→ only once

```
Car2 i20 = new Car2();
i20.start();
```

Output

You, 1 second ago | 1 author (You)

```java
interface Radio2 {
    String data = "Radio"; // static

    void start();
}
```

You, 1 second ago | 1 author (You)

```java
interface MyCar2 {
    String data = "Car"; // static

    void start();
}
```

You, 1 second ago | 1 author (You)

```java
class Car2 implements Radio2, MyCar2 {
    @Override
    public void start() {
        System.out.println(x: "Radio & Car
        Starts Together");
    }

    public void fun() {
        System.out.println(Radio2.data);
        System.out.println(MyCar2.data);
    }
}
```

} static property
↳ Interface name
⇓
ambiguity ∝

Q) What is the difference between class & interface?

| Class | Interface |
| --- | --- |
| ① blueprint of an object | ① blueprint of a class |
| ② achieve encapsulation | ② achieve 100% abstraction |
| ③ it can be instantiated | ③ it cannot be instantiated |
| ④ constructors, this, super ✓ | ④ constructor, this, super ✗ |
| ⑤ properties → static/nonstatic nonfinal/final | ⑤ properties → public static final |
| ⑥ methods → abstract/concrete | ⑥ all methods are abstract & public |

| Class | Interface |
| --- | --- |
| The keyword used to create a class is “class” | The keyword used to create an interface is “interface” |
| A class can be instantiated i.e, objects of a class can be created. | An Interface cannot be instantiated i.e, objects cannot be created. |
| Classes does not support multiple inheritance. | Interface supports multiple inheritance. |
| It can be inherit another class. | It cannot inherit a class. |
| It can be inherited by another class using the keyword ‘extends’. | It can be inherited by a class by using the keyword ‘implements’ and it can be inherited by an interface using the keyword ‘extends’. |
| It can contain constructors. | It cannot contain constructors. |
| It cannot contain abstract methods. | It contains abstract methods only. |
| Variables and methods in a class can be declared using any access specifier(public, private, default, protected) | All variables and methods in a interface are declared as public. |
| Variables in a class can be static, final or neither. | All variables are static and final. |

Q) What are the differences between abstract classes & interfaces?

Similarities → to achieve abstraction, cannot instantiate

| Abstract class | Interface |
| --- | --- |
| ① 0-100% abstraction | ① 100% abstraction |
| ② by defaulit :→ concrete method | ② by defaulit :→ abstract method |
| ③ variables → instance variables, final/static variable | ③ variables → final/static |
| ④ multiple inheritance (✗) | ④ class can implement multiple interfaces |
| ⑤ access modifier → public/default/protected/private | ⑤ access modifier → public |

| Abstract class | Interface |
| --- | --- |
| 1) Abstract class can **have abstract and non-abstract** methods. | Interface can have **only abstract** methods. Since Java 8, it can have **default and static methods** also. |
| 2) Abstract class **doesn't support multiple inheritance**. | Interface **supports multiple inheritance**. |
| 3) Abstract class **can have final, non-final, static and non-static variables**. | Interface has **only static and final variables**. |
| 4) Abstract class **can provide the implementation of interface**. | Interface **can't provide the implementation of abstract class**. |
| 5) The **abstract keyword** is used to declare abstract class. | The **interface keyword** is used to declare interface. |
| 6) An **abstract class** can extend another Java class and implement multiple Java interfaces. | An **interface** can extend another Java interface only. |
| 7) An **abstract class** can be extended using keyword "extends". | An **interface** can be implemented using keyword "implements". |
| 8) A Java **abstract class** can have class members like private, protected, etc. | Members of a Java interface are public by default. |