



WELCOME TO THE PRESENTATION



Interface



Java™



TEAM MEMBER

22211118

MAHBUBA AKTER
JOTY

22111119

ABU NAIM PIAS

22111120

MD MONIR AHMED

22111121

HIMEL SARDER




TOPICS FOR DISCUSSION

- What is interface
- Why use Java interface
- Relationship between class and interface
- Interface Declaration
- Interface Method
- Interface Properties
- Example
- Multiple inheritance is not supported through class in java, but it is possible by an interface, why?
- Diamond Problem





WHAT IS INTERFACE IN JAVA

- Interface is a reference type in Java. It is similar to class. It has static constants.
 - It is a collection of abstract methods.
 - There can be only abstract methods in the java interface not method body. It is used to achieve fully abstraction and multiple inheritance in Java.
- 

WHY USE JAVA INTERFACE

It is used to achieve abstraction.

1

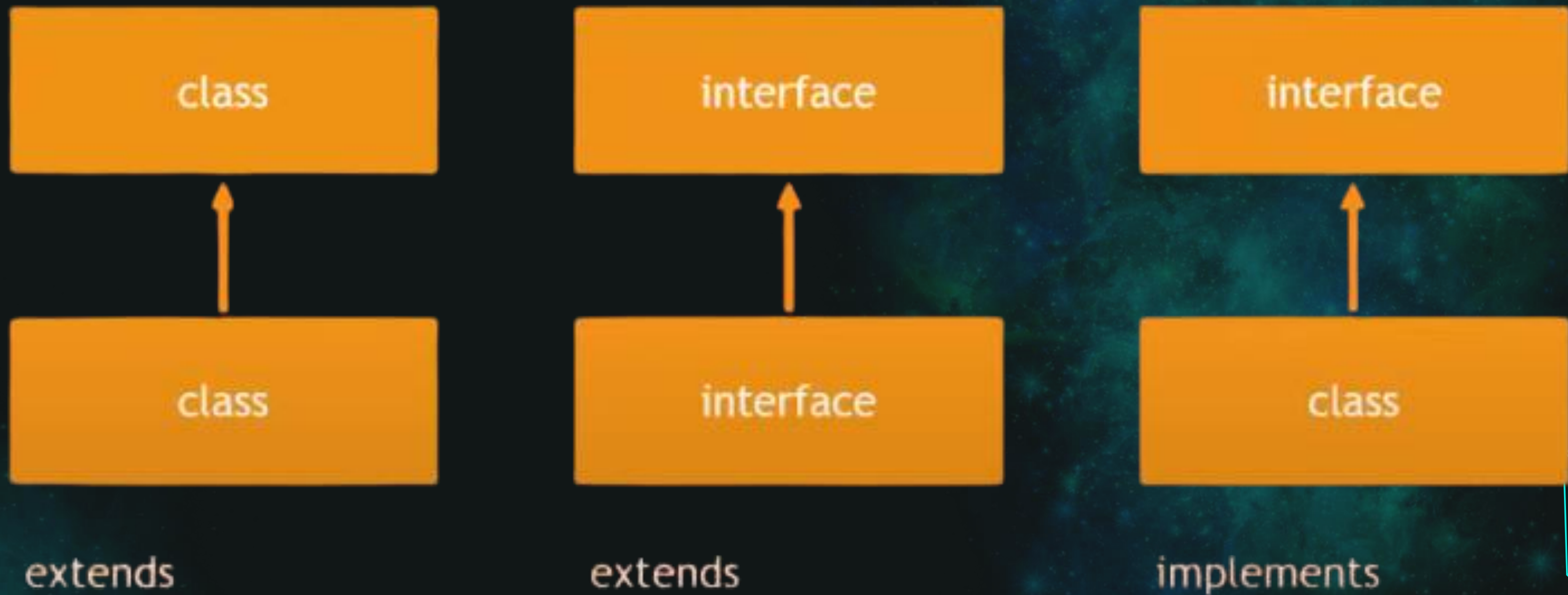
2

By interface, we can support the functionality of multiple inheritance.

It can be used to achieve loose coupling.


3

RELATIONSHIP BETWEEN CLASS AND INTERFACE





How interface is different from a class?

1. You can't instantiate an interface.
 2. Interface doesn't contain constructor.
 3. All the methods in interface are abstract.
 4. An interface can't have instance variables.
 5. An interface can extend multiple interface.
- 

INTERFACE DECLARATION

```
interface <interface_name>{  
  
    // declare constant fields  
  
    // declare methods that abstract  
    // by default.  
  
}
```

To declare an interface, use the interface keyword.

INTERFACE METHOD

An interface is a collection of abstract methods.

```
interface Animal {
```

```
    eat(); -----> Compiler -----> public abstract void eat();  
}
```

The Java Compiler adds public and abstract keywords before the interface method.

The background features a dark, textured teal and blue gradient. In the top-left and bottom-right corners, there are abstract geometric wireframe structures composed of interconnected lines forming various polygons.

INTERFACE PROPERTIES

Interfaces have the following properties

- Each method in an interface is also implicitly abstract, so the abstract keyword is not needed.
- Methods in an interface are implicitly public.
- A class can inherit from just one superclass, but can implement multiple interfaces!

EXAMPLE

```
// Define an interface named Animal
interface Animal {
    // Declare a method named makeSound
    void makeSound();
}

// Define a class named Dog that implements the Animal interface
class Dog implements Animal {
    // Implement the makeSound method specified by the Animal interface
    @Override
    public void makeSound() {
        System.out.println("Woof!");
    }
}

// Define a class named Cat that implements the Animal interface
class Cat implements Animal {
    // Implement the makeSound method specified by the Animal interface
    @Override
    public void makeSound() {
        System.out.println("Meow!");
    }
}
```


EXAMPLE

```
// Main class to test the Animal interface and its implementations
public class Main {
    public static void main(String[] args) {
        // Create instances of Dog and Cat
        Animal dog = new Dog();
        Animal cat = new Cat();

        // Call the makeSound method for each animal
        dog.makeSound();
        cat.makeSound();
    }
}
```

OUTPUT

```
C:\Users\LEN0NV0\.jdk\openjdk-21.0.2\bin\java.exe
```

```
Woof!
```

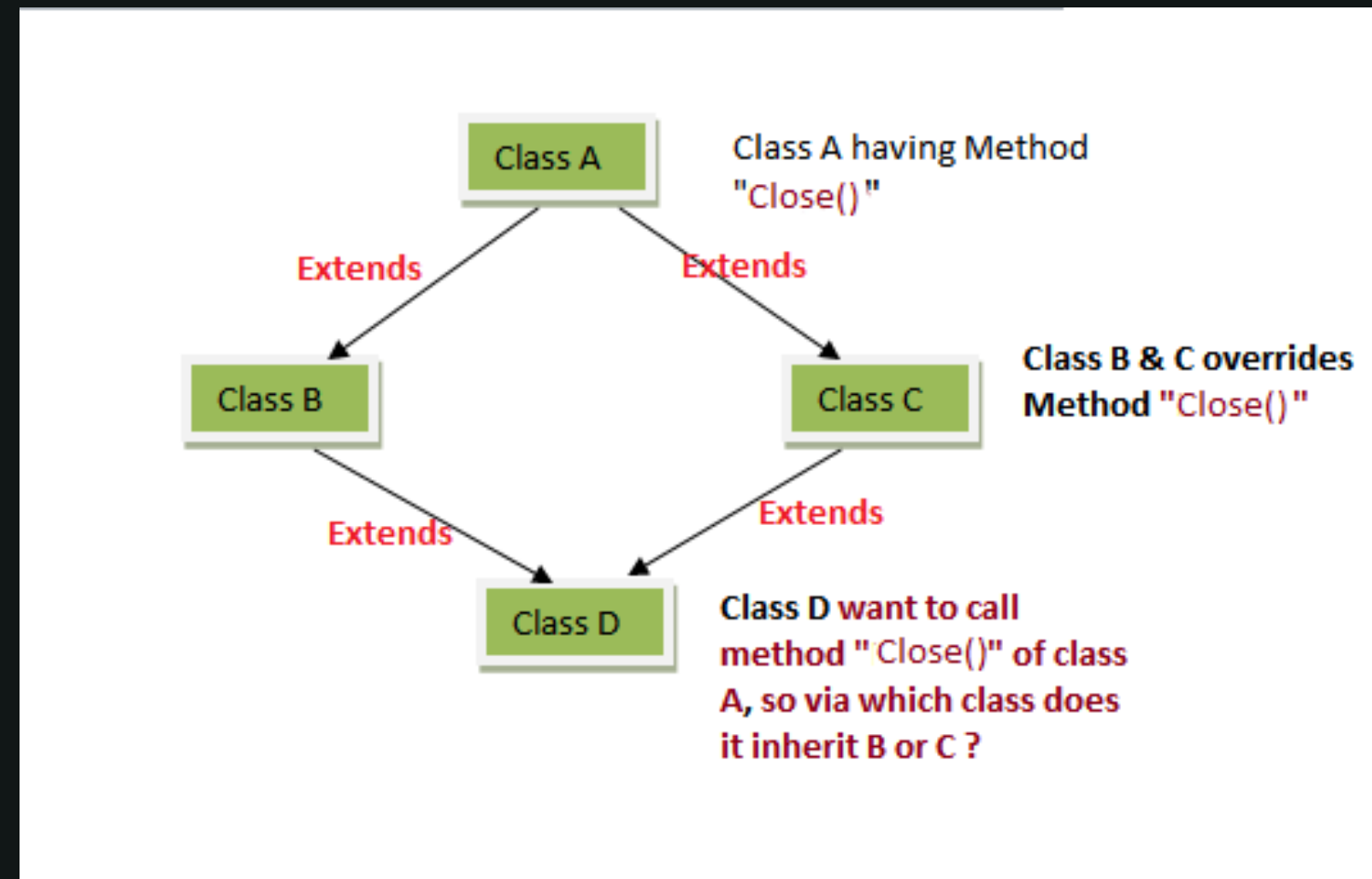
```
Meow!
```

```
Process finished with exit code 0
```

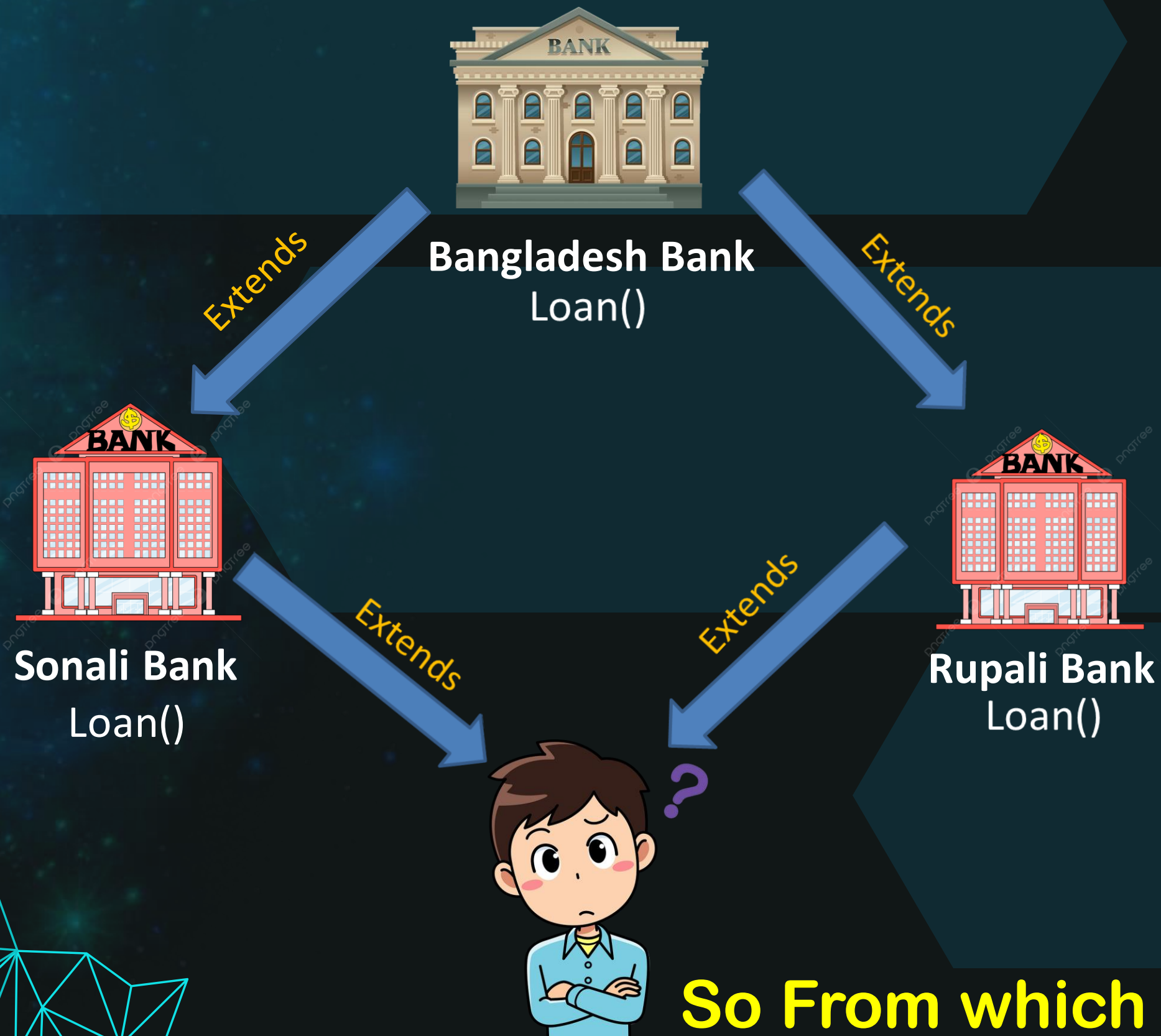

MULTIPLE INHERITANCE IS NOT SUPPORTED THROUGH CLASS IN JAVA, BUT IT IS POSSIBLE BY AN INTERFACE, WHY?

Multiple inheritance is not supported in the case of class because of ambiguity problem. However, it is supported in case of an interface because there is no ambiguity problem.

Inheritance in Java allows a child class to inherit properties from a parent class. However, when a child class inherits from more than one parent class, and both parents have methods with the same name and parameters, confusion arises. This situation is known as the **Diamond Problem**.



Diamond Problem with story



Bangladesh Bank: This serves as the parent class, analogous to the common ancestor in the diamond problem.

Sonali Bank and Rupali Bank: These are the child classes, both inheriting from Bangladesh Bank. They offer Motin a loan of 1 lakh taka each.

Motin: He is faced with the decision of choosing between the two banks or going directly to Bangladesh Bank for a loan.

So From which Bank Motin will take loan?

ERROR!

```
/tmp/RXc6GjteYE/BangladeshBank.java:19: error: '{'  
    expected
```

```
class Motin extends SonaliBank, RupaliBank {  
                                     ^
```

```
1 error
```

```
=== Code Exited With Errors ===
```

In the above example, the Motin extends both SonaliBank and RupaliBank. Both parent classes define a loan() method. When we call b.loan(), the compiler is unsure which definition of loan() to inherit, leading to ambiguity

```
class BangladeshBank {  
    void loan() {  
        System.out.println("Please take loan");  
    }  
}
```

```
class SonaliBank extends BangladeshBank{  
    void loan() {  
        System.out.println("Please take loan from Sonali Bank");  
    }  
}
```

```
class RupaliBank extends BangladeshBank{  
    void loan() {  
        System.out.println("Please take loan from Rupali Bank");  
    }  
}
```

```
class Motin extends SonaliBank, RupaliBank {  
    public static void main(String[] args) {  
        Motin b = new Motin();  
        b.loan();  
    }  
}
```



```
java -cp /tmp/OLkWurn1NM/Main
```

```
Don't need any loan
```

```
=== Code Execution Successful ===
```

Interfaces:

- BangladeshBank: This interface declares a single method loan().
- SonaliBank and RupaliBank: Both interfaces extend BangladeshBank and declare the same method loan().

Class Motin:

- This class implements both SonaliBank and RupaliBank interfaces.
- It provides an implementation for the loan() method, which simply prints "Don't need any loan" when called.

Class Main:

- This is the main class of the program.
- In the main method, an instance of Motin is created.
- The loan() method of the Motin instance is then called, resulting in the message "Don't need any loan" being printed to the console.

```
interface BangladeshBank {  
    void loan();  
}
```

```
interface SonaliBank extends BangladeshBank {  
    void loan();  
}
```

```
interface RupaliBank extends BangladeshBank {  
    void loan();  
}
```

```
class Motin implements SonaliBank, RupaliBank {  
    public void loan() {  
        System.out.println("Don't need any loan");  
    }  
}
```

```
public class Main {  
    public static void main(String[] args) {  
        Motin motin = new Motin();  
        motin.loan();  
    }  
}
```




Thank you for your attention

**Please clap and don't ask difficult
questions**