

Java and Multiple Inheritance

Difficulty Level : Medium Last Updated : 17 Dec, 2021

Multiple Inheritance is a feature of an object-oriented concept, where a class can inherit properties of more than one parent class. The problem occurs when there exist methods with the same signature in both the superclasses and subclass. On calling the method, the compiler cannot determine which class method to be called and even on calling which class method gets the priority.

Note: Java doesn't support Multiple Inheritance

Example 1:

```
// Java Program to Illustrate Unsupportance of
// Multiple Inheritance

// Importing input output classes
import java.io.*;

// Class 1
// First Parent class
class Parent1 {

    // Method inside first parent class
    void fun() {

        // Print statement if this method is called
        System.out.println("Parent1");
    }
}

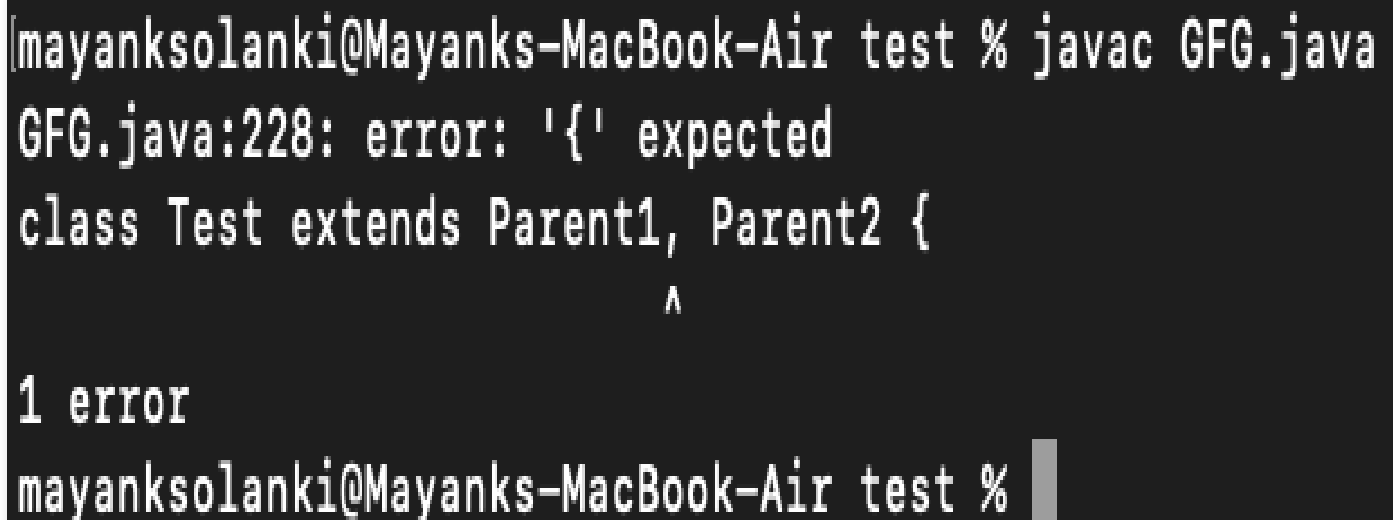
// Class 2
// Second Parent Class
class Parent2 {

    // Method inside first parent class
    void fun() {

        // Print statement if this method is called
        System.out.println("Parent2");
    }
}
```

```
    }  
}  
  
// Class 3  
// Trying to be child of both the classes  
class Test extends Parent1, Parent2 {  
  
    // Main driver method  
    public static void main(String args[]) {  
  
        // Creating object of class in main() method  
        Test t = new Test();  
  
        // Trying to call above functions of class where  
        // Error is thrown as this class is inheriting  
        // multiple classes  
        t.fun();  
    }  
}
```

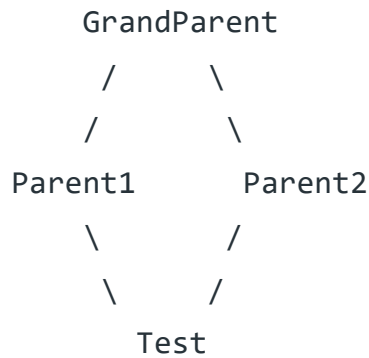
Output: Compilation error is thrown



```
mayanksolanki@Mayanks-MacBook-Air test % javac GFG.java  
GFG.java:228: error: '{' expected  
class Test extends Parent1, Parent2 {  
                           ^  
1 error  
mayanksolanki@Mayanks-MacBook-Air test %
```

Conclusion: As depicted from code above, on calling the method `fun()` using `Test` object will cause complications such as whether to call `Parent1's fun()` or `Parent2's fun()` method.

Example 2:



The code is as follows

```

// Java Program to Illustrate Unsupportance of
// Multiple Inheritance
// Diamond Problem Similar Scenario

// Importing input output classes
import java.io.*;

// Class 1
// A Grand parent class in diamond
class GrandParent {

    void fun() {

        // Print statement to be executed when this method is called
        System.out.println("Grandparent");
    }
}

// Class 2
// First Parent class
class Parent1 extends GrandParent {
    void fun() {

        // Print statement to be executed when this method is called
        System.out.println("Parent1");
    }
}

// Class 3
// Second Parent Class
class Parent2 extends GrandParent {
    void fun() {

        // Print statement to be executed when this method is called
        System.out.println("Parent2");
    }
}
  
```

```
}

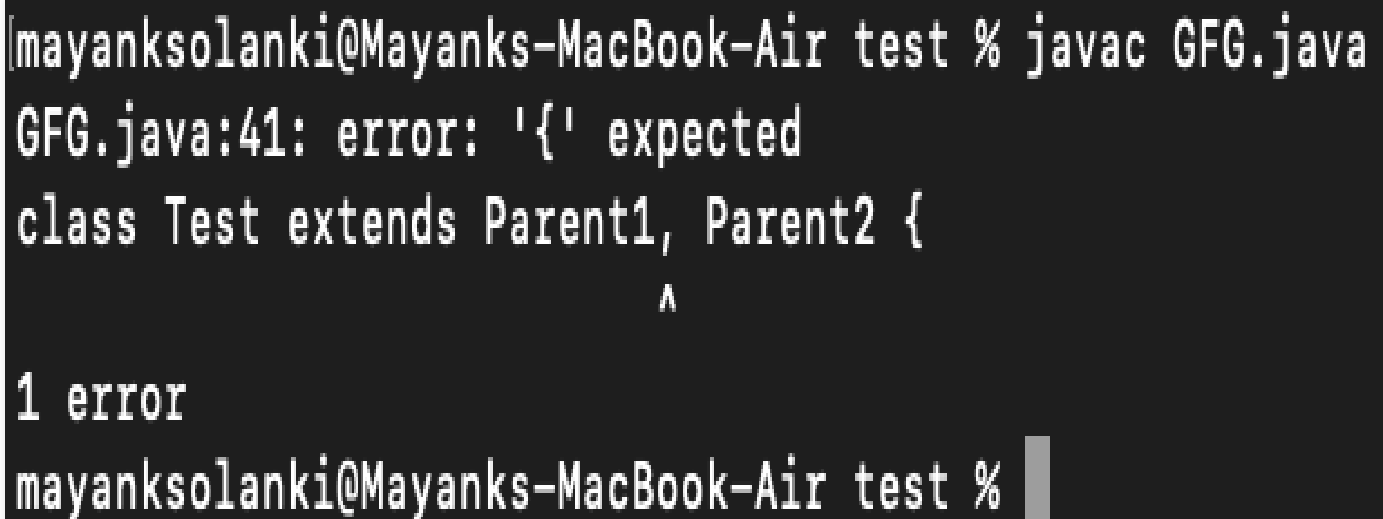
// Class 4
// Inheriting from multiple classes
class Test extends Parent1, Parent2 {

    // Main driver method
    public static void main(String args[]) {

        // Creating object of this class i main() method
        Test t = new Test();

        // Now calling fun() method from its parent classes
        // which will throw compilation error
        t.fun();
    }
}
```

Output:



```
mayanksolanki@Mayanks-MacBook-Air test % javac GFG.java
GFG.java:41: error: '{' expected
class Test extends Parent1, Parent2 {
                        ^
1 error
mayanksolanki@Mayanks-MacBook-Air test %
```

Again it throws compiler error when run() method as multiple inheritances cause a diamond problem when allowed in other languages like C++. From the code, we see that: On calling the method fun() using Test object will cause complications such as whether to call Parent1's fun() or Child's fun() method. Therefore, in order to avoid such complications, Java does not support multiple inheritances of classes.

Multiple inheritance is not supported by Java using classes, handling the complexity that causes due to multiple inheritances is very complex. It creates problems during various operations like casting, constructor chaining, etc, and the above all reason is that there

are very few scenarios on which we actually need multiple inheritances, so better to omit it for keeping things simple and straightforward.

How are the above problems handled for Default Methods and Interfaces?

Java 8 supports default methods where interfaces can provide a default implementation of methods. And a class can implement two or more interfaces. In case both the implemented interfaces contain default methods with the same method signature, the implementing class should explicitly specify which default method is to be used, or it should override the default method.

Example 3:

```
// Java program to demonstrate Multiple Inheritance
// through default methods

// Interface 1
interface PI1 {

    // Default method
    default void show()
    {

        // Print statement if method is called
        // from interface 1
        System.out.println("Default PI1");
    }
}

// Interface 2
interface PI2 {

    // Default method
    default void show()
    {

        // Print statement if method is called
        // from interface 2
        System.out.println("Default PI2");
    }
}

// Main class
// Implementation class code
class TestClass implements PI1, PI2 {
```

```
// Overriding default show method
public void show()
{

    // Using super keyword to call the show
    // method of PI1 interface
    PI1.super.show();

    // Using super keyword to call the show
    // method of PI2 interface
    PI2.super.show();
}

// Mai driver method
public static void main(String args[])
{

    // Creating object of this class in main() method
    TestClass d = new TestClass();
    d.show();
}
}
```

Output

Default PI1

Default PI2

Note: If we remove the implementation of default method from “TestClass”, we get a compiler error. If there is a diamond through interfaces, then there is no issue if none of the middle interfaces provide implementation of root interface. If they provide implementation, then implementation can be accessed as above using super keyword.

Example 4:

```
// Java program to demonstrate How Diamond Problem
// Is Handled in case of Default Methods

// Interface 1
```

```
interface GPI {

    // Default method
    default void show()
    {

        // Print statement
        System.out.println("Default GPI");
    }
}

// Interface 2
// Extending the above interface
interface PI1 extends GPI {
}

// Interface 3
// Extending the above interface
interface PI2 extends GPI {
}

// Main class
// Implementation class code
class TestClass implements PI1, PI2 {

    // Main driver method
    public static void main(String args[])
    {

        // Creating object of this class
        // in main() method
        TestClass d = new TestClass();

        // Now calling the function defined in interface 1
        // from whom Interface 2 and 3 are deriving
        d.show();
    }
}
```

Output

Default GPI