# **Advanced Programming (Java)**

**Diamond Problem, Default Methods, Inner Classes and Generics**

**Week 11 – Reading Material and Practice Material**

# What is Diamond Problem in Java

In **Java, the diamond problem** is related to multiple inheritance. Sometimes it is also known as the **deadly diamond problem** or **deadly diamond of death**. In this session, we will learn **what is the demand problem in Java** and **what is the solution to the diamond problem**.

Before moving to the diamond problem let's have a look at [**inheritance in Java**](https://www.javatpoint.com/inheritance-in-java).

# Inheritance in Java

Inheritance is a relation between two classes, the parent and child class. The child class (sub-class) inherits all the properties of the parent class (super-class). To define the relation, we use **extends** keyword. For example:

1. **public** **class** A **extends** B
2. {
3. }

When we inherit the properties of a class into another class, a copy of the super-class (parent class) is created in the sub-class (child class) object. Hence, by using the sub-class object, we can access the member of super-class, also.

# Multiple Inheritance

It is a feature of an object-oriented concept, where a class can inherit properties of more than one parent class. The feature creates a problem when there exist methods with the same name and signature in both the super-class and sub-class. When we call the method, the compiler gets confused and cannot determine which class method to be called and even on calling which class method gets the priority.

Let's understand the concept through an example.

**A.java**

**class** A

{

**public** **void** display()

{

System.out.println("class A display() method called");

}

}

**class** B **extends** A

{

@Override

**public** **void** display()

{

System.out.println("class B display() method called");

}

}

**class** C **extends** A

{

@Override

**public** **void** display()

{

System.out.println("class C display() method called");

}

}

//not supported in Java

**public** **class** D **extends** B,C

{

**public** **static** **void** main(String args[])

{

D d = **new** D();

//creates ambiguity which display() method to call

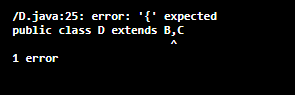
d.display();

}

}

* Class B and class C inherits the class A. The display() method of class A is overridden by the class B and class C.
* Class D inherits the class B and class C **(which is invalid in Java).** Assume that we need to call the display() method by using the object of class D, in this scenario Java compiler does not know which display() method to call. Therefore, it creates ambiguity.

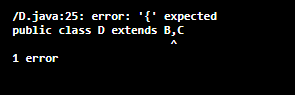
When we compile the above program, it shows the **compiler error**, as we have shown below.



Due to these complications and ambiguities, Java does not support multiple inheritance. It creates problems during various operations, for example, constructor chaining and casting. Hence, it will be good to avoid it for making things simple.

# The Diamond Problem

The diamond problem is a common problem in Java when it comes to inheritance. Inheritance is a very popular property in an object-oriented programming language, such as [C++](https://www.javatpoint.com/cpp-tutorial), [Java](https://www.javatpoint.com/java-tutorial), etc. There are different types of inheritance such as, single, multiple, multi-level, and hybrid inheritance. But remember that **Java does not support the multiple inheritance** because of the diamond problem.



As simple inheritance allows a child class to derive properties from one super-class. for example, if class B inherits properties from only one super-class A, then it is called simple inheritance, and Java supports them.

Multi-level inheritance allows a child class to inherit properties from a class that can inherit properties from some other classes. For example, class C can inherit its property from B class which itself inherits from A class. Java also supports them.

What Java does not allow is multiple inheritance where one class can inherit properties from more than one class. It is known as the **diamond problem**. In the above figure, we find that class D is trying to inherit form class B and class C, that is not allowed in Java.

It is an ambiguity that can rise as a consequence of allowing multiple inheritance. It is a serious problem for other OPPs languages. It is sometimes referred to as the **deadly diamond of death**.

# The Solution of Diamond Problem

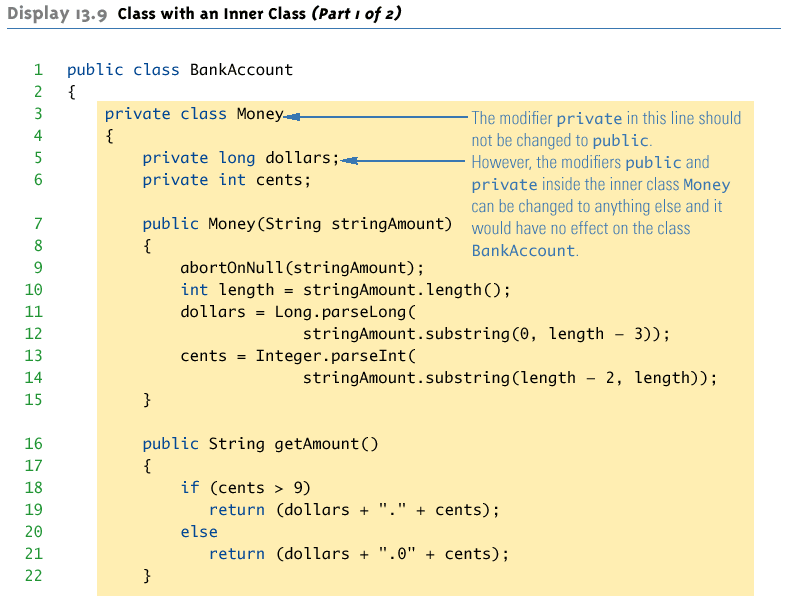
The solution to the diamond problem is **default methods** and **interfaces**. We can achieve multiple inheritance by using these two things.

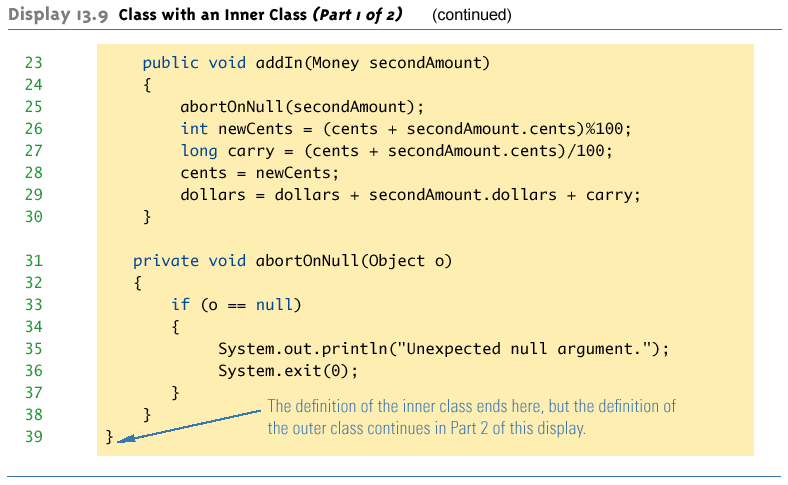
The **default** method is similar to the abstract method. **The only difference is that it is defined inside the interfaces with the default implementation**. We need not to override these methods. Because they are already implementing these interfaces.

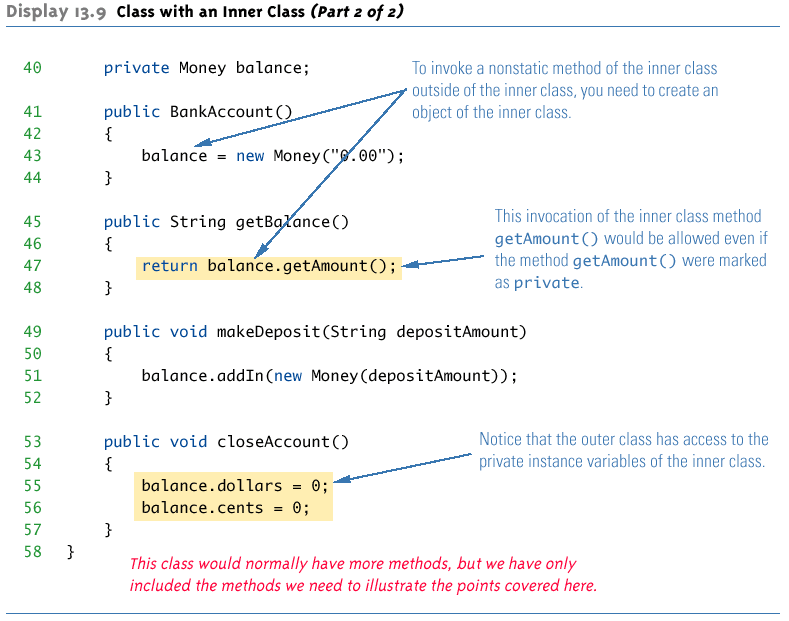
# Inner Classes

* **Inner (or nested) classes are classes defined within other classes:**
  + The class that includes the inner class is called the outer class
* **There are four categories of inner classes in Java:**
  + Inner classes (non-static).
  + Static inner classes.
  + **Local classes (defined inside a block of Java code).**
  + **Anonymous classes (defined inside a block of Java code).**

# Example Program Inner Classes







# Local Class Example:

class LocalClassExample{

private String name = “UoHB";

public void method ( ) {

int j = 20;

final int k = 30;

class Local {

public void test ( ) {

//System.out.println(j); *//Error as j is not final*

System.out.println(k); *//OK k is final*

*//Like an inner class, instance variables of*

*//the enclosing object can be accessed.*

System.out.println ( name ) ;

}

}

Local loc = new Local ( ) ;

loc.test ( ) ;

}

public static void main ( String [ ] args ) {

LocalClassExample obj = new LocalClassExample ( );

obj.method ( ) ;

}

}

# References:

[**https://www.javatpoint.com/what-is-diamond-problem-in-java**](https://www.javatpoint.com/what-is-diamond-problem-in-java)

[**https://www.w3schools.com/java/java\_inner\_classes.asp**](https://www.w3schools.com/java/java_inner_classes.asp)

[**https://www.tutorialspoint.com/java/java\_innerclasses.htm**](https://www.tutorialspoint.com/java/java_innerclasses.htm)

[**https://www.geeksforgeeks.org/inner-class-java/**](https://www.geeksforgeeks.org/inner-class-java/)

[**https://www.javatpoint.com/java-inner-class**](https://www.javatpoint.com/java-inner-class)

[**https://www.geeksforgeeks.org/generics-in-java/**](https://www.geeksforgeeks.org/generics-in-java/)

[**https://www.tutorialspoint.com/java/java\_generics.htm**](https://www.tutorialspoint.com/java/java_generics.htm)

[**https://www.javatpoint.com/generics-in-java**](https://www.javatpoint.com/generics-in-java)

[**https://www.programiz.com/java-programming/generics**](https://www.programiz.com/java-programming/generics)

[**https://docs.oracle.com/javase/tutorial/java/data/autoboxing.html**](https://docs.oracle.com/javase/tutorial/java/data/autoboxing.html)