Order of Constructor invocation in Inheritance

- **If the parent class does not contain any constructor:**
  - In Kotlin, if the parent class has no primary or secondary constructors defined, it means the class is using the default constructor (implicitly provided). In this case, you **do not need to call any constructor explicitly** in the child class.
  - Example:

```
kotlin
Copy code
open class Parent {
    // No explicit constructors
}

class Child : Parent() {
    // No need to call any constructor explicitly
}
```

- **If the parent class contains one primary constructor:**
  - When a parent class has a primary constructor, the child class **must** call the parent's primary constructor. This is done by using the constructor invocation syntax in the class declaration (`Parent()`).
  - Example:

```
kotlin
Copy code
open class Parent(val name: String)

class Child(name: String) : Parent(name) {
    // Must call Parent's primary constructor
}
```

- **If the parent class contains one primary and one secondary constructor:**
  - If the parent class has both a primary and a secondary constructor, the child class **must call the primary constructor** either directly or through one of the parent's secondary constructors that ultimately calls the primary one.
  - Example: directly

```
kotlin
Copy code
open class Parent(val name: String) {
    constructor(name: String, age: Int) : this(name) {
        // secondary constructor
    }
}

class Child(name: String) : Parent(name) {
    // Primary constructor of Parent is called
}
```

- If a parent class has both a primary and a secondary constructor, the child class can call the parent's **secondary constructor**, which will in turn call the **primary constructor**. The secondary constructor must delegate to the primary constructor (using this(...)), so when the child class calls the secondary constructor using super(...), the primary constructor is also called indirectly.
- Example: indirectly

```
kotlin
Copy code
// Parent class with a primary and a secondary constructor
open class Parent(val name: String) {

    // Secondary constructor delegating to the primary constructor
    constructor(name: String, age: Int) : this(name) {
        println("Secondary constructor called with age: $age")
    }
}

// Child class calling the parent's secondary constructor
class Child : Parent {
    constructor(name: String, age: Int) : super(name, age) {
        println("Child constructor called")
    }
}

fun main() {
    val child = Child("Ansh", 25)
}
```

Explanation:

1. **Parent Class:**
   - The **primary constructor** accepts a single argument (`name: String`).
   - The **secondary constructor** takes two arguments (`name: String, age: Int`) and delegates to the primary constructor using `this(name)`.
2. **Child Class:**
   - The child class has a constructor that accepts the same parameters as the parent's secondary constructor.
   - It calls the parent's secondary constructor using `super(name, age)`.
3. **Execution Flow:**
   - When `Child("Ansh", 25)` is instantiated:
     1. The **child class** calls the parent's **secondary constructor** (`super(name, age)`).
     2. The **secondary constructor** of the parent calls the **primary constructor** (`this(name)`).
     3. The **primary constructor** of the parent is executed first.
     4. After the primary constructor finishes, the **secondary constructor**'s additional logic is executed.
     5. Finally, the control returns to the **child class** constructor, and its logic is executed.

- **If the parent class contains only a secondary constructor:**

  - If the parent class has only a secondary constructor, the child class **must call this secondary constructor** explicitly, since there is no primary constructor to rely on.
  - Example:

```
kotlin
Copy code
open class Parent {
    constructor(name: String) {
        // secondary constructor
    }
}

class Child : Parent {
    constructor(name: String) : super(name) {
        // Must call Parent's secondary constructor
    }
}
```

Points to remember-

- This is a keyword is a type of reference variable that refer to object of current class, and can be used to refer current instance varible of the class Ex- this.a, this.b
- This is also used in constructor chaining from secondary constructor of a class.
- Super keyword is a type of refernce variable that refers to object of parent class, and can be used to refer immediate parent class instance variable Ex- super.p_a, super.p_b
- Super is also used to invoke parent class constructor explicitely
- The first stament of any constructor can either be this() or super() calling statement but not both-

```
// Base class with primary constructor

open class A(var a: String, var b: Int) {

   constructor(A_y: Int, A_x: String) : this(A_x, A_y) { println("Secondary
constructor from class A") }

   fun displayA() { println("a= $a") println("b= $b") }

}

class B(var c: Double) :A {

   //Any secondary constructor delegate to primary constructor

constructor(A_y: Int, A_x: String, B_z: Double ): super(A_y, A_x), this(B_z){ }

}
```

Gives ERROR

The error in your code comes from an incorrect use of constructor delegation in the child class `B`. In Kotlin, a secondary constructor in a child class must first delegate to primary constructor of the same class then a constructor of the parent class, but not both simultaneously as you're trying to do.

The correct way is to delegate first to the primary constructor of `B`, and then to the parent class constructor using `super`. Here's how you can fix it:

Fixed Code

```
kotlin
Copy code
// Base class with primary constructor
open class A(var a: String, var b: Int) {

    constructor(A_y: Int, A_x: String) : this(A_x, A_y) {
        println("Secondary constructor from class A")
    }

    fun displayA() {
        println("a= $a")
        println("b= $b")
    }
}

// Child class B
class B(var c: Double) : A {

    // Any secondary constructor delegate to primary constructor first, then to the
parent
    constructor(A_y: Int, A_x: String, B_z: Double) : this(B_z) {
        println("Secondary constructor from class B")
        // Call the parent class constructor
        super.a = A_x
        super.b = A_y
    }

    fun displayB() {
        println("Class A a is: $a")
        println("Class A b is: $b")
        println("Class B c is: $c")
    }
}

fun main() {
    // Create an object of B with the secondary constructor
    val objB = B(10, "Hello", 20.5)
    objB.displayA()
    objB.displayB()
}
```

Changes made:

1. In the secondary constructor of `B`, I first delegate to the primary constructor using `this(B_z)`, which initializes the `c` property.
2. Then, I set the values for `a` and `b` using `super.a = A_x` and `super.b = A_y` to manually initialize the parent class properties.
3. This ensures that `A`'s primary constructor is used indirectly since the properties `a` and `b` are being set correctly, and the delegation rules are followed.

This way, you will avoid the delegation error and achieve the desired behavior of calling both the parent and child constructors correctly.

® Ansh Vikalp