Q) Is constructor overloading possible by Only primary constructor and no Secondary Constructor?

Ans- In Kotlin, **constructor overloading** is possible using **secondary constructors or a mixture of one primary constructor and multiple secondary constructors**, not via the primary constructor alone. Because as we already know that a class can have only one primary constructor to initialize properties directly. So having only single primary constructor how overloading can be done. You can have multiple secondary constructors that allow different ways to create objects.

So overloading can be done in either below ways:-

- One Primary Constructor + One or more Secondary constructors
- Two or more Secondary Constructors
- Primary constructor with init block + One or more Secondary constructors

Constructor Overloading and Chaining using this()

Constructor overloading means two more constructor present in same class but either differ in no of parameter Or their types or order. Whereas constructor chaining means when one constructor invokes another constructor of same class that has been overloaded.

- A secondary constructor can call primary constructor
- A secondary constructor can another secondary constructor
- If the class has a primary constructor, all secondary constructors must either directly or indirectly delegate "chain" to it using the this() keyword.

1. Constructor Chaining with Primary and Secondary Constructors

When you have both a **primary constructor** and one or more **secondary constructors**, the secondary constructors must delegate to the primary constructor using `this()`. This ensures that the primary constructor is always called to initialize the properties defined in the class.

Example:

```
//Primary Constructor
class Person(val name: String, var age: Int) {

    // Secondary constructor #1 chaining to the primary constructor
    constructor() : this("Unknown", 0) { // 0 parameter
        println("Default constructor called")
    }

// Secondary Overloaded constructor #2 chaining to the primary constructor
    constructor(name: String) : this(name, 0) { // 1 parameter
        println("Constructor with name called")
    }
}
```

In this example:

- The primary constructor takes two parameters (`name` and `age`).
- The first secondary constructor takes no arguments and calls the primary constructor with default values ("Unknown", 0).
- The second secondary constructor takes just `name` and passes a default value for `age` (`0`), then calls the primary constructor.
- The `this()` keyword is used in secondary constructors to **chain** them to the primary constructor.

When an object is created using any constructor, the chain ensures that the primary constructor is always called to initialize the class:

```
val person1 = Person()              // "Default Secondary constructor called"
val person2 = Person("John")        // "Constructor with name called"
val person3 = Person("John", 25)    // No secondary constructor is called
```

2. Constructor Chaining Between Secondary Constructors Only

Secondary constructors can also chain to other **secondary constructors**. However, eventually, they must always delegate to the **primary constructor**.

Example:

```
class Rectangle {
    var length: Int
    var breadth: Int

    //Secondary constructor #1
    constructor(length: Int, breadth: Int) {
        this.length = length
        this.breadth = breadth
        println("Primary constructor called")
    }

    // Secondary constructor #2 chaining to another secondary constructor #1
    constructor(side: Int) : this(side, side) {
        println("Square constructor called")
    }

    // Another secondary constructor #3 calling secondary constructor #2
    constructor() : this(1) {
        println("Default constructor called")
    }
}
```

Here's what happens:

- The primary constructor takes two parameters (`length` and `breadth`).
- The first secondary constructor takes one parameter (`side`) and chains to the primary constructor with the same value for both `length` and `breadth`.
- The second secondary constructor takes no arguments and chains to the first secondary constructor with a default value (`side = 1`).

When an object is created:

```
val rect1 = Rectangle()          // Chained: Default -> Square -> Primary
val rect2 = Rectangle(5)         // Chained: Square -> Primary
val rect3 = Rectangle(4, 6)      // Calls only the primary constructor
```

3. Constructor Chaining with `init` Blocks

If you use an `init` block in a class with constructor chaining, it gets called each time the primary constructor is invoked. The `init` block runs before the secondary constructor body is executed.

Example with `init` Block:

```
class Employee(val name: String, val salary: Int) {

    // Init block
    init {
        println("Employee initialized with name: $name and salary: $salary")
    }

    // Secondary constructor chaining to the primary constructor
    constructor() : this("Unknown", 0) { // Default or zero arg constructor
        println("Default employee created")
    }
}
```

Here:

- The `init` block runs after the primary constructor is called.
- The secondary constructor first calls the primary constructor (using `this("Unknown", 0)`), which in turn runs the `init` block.

Output when creating objects:

```
// "Employee initialized with name: Unknown and salary: 0", "Default employee created"
val employee1 = Employee()

// "Employee initialized with name: John and salary: 5000"
val employee2 = Employee("John", 5000)
```

Tip:

When designing constructors, use default parameter values where possible to avoid needing multiple secondary constructors, simplifying your code while still providing flexible ways to initialize objects.

Ideal Sequence of Constructor Overloading

Let's look at another example using a `Shape` class with secondary constructor overloading to initialize different shapes like circles and rectangles.

```
class Shape(type: String) { //Primary constructor initlaizes values in init block
    var type: String
    var radius: Double = 0.0
    var length: Int = 0
    var width: Int = 0

  // Primary constructor for a generic shape
init{
        println("Primary constructor called: Shape type is $type")
        this.type= type
}


    // Overloaded secondary constructor for circles 1 param type Double
    constructor(radius: Double) : this("Circle") {
        this.radius = radius
        println("Circle constructor called with radius: $radius")
    }

    // Overloaded secondary constructor for rectangles 2 param type Int,Int
    constructor(length: Int, width: Int) : this("Rectangle") {
        this.length = length
        this.width = width
        println("Rectangle constructor called with length: $length, width: $width")
    }

    // Another secondary constructor with default values for squares 1 param type Int
    constructor(side: Int) : this(side, side) {
        println("Square constructor called with side: $side")
    }
}
```

Explanation:

- **Primary Constructor**: `constructor(type: String)` sets the type of the shape.
- **Secondary Constructor #1**: Initializes a circle with a `radius` and sets the shape type to `"Circle"`.
- **Secondary Constructor #2**: Initializes a rectangle with `length` and `width` and sets the shape type to `"Rectangle"`.
- **Secondary Constructor #3**: Initializes a square by chaining to the rectangle constructor, passing the same value for both `length` and `width`.

Usage:

```
val circle = Shape(5.7)
// Output: "Primary constructor called: Shape type is Circle", "Circle constructor called
with radius: 5"

val rectangle = Shape(10, 20)
// Output: "Primary constructor called: Shape type is Rectangle", "Rectangle constructor
called with length: 10, width: 20"

val square = Shape(10)
// Output: "Primary constructor called: Shape type is Rectangle", "Rectangle constructor
called with length: 10, Width: 10, "Square constructor called with side: 10"
```