Data Class

In Kotlin, a **data class** is a special class designed to hold data. It automatically provides several useful functions without requiring much boilerplate code, such as `equals()`, `hashCode()`, `toString()`, and `copy()`.

Here's a breakdown of the key features of **data classes**:

1. Primary Constructor

A data class must have at least one parameter in its primary constructor, and these parameters should be marked as `val` or `var` so that they can be used for property generation.

2. Automatically Generated Functions

When you declare a class as a data class, Kotlin automatically generates the following overridden methods:

- **`equals()`**: Compares two objects for equality based on the property values.
- **`hashCode()`**: Generates a hash code based on the property values.
- **`toString()`**: Returns a string representation of the object with state.
- **`copy()`**: Allows you to create a copy of an object, optionally changing some of its properties.

3. `componentN()` Functions

Each property declared in the primary constructor of a data class has a corresponding `componentN()` function, where `N` is the index of the property. This enables destructuring declarations.

4. Custom Functions

Like a normal class you can define your own methods inside data class more precisely you can do anything that you can do with your normal class.

Example of a Data Class:

```
data class User(val name: String, val age: Int)
```

This creates a class `User` with two properties: `name` and `age`.

Generated Methods:

1. **`toString()`**:

   ```
   val user = User("Ansh", 25)
   println(user) // Output: User(name=Ansh, age=25)
   ```

2. **`equals()`**:

   ```
   val user1 = User("Ansh", 25)
   val user2 = User("Ansh", 25)
   println(user1 == user2) // Output: true prop values are checked
   ```

3. **`copy()`**:

```
val user1 = User("Ansh", 25)
val user2 = user1.copy(age = 26)
println(user2) // Output: User(name=Ansh, age=26)
```

4. **Destructuring**: Fully

```
val user = User("Ansh", 25)
val (name, age) = user
println(name) // Output: Ansh
println(age)  // Output: 25
```

5. **Destructuring**: Partially

   You don't have to destructure all properties. Use underscores _ for the properties you want to ignore.

```
val user = User("Kishu", 15)
val (name,_) = user
println(name) // Output: Kishu
```

   A **destructuring declaration** allows you to unpack an object into multiple variables simultaneously. Instead of accessing each property individually, you can extract them in a single statement.

How Do They Work?

Under the hood, destructuring declarations rely on a series of `componentN()` functions (where `N` is the position of the property) defined in the class. For example, `component1()`, `component2()`, etc. **Data classes** in Kotlin automatically generate these `componentN()` functions based on their properties, enabling destructuring out of the box.

Rules for Data Classes:

- The primary constructor must have at least one parameter.
- All primary constructor parameters need to be marked `val` or `var`.
- Data classes cannot be abstract, open, sealed, or inner.
- **Data classes can extend other classes**, including regular classes or abstract classes but no other data class. If a data class extends another class, the primary constructor of the data class must **delegate required parameters** to the parent class primary constructor (same concept with normal class in inheritance).

Data classes are especially useful for modeling simple data containers, making code more concise and readable.

Q) Does a normal Concrete class also have these utility methods, if yes then why don't they give same output. Can we override them?

Ans- In Kotlin, **normal classes** (non-data classes) **do not automatically** have the implementations for methods like `equals()`, `hashCode()`, `toString()`, or `copy()` generated for them. If you define a **normal class**, you would need to manually override these methods if you want behavior similar to that of a data class.

Key Differences Between Data Classes and Normal Classes:

1. **`equals()`**:
   - **Data Class**: Automatically compares the values of properties.
   - **Normal Class**: By default, compares object references (i.e., checks if two variables point to the same object).

```
class NormalUser(val name: String, val age: Int)

data class DataUser(val name: String, val age: Int)

val normal1 = NormalUser("Ansh", 25)
val normal2 = NormalUser("Ansh", 25)
println(normal1 == normal2) // Output: false (reference comparison)

val data1 = DataUser("Ansh", 25)
val data2 = DataUser("Ansh", 25)
println(data1 == data2) // Output: true (value comparison)
```

2. **`hashCode()`**:
   - **Data Class**: Automatically computes a hash code based on the properties' values.
   - **Normal Class**: If not overridden, it uses the default implementation from `Any`, which is based on the object's memory address.

```
println(data1.hashCode())   // Consistent with the property values
println(normal1.hashCode()) // Memory address-based, different even if values are
the same
```

3. **`toString()`**:
   - **Data Class**: Automatically generates a readable string representing the object with its property values.
   - **Normal Class**: Uses the default implementation from `Any`, which typically returns the class name followed by the memory address.

```
println(data1.toString())   // Output: DataUser(name=Ansh, age=25)
println(normal1.toString()) // Output: NormalUser@HASHCODE
```

4. **`copy()`**:
   - **Data Class**: Automatically provides a `copy()` method to make a shallow copy of an object, with the option to modify some properties.
   - **Normal Class**: There is no `copy()` method by default, and you'd have to manually implement a method if you wanted to clone objects.

```
val copyDataUser = data1.copy(age = 26)
println(copyDataUser) // Output: DataUser(name=Ansh, age=26)

// For normal class, you'd need to manually implement a cloning method
```

5. **Destructuring Declarations**:
    - **Data Class**: Automatically supports destructuring declarations (using `componentN()` functions).
    - **Normal Class**: Does not support destructuring unless you manually implement the `componentN()` functions.

```
val (name, age) = data1 // Works because data classes have componentN functions.
// Destructuring is not supported for normal classes without custom implementation.
```

Summary

In **normal classes**, if you want the same functionality as a **data class**, you would have to manually implement methods like `equals()`, `hashCode()`, `toString()`, and `copy()`. In contrast, **data classes** provide these methods out of the box, making them very convenient for classes that primarily hold data.