

Data Classes

Data Classes



Classes whose main
purpose is to hold data.

Data Classes

We frequently create classes whose main purpose is to hold data. In such a class some standard functionality and utility functions are often mechanically derivable from the data. In Kotlin, this is called a *data class* and is marked as `data`:

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

The compiler automatically derives the following members from all properties declared in the primary constructor:

- `equals()` / `hashCode()` pair;
- `toString()` of the form `"User(name=John, age=42)"`;
- [componentN\(.\) functions](#) corresponding to the properties in their order of declaration;
- `copy()` function (see below).

To ensure consistency and meaningful behavior of the generated code, data classes have to fulfill the following requirements:

- The primary constructor needs to have at least one parameter;
- All primary constructor parameters need to be marked as `val` or `var`;
- Data classes cannot be abstract, open, sealed or inner;

Properties Declared in the Class Body

Note that the compiler only uses the properties defined inside the primary constructor for the automatically generated functions. To exclude a property from the generated implementations, declare it inside the class body:

```
data class Person(val name: String) {  
    var age: Int = 0  
}
```

Only the property `name` will be used inside the `toString()`, `equals()`, `hashCode()`, and `copy()` implementations, and there will only be one component function `component1()`. While two `Person` objects can have different ages, they will be treated as equal.

```
val person1 = Person("John")  
val person2 = Person("John")  
person1.age = 10  
person2.age = 20
```

Copying

It's often the case that we need to copy an object altering *some* of its properties, but keeping the rest unchanged. This is what `copy()` function is generated for. For the `User` class above, its implementation would be as follows:

```
fun copy(name: String = this.name, age: Int = this.age) = User(name, age)
```

This allows us to write:

```
val jack = User(name = "Jack", age = 1)  
val olderJack = jack.copy(age = 2)
```

Data Classes and Destructuring Declarations

Component functions generated for data classes enable their use in [destructuring declarations](#):

```
val jane = User("Jane", 35)
val (name, age) = jane
println("$name, $age years of age") // prints "Jane, 35 years of age"
```

Standard Data Classes

The standard library provides `Pair` and `Triple`. In most cases, though, named data classes are a better design choice, because they make the code more readable by providing meaningful names for properties.