Sealed Classes

We'll cover the following How sealed classes work in Kotlin Instantiating objects

In Kotlin, on one extreme we have final classes—that is, classes not marked as open—which can't have any derived classes. On the other extreme we have open and abstract classes, and there's no telling which class may inherit from them. It'll be nice to have a middle ground for a class to serve as a base to only a few classes, which the author of the class specifies.

How sealed classes work in Kotlin

Kotlin's sealed classes are open for extension by other classes defined in the same file but closed—that is, final or not open—for any other classes.

Here's a sealed class Card, along with a few classes that inherit from it, all within the same file Card.kt.

```
sealed class Card(val suit: String)

class Ace(suit: String) : Card(suit)

class King(suit: String) : Card(suit) {
   override fun toString() = "King of $suit"
}

class Queen(suit: String) : Card(suit) {
   override fun toString() = "Queen of $suit"
}

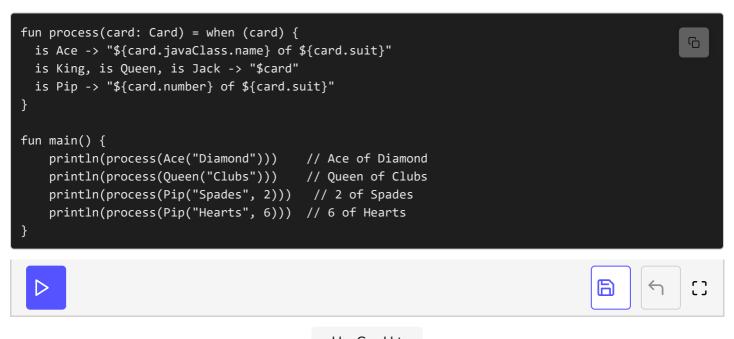
class Jack(suit: String) : Card(suit) {
   override fun toString() = "Jack of $suit"
}

class Pip(suit: String, val number: Int) : Card(suit) {
   init {
     if (number < 2 || number > 10) {
        throw RuntimeException("Pip has to be between 2 and 10")
     }
}
```

The constructors of sealed classes aren't marked private, but they're considered private. The derived classes of a sealed class, like Ace, for example, may have any number of instances and may have state—that is, their own properties—and methods. In addition to deriving classes, you may also derive singleton objects from sealed classes. In this example, there can be only five derived classes of Card. Any attempt to inherit from Card by any classes written in any other files will fail compilation.

Instantiating objects

Since the constructors of sealed classes are considered to be private, we can't instantiate an object of these classes. However, we can create objects of classes that inherit from sealed classes, assuming their constructors aren't marked private explicitly. Let's create instances of the derived classes of Card:



Use Card.kt

Creating instances of the derived classes of a sealed class is straightforward. However, when used within a when expression, you should not write the else path. If there's a path for all the derived types of a sealed class in when, then placing an else will result in a warning for the path that will never be taken. If there's no path for any of the derived class, the compiler will insist that you add a path for the missing cases or that you add an else path. Even if the compiler suggests adding an else path, do not add it, and don't ignore any warnings the

added, then instead of getting a compilation error to alert that the new case isn't handled properly, the program may execute an unintended piece of code in the else path.

The derived classes of a sealed class may have any number of instances. A special case of this is enum, which restricts the number of instances to one for each subclass.

The next lesson will discuss enums.