Android: A quick Kotlin Eksperience

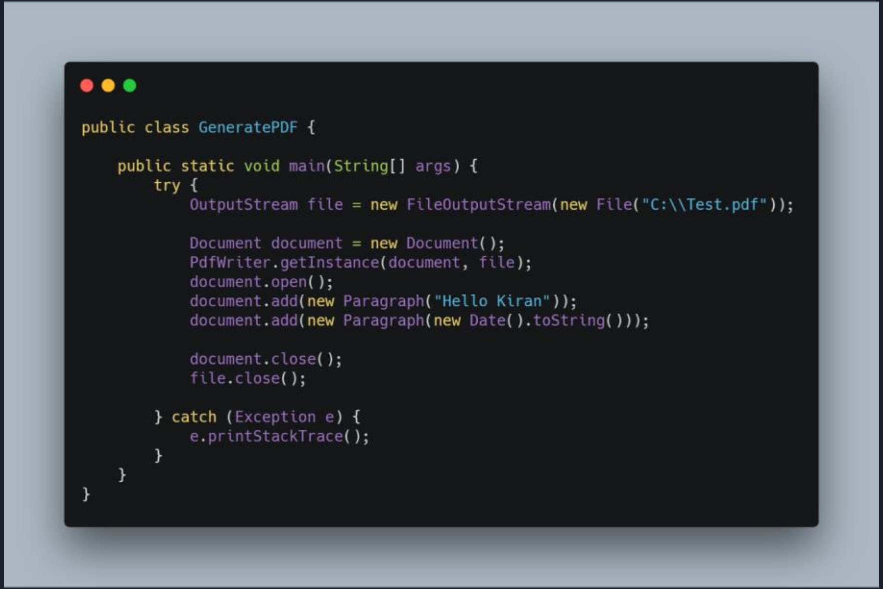
Benefits of using Kotlin

1. No Semi-Colon
2. Default Values
3. Extension Functions
4. Data Classes
5. Parcelize
6. Unit Functions
7. Null Operations

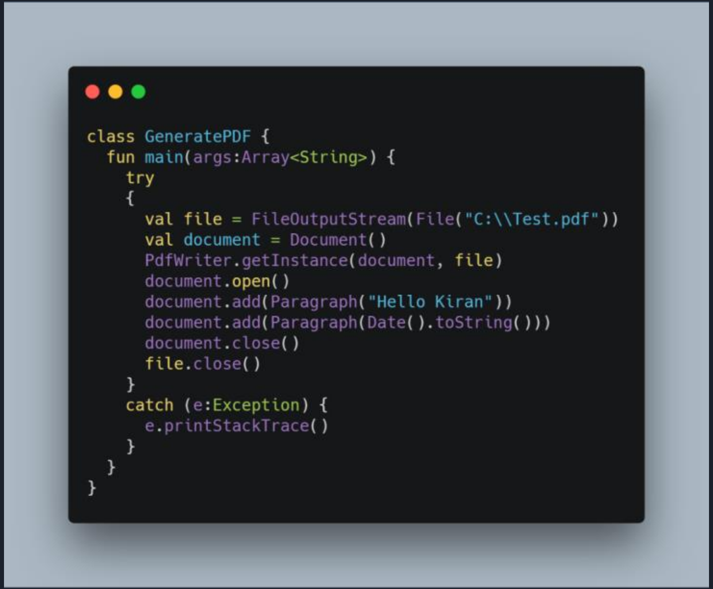
No Semicolons

This is probably the thing smallest things but has a huge effect on readability of code.

A couple things to note is that there’s no new keyword and you don’t need to declare the type

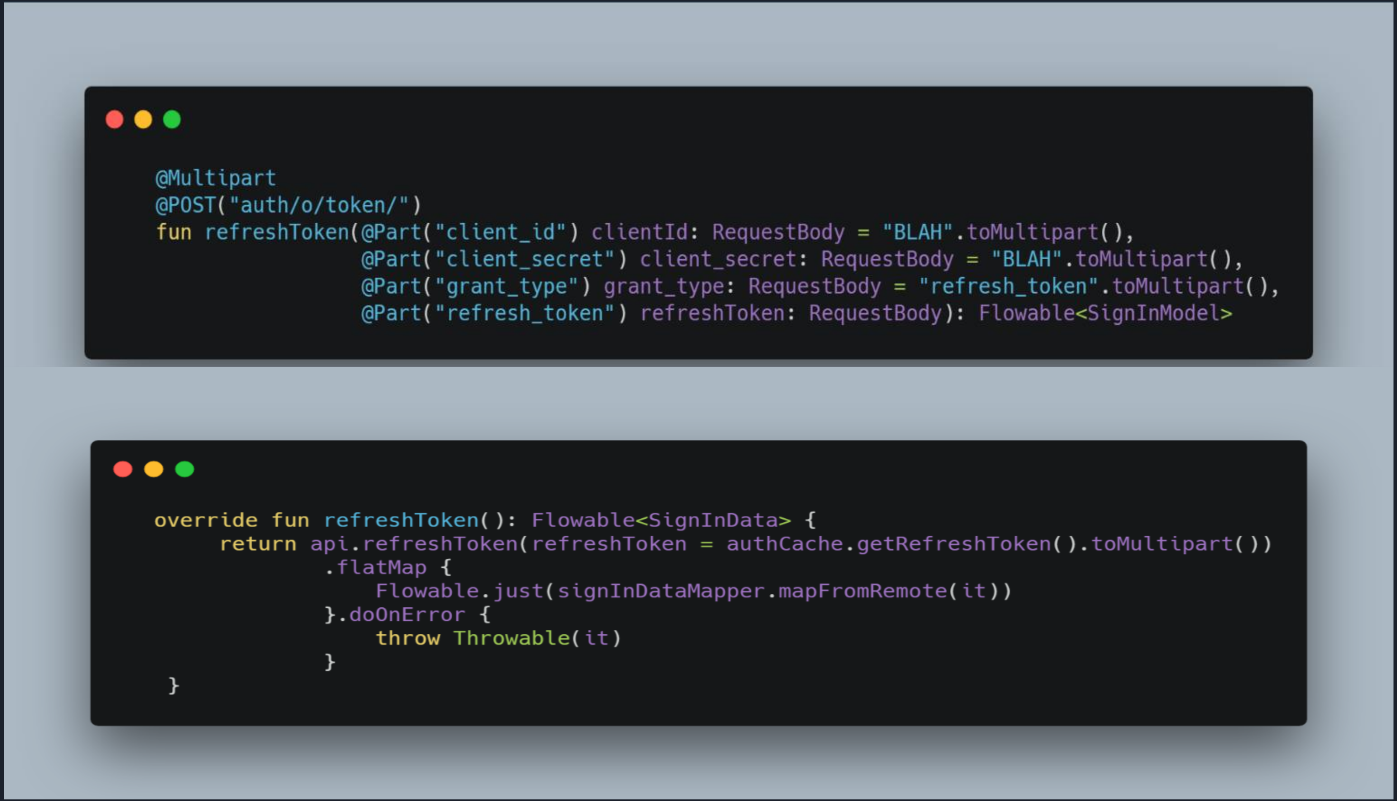


```
public class GeneratePDF {

    public static void main(String[] args) {
        try {
            OutputStream file = new FileOutputStream(new File("C:\\Test.pdf"));

            Document document = new Document();
            PdfWriter.getInstance(document, file);
            document.open();
            document.add(new Paragraph("Hello Kiran"));
            document.add(new Paragraph(new Date().toString()));

            document.close();
            file.close();

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```



```
class GeneratePDF {
  fun main(args:Array<String>) {
    try
    {
      val file = FileOutputStream(File("C:\\Test.pdf"))
      val document = Document()
      PdfWriter.getInstance(document, file)
      document.open()
      document.add(Paragraph("Hello Kiran"))
      document.add(Paragraph(Date().toString()))
      document.close()
      file.close()
    }
    catch (e:Exception) {
      e.printStackTrace()
    }
  }
}
```

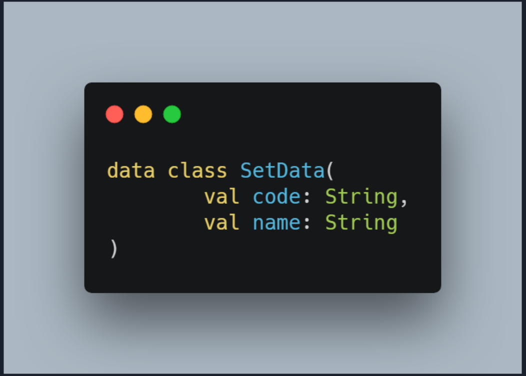
Default Values



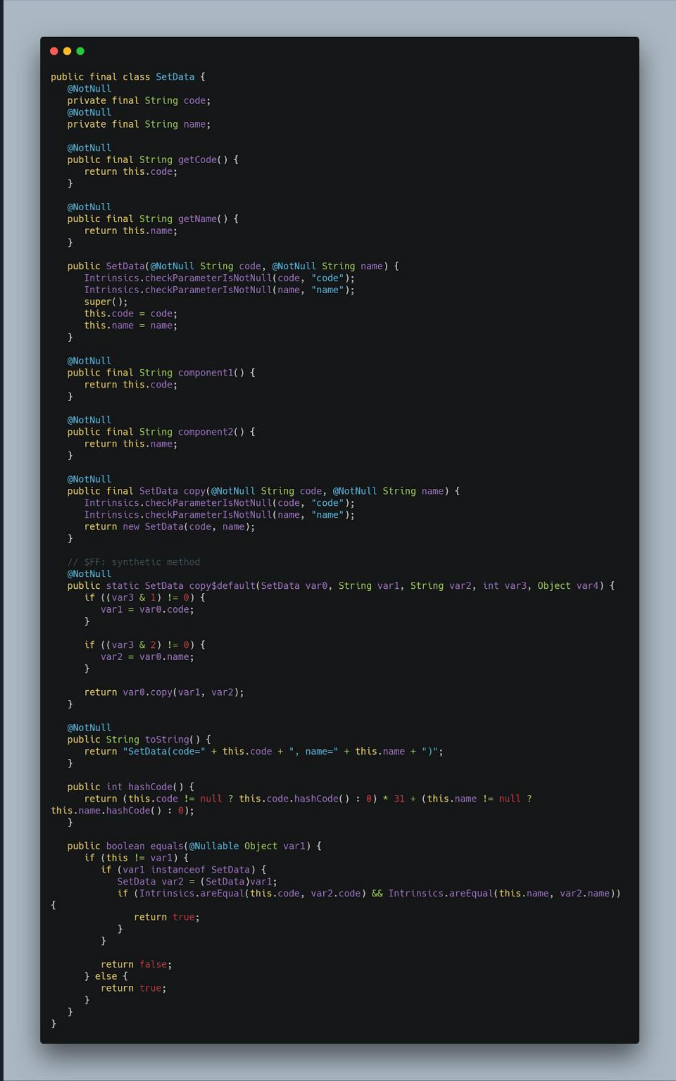
```
@Multipart
@POST("auth/o/token/")
fun refreshToken(@Part("client_id") clientId: RequestBody = "BLAH".toMultipart(),
                 @Part("client_secret") client_secret: RequestBody = "BLAH".toMultipart(),
                 @Part("grant_type") grant_type: RequestBody = "refresh_token".toMultipart(),
                 @Part("refresh_token") refreshToken: RequestBody): Flowable<SignInModel>
```

```
override fun refreshToken(): Flowable<SignInData> {
     return api.refreshToken(refreshToken = authCache.getRefreshToken().toMultipart())
               .flatMap {
                    Flowable.just(signInDataMapper.mapFromRemote(it))
               }.doOnError {
                    throw Throwable(it)
               }
 }
```

Data Classes - Code



```kotlin
data class SetData(
        val code: String,
        val name: String
)
```

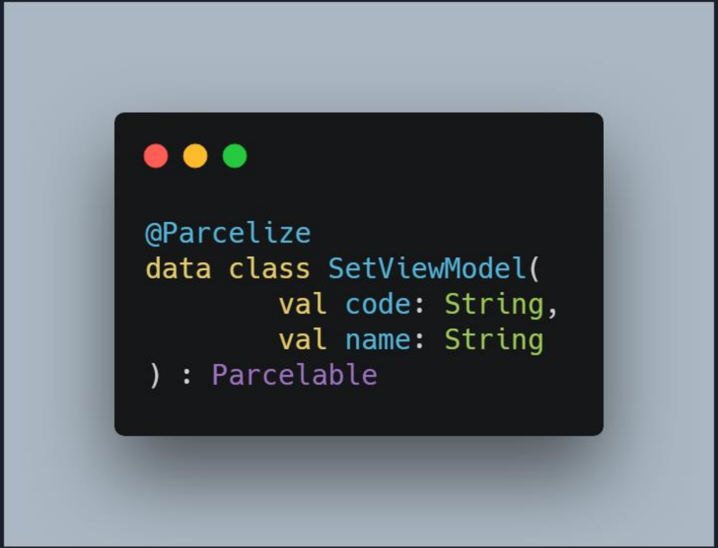


```java
public final class SetData {
   @NotNull
   private final String code;
   @NotNull
   private final String name;

   @NotNull
   public final String getCode() {
      return this.code;
   }

   @NotNull
   public final String getName() {
      return this.name;
   }

   public SetData(@NotNull String code, @NotNull String name) {
      Intrinsics.checkParameterIsNotNull(code, "code");
      Intrinsics.checkParameterIsNotNull(name, "name");
      super();
      this.code = code;
      this.name = name;
   }

   @NotNull
   public final String component1() {
      return this.code;
   }

   @NotNull
   public final String component2() {
      return this.name;
   }

   @NotNull
   public final SetData copy(@NotNull String code, @NotNull String name) {
      Intrinsics.checkParameterIsNotNull(code, "code");
      Intrinsics.checkParameterIsNotNull(name, "name");
      return new SetData(code, name);
   }

   // $FF: synthetic method
   @NotNull
   public static SetData copy$default(SetData var0, String var1, String var2, int var3, Object var4) {
      if ((var3 & 1) != 0) {
         var1 = var0.code;
      }

      if ((var3 & 2) != 0) {
         var2 = var0.name;
      }

      return var0.copy(var1, var2);
   }

   @NotNull
   public String toString() {
      return "SetData(code=" + this.code + ", name=" + this.name + ")";
   }

   public int hashCode() {
      return (this.code != null ? this.code.hashCode() : 0) * 31 + (this.name != null ?
this.name.hashCode() : 0);
   }

   public boolean equals(@Nullable Object var1) {
      if (this != var1) {
         if (var1 instanceof SetData) {
            SetData var2 = (SetData)var1;
            if (Intrinsics.areEqual(this.code, var2.code) && Intrinsics.areEqual(this.name, var2.name))
{
               return true;
            }
         }

         return false;
      } else {
         return true;
      }
   }
}
```

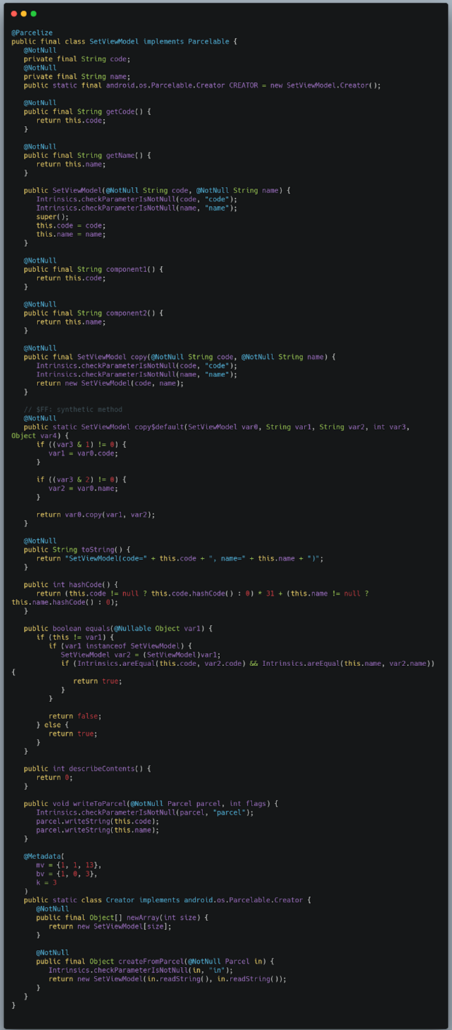
Kotlin data class benefits

1. The properties declared in the constructor: this technically is not exclusive to a data class, but it avoids all the boilerplate of getters and setters, in addition to the constructor
2. Provides the equals() & hashCode() functions
3. Provides a copy() method, very useful when we use immutable objects.

Parcelize - Code



```kotlin
@Parcelize
data class SetViewModel(
        val code: String,
        val name: String
) : Parcelable
```

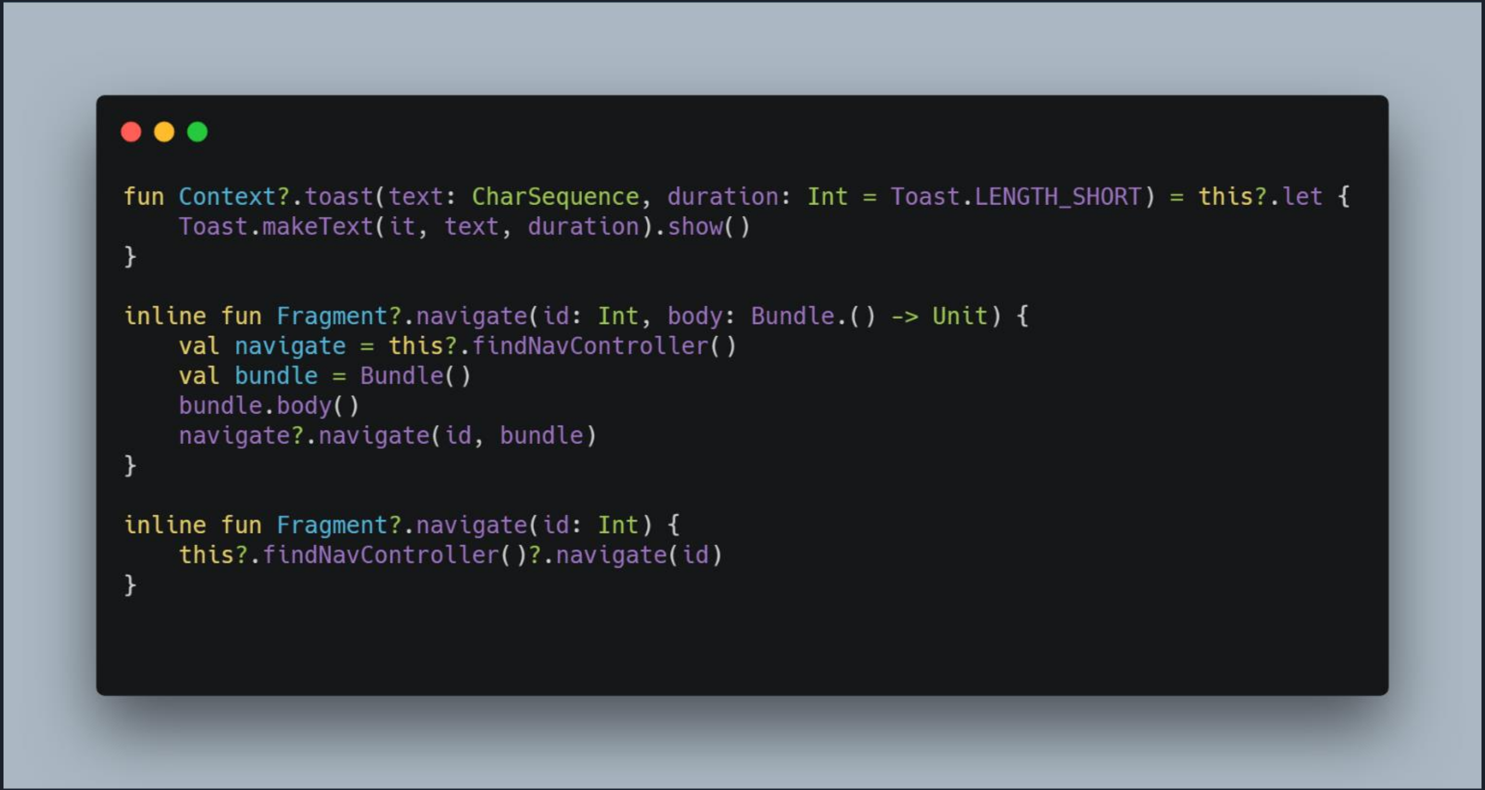


```java
@Parcelize
public final class SetViewModel implements Parcelable {
   @NotNull
   private final String code;
   @NotNull
   private final String name;
   public static final android.os.Parcelable.Creator CREATOR = new SetViewModel.Creator();

   @NotNull
   public final String getCode() {
      return this.code;
   }

   @NotNull
   public final String getName() {
      return this.name;
   }

   public SetViewModel(@NotNull String code, @NotNull String name) {
      Intrinsics.checkParameterIsNotNull(code, "code");
      Intrinsics.checkParameterIsNotNull(name, "name");
      super();
      this.code = code;
      this.name = name;
   }

   @NotNull
   public final String component1() {
      return this.code;
   }

   @NotNull
   public final String component2() {
      return this.name;
   }

   @NotNull
   public final SetViewModel copy(@NotNull String code, @NotNull String name) {
      Intrinsics.checkParameterIsNotNull(code, "code");
      Intrinsics.checkParameterIsNotNull(name, "name");
      return new SetViewModel(code, name);
   }

   // $FF: synthetic method
   @NotNull
   public static SetViewModel copy$default(SetViewModel var0, String var1, String var2, int var3,
Object var4) {
      if ((var3 & 1) != 0) {
         var1 = var0.code;
      }

      if ((var3 & 2) != 0) {
         var2 = var0.name;
      }

      return var0.copy(var1, var2);
   }

   @NotNull
   public String toString() {
      return "SetViewModel(code=" + this.code + ", name=" + this.name + ")";
   }

   public int hashCode() {
      return (this.code != null ? this.code.hashCode() : 0) * 31 + (this.name != null ?
this.name.hashCode() : 0);
   }

   public boolean equals(@Nullable Object var1) {
      if (this != var1) {
         if (var1 instanceof SetViewModel) {
            SetViewModel var2 = (SetViewModel)var1;
            if (Intrinsics.areEqual(this.code, var2.code) && Intrinsics.areEqual(this.name, var2.name))
{
               return true;
            }
         }

         return false;
      } else {
         return true;
      }
   }

   public int describeContents() {
      return 0;
   }

   public void writeToParcel(@NotNull Parcel parcel, int flags) {
      Intrinsics.checkParameterIsNotNull(parcel, "parcel");
      parcel.writeString(this.code);
      parcel.writeString(this.name);
   }

   @Metadata(
      mv = {1, 1, 13},
      bv = {1, 0, 3},
      k = 3
   )
   public static class Creator implements android.os.Parcelable.Creator {
      @NotNull
      public final Object[] newArray(int size) {
         return new SetViewModel[size];
      }

      @NotNull
      public final Object createFromParcel(@NotNull Parcel in) {
         Intrinsics.checkParameterIsNotNull(in, "in");
         return new SetViewModel(in.readString(), in.readString());
      }
   }
}
```

Parcelize

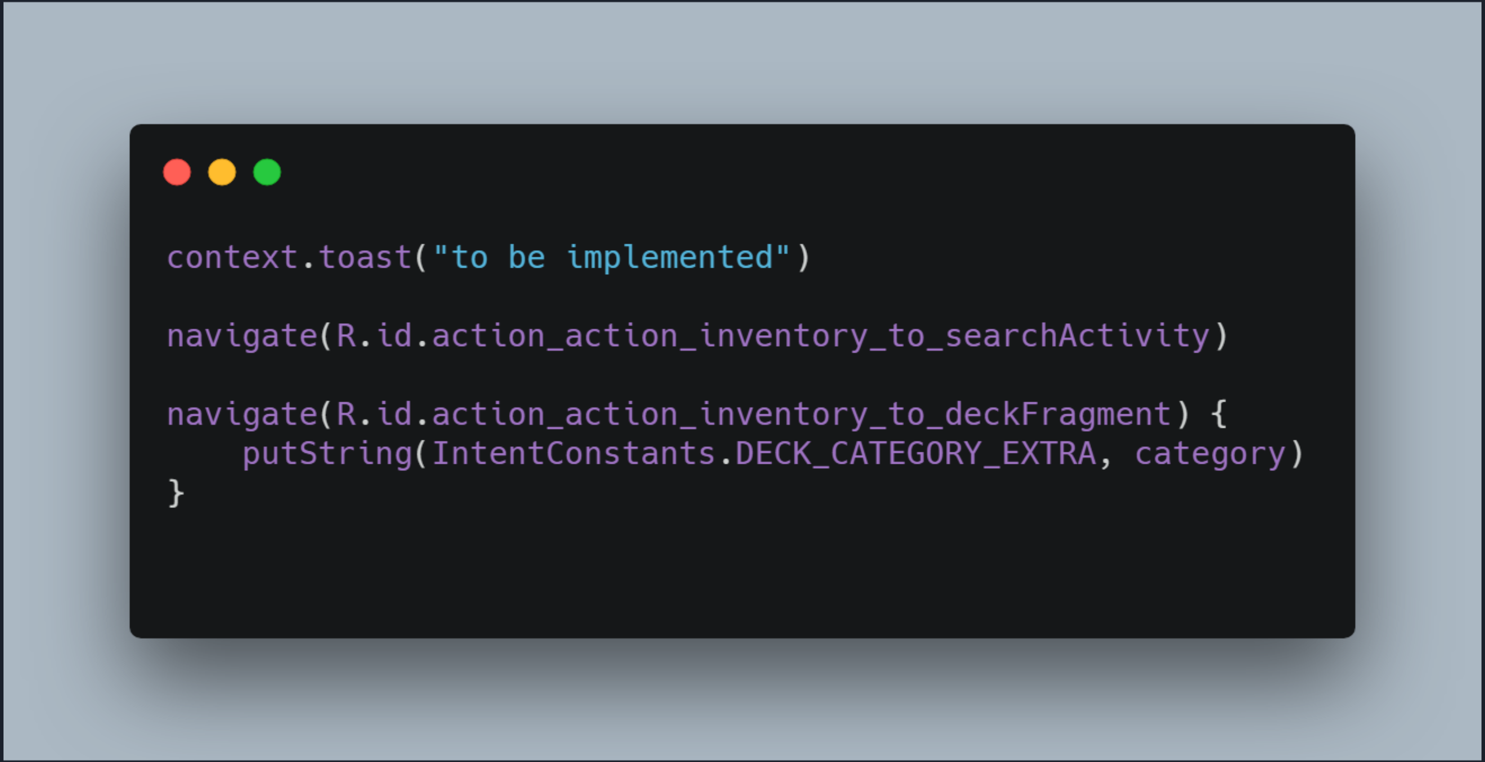
The old implementation of Parcelize, required you to write a **writeToParcelize** class and create a new **Creator** class and was generally more work than it should have been. The more parameters you had in the Model Class the longer your code would inevitably become

Extension Functions - Code



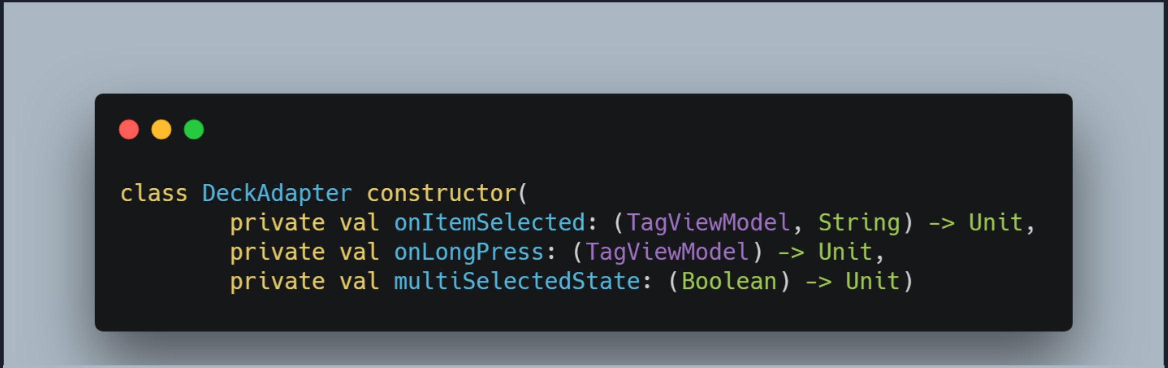
```kotlin
fun Context?.toast(text: CharSequence, duration: Int = Toast.LENGTH_SHORT) = this?.let {
    Toast.makeText(it, text, duration).show()
}

inline fun Fragment?.navigate(id: Int, body: Bundle.() -> Unit) {
    val navigate = this?.findNavController()
    val bundle = Bundle()
    bundle.body()
    navigate?.navigate(id, bundle)
}

inline fun Fragment?.navigate(id: Int) {
    this?.findNavController()?.navigate(id)
}
```

Extension Function Usage

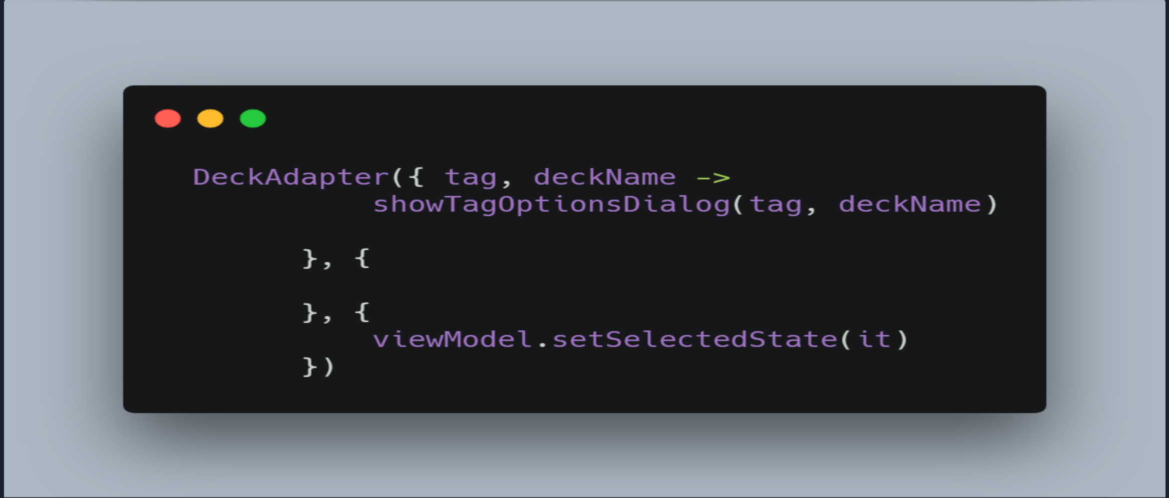


```kotlin
context.toast("to be implemented")

navigate(R.id.action_action_inventory_to_searchActivity)

navigate(R.id.action_action_inventory_to_deckFragment) {
    putString(IntentConstants.DECK_CATEGORY_EXTRA, category)
}
```

Unit functions

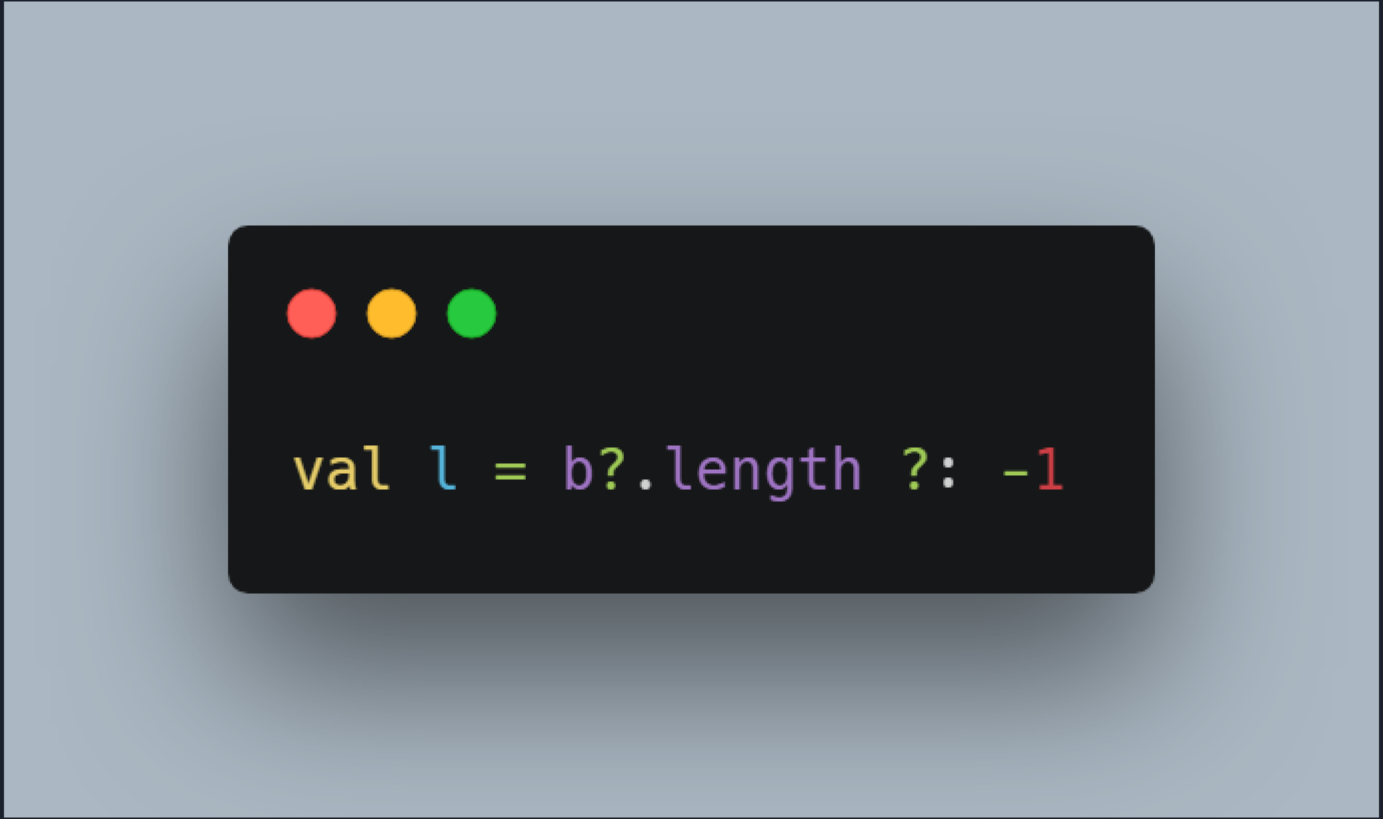


```
class DeckAdapter constructor(
        private val onItemSelected: (TagViewModel, String) -> Unit,
        private val onLongPress: (TagViewModel) -> Unit,
        private val multiSelectedState: (Boolean) -> Unit)
```



```
DeckAdapter({ tag, deckName ->
            showTagOptionsDialog(tag, deckName)

        }, {

        }, {
            viewModel.setSelectedState(it)
        })
```

Null Operators



```
val l = b?.length ?: -1
```