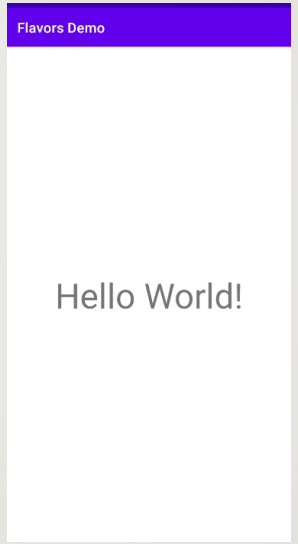
ANDROID FLAVORS

Create different versions of your app from a single project!!

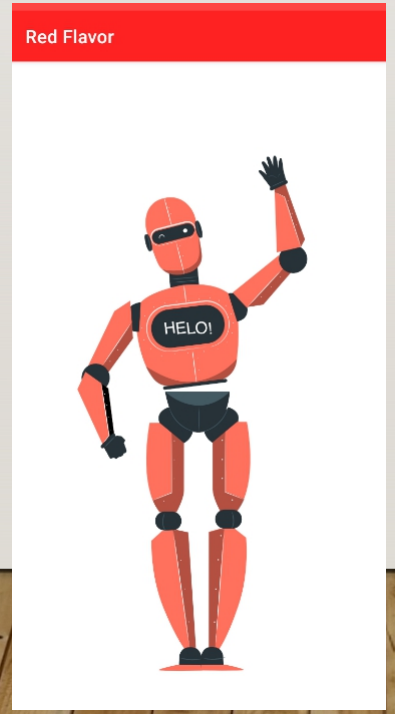
by Symela Komini

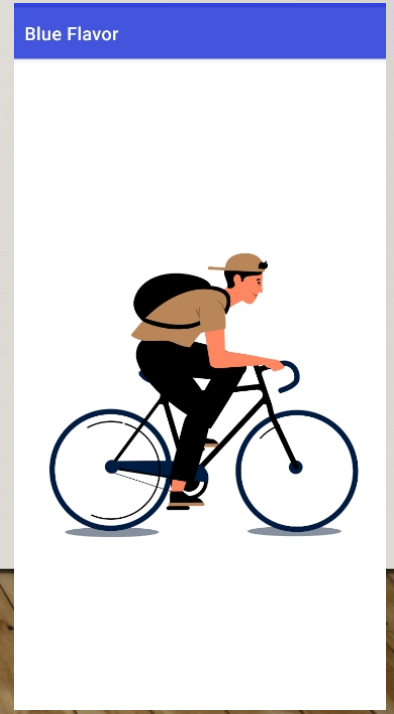
What are the flavors and why do we use them?

- Multiple versions of an app, with different:
  - Appearance,
  - Functionality,
  - Target Devices



Basic App





Before we start the implementation...

- Let's define a few concepts for the sake of this presentation.
  - **Flavor / ProductFlavors** : version of the app (eg. flavorLive, taza, google etc)
  - **Dimension / FlavorDimension** : the group that a flavor belongs to (eg. environment, AppFlavor, store). We can define more than one group of flavors in our app (multi-dimension flavors)
  - **BuildType** : the type of build we want (eg. debug, snapshot, release etc)
  - **BuildVariant** : the result of combining Flavors and BuildTypes (eg. FlavorLiveVodafoneGoogleDebug)

Configure flavors

- Add to app build.gradle file:
  - flavor dimensions
  - product flavors

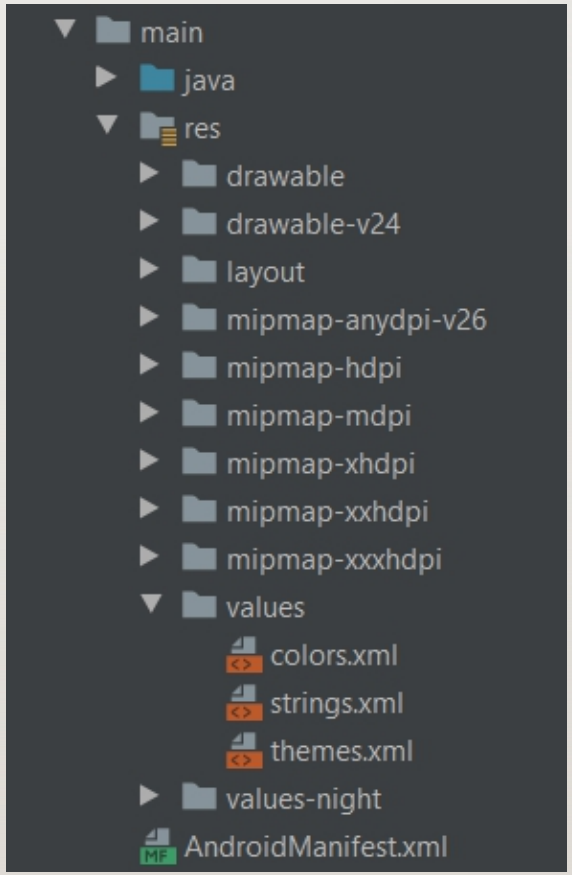
- Remember…
  - all product flavors must belong to a dimension
  - if you use **only one** flavor dimension, 'dimension' property is **optional**, and the plugin automatically assigns all the module's flavors to that dimension

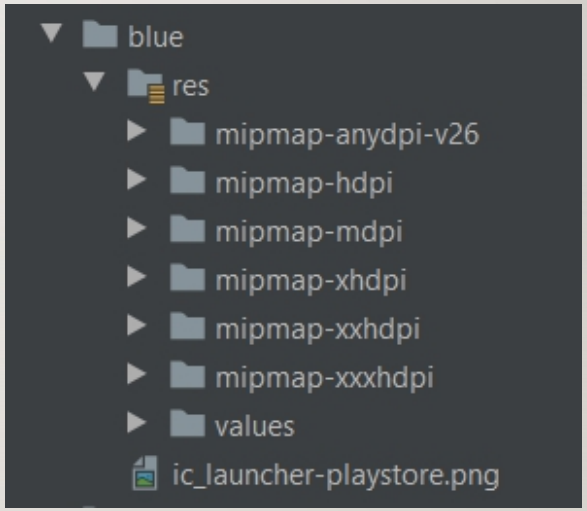
```
flavorDimensions 'color'

productFlavors {

    red {
        dimension 'color'
        applicationId 'com.android.flavorstutorial.red'
    }

    green {
        dimension 'color'
        applicationId 'com.android.flavorstutorial.green'
    }

    blue {
        dimension 'color'
        applicationId 'com.android.flavorstutorial.blue'
    }
}
```

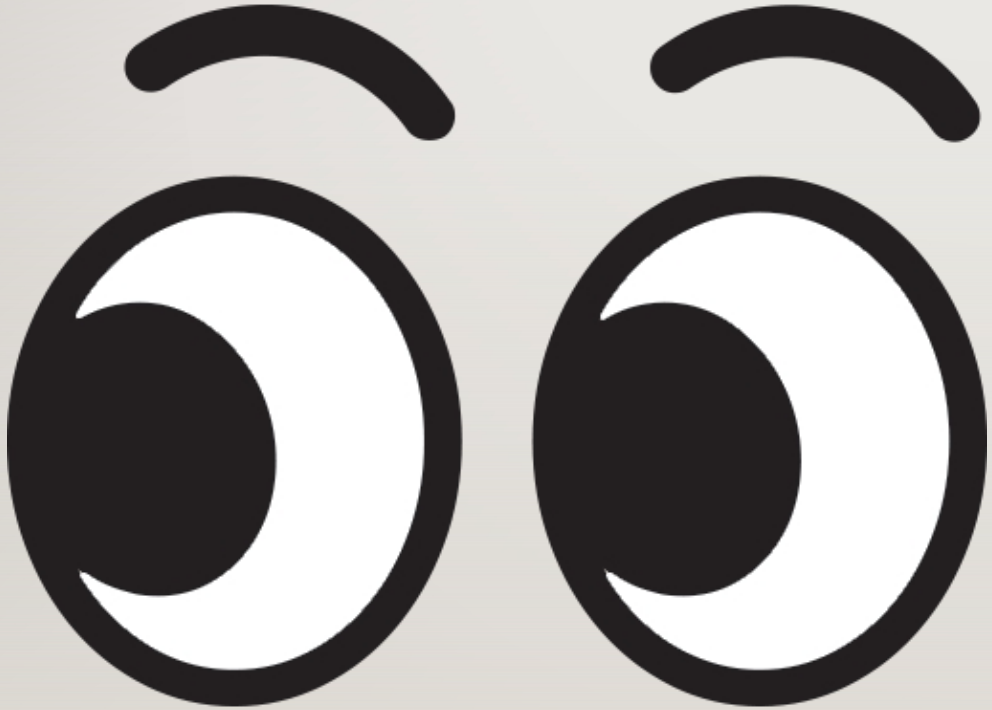
How to customize each flavor?

- Anything flavor specific must be inside the current flavor folder:
  - for **red** flavor, all the assets and code should be in **app/src/red** folder,
  - for **green** flavor, in **app/src/green** folder, and,
  - for **blue** flavor, in **app/src/blue** folder

- Assets, resources and code in flavor folders, **must** have the **same structure** as the ones in app main folder





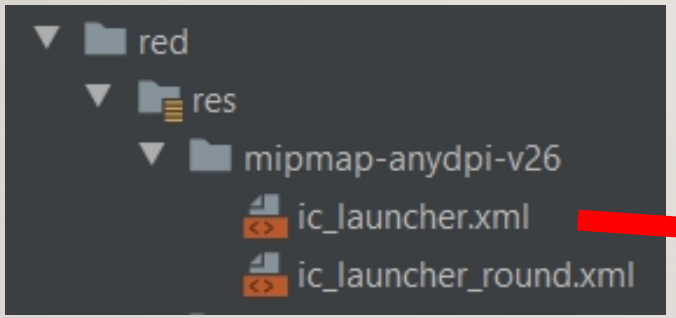


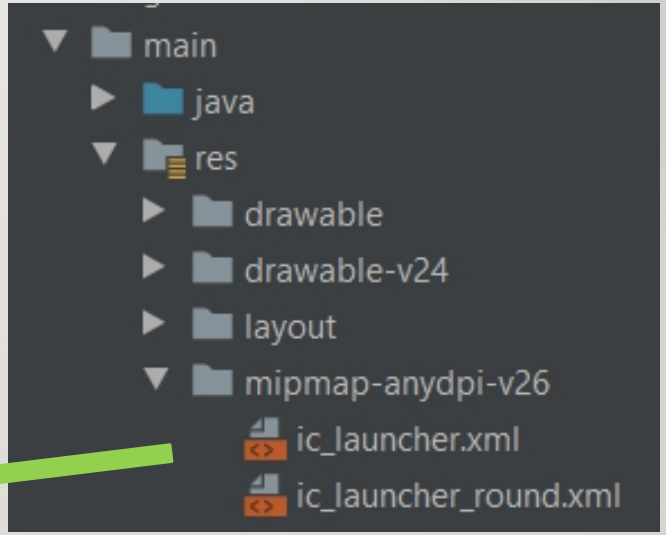
So… How does flavor magic happen?

File hierarchy and overriding!!

- Let's say we want to build red flavor
- During build time:
  - app/src/red folder is being merged with app/src/main folder, while green and blue flavor folders are ignored
  - If there are files in both folders with the same name, flavor files override main files (eg different launcher icon per flavor)
  - Values files are merged and items with the same name are overridden

Example





red\...\strings.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Red Flavor</string>
</resources>
```

main\...\strings.xml

```xml
<resources>
    <string name="app_name">Flavors Demo</string>
</resources>
```

Build variants

- The number of available build variants varies depending on the number of dimensions, the number of flavors per dimension and the number of build types we have configured.

Let's do some math!

- Suppose we have only **one** flavor dimension, **N** flavors in this dimension and **B** built types configured in app build.gradle.

Number of Build Variants = N x B **Build Variants name format = NB**

- Suppose we have **2** flavor dimensions, **N** flavors in first dimension and **M** in second, as well as **B** built types.

Number of Build Variants = N x M x B **Build Variants name format = NMB**

- And so on, you get the point....

Variant filters

- The number of available build variants can escalate quickly and make build variants complicated and chaotic...
- Variant filter to the rescue!
  - We keep only the combinations of flavors and build types that we want!
- Example:
  - 2 flavors: red and blue
  - 3 build types: debug, local and release
  - We don't want redDebug and blueLocal build variants

```
variantFilter { variant ->
    def names = variant.flavors*.name
    // To check for a certain build type, use variant.buildType.name == "<buildType>"
    if ((names.contains("red") && variant.buildType.name == "debug") ||
            (names.contains("blue") && variant.buildType.name == "local")) {
        // Gradle ignores any variants that satisfy the conditions above.
        setIgnore(true)
    }
}
```

Thank you!