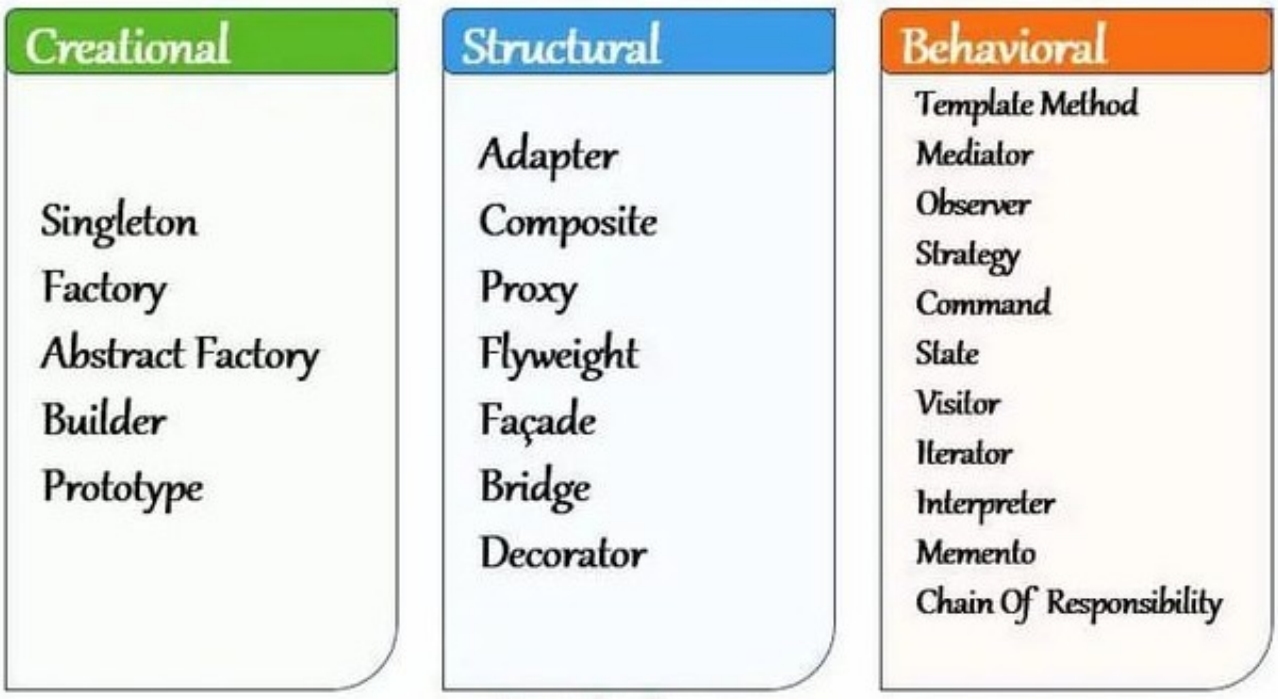
Design patterns

Design patterns are the best solutions to the common problems encountered in software design.

Creational Patterns:
These patterns focus on object creation mechanisms

Structural Patterns:
Structural patterns deal with object composition and class relationships, helping to define how objects are connected to one another.

Behavioral Patterns:
Behavioral patterns focus on how objects interact and communicate with each other



Creational

Singleton

The singleton pattern ensures that only one instance of a class is ever created.

1 – Using “**object**” keyword

```
object MySingleton {

    fun doSomeThing(){
        // your code
    }
}
```

2 – With **“ parameters”**

```
class MySingleton private constructor(private val param: String) {

    companion object {
        @Volatile
        private var INSTANCE: MySingleton? = null

        @Synchronized
        fun getInstance(param: String): MySingleton {
           return INSTANCE ?: MySingleton(param).also { INSTANCE = it }
        }
    }
}
```

3 – Using **“ Application Class “** & **“ lazy initializer “**

```kotlin
class MyApp: Application() {

    companion object {
        lateinit var appModule: AppModule
    }

    override fun onCreate() {
        super.onCreate()
        appModule = AppModuleImpl(this)
    }
}

interface AppModule {
    val authApi: AuthApi
    val authRepository: AuthRepository
}

class AppModuleImpl(private val appContext: Context): AppModule {
    override val authApi: AuthApi by lazy {
        Retrofit.Builder()
            .baseUrl("https://my-url.com")
            .addConverterFactory(GsonConverterFactory.create())
            .build()
            .create()
    }
    override val authRepository: AuthRepository by lazy {
        AuthRepositoryImpl(authApi)
    }
}
```