Coding Standard (Kotlin)

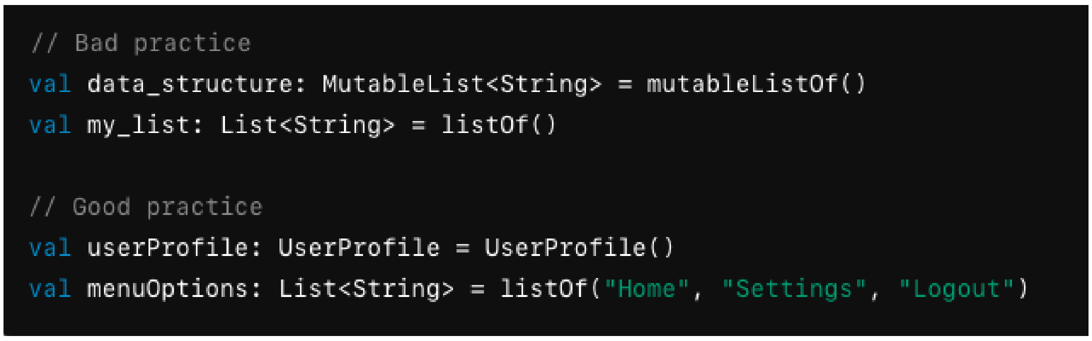
1.Naming Convention

1.1.1 General

- Use descriptive but concise names that reflect the purpose of variables or functions. Avoid names that are overly generic or too detailed.

- **Bad practice :**
  data_structure
  my_list
  info_map,

- **Good practice :**
  userProfile – Describes a profile object related to the user.
  menuOptions – Defines options within a menu.



```kotlin
// Bad practice
val data_structure: MutableList<String> = mutableListOf()
val my_list: List<String> = listOf()

// Good practice
val userProfile: UserProfile = UserProfile()
val menuOptions: List<String> = listOf("Home", "Settings", "Logout")
```

1.1.2 CamelCase for Abbreviations :

- When using CamelCase, capitalize all letters of abbreviations or acronyms (e.g., HTTP, URL, XML, JSON). This makes the names clearer and more in line with widely adopted practices.

- **Bad Practice:**
  HttpServer
  xmlParser
  jsonResponse
- **Good Practice:**
  HTTPServer
  XMLParser
  JSONResponse

```
// Bad practice
fun createHttpServer(): Server {
    // implementation
}

val xmlParser: XMLParser = XMLParser()

// Good practice
fun createHTTPServer(): Server {
    // implementation
}
```

1.2. Packages

- Package names are written in **lowercase** and typically follow the reverse domain name convention, just like in Java.
- Avoid using underscores or capital letters in package names.

```
package com.example.myapp
```

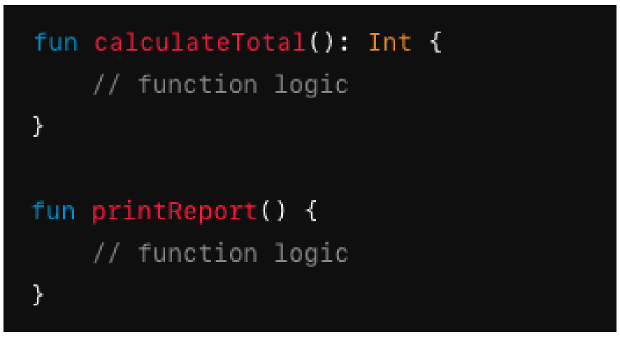
1.3 Classes and Interfaces

- **PascalCase** (UpperCamelCase) is used for class and interface names.
- Class names should be **nouns**, while interface names can be adjectives or nouns depending on the usage.

```
class Person
interface Drivable
```

2. Functions and Methods

- Use **camelCase** for function names.
- Function names should describe actions, and typically be **verbs**.



```kotlin
fun calculateTotal(): Int {
    // function logic
}

fun printReport() {
    // function logic
}
```

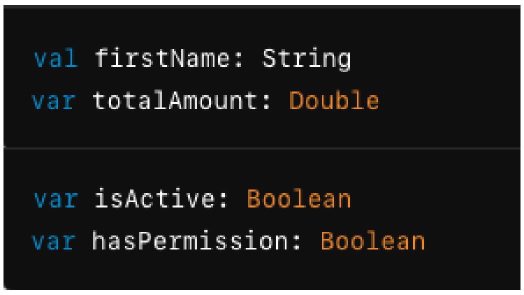
3. Variables and Properties

3.1. Variables

- Variable and property names should also use **camelCase**.
- Use short but meaningful names that describe the purpose of the variable.

Boolean properties should follow naming conventions such as:

- **isX, hasX, canX, shouldX**

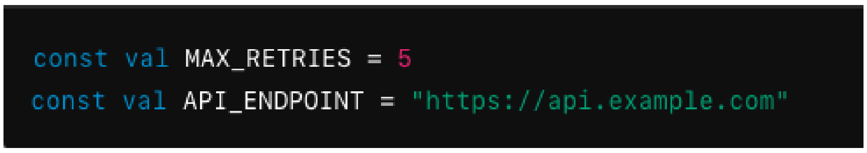


```kotlin
val firstName: String
var totalAmount: Double

var isActive: Boolean
var hasPermission: Boolean
```

3.2. Constants (Immutable Variables)

Use **UPPER_SNAKE_CASE** for constants, declared with **const val**



```kotlin
const val MAX_RETRIES = 5
const val API_ENDPOINT = "https://api.example.com"
```

4. Enums

Enum names should follow **PascalCase**, and enum constants should be written in **UPPER_SNAKE_CASE**.

```
enum class Color {
    RED, GREEN, BLUE
}
```

5. Type Parameters (Generics)

Use single, uppercase letters such as T, E, K, V, or descriptive names starting with T.

```
class Box<T> (val value: T)
```

```
class Mapper<InputType, OutputType> {
    fun map(input: InputType): OutputType {
        // logic
    }
}
```

6. Test Method Names

- Kotlin allows more descriptive function names, and when writing test methods, use **camelCase** with descriptive names.
- It's common to prefix test functions with should or test.

```
fun shouldReturnCorrectTotalWhenItemsAdded() {
    // test logic
}

fun testUserCanLoginWithValidCredentials() {
    // test logic
}
```

7. Companion Object and Factory Names

- Companion object names should be **PascalCase** if given a name. If unnamed, refer to the default name companion.
- Factory methods should follow function naming conventions.

```kotlin
class Database {
    companion object Factory {
        fun create(): Database = Database()
    }
}
```

8. Extension Functions

- Use **camelCase** and keep the function names consistent with regular function naming conventions.
- The function name should describe what it extends and what it does.

```kotlin
fun String.isValidEmail(): Boolean {
    return this.contains("@")
}
```

9. Annotations

- Annotations are usually **PascalCase**, similar to class and interface naming.

```kotlin
@Serializable
class User(val name: String)
```

10. Coroutines (Suspending Functions)

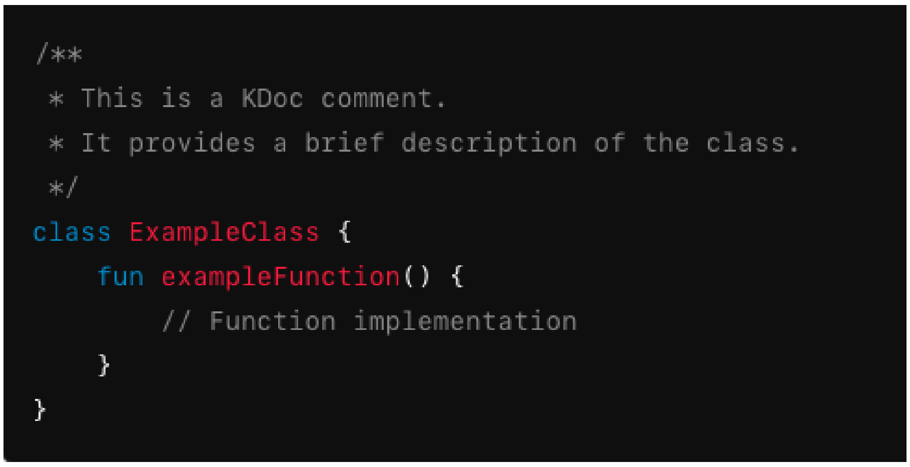
- Suspending functions in Kotlin typically include the suspend keyword. The name can reflect an asynchronous action but follows the same naming conventions as functions.

```kotlin
suspend fun fetchData(): Data {
    // asynchronous fetching logic
}
```

11. White Spaces

11.1 indentation:

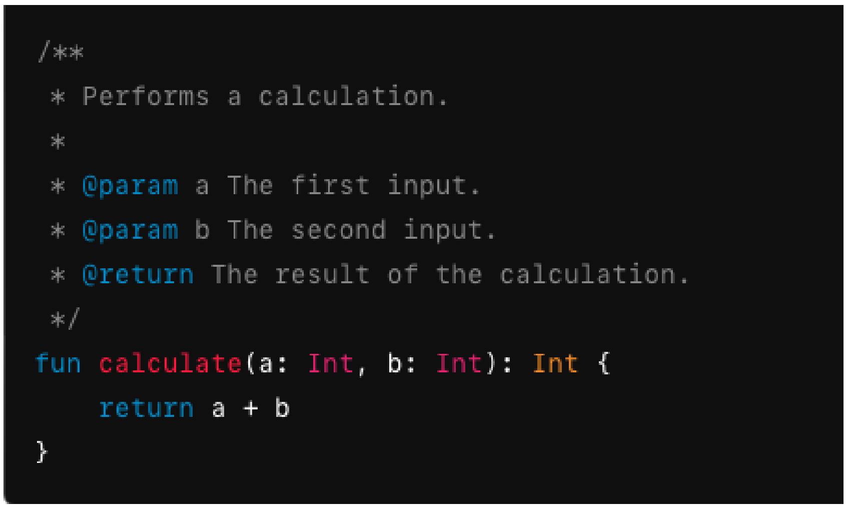
- Use **4 spaces** for indentation, and do not use tabs. This makes the code and comments uniform across different text editors.



```kotlin
/**
 * This is a KDoc comment.
 * It provides a brief description of the class.
 */
class ExampleClass {
    fun exampleFunction() {
        // Function implementation
    }
}
```

11.2 Spaces After KDoc Tags:

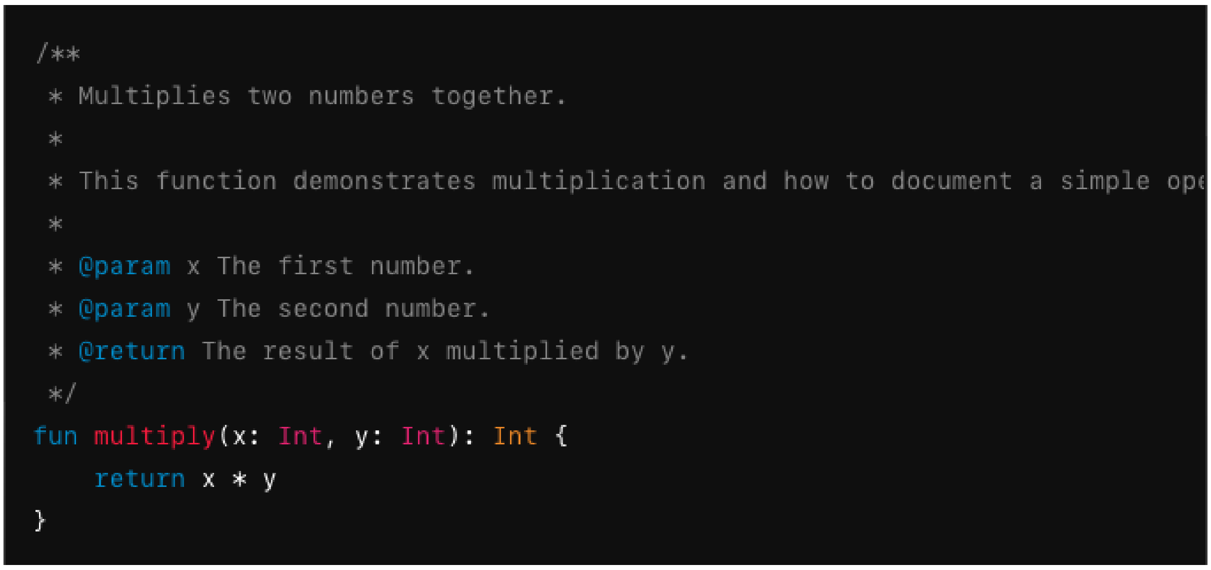
- After the @param, @return, and other KDoc tags, there should be **exactly one space** before the description starts.



```kotlin
/**
 * Performs a calculation.
 *
 * @param a The first input.
 * @param b The second input.
 * @return The result of the calculation.
 */
fun calculate(a: Int, b: Int): Int {
    return a + b
}
```

11.3 Blank Lines Between Sections:

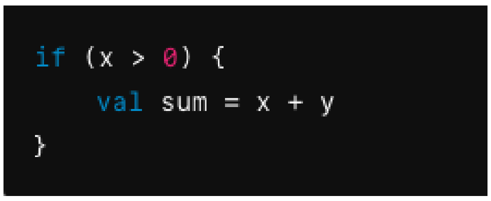
- Use **one blank line** to separate different sections of the KDoc, like the description and tag sections (@param, @return, etc.), for clarity.



```
/**
 * Multiplies two numbers together.
 *
 * This function demonstrates multiplication and how to document a simple ope
 *
 * @param x The first number.
 * @param y The second number.
 * @return The result of x multiplied by y.
 */
fun multiply(x: Int, y: Int): Int {
    return x * y
}
```

11.4 **Spaces Around Operators and Keywords**:

- Always use spaces around operators like =, +, -, *, and between control keywords and their parentheses, such as if (condition) and for (item in list).



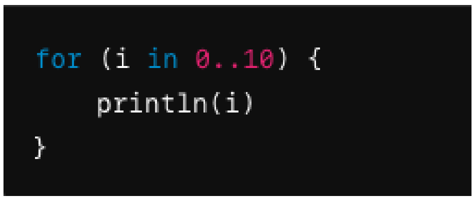
```
if (x > 0) {
    val sum = x + y
}
```

11.5 **No Trailing White Spaces**:

- Avoid trailing white spaces at the end of lines. Many editors can highlight or automatically remove them.

11.6 Spaces Before and After Curly Braces:

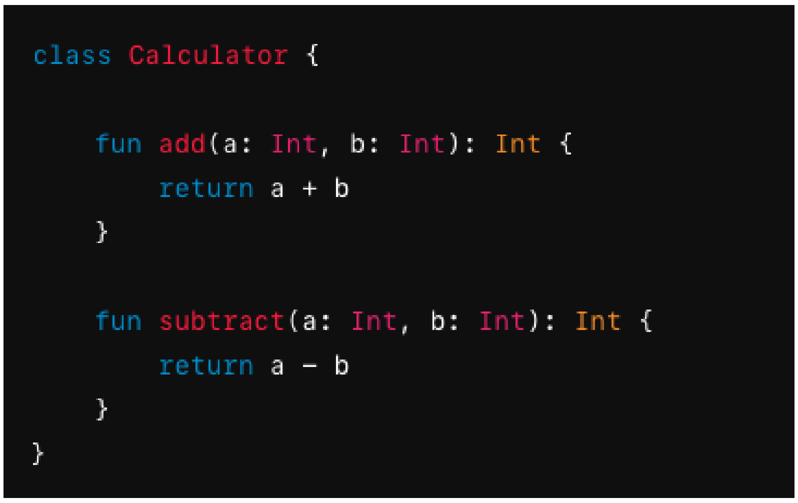
- Ensure **one space** before { in control structures and after } in the closing block.



```
for (i in 0..10) {
    println(i)
}
```

11.7 Blank Lines Between Methods:

- Place a **blank line** between methods and class members to increase readability.



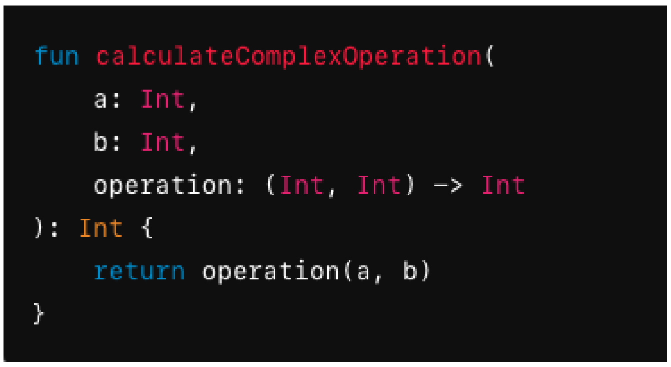
```
class Calculator {

    fun add(a: Int, b: Int): Int {
        return a + b
    }

    fun subtract(a: Int, b: Int): Int {
        return a - b
    }
}
```

11.8 Alignment of Parameters in Method Signatures:

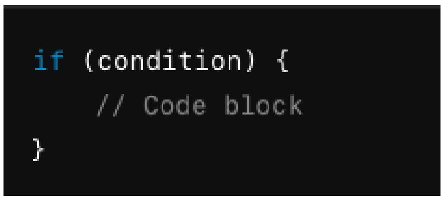
- When method signatures are long and span multiple lines, align subsequent lines properly by starting them with **four spaces** after the opening parenthesis.



```
fun calculateComplexOperation(
    a: Int,
    b: Int,
    operation: (Int, Int) -> Int
): Int {
    return operation(a, b)
}
```

12. Curly Braces

- Open curly braces { should not be followed by a new line. They should be at the end of the line.
- Closing curly braces } should be on a new line aligned with the beginning of the corresponding block.

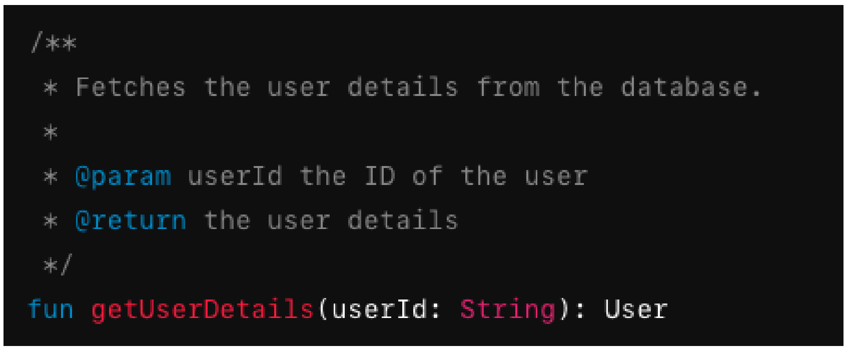


```
if (condition) {
    // Code block
}
```

13.Commenting

13.1. KDoc Commenting

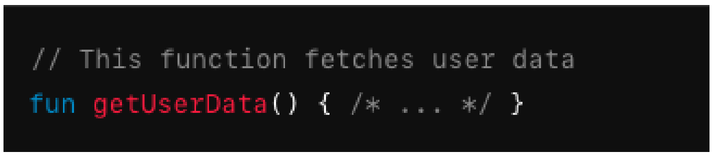
- Use KDoc (/** */) for documenting functions, classes, and parameters.
- Always document public APIs and provide a description of parameters and return types.



```
/**
 * Fetches the user details from the database.
 *
 * @param userId the ID of the user
 * @return the user details
 */
fun getUserDetails(userId: String): User
```

13.2 Single line and Multi line comments

- Comment every **3-7 lines of code**.
- Use single-line comments (//) for short explanations.
- Use multi-line comments (/* */) for detailed documentation or block comments.



```
// This function fetches user data
fun getUserData() { /* ... */ }
```

14. Exception Handling

- Always handle exceptions using **try-catch** blocks.
- Provide meaningful messages for exceptions.

```
try {
    // Code that might throw an exception
} catch (e: Exception) {
    println("An error occurred: ${e.message}")
}
```

15. Import Formatting

- **Group imports logically** by keeping similar types of imports together (standard library imports, third-party imports, project-specific imports).
- **Avoid wildcard imports** (e.g., import kotlin.*). Instead, import only what is necessary.
- **Static imports** should follow regular imports, ensuring that they are placed after the non-static imports.

```
// Standard library imports
import kotlin.collections.List

// Third-party library imports
import com.example.myapp.utilities.Logger

// Project-specific imports
import com.mycompany.project.models.User
import com.mycompany.project.services.UserService

// Static imports
import com.mycompany.project.constants.AppConstants.MAX_USERS
```

16. File and Folder Structure

- Organize files logically by **feature or module**, grouping related components (e.g., models, services, views) in the same folder.
- Use consistent folder naming and structure to make the project easy to navigate. Folder names should follow **lowercase naming conventions** with no underscores.

```
/src
    /user
        User.kt
        UserService.kt
        UserRepository.kt
    /auth
        AuthController.kt
        AuthService.kt
    /config
        AppConfig.kt
```

17. Code Reusability

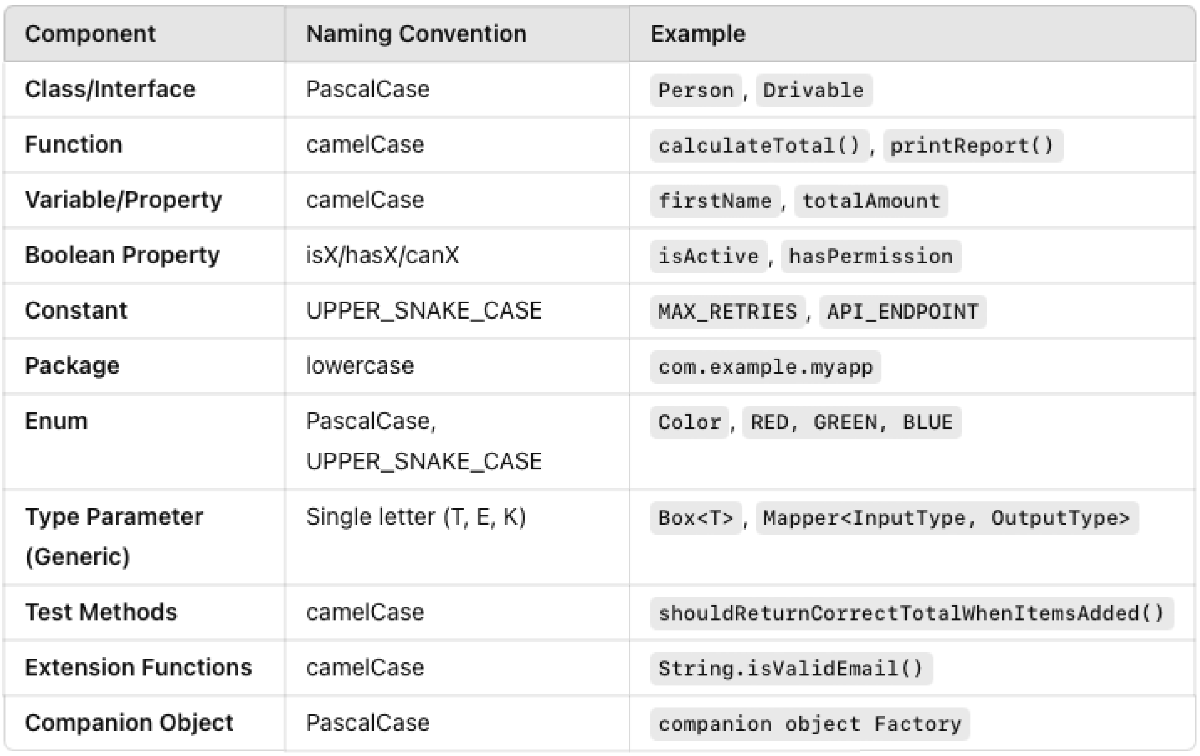
- Break down large tasks into **smaller, reusable functions**. This increases readability, maintainability, and testability of the code.
- Avoid code duplication by creating methods that handle repetitive logic and can be reused in multiple parts of the application

Example: Instead of repeating the same logic in multiple places, extract it into a reusable function:

```kotlin
// Reusable function to validate user input
fun validateInput(input: String): Boolean {
    return input.isNotEmpty() && input.length >= 5
}

// Usage in different parts of the application
val isUsernameValid = validateInput(username)
val isPasswordValid = validateInput(password)
```

Summery Table :



Component	Naming Convention	Example
Class/Interface	PascalCase	`Person`, `Drivable`
Function	camelCase	`calculateTotal()`, `printReport()`
Variable/Property	camelCase	`firstName`, `totalAmount`
Boolean Property	isX/hasX/canX	`isActive`, `hasPermission`
Constant	UPPER_SNAKE_CASE	`MAX_RETRIES`, `API_ENDPOINT`
Package	lowercase	`com.example.myapp`
Enum	PascalCase, UPPER_SNAKE_CASE	`Color`, `RED, GREEN, BLUE`
Type Parameter (Generic)	Single letter (T, E, K)	`Box<T>`, `Mapper<InputType, OutputType>`
Test Methods	camelCase	`shouldReturnCorrectTotalWhenItemsAdded()`
Extension Functions	camelCase	`String.isValidEmail()`
Companion Object	PascalCase	`companion object Factory`

References:

1. Official Kotlin Documentation (KDoc):
https://kotlinlang.org/docs/kotlin-doc.html

2. Documentation with KDoc for Kotlin/Android:
https://medium.com/@drflakelorenzgerman/documentation-with-kdoc-for-kotlin-android-a93c99dfe74

3. Kotlin Style Guide by Android Developers:
https://developer.android.com/kotlin/style-guide

4. Kotlin Style Guide on GitHub:
https://github.com/ditn/KotlinStyleGuide