

Escuela Politécnica Nacional

Programación II

2024-A



Proyecto final

Tema:

Sistema de ingreso estudiantil

Grupo 6

Profesor:

Ing. Paccha Angamarca Patricio Michael

Integrantes:

Rios Condoy Marco David

Rosas Borja Ariel Alexander

Sacoto Solorzano Jonathan David

Sanchez Vistin Katherine Abigail

Quito, 2024

Estándares de Codificación
Sistema de ingreso estudiantil (Kotlin)
2024A

Contenido

1. Marco general	4
2. Nomenclatura.....	4
2.2. Reglas específicas.....	4
2.2.1. Clases.....	4
2.2.2. Métodos	5
3. Identificadores	5
3.1. Parámetros	5
3.2. Variables	5
4. Documentación.....	6
4.1. Encabezado	6
4.2. Nomenclatura de Scripts de Bases de Datos	6
5. Formato del Código	7
5.1. Indentación	7
5.2. Longitud de líneas	7
5.3. Líneas en blanco	7
6. Buenas prácticas	7
6.1. Uso de val y var	7
6.2. Expresiones lambda.....	8
6.3. Estructuras de control	8
7. Recursos	8
7.1. Imágenes.....	8
7.2. Strings.....	9
8. Recomendaciones	9
9. Referencias	9

1. Marco general

Este documento establece los estándares de codificación para proyectos desarrollados en Kotlin, con el objetivo de asegurar la calidad, legibilidad y mantenibilidad del código. Estos estándares están basados en las convenciones de la comunidad Kotlin y se aplican a todos los desarrollos en la Escuela Politécnica Nacional.

2. Nomenclatura

2.1. Regla general

- Usar nombres en singular para todos los objetos a crear.
- No usar palabras reservadas en los nombres de los objetos, en el caso de ser absolutamente necesario usar ``.
- Los nombres de los objetos deben utilizar la convención UpperCamelCase para clases y lowerCamelCase para variables y métodos.
- Evitar el uso de abreviaciones que dificulten la comprensión.

2.2. Reglas específicas

2.2.1. Clases

Reglas:

- Los nombres de las clases deben ser descriptivos del propósito del sistema.
- Usar mayúsculas y evitar espacios en los nombres.
- Usar un formato consistente para todas las clases relacionadas.

Formato:

<NombreClase>

Ejemplo:

```
class QRCodeStudentValidator {  
  
    // Código de la clase  
  
}
```

2.2.2. Métodos

Reglas:

- El nombre de los métodos debe reflejar claramente su funcionalidad.
- Usar `lowerCamelCase` para los nombres de los métodos.

Formato:

<nombreMetodo>

Ejemplo:

```
fun validateAccess(scannedData: String): String {  
    // Código del método  
}
```

3. Identificadores

3.1. Parámetros

Reglas:

- Los parámetros deben llevar un prefijo seguido de `UpperCamelCase` para mayor claridad.

Formato:

<tipoDato><NombreParámetro>

Ejemplo:

```
fun processStudentData(studentId: Int, studentName: String) {  
    // Código del método  
}
```

3.2. Variables

Reglas:

- Las variables deben seguir la convención `lowerCamelCase`.

Formato:

<nombreVariable>

Ejemplo:

```
val studentCount: Int = 0
```

4. Documentación

4.1. Encabezado

Cada clase y método en tu proyecto debe tener un encabezado que incluya detalles del autor, descripción, y validación.

Ejemplo:

```
/
```

```
* Proyecto: Sistema de Ingreso Estudiantil
```

```
* Autor: Katherine Sanchez
```

```
* Descripción: Clase para validar el acceso de estudiantes mediante códigos QR.
```

```
* Fecha: 20240808
```

```
*/
```

```
class QRCodeStudentValidator {
```

```
    // Código de la clase
```

```
}
```

4.2. Nomenclatura de Scripts de Bases de Datos

Reglas:

- Los scripts de modificación de vistas, procedimientos y funciones deben seguir la nomenclatura.

Formato:

<NOMBRE_SISTEMA><MODULO><DESCRIPCION>.sql

Ejemplo:

QRCodeStudentValidatorDBCreateTables.sql

5. Formato del Código

5.1. Indentación

- Usa una indentación de 4 espacios, sin tabulaciones.

Ejemplo:

```
fun exampleFunction() {  
    val value = "example"  
}
```

5.2. Longitud de líneas

- Limita la longitud de las líneas a 100 caracteres. Si una línea es demasiado larga, divídela en varias líneas.

Ejemplo:

```
val longString = "This is a very long string that should be broken " +  
    "into multiple lines for better readability."
```

5.3. Líneas en blanco

- Inserta líneas en blanco para separar lógicamente bloques de código y mejorar la legibilidad.

Ejemplo:

```
fun initialize() {  
    setupUI()  
    setupListeners()  
}
```

6. Buenas prácticas

6.1. Uso de val y var

- Prefiere val en lugar de var para definir variables inmutables.

Ejemplo:

```
val name = "John" // immutable
```

```
var age = 30 // mutable
```

6.2. Expresiones lambda

- Usa expresiones lambda cuando sea adecuado para mejorar la concisión y legibilidad del código.

Ejemplo:

```
val numbers = listOf(1, 2, 3, 4, 5)
```

```
val doubled = numbers.map { it * 2 }
```

6.3. Estructuras de control

- Utiliza estructuras de control como when de manera clara y legible.

Ejemplo:

```
when (validationStatus) {  
    "Valid" -> // acción para código válido  
    "Revision" -> // acción para revisión  
    "Invalid" -> // acción para código inválido  
    else -> // acción predeterminada  
}
```

7. Recursos

7.1. Imágenes

- Las imágenes deben estar ubicadas en la carpeta res/drawable y nombradas en minúsculas usando guiones bajos.

Ejemplo:

```
valid_student.png, invalid_student.png
```


7.2. Strings

- Todos los textos mostrados en la interfaz de usuario deben estar en el archivo `res/values/strings.xml` para soportar la localización.

Ejemplo:

```
<string name="scan_qr">Escanear QR</string>
```

8. Recomendaciones

8.1. Manejo de errores y excepciones

Es fundamental manejar las excepciones de manera adecuada para evitar errores inesperados en el código. Utiliza bloques `try-catch` para capturar y manejar excepciones específicas, en lugar de capturar excepciones generales como `Exception`. Esto mejora la seguridad y claridad del código, permitiendo una mejor gestión de errores.

8.2. Fomentar la inmutabilidad

Siempre que sea posible, utiliza `val` en lugar de `var` para declarar variables inmutables. La inmutabilidad ayuda a prevenir errores al asegurar que los valores no sean modificados después de su asignación inicial, lo que contribuye a la estabilidad y predictibilidad del código.

8.3. Uso de funciones de extensión

Las funciones de extensión son una poderosa herramienta en Kotlin que permite añadir funcionalidades a clases existentes sin modificar su código original. Esto mejora la modularidad y reutilización del código, facilitando su mantenimiento y evolución.

9. Referencias

Kotlin Documentation. (n.d.). *Coding conventions*. Kotlin. Retrieved August 8, 2024, from <https://kotlinlang.org/docs/coding-conventions.html>

Moskala, M. (n.d.). *Effective Kotlin*. Leanpub. Retrieved August 8, 2024, from <https://leanpub.com/effectivekotlin>

Android Developers. (n.d.). *Kotlin style guide*. Android Developers. Retrieved August 8, 2024, from <https://developer.android.com/kotlin/style-guide>