# ESTANDAR DE PROGRAMACION


PROYECTO:

# APLICACIÓN MÓVIL PARA EL SEGUIMIENTO MUNDIAL DE CASOS DE COVID-19 CON DATOS EN TIEMPO REAL

INTEGRANTES:

- PILCO QUISPE, Mireya Flavia
- SALAMANCA CONTRERAS, Fiorella Rosmery
- ZAVALA VENEGAS, Luis Ángel

TACNA - PERÚ
2020

HISTORIAL DE VERSIONES

| ***Fecha*** | ***Versión*** | ***Descripción*** | ***Autor*** |
|---|---|---|---|
| 26/05/2020 | 1.0 | Creación del documento | MP, FS, LZ |

# ESTANDAR DE PROGRAMACION

## 1. OBJETIVO

El objetivo del presente documento es definir los estándares de programación en cuanto a denominación de archivos, distribución de pantallas, reportes, etc. con el fin de desarrollar una programación fácil de entender por cualquier programador y contar con una herramienta de fácil documentación de los sistemas de información.

## 2. ALCANCES

El estándar de programación se utilizará en todos los desarrollos de software realizados para la Aplicación Móvil para el Seguimiento Mundial de Casos De Covid-19 con Datos en Tiempo Real.

## 3. DEFINICIONES

- ✔ Metodología de Programación Permite realizar mantenimientos en menor tiempo y costo, migrar código consistentemente, habilita a los programadores moverse entre proyectos diferentes y asegura la comunicación técnica entre desarrolladores.
- ✔ Sistema Conjunto de elementos que interactúan entre sí para lograr un objetivo común.

## 4. RESPONSABILIDAD

- ✔ De los analistas programadores Cada analista programador deberá utilizar el estándar de programación en los desarrollos de software posteriores a la fecha de aprobación del mismo.

- ✔ Es responsabilidad del Analista Supervisor controlar la aplicación del estándar de programación en los desarrollos de software posteriores a la fecha de aprobación del mismo.

# 5. DESARROLLO

## 5.1. ARQUITECTURA DE PROGRAMACIÓN:

a. **Dimensión**

- Para las dimensiones, use " **dp"** y para el tamaño del texto, use " **sp".** No use " **px"**.

b. **Nombre de archivo**

- Layout: el archivo se incluye con el archivo de actividad, use - " **_activity.xml** "
- View: archivo incluido con la vista, use " **_view.xml** "
- Activity: archivo incluido con **_Activity.java**
- Service: archivo incluido con **_Service.java**

**c. string.xml**

- **Mensajes**: nombre del prefijo con " **msg**_ "
- **Etiquetas**: nombre del prefijo con " **lbl**_ "

d. **colores.xml**

- **Nombre de color** : prefijo de color con " **color**_ "

e. **Fuente**

- cree la carpeta " **fonts** " y coloque el ". **ttf** "dentro

**f. Widget**

- EditText: prefija el nombre con **edtitext**_
- Botón: prefija el nombre con **btn**_
- ImageView: prefija el nombre con **imageview**_
- TextView: prefija el nombre con **textview**_
- RadioButton: prefija el nombre con **radiobutton**_
-

g. **raw** : carpeta sin procesar para XML estático, archivos json o archivos multimedia

h. Nombre del icono de prefijo con " **ic**_ " e imagen de fondo con " **bg**_ "

i. Se deben usar imágenes de diferentes dimensiones para ldpi, mdpi, hdpi y xhdpij.

j. Para todos los mensajes y etiquetas, almacene los datos en string.xmlk.

k. style.xml : todas las propiedades del widget deben definirse aquíl.

l. paquete : agrupa todos los códigos de módulo relevantes en un solo paquete

## 5.2. CÓDIGO ESTILO DIRECTRICES PARA LOS CONTRIBUYENTES

**Estándar de Java**

- Lanza la excepción al llamador de tu método
- Lanza una nueva excepción que sea apropiada para tu nivel de abstracción
- Maneje el error con gracia y sustituya un valor apropiado en el bloque catch {}

**No use finalizadores**

- Pros: puede ser útil para hacer la limpieza, particularmente de recursos externos.
- Contras: no hay garantías de cuándo se llamará a un finalizador, o incluso si será llamado en absoluto.
- Decisión: no utilizamos finalizadores

**Importaciones totalmente calificadas**

Cuando desee utilizar la barra de clase del paquete foo, hay dos formas posibles para importarlo:

a) . import foo.*;

Pros: Reduce potencialmente el número de declaraciones de importación

b) import foo.Bar;

Pros: hace obvio qué clases se usan realmente. Hace código más legible para mantenedores

Decisión : utilice este último para importar todo el código de Android. Una excepción explícita es hecho para bibliotecas estándar de java ( java.util. * , java.io. * , etc.) y código de prueba de unidad( junit.framework. * )

**Usar comentarios estándar de Javadoc**

- Cada archivo debe tener una declaración de derechos de autor en la parte superior.
- Luego deben seguir una declaración de paquete y declaraciones de importación.
- Y luego está la declaración de clase o interfaz.

```
/*
* Copyright (C) 2010 The Android Open Source Project
*
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
*/
```

**Comentario de varias líneas para**

```
/**
* Does X and Y and provides an abstraction for Z.
*/
```

Cada clase y método público no trivial que escriba debe contener un comentario Javadoc con al menos una oración que describe lo que hace la clase o el método.

Ejemplos:

```
/** Returns the correctly rounded positive square root of a double value. */
static double sqrt(double a) {
 }
```

O

```
/**
* Constructs a new String by converting the specified array of
* bytes using the platform's default character encoding.
*/
public String(byte[] bytes) {
```

**Escribir métodos cortos**

- Si un método supera las 40 líneas más o menos, piense si puede dividirse sin dañar la estructura del programa.

**Definir campos en lugares estándar**

- Los campos deben definirse en la parte superior del archivo o inmediatamente antes de los métodos que los usan

**Alcance variable límite**

- El alcance de las variables locales debe mantenerse al mínimo
- Cada variable debe declararse en el bloque más interno que encierra todos los usos de la variable
- Las variables locales deben declararse en el punto en que se utilizan por primera vez

```
Set s = null;
try {
 s = (Set) cl.newInstance();
} catch(IllegalAccessException e) {
 throw new IllegalArgumentException(cl + " not accessible");
} catch(InstantiationException e) {
 throw new IllegalArgumentException(cl + " not instantiable");
}
```

Las variables de bucle deben declararse en la declaración for a menos que haya una razón convincente para hacer lo contrario:

```
for (int i = 0; i n; i++) {
 doSomething(i);
}
```

**Declaraciones de importación de Statements**

El orden de las declaraciones de importación es:
1. Imports de Android
2. Imports de terceros ( com , junit , net , org )
3. java y javaxPara coincidir exactamente con la configuración de IDE, las importaciones deben ser:

- Alfabético dentro de cada grupo, con letras mayúsculas antes de minúsculas (p. Ej.Z antes de a).
- Debe haber una línea en blanco entre cada grupo principal ( android , com , junit , net , org ,java , javax )

Este estilo fue elegido de tal manera que:

- Las importaciones que las personas quieren ver primero tienden a estar en la parte superior (android)
- Las importaciones que las personas quieren ver al menos tienden a estar en la parte inferior (Java)
- Los humanos pueden seguir fácilmente el estilo
- IDEs pueden seguir el estilo

**Usar espacios para sangría**

- Utilizamos 4 espacios de sangría para bloques.
- Nunca usamos pestañas.
- Utilizamos 8 sangrías de espacio para los ajustes de línea, incluidas las llamadas a funciones y las asignaciones

Si
```
String s =
 somethingVeryLong(…);
```

No

```
String s =
 somethingVeryLong(…);
```

**Siga las convenciones de nomenclatura de campo**

- Los nombres de campo no públicos y no estáticos comienzan con m.
- Los nombres de campos estáticos comienzan con s.
- Otros campos comienzan con una letra minúscula.
- Los campos finales estáticos públicos (constantes) son ALL_CAPS_WITH_UNDERSCORES

```
public class MyClass {
 public static final int SOME_CONSTANT = 42;
 public int publicField;
 private static MyClass sSingleton;
 int mPackagePrivate;
 private int mPrivate;
protected int mProtected;
}
```

**Usar Standard Brace Style**

- Los frenos no van en su propia línea; van en la misma línea que el código antes que ello

```
class MyClass {
 int func() {
 if (something) {
 // ...
 } else if (somethingElse) {
 // ...
 } else {
 // ...
 }
 }
```

}

**Limite la longitud de la línea**

- Cada línea de texto en su código debe tener como máximo 100 caracteres de longitud
- Excepción: si una línea de comentarios contiene un comando de ejemplo o una URL literal de más de 100caracteres, esa línea puede tener más de 100 caracteres para facilitar el corte y el pegado.
- Excepción: las líneas de importación pueden superar el límite porque los humanos rara vez las ven. Esto también simplifica la escritura de herramientas

**Usar anotaciones Java estándar**

- Las anotaciones (por ejemplo, @Override) se pueden enumerar en la misma línea con el elemento del lenguaje.
- Si hay múltiples anotaciones, o anotaciones parametrizadas, cada una de ellas debería estar listada uno por línea en orden alfabético.

**Las prácticas estándar de Android para las anotaciones predefinidas en Java son:**

- **@Deprecated :**

  ➢ La anotación @Deprecated se debe usar siempre que se use el anotado. Se desaconseja el elemento.

  ➢Si usa la anotación @Deprecated, también debe tener una @deprecatedEtiqueta Javadoc y debería nombrar una implementación alternativa.

  ➢ Además, recuerde que todavía se supone que un método @Deprecated funciona.

  Nota: si ve un código antiguo que tiene una etiqueta Javadoc @deprecated, agregue el@ Anotación reducida.

- **@Override:**

  ➢ La anotación @Override debe usarse siempre que un método anule el declaración o implementación de una superclase

Nota: si usa la etiqueta Javadoc @inheritdocs y deriva de una clase (no una interfaz), también debe anotar que el método @Sobre la clase principal método

**Tratar los acrónimos como palabras**

- Trate los acrónimos y las abreviaturas como palabras en nombres de variables, métodos y clases.
- Los nombres son mucho más legibles:

| Bueno | Malo |
|---|---|
| **XmlHttpRequest** | **XMLHTTPRequest** |
| **getCustomerId** | **getCustomerID** |
| **class Html** | **class HTML** |
| **String url** | **String URL** |
| **long id** | **long ID** |

**Use TODO comentarios**

- Utilice los comentarios TODO para el código que es temporal, una solución a corto plazo o lo suficientemente bueno pero no es perfecto.
- TODO debe incluir la cadena TODO en mayúsculas, seguida de dos puntos

// TODO: Remove this code after the UrlTable2 has been checked in.
Y
// TODO: Change this to use a flag instead of a constant.

**Inicie sesión con moderación**

- Las instalaciones de registro proporcionan cinco niveles diferentes de registro.
  - ➢ Error
  - ➢ Advertencia
  - ➢ Informativo

➢ Depura

- **ERROR:**

  ➢ Este nivel de registro debe usarse cuando algo fatal ha sucedido➢ Este nivel siempre se registra.

- **ADVERTENCIA:**

  ➢ Este nivel de registro debería usarse cuando sucediera algo grave e inesperado,

  ➢ Este nivel siempre se registra.

- **INFORMATIVO:**

  ➢ Este nivel de registro debería ser para notar que algo interesante para la mayoría la gente

  ➢ Este nivel siempre se registra.

- **DEPURACIÓN:**

  ➢ Este nivel de registro se debe utilizar para tener más en cuenta lo que está sucediendo en el dispositivo eso podría ser relevante para investigar y depurar comportamientos inesperados.

- **VERBOSA:**

  ➢ Este nivel de registro debe usarse para todo lo demás.

  ➢ Cualquier construcción de cadenas se eliminará de las versiones de lanzamiento y debe aparecer dentro el bloque if (LOCAL_LOGV)

**Descripciones**

Todo código que se encuentre en algún objeto, método, procedimiento u otro deberá ser descrito de manera clara y breve. Sobre las Descripciones

Cada procedimiento, programa o cualquier otro código, deberá incluir una cabecera en la cual se indicará el nombre del programa, procedimiento o función.

***---------- ***

*** Programa Principal ***
*** Nombre del programa: ***
*** Programador: ***
*** Fecha Creación: ***
*** Fecha de última modificación: ***
***---------- ***

### 6.5. SOBRE LA DOCUMENTACIÓN

Durante el desarrollo del sistema cada fase terminada deberá ser documentada, según el estándar de documentación vigente. La responsabilidad de la documentación de cada sistema de información recaerá en el jefe de cada proyecto de software.

Se deberá indicar en la documentación el código identificador asignado para cada tabla que conforma el sistema desarrollado.

Todo sistema que se encuentre en producción deberá tener un registro de bitácora, donde se detallen las modificaciones y/o adecuaciones que se realicen sobre el mismo.