Repository: https://github.com/Phalord/Proyecto-Construccion

Github usernames:

@Phalord

@2dumb2program

@DrearyStudent

1. Formato
    a. Llaves
        i. Las llaves serán usadas donde sea opcional
            - Las llaves serán usadas con “if”, “else”, “for”, “do” y “while”, incluso si el cuerpo está vació o solo tiene una declaración
        ii. Bloques no vacíos: Estilo K & R
            - Las llaves seguirán el estilo “Llaves Egipcias” para los *bloques no vacíos*
                - No salto de línea antes de la llave de apertura
                - Salto de línea después de la llave de apertura
                - Salto de línea antes de la llave de cierre
                - Llave de cierre después de la llave de cierre, *solo* si esa llave termina una declaración o termina el cuerpo de un método, constructor o una clase. No hay salto de línea si después de la llave hay un “else” o una coma.

```
return () -> {
    while (condition()) {
        method();
    }
};

return new MyClass() {
    @Override public void method() {
        if (condition()) {
            try {
                something();
            } catch (ProblemException e) {
                recover();
            }
        } else if (otherCondition()) {
            somethingElse();
        } else {
            lastThing();
        }
    }
};
```

        iii. Bloques vacíos: Deben ser concisos

- Un bloque vacío debe seguir el estilo K & R. Alternativamente, debe ser cerrado inmediatamente después de abierto, sin caracteres o salto de línea en medio ({}), al menos que sea parte de una declaración multi-bloque

```
//Esto es aceptable
void doNothing() {}

//Esto también es aceptable
void doNothingElse() {

}

//Esto no es aceptable: No debe haber bloques
//vacíos concisos en un multibloque
try {
    doSomething();
} catch (Exception e) {}
```

iv. Sangría de Bloque: +4 espacios

- Cada que un bloque o una línea del bloque es abierta, la sangría incrementa por cuatro espacios. Cuando el bloque termina, la sangría regresa a su anterior nivel de sangría. La sangría aplica tanto para código como comentarios a través del bloque

```
try {
    something();
} catch (ProblemException e) {
    recover();
}
```

b. Una declaración por Línea
   i. Cada declaración es seguida de un salto de línea
c. Límite de Columna: 100 caracteres
   i. El código Java tiene un límite de columna de 100 caracteres. Un "carácter" es cualquier punto de código Unicode. A excepción de lo mostrado abajo, cualquier línea que pueda exceder este límite debe ser ajustado, como explicado en la siguiente sección, Envoltura de Línea.
   ii. Excepciones:
      - Líneas donde sea imposible obedecer el límite (por ejemplo, una URL larga, o un método de referencia JSNI largo
      - Declaraciones *package* e *import*
      - Líneas de comando, en un comentario, que puedan ser copiadas y pegadas en un *Shell*
d. Envoltura de Línea
   i. **Nota de Terminología**: Cuando código que puede, legalmente, ocupar una sola línea de código es divida en múltiples líneas, la actividad se llama "Envoltura de código"

ii. No existe una formula comprehensiva o determinística que muestre exactamente cómo envolver una línea en cada situación. Regularmente, existen varias formas válidas de envolver la misma pieza de código.
iii. Aunque la razón típica para envolver código sea prevenir sobrepasar el límite de columna, incluso código que puede caber en una sola línea puede ser envuelto por decisión propia del autor.
iv. Dónde saltar:
    - Cuando se aplica un salto de línea a un operador sin-asignación, el salto se hace antes del símbolo.
        - Esto también aplica a los siguientes símbolos "parecidos a operadores":
            - Separador punto (.)
            - El doble dos puntos de referencia (::)
            - El ampersand en un tipo enlazado (<T extends Foo & Bar>)
            - Un pipe en un bloque catch (catch (FooException | BarException e))
        - Cuando una línea es saltada en un operador de asignación, regularmente, el salto viene después del símbolo, pero también se acepta hacerlo antes
        - El nombre de un método o un constructor permanece atado a su paréntesis de apertura que le sigue (*(*).
        - Una coma (,) permanece atada al token que la precede
        - Una línea nunca se salta adyacente a una flecha en una lambda, excepto que un salto venga inmediatamente después de la flecha si el cuerpo de la lambda consiste en una simple expresión sin llaves
        -
```
MyLambda<String, Long, Object> lambda =
  (String label, Long value, Object obj) -> {

  };

Predicate<String> predicate = str ->
  longExpressionInvolving(str);
```
v. Sangría de continuación de línea: al menos +2 espacios
    - Cuando se envuelve una línea, cada línea después de la primera continuación lleva una sangría de al menos +2 espacios de la línea original.
    - Cuando hay múltiples líneas de continuación, la sangría puede variar más allá de +2 espacios como sea deseado.

e. Espacio Vertical
    i. Un único espacio aparece siempre:
        - Entre miembros consecutivos o inicializadores de una clase: campos, constructores, métodos, clases anidadas, inicializadores estáticos, e instanciadores

Repository: https://github.com/Phalord/Proyecto-Construccion

2. Restricciones
    a. Comparación de Booleanos
        i. No invertir el significado natural del lenguaje; preferible usar “si el estado actual no está activado” que “si activo no es el estado actual”

```
//Correcto
!active

//Incorrecto
active == false
```

    b. for loops VS for-each loop
        i. Cuando se esté iterando en un elemento iterable, se prefiere hacer uso de:

```
//Correcto
for (String name: names) {
    doSomething(name);
}

//Incorrecto
for (int i = 0; i < names.lenght; i++) {
    doSomething(names[i]);
}
```

    c. Concatenación
        i. Evitar el uso del operador “+” para concatenar cadenas. Usa estándares de Java creados para esos propósitos como String.forma( ) y StringBuilder ( )

```
//Correcto
log.debug(String.format("found %s items", amount));

//Incorrecto
log.debug("found " + amount + " items");
```

    d. Excepciones
        i. No comerse excepciones, atrapar todas que se puedan o hacer que hagan algo

Repository: https://github.com/Phalord/Proyecto-Construccion

```
//Correcto
try {
    do();
} catch (SomethingWeCanHandleException e) {
    log.error(e);
    notifyUser(e);
} finally {
    cleanUp();
}

//Incorrecto
try {
    do();
} catch (Throwable e) {
    log.error(e);
}

try {
    do();
} catch (SomethingWeCanHandleException e) {

}
```

e. Tipos Generales
    i. Evitar usar declaraciones de tipo “Dios” con las clases que soporten genéricos

```
//Correcto
List<String> people = Arrays.asList("you", "me");

//Incorrecto
List people = Arrays.asList("you", "me");
```

f. Uso de *final*
    i. Al implementar servicios y clases que sean más que javabeans u objetos para transferir datos entre capas, mantente seguro de usar la palabra clave “final” para comunicar las intenciones con respecto a las subclases, el uso de constantes y valores que una vez declarados sean inmutables.

Repository: https://github.com/Phalord/Proyecto-Construccion

```
public final class ThisShouldNeverBeExtended {
 ...
}

public final neverOverrideThisMethod() {

}

private final int thisFieldValueWillNeverChangeOnceSet;

final int thisLocalVariableWillNeverChangeOnceSet;
```

g. Nombra los valores de retorno como “resultado”
    i. Considera el uso de “resultado” como el nombre de la variable de retorno. Esto facilita la dura tarea de depurar e incrementa la legibilidad del código

```
public Object doSomething() {
    Object resultado = null;
    if (something) {
        resultado = new Object();
    }
    return resultado;
}
```

h. Considera a los setters y getters para acceder a los campos
    i. Dentro de la misma clase, considere usar getters y setters para acceder a los valores de variables para que siempre se apliquen la inicialización diferida y otra lógica prevista implementada en los getters y setters

```
//Correcto
public void cambiarNombe(String nombre) {
    setNombre(nombre);
}

//Incorrecto
public void cambiarNombre(String nombre) {
    this.nombre = nombre;
}
```

3. Seguridad (Programación defensiva)
    a. Pruebas
        i. Uso de la metodología de TTD. Test driven development
            - Escribir siempre primero los casos de prueba unitarios (tomar una parte del código y ver que funciona correctamente.
        ii. Uso de casos frontera
            - Escribir casos que no están dentro de la funcionalidad normal.
    b. Excepciones
        i. Capturar excepciones

Repository: https://github.com/Phalord/Proyecto-Construccion

- Las excepciones deben ser atrapadas en un bloque try-catch
- Las respuestas típicas son registrarla, o si se considera "imposible", deben ser lanzadas como AssertionError).

```
try {
    int i = Integer.parseInt(response);
    return handleNumericResponse(i);
} catch (NumberFormatException ok) {
    //no es numerico; está bien, sólo continúa
}
return handleTextResponse(response);
```

c. Miembros estáticos
   i. Revisar referencias
      - Cuando se revisa una referencia a un miembro de una clase estática, se califica con el nombre de esa clase, no con una referencia o expresión del tipo de esa clase.

4. Convención de nombres
   a. Reglas en común para todos los identificadores
      i. Los identificadores solo usarán caracteres de ASCII y, en unos cuantos casos a mencionar, subrayado. Así, cada nombre de identificador válido coincide con la expresión regular “\w+”.
   b. Reglas por tipo de identificador
      i. Paquetes
         - Los nombres de los paquetes se escriben en minúsculas, con palabras consecutivas concatenadas (si usar guion bajo). Por ejemplo, “com.example.deepspace” en lugar de “com.example.deepSpace” o “com.example.deep_space”.
      ii. Clases
         - Los nombres de las clases se escriben en UpperCamelCase. Normalmente, se utilizan sustantivos o frases nominales. Por ejemplo, “Character” o “ImmutableList”.
         - Las interfaces también se pueden nombrar como sustantivos o frases nominales (por ejemplo, “List”), pero, en ocasiones, pueden ser adjetivos o frases adjetivas (por ejemplo. “Readable”). No existen reglas específicas o convenciones ya establecidas para el nombramiento de tipos de anotaciones.
         - Las clases de prueba se nombran iniciando con el nombre de la clase que se está probando, seguido del término “Test”. Por ejemplo, “HashTest” o “HashIntegrationTest”
      iii. Métodos
         - Normalmente son verbos o frases verbales. Por ejemplo, “sendMessage” o “stop”.
         - Se puede agregar guion bajo a los métodos de prueba JUnit para separar los componentes lógicos del nombre.
      iv. Constantes

- Los nombres de constantes usan CONSTANT_CASE: Todas las letras en mayúsculas y cada palabra separada por un guion bajo.
- Las constantes son campos estáticos cuyo contenido es inmutable y cuyos métodos no tienen efectos secundarios notables. Esto incluye datos primitivos, cadenas, tipos inmutables y colecciones de éstos últimos. Si alguno de los estados observables de la instancia se puede cambiar, no es una constante.

v. Campos no constantes
- Normalmente, son sustantivos o frases nominales. Por ejemplo, “computedValues” o “index”.

vi. Parámetros
- Se deben evitar los nombres de un carácter en los métodos públicos

vii. Variables locales
- Incluso cuando son finales e inmutables, estas variables no se consideran constantes y no se deben diseñar así.

viii. Tipo de variable
- Cada tipo de variable se nombra como uno de los siguientes estilos:
  - Una sola letra mayúscula, seguido opcionalmente de un solo número (por ejemplo, “E”, “T”, “X”, “T2”)
  - Un nombre de la forma usada para las clases, seguido de una letra en mayúscula.

c. Camel Case

i. Para convertir una frase a Camel Case, se comienza con el nombre en forma de prosa. Después, se sigue el siguiente proceso:
ii. Convertir la frase a ASCII simple y eliminar apostrofes
iii. Dividir el resultado en palabras divididas por espacios y cualquier puntación restante.
iv. Convertir todo a minúscula
v. Unir todas las palabras en un solo identificador

Prosa	Correcto	Incorrecto
"XML HTTP request"	XmlHttpRequest	XMLHTTPRequest
"new customer ID"	newCustomerId	newCustomerID
"inner stopwatch"	innerStopwatch	innerStopWatch
"supports IPv6 on iOS?"	supportsIpv6OnIos	supportsIPv6OnIOS
"YouTube importer"	YouTubeImporter	