Laboratorio I

I Semestre 2024

Vargas Ramírez Gustavo Isaías

200409141

Programación Orientada a Objetos

Ingeniería en Computación

Instituto Tecnológico de Costa Rica

Profesor Luis Pablo Soto Chaves

PARTE I.

1. ¿Qué es un estándar de codificación?
   Es un conjunto de criterios que se establecen para ayudar a los programadores a comunicar y colaborar eficientemente en proyectos grandes de tal forma que produzcan productos software de calidad.

2. Mencionar los 4 estándares de codificación para el uso de identificadores, sus características y ejemplos para uno.
   - Convención de nomenclatura Húngara.
     - Se utiliza una o mas letras en minúscula al inicio para identificar su rango de cobertura y el tipo de identificador.
     - Ejemplo: int iStudentNumber

   - Convención de nomenclatura Camel.
     - Cada palabra debe iniciar con mayúscula y todo lo demás en minúscula, a excepción de la primera letra que también es minúscula.
     - Ejemplo: printEmployeePaychecks( )

   - Convención de nomenclatura Pascal
     - Similar a Camel, pero la primera letra va también en mayúscula.
     - Ejemplo: public void DisplayInfo( ) ;
       String UserName( ) ;

   - Convención de nomenclatura con guiones bajos.
     - Todo en minúscula, cada palabra separada por guion bajo. _
     - Ejemplo: print_employee_paychecks( ) ;

3. ¿Qué beneficios se obtienen con la aplicación de un estándar de codificación?
   Mayor eficiencia en los códigos que se crean,

4. ¿En qué partes se divide o clasifica un estándar de codificación?
   a. Layout
   b. Naming
   c. Comment
   d. Coding.

PARTE II.

Source file basics	
Extensión del archivo	.java
Codificación del archivo	UTF-8
Nombre del archivo	Nombre sensible a mayúsculas con el mismo nombre de la clase declarada en el archivo. Solamente contiene una única clase.
Caracteres de espaciado	Se utiliza el espacio para indentación del código La tabulación no se utiliza
Secuencias de escape	En lugar de la correspondiente secuencia de escape octal, se utilizarán: \b, \t, \n, \f, \r, \", \' y \\
Source file structure	
Orden de los elementos del archivo	• Información de derechos de autor • Paquete al que pertenece el archivo • No se realiza ajuste de línea (line-wrap)

	• Importación de clases
	  • No se realiza importación con wildcards (.*)
	  • No se realiza ajuste de línea (line-wrap)
	• Código de la clase

Estructura de la clase	• Debe existir un orden y justificación de cómo se agregan los métodos en cada clase. Sea este cronológico o basado en otra variable. • La sobrecarga de métodos y constructores deben colocarse de forma consecutiva.
Formating	
Uso de llaves	Se utiliza en bloques **if, else, for, do, while**; aún cuando el cuerpo se encuentre vacío o tenga una sola instrucción.
Llaves en bloques no vacíos	• No salto de línea antes de una llave de apertura. • No salto de línea después de una llave de apertura. • No salto de línea antes de una llave de cierre. • Si salto de línea después de una llave de cierre, solo si esa llave termina un método, constructor o nombre de clase.
Llaves en bloques vacíos	Las llaves deben cerrarse inmediatamente: "{}"
Identación en bloques	+2 espacios simples
Instrucciones por línea	Una instrucción por línea, seguida por un salto de línea (Enter)
Limite por columna	Tiene un límite de 100 caracteres.
Envoltura de líneas (line-wrapping)	• Lo recomendado es evitar esta práctica, pero se puede usar a discreción. • La siguiente línea debe tener indentación de +4 espacios.
Espacios en blanco verticales	Siempre aparece: • Entre miembros o inicializadores de clase consecutivos, a excepción de cuando se enumeran constantes. • Pueden aparecer en cualquier parte que mejore la lectura y organicen el código. • Son permitidas múltiples líneas en blanco consecutivas, pero no son requerimiento ni recomendadas.
Espacios en blanco horizontales	• Aparecen para separar palabras reservadas, tales como if, for catch de una apertura de paréntesis "(".

	• Para separar else, catch de una llave de cierre "}". • Antes de una apertura de llaves "{". • Para separar operadores. • Para separar ":" y la flecha "->", //.
Justificación horizontal	No es requerida.
Declaración de variables	• Solamente una variable por declaración: Int a, b; no es permitido. • Dentro de un loop "for" si es permitido. • Se deben declarar lo más cercano al punto donde van a ser utilizadas.
Arreglos	• Se pueden declarar en forma de bloque. • Las llaves cuadradas deben ir en el tipo, no en la variable. Ejemplo: String[] args. NO String args[]
Switch	• Incluyen una o más etiquetas, seguidas de una o más instrucciones cada una, las ultima puede no tener instrucciones. • Indentación de +2 espacios. • Se puede comentar //fall through si queremos indicar que la ejecución va a continuar a la siguiente etiqueta del switch. • Default label: siempre va al final, aunque no tenga código.
Anotaciones	• Anotaciones por tipo • Anotaciones de clases. • Anotaciones de métodos y constructores. • Anotaciones de campos.
Comentarios	• Se pueden usar // o /* … */ • Para comentarios multilíneas se utiliza /* … */ y cada línea intermedia inicia con un *
Modificadores	• Cuando estén presentes, aparecen en el orden recomendado por la especificación del lenguaje Java. `public protected private abstract default static final transient volatile synchronized native strictfp`

<table>
<tr><td>Literales numéricos</td><td>

- Números Long usan la L mayúscula al final, nunca minúscula: 300000000000L

</td></tr>
<tr><td colspan="2">Naming</td></tr>
<tr><td>Reglas comunes</td><td>

- Deben utilizar solamente letras y números, pocas veces _. Un identificador valido es comparado con la regex <code>\w+</code>

</td></tr>
<tr><td>Reglas por tipo de identificador</td><td>

- Package names usan solo minúsculas.
- El nombre de clases debe tener mayúscula cada primera letra que inicie una palabra: <code>Character</code> or <code>ImmutableList</code>.
- Métodos inician con minúscula y el resto de palabras inician en mayúscula: sendMessage or stop.
- Constantes todo con mayúscula: <code>UPPER_SNAKE_CASE, NUMBER</code>
- Nombres de No constantes, parámetros y variables locales: *lowerCamelCase*.

</td></tr>
<tr><td>Camel case: defined</td><td>

- Convertir la frase en palabras planas usando el lenguaje ASCII
- Poner todo en minúscula
- Finalmente poner mayúscula la primera letra de cada palabra, o la primera letra de cada palabra excepto la primera dependiendo si es upper o lower camel case.

<table>
<tr><td>"new customer ID"</td><td>newCustomerId</td></tr>
<tr><td>"inner stopwatch"</td><td>innerStopwatch</td></tr>
</table>

</td></tr>
<tr><td colspan="2">Programming Practices</td></tr>
<tr><td>@override</td><td>

- Usar siempre que un método de clase tenga precedencia sobre el método de la super clase.
- Puede omitirse cuando el método padre tiene @deprecated.

</td></tr>
<tr><td>Excepciones</td><td>

- Siempre que se usa catch debe haber una acción, en las pocas veces que se justifique no tener acción se debe comentar el motivo.
- Si el nombre de la excepción comienza con “expected”, es permitido no comentarlo.

</td></tr>
</table>

Miembros estáticos	Para refernciar un miembro estatico se utiliza el nombre de la clase y no la referencia al tipo de clase:
	`Foo aFoo = ...;`
	`Foo.aStaticMethod(); // good`
	`aFoo.aStaticMethod(); // bad`
	`somethingThatYieldsAFoo().aStaticMethod(); // very bad`

Finalizadores	No usarlos.
Javadoc	

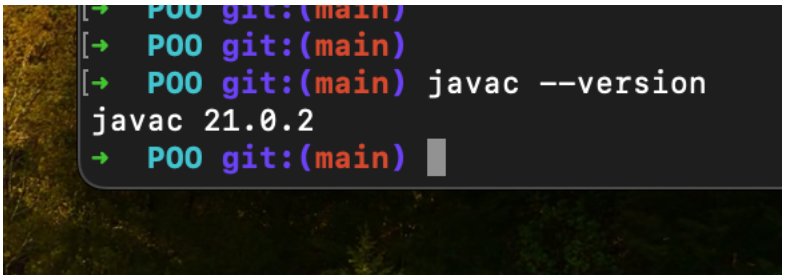
Formatting	/** * Multiple lines of Javadoc text are written here, * wrapped normally... */ public int method(String p1) { ... } o /** An especially short bit of Javadoc. */
Párrafos	• Solamente una linea en blanco con "*" separa cada párrafo. • Todos lo párrafos excepto el primero llevan <p> antes de la primera palabra sin espacio. • y <table> no vienen precedidos por <p>
Block Tags	• Los tags `@param`, `@return`, `@throws`, `@deprecated,` siempre deben tener descripción. • Si el tag no cabe en una linea, la siguiente linea debe venir con una indentación de 4 o más espacios después de la posición del @.
Summary fragment	• Todo block en Javadoc inicia con un summary fragment. • Es una frase que da contexto a la clase o método: Ejemplo: /** Returns the customer ID. */.

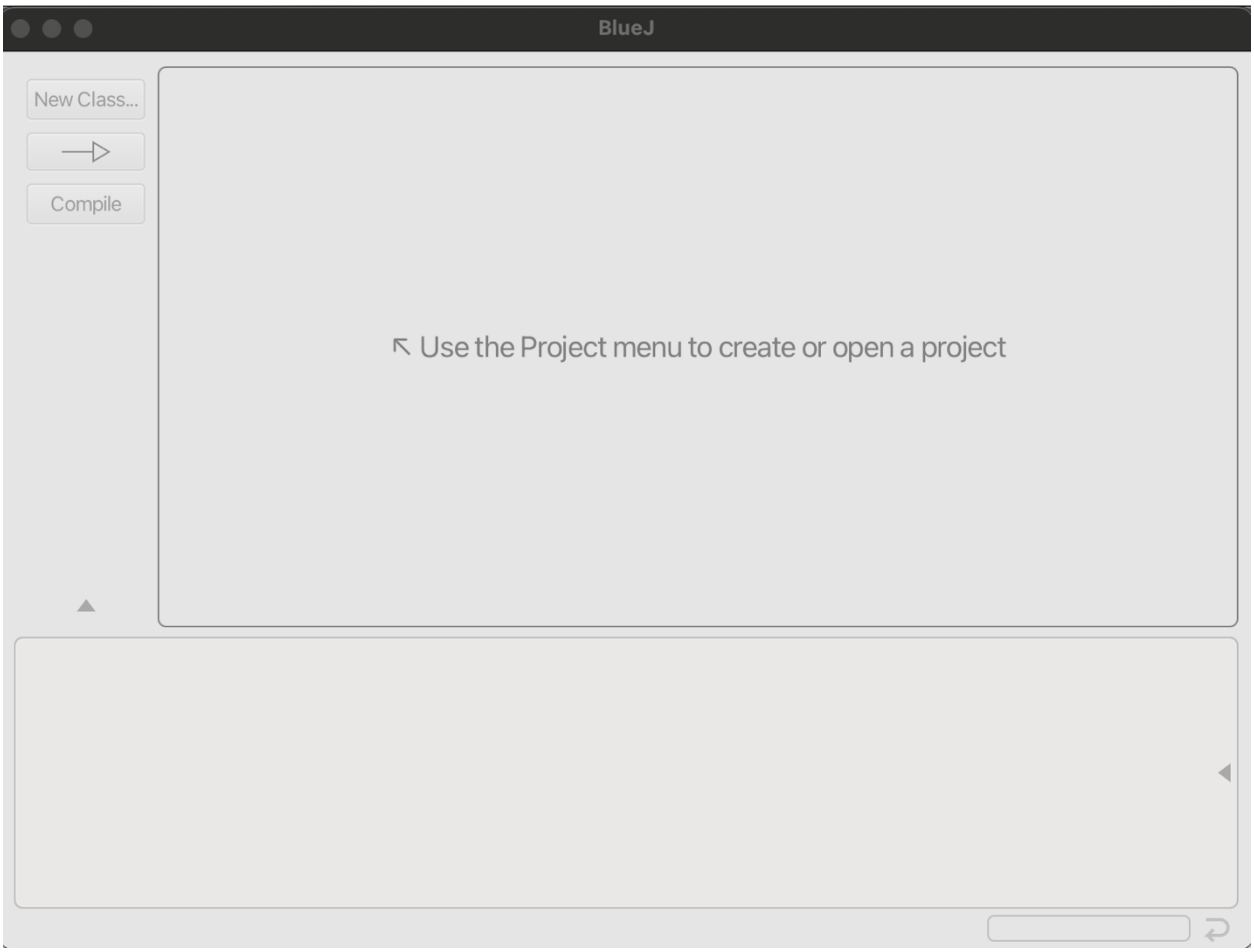
Referencias:

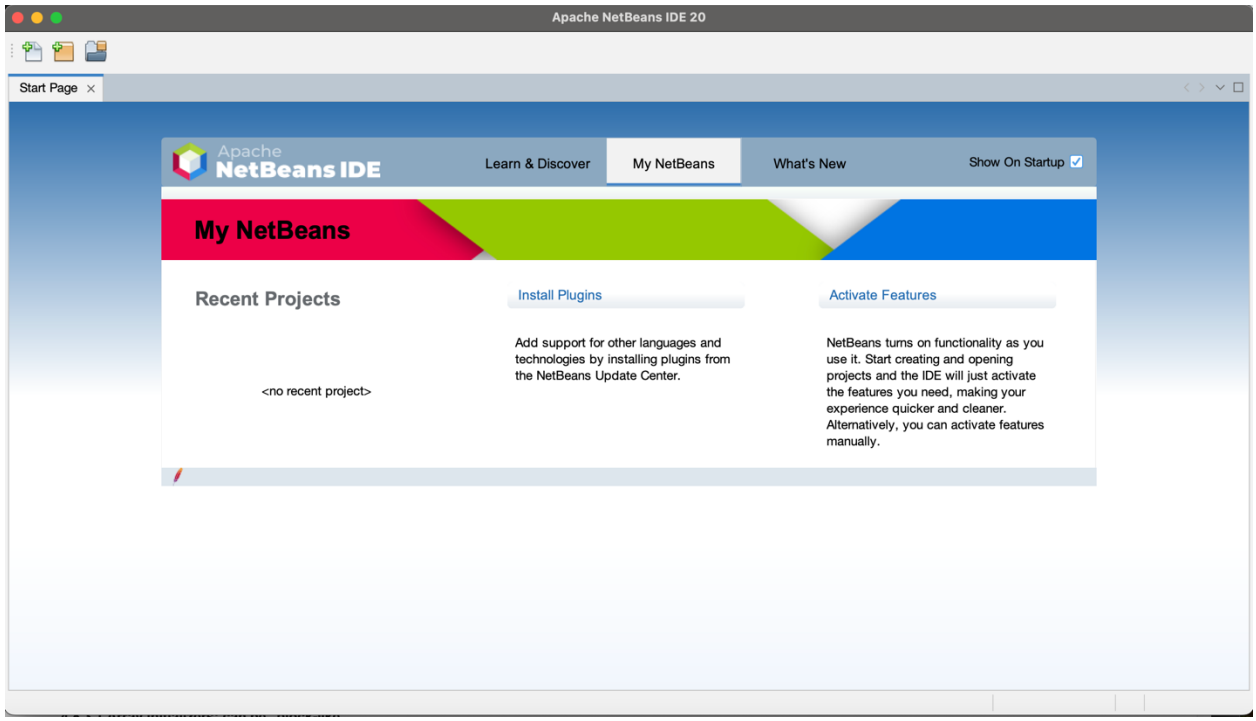
Google. (s.f.). Google Java Style Guide. Recuperado el 12 Febrero de 2024, de https://google.github.io/styleguide/javaguide.html

Wang, Y., Wang, S., Li, X., Li, H., & Du, J. (s.f). Identifier Naming Conventions and Software Coding Standards: A Case Study in One School of Software. Recuperado de https://drive.google.com/file/d/1mt5qln2TOsEuzrsg7-yf-1B5YxscKQnq/view

EVIDENCIAS


POO git:(main)
POO git:(main) javac --version
javac 21.0.2
POO git:(main)


BlueJ
New Class...
Compile
Use the Project menu to create or open a project


Apache NetBeans IDE 20
Start Page
Apache NetBeans IDE
Learn & Discover
My NetBeans
What's New
Show On Startup
My NetBeans
Recent Projects
<no recent project>
Install Plugins
Add support for other languages and technologies by installing plugins from the NetBeans Update Center.
Activate Features
NetBeans turns on functionality as you use it. Start creating and opening projects and the IDE will just activate the features you need, making your experience quicker and cleaner. Alternatively, you can activate features manually.