**GUÍA DE APRENDIZAJE**

**Control de acceso**

|  |  |
| --- | --- |
| **OBJETIVO DE APRENDIZAJE** | **OA 4**. Construir aplicaciones computacionales basadas en programación orientada a objetos, de manera de cumplir con las exigencias técnicas y de los usuarios. |
| **APRENDIZAJE ESPERADO** | **AE 5.** Construye programas, utilizando los componentes del paradigma orientado a objetos, a partir del análisis de problemáticas y requerimientos, modelando el comportamiento del sistema.**AE 1.** Construye unidades de prueba para verificar el correcto funcionamiento de la codificación realizada, de acuerdo a exigencias técnicas de confiabilidad.**AE 2.** Detecta y corrige errores de codificación, analizando el comportamiento del código de programación, de acuerdo a especificaciones y manuales de referencia. |
| **CRITERIOS DE EVALUACIÓN** | **5.1.** Construye los componentes a utilizar en el programa a partir del análisis de requerimientos para resolución de problema, resguardando los principios de POO.**1.1.** Realiza pruebas para detectar problemas previos a la codificación de una unidad de software con una herramienta de software disponible en el mercado.**1.2.** Construye una unidad de prueba para detectar excepciones en un entorno de lenguaje con una herramienta compatible que permita desarrollar, configurar e Implementar aplicaciones en función del lenguaje utilizado.**1.3.** Construye una unidad de prueba para detectar casos límites, en un entorno de lenguaje, con una herramienta compatible, propia del software utilizado como marco de trabajo.**2.1.** Utiliza la herramienta de depuración de un entorno de desarrollo, para revisar la codificación de clases programadas.**2.2.** Revisa y corrige las variables, del programa desarrollado para la solución de un caso, indagando en las funciones internas de acuerdo a protocolos de revisión.**2.3.** Revisa las variables de una función de clases programadas, utilizando opciones del lenguaje empleado, para asegurar su correcto funcionamiento. |
| **OBJETIVOS DE APRENDIZAJE GENÉRICOS** | **A – B – C – G - H** |

## CONTROL DE ACCESO

Los controles de acceso, también llamados modificadores de acceso, son un conjunto de palabras clave que permiten controlar la visibilidad de los objetos (clases), atributos y métodos de una aplicación desde otras partes de la misma.

Los modificadores de acceso permiten dar un nivel de seguridad mayor a las aplicaciones, restringiendo el acceso a diferentes atributos, métodos y constructores, asegurando de esta forma una ruta especificada por el programador para acceder a la información.



**Por un lado, el nivel de acceso *public* permite acceder a los elementos desde cualquier clase, siendo el más permisivo de todos, mientras que, por otro lado, el nivel de acceso *private*, es el más restrictivo, ya que básicamente el elemento solo puede ser accedido únicamente por la misma clase.**





## ÁMBITO

En Java, al igual que en cualquier lenguaje de programación, cada variable posee un **ámbito** o ***scope***, que consiste en un segmento del programa donde la variable es válida y puede ser usada.

Cada variable declarada dentro de los corchetes de una clase **{}** con un modificador de acceso privado, pero fuera de cualquier método, tiene un alcance de clase. Como resultado, estas variables se pueden usar en cualquier parte de la clase, pero no fuera de ella:

public class AmbitoDeClase{
 private int monto = 0;
 public void metodo() {
 monto++;
 }
 public void otroMetodo() {
 int otroMonto = monto + 4;
 }
}

Cuando se declara una variable dentro de un método, tiene alcance de método y solo será válida dentro del mismo:

public class AmbitoDeMetodo{
 public void metodoA() {
 int area = 2;
 }
 public void metodoB() {
 *// Error de compilación, “area” no está definida como variable*
 area = area + 2;
 }
}

Si declaramos una variable dentro de un bucle, solo estará disponible dentro del él:

public class AmbitoDeBucle{
 public static void main(String[] args) {
 int num = 5, fact = 1;
 for(int i = 1; i<=num; i++)
 {
 fact \*= i;
 }
 *//Error de compilación, "i" no está definida como variable*
 System.out.println(i);
 }
}

## THIS

La palabra clave ***this***puede ser usada para invocar a un constructor. Sin embargo, su uso más frecuente en Java tiene lugar en otro contexto: cuando existe sobrecarga de nombres. La sobrecarga de nombres se da cuando tenemos una variable local de un método o constructor, o un parámetro formal de un método o constructor, con un nombre idéntico al de un campo de la clase.

Sin ***this***:

public class Bebe {
 int comida;
 void alimentar (int comida) {
 *//¿Cuál comida es cuál?*
 comida = comida + comida;
 }
}

Con ***this***:

public class Bebe {
 int comida;
 void alimentar (int comida) {
 *//this hace referencia a la variable de la clase*
 this.comida = this.comida + comida;
 }
}

**ENCAPSULAMIENTO**

En Java, los **getters** y **setters** son dos métodos convencionales que son usados para obtener y actualizar el valor de una variable. El siguiente código es un ejemplo de una clase con una variable privada y su **get/set**.

public class MetodosGetSet {
 private int numero;

 public int getNumero() {
 return this.numero;
 }
 public void setNumero(int n) {
 this.numero = n;
 }
}

Dado que la variable **“numero”** es privada, el código desde el exterior de esta clase no puede acceder a ella directamente. Por el contrario, el código externo si puede invocar a los métodos **“get”** y **“set”** para tanto lectura como escritura de la variable.

En resumen, el método **“get”** es un método que lee la variable de la clase y el método **“set”** actualiza su valor. De esta manera, el programador puede controlar cómo se accede a sus variables importantes y cómo se actualizan de manera adecuada, añadiendo una capa de seguridad al código. Esta buena práctica en programación también es conocida como **encapsulamiento**.

## RETOS

|  |
| --- |
| **¡Ya trabajaste en los Retos anteriores!** **Ahora es tu turno y aplica los pasos efectuados para los siguientes desafíos.****Recuerda consultar con el/la docente en caso dudas o comentarios.** |

1. Michi-Salud es una prestigiosa clínica de gatos que desea modernizar la administración de sus clientes, y te contrataron para entregar una solución tecnológica. Sigue las siguientes indicaciones:
	1. Crea una clase Gato que siga las siguientes condiciones
		* Atributos: ID, nombre raza, edad, sexo, peso.
		* El atributo ID es un identificador único para cada gato y no puede haber copias de ese número en las instancias creadas.
	2. No pueden acceder directamente a los datos (utilizar técnicas de encapsulamiento).
	3. Constructores
		* Constructor 1: Por defecto. El ID se genera automáticamente.
		* Constructor 2: Con los atributos creados anteriormente. El ID debe generarse automáticamente.
	4. Métodos a implementar
		* **esHembra:** retorna si el gato es Hembra o Macho.
		* **esObeso:** devuelve si el gato está con sobrepeso (sobre 11.2 kilos).
		* **toString:** devuelve la información del Gato.
	5. La clase ejecutable debe hacer lo siguiente:
		* Crea 2 objetos de la clase Gato, usando los 2 constructores mencionados.
		* Para el objeto que no tiene definidos sus atributos, usar los métodos set de la clase.
		* Por cada objeto creado, comprobar su sexo, si es obeso y la información de cada Gato.
		* **Bonus 1:** Crear un programa que pida los datos por teclado y entregue por pantalla el resultado de las funcionalidades implementadas.
		* **Bonus 2:** ¿Qué otro método puede ser implementado? ¡Usa tu imaginación!
	6. Entregable



* 1. Consideraciones









1. Actualmente algunos Navegadores web sugieren generar una contraseña con **L3Tr4$ 4L3aToR!@s** para que ésta sea una [**contraseña fuerte**](https://securityinside.info/hablemos-de-contrasenas-fuertes-lo-tenemos-claro/).
2. Deberán crear una clase Usuario con los atributos nombre y contraseña. Ambos atributos deben estar encapsulados.
3. La creación de la contraseña se gestiona mediante un método que la crea automáticamente de **MANERA ALEATORIA**
	* + Características de la contraseña:
			1. **Números:** 0123456…
			2. **Minúsculas:** qazwsxedcrfvtgb...
			3. **Mayúsculas:** UJMYHNTGBRFV...
			4. **Caracteres especiales:** )(/&%$#.[]¨\*?¡
		+ **Sugerencia:** Usar [Random](https://docs.oracle.com/javase/8/docs/api/java/util/Random.html).
		+ **Bonus:** Crear un [Login](https://www.lawebdelprogramador.com/diccionario/login/#:~:text=Entrada%20de%20identificaci%C3%B3n%2C%20conexi%C3%B3n.,Sin%C3%B3nimo%20de%20nombre%20de%20usuario) con el nombre de usuario y contraseña creada automáticamente
4. Entregable



1. Consideraciones













## RECURSOS

1. Programación ATS. Módificadores de acceso <https://www.youtube.com/watch?v=1MOjUdz0Y3Q&feature=emb_title>
2. Programación ATS. Encapsulamiento y métodos accesores <https://www.youtube.com/watch?time_continue=1&v=8LyjCiuynKw&feature=emb_title>
3. Programación ATS. Curso completo de programación en Java: <https://www.youtube.com/watch?v=2ZXiuh0rg3M&list=PLWtYZ2ejMVJkjOuTCzIk61j7XKfpIR74K>
4. TutorialesYa. Curso completo de programación en java: <http://www.tutorialesprogramacionya.com/javaya/>
5. La Geekipedia de Ernesto. Curso de Java desde cero:
<https://www.youtube.com/watch?v=L1oMLsiMusQ&list=PLyvsggKtwbLX9LrDnl1-K6QtYo7m0yXWB>
6. Java desde cero:
<https://javadesdecero.es/>
7. w3schools:
<https://www.w3schools.com/java/default.asp>
8. CodingBat:
<https://codingbat.com/java>
9. Learn Java:
<https://www.learnjavaonline.org/>
10. CodeGym:
<https://codegym.cc/>
11. Programming Skills:
<http://www.pskills.org/corejava.jsp>
12. Tutorials Point:
<https://www.tutorialspoint.com/java/java_generics.htm>
13. Java Tests:
<https://www.studytonight.com/tests/?subject=java>
14. JavatPoint:
<https://www.javatpoint.com/core-java-quiz>
15. Recopilación de ejercicios Java POO
<https://www.academia.edu/36182478/Recopilaci%C3%B3n_de_Ejercicios_Java_POO_Programaci%C3%B3n_Orientada_a_Objetos>
16. Curso de programación orientada a objetos: <https://www.curriculumnacional.cl/614/w3-article-209545.html>
17. Curso de desarrollo de videojuegos: <https://www.curriculumnacional.cl/614/w3-article-209538.html>
18. Curso de desarrollo de aplicaciones móviles: <https://www.curriculumnacional.cl/614/w3-article-209536.html>
19. makigas. Tutorial de GIT: <https://www.youtube.com/watch?v=jSJ8xhKtfP4&list=PLTd5ehIj0goMCnj6V5NdzSIHBgrIXckGU>

## REFERENCIAS

1. MIT. (2010). Introduction to Programming in Java, Lecture 5: Access Control, Class Scope, Packages, Java Api. Recuperado 20 de agosto de 2020, de <https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-092-introduction-to-programming-in-java-january-iap-2010/lecture-notes/MIT6_092IAP10_lec05.pdf>
2. Velarde, O., Murillo, M., Gómez, L., & Castillo, F. (2006). Introducción a La Programación Orientada a Objetos. Pearson.