

Programación Orientada a Objetos

Guía didáctica



Unidad Académica Técnica y Tecnológica

Tecnología Superior en Transformación Digital de Empresas

Programación Orientada a Objetos

Guía didáctica

Carrera	PAO Nivel
▪ <i>Tecnología Superior en Transformación Digital de Empresas</i>	I

Autor:

Bustamante Granda Wayner Xavier



D S O F _ 1 0 6 6

Asesoría virtual
www.utpl.edu.ec

Universidad Técnica Particular de Loja

Programación Orientada a Objetos

Guía didáctica

Bustamante Granda Wayner Xavier

Diagramación y diseño digital:

Ediloja Cía. Ltda.

Telefax: 593-7-2611418.

San Cayetano Alto s/n.

www.ediloja.com.ec

edilojacialtda@ediloja.com.ec

Loja-Ecuador

ISBN digital - 978-9942-39-805-5



Reconocimiento-NoComercial-CompartirIgual 4.0 Internacional (CC BY-NC-SA 4.0)

Usted acepta y acuerda estar obligado por los términos y condiciones de esta Licencia, por lo que, si existe el incumplimiento de algunas de estas condiciones, no se autoriza el uso de ningún contenido.

Los contenidos de este trabajo están sujetos a una licencia internacional Creative Commons – **Reconocimiento-NoComercial-CompartirIgual 4.0 (CC BY-NC-SA 4.0)**. Usted es libre de **Compartir** – copiar y redistribuir el material en cualquier medio o formato. **Adaptar** – remezclar, transformar y construir a partir del material citando la fuente, bajo los siguientes términos: **Reconocimiento**– debe dar crédito de manera adecuada, brindar un enlace a la licencia, e indicar si se han realizado cambios. Puede hacerlo en cualquier forma razonable, pero no de forma tal que sugiera que usted o su uso tienen el apoyo de la licenciante. **No Comercial**-no puede hacer uso del material con propósitos comerciales. **Compartir igual**-Si remezcla, transforma o crea a partir del material, debe distribuir su contribución bajo la misma licencia del original. No puede aplicar términos legales ni medidas tecnológicas que restrinjan legalmente a otras a hacer cualquier uso permitido por la licencia. <https://creativecommons.org/licenses/by-nc-sa/4.0/>

Índice

1. Datos de información	8
1.1. Presentación de la asignatura	8
1.2. Competencias genéricas de la UTPL.....	8
1.3. Competencias específicas de la carrera	8
1.4. Problemática que aborda la asignatura	8
2. Metodología de aprendizaje	9
3. Orientaciones didácticas por resultados de aprendizaje	10
 Primer bimestre.....	 10
 Resultado de aprendizaje 1 y 2	 10
 Contenidos, recursos y actividades de aprendizaje.....	 10
 Semana 1	 11
 Unidad 1. Programación estructurada.....	 11
1.1. Introducción programación.....	11
1.2. Pensamiento computacional.....	11
1.3. Programación estructurada.....	12
1.4. Pseudocódigo	14
1.5. Estructuras de datos	14
1.6. Mini especificaciones:.....	23
1.7. Algoritmos:.....	23
Actividades de aprendizaje recomendadas	24
 Semana 2	 25
1.8. Estructura secuencial:.....	25
1.9. Estructuras de selección.....	26
1.10.If	26
1.11.Switch.....	27
1.12.Estructuras repetitivas: Do-While y While	29
Actividades de aprendizaje recomendadas	33
 Semana 3	 33
1.13.Estructuras repetitivas: for.....	33
Actividades de aprendizaje recomendadas	36

Semana 4	36
1.14.Arreglos	36
Actividades de aprendizaje recomendadas	38
Semana 5	39
1.15.Variables globales, locales y parámetros.....	39
1.16.Funciones y métodos:	40
Actividades de aprendizaje recomendadas	43
Autoevaluación 1	44
Semana 6	50
Unidad 2. Programación orientada a objetos.....	50
2.1. Introducción a programación orientada a objetos	50
2.2. Tipos de datos objetos	50
2.3. Atributos.....	52
2.4. Métodos	52
2.5. Modificadores	53
Actividades de aprendizaje recomendadas	54
Semana 7	54
2.6. Objetos	55
2.7. Clases	56
2.8. Constructor.....	57
Actividades de aprendizaje recomendadas	62
Autoevaluación 2.....	64
Semana 8	67
Actividades finales del bimestre	67
Actividades de aprendizaje recomendadas	67
Segundo bimestre	69
Resultado de aprendizaje 3.....	69
Contenidos, recursos y actividades de aprendizaje.....	69

Semana 9	69
 Unidad 3. Programación orientada a objetos avanzada	69
3.1. Programación avanzada	69
3.2. Herencia	70
Actividades de aprendizaje recomendadas	71
Semana 10	72
3.3. Ejercicios con herencia:	72
Actividades de aprendizaje recomendadas	73
Semana 11	73
3.4. Polimorfismo:.....	73
Actividades de aprendizaje recomendadas	75
Semana 12	75
3.5. Herencia y polimorfismo:	75
Actividades de aprendizaje recomendadas	77
Semana 13	78
3.6. Métodos y sus tipos:	78
3.7. Parámetros en los métodos.....	81
3.8. ArrayList con objetos.....	83
Actividades de aprendizaje recomendadas	84
Semana 14	85
3.9. Ambientes de desarrollo cloud:	85
3.10.AWS cloud 9	86
Actividades de aprendizaje recomendadas	88
Semana 15	88
3.11.Gestores de paquetes java: Maven:.....	89
3.12.Configurar el uso del AWS SDK para Java	92
3.13.Gestores de paquetes java: Gradle	95
3.14.Compile y ejecute el código:	98
Actividades de aprendizaje recomendadas	100

Autoevaluación 3	101
Semana 16	105
Actividades finales del bimestre	105
Actividades de aprendizaje recomendadas	105
4. Glosario.....	107
5. Solucionario	108
6. Referencias bibliográficas	112
7. Anexos	113



1. Datos de información

1.1. Presentación de la asignatura



1.2. Competencias genéricas de la UTPL

- Comunicación en inglés.

1.3. Competencias específicas de la carrera

1. Desarrolla aplicaciones empresariales aplicando enfoques centrados en la nube.

1.4. Problemática que aborda la asignatura

Con el avance de la tecnología y la creciente demanda de las organizaciones por automatizar sus procesos, se ha generado una alta demanda para el desarrollo de soluciones informáticas (software) sean ellas: móviles, web y de escritorio. La asignatura de Programación Orientada a Objetos con el

apoyo de un proceso de enseñanza / aprendizaje sólido, basado en el uso de: buenas prácticas, técnicas y metodologías; pretende generar habilidades que le permitan al alumno desarrollar: pensamiento computacional, pensamiento crítico, crear soluciones de software con complejidad, escalabilidad, modularidad y reutilización de código, preceptos de la programación orientada a objetos, lo que la convierte en una metodología de programación eficaz y eficiente.



2. Metodología de aprendizaje

La metodología de aprendizaje a implementar para la asignatura de Fundamentos del *Hardware* mezcla las bondades del aprendizaje basado en competencias en donde se fomenta el desarrollo de un conjunto de habilidades tanto generales como específicas que le sean de provecho durante su formación profesional, para cada lección existen actividades que evaluarán su entendimiento de los contenidos y el desarrollo de las competencias necesarias a través de rúbricas, con el aprendizaje basado en proyectos por lo cual el estudiante deberá escoger al inicio de la asignatura uno de los casos sobre el cual irá desarrollando cada una de las actividades que le permitirán al final del curso la entrega de un programa computacional aplicando los paradigmas fundamentales de la programación orientada a objetos.



3. Orientaciones didácticas por resultados de aprendizaje



Primer bimestre

Resultado de aprendizaje 1 y 2

- Utiliza el pensamiento computacional para la resolución de problemas.
- Establece estructuras de programas a través de objetos.

Para comprender integralmente todos los aspectos de la programación, es necesario iniciar con conceptos fundamentales de programación, para luego desarrollar un pensamiento computacional, mismo que nos va a servir a lo largo de toda la carrera.

Para abordar adecuadamente los resultados de aprendizajes tanto de la unidad I y II desarrollados en el primer bimestre, es importante contemplar los siguientes conceptos: programación, pensamiento computacional, programación estructurada, pseudocódigo, estructuras de datos, operaciones aritméticas, estructuras secuenciales, de selección y repetitivas. Además, se tiene previsto revisar temáticas con tipos de datos no primitivos como son los arreglos, y al finalizar la unidad I se cierra con funciones y métodos.

Finalmente, las temáticas abordadas en la unidad II, se inicia con el nuevo paradigma de programación orientada a objetos, donde se contempla la revisión de tipos de datos, objetos, clases y constructores.

Contenidos, recursos y actividades de aprendizaje



Semana 1

En el primer bimestre de la asignatura de Programación Orientada a Objetos, se inicia con el desarrollo de habilidades que le permitan al estudiante adquirir un pensamiento computacional idóneo para resolver problemas mediante el desarrollo de aplicativos, generando en ellos pensamiento computacional basado en lógica de programación. Es así como en la unidad uno se contempla revisar conceptos fundamentales de programación estructurada, uso de variables, estructuras de selección y repetitivas, arreglos y funciones.

Toda vez que el alumno desarrolle las habilidades planteadas en la unidad uno, se desea propiciar nuevas habilidades propias que exige el paradigma de programación orientada a objetos, es ahí donde, la Programación Orientada a Objetos toma su real dimensión propia de la asignatura. Este paradigma contempla la revisión de temas como: atributos, métodos, objetos, clases, constructores y modificadores, nos permitirá alcanzar los resultados de aprendizajes planteados.

Unidad 1. Programación estructurada

1.1. Introducción programación

Para iniciar el estudio es importante tener claro el concepto de programa, (López, 2013) menciona que “un programa es un conjunto de instrucciones que guían a la computadora para resolver algún problema o realizar alguna actividad”; y que además se compone de tres elementos fundamentales: los hechos representados por datos a lo que le llama “estructura de datos”, las acciones que el computador sabe – hacer llamadas operaciones primitivas elementales y las estructuras de control que consiste en las formas lógicas como un computador entienden los programas desarrollados.

1.2. Pensamiento computacional

Desarrollar un pensamiento computacional resulta necesario para todo programador y se refiere a la capacidad del programador para crear una

solución que pueda ser interpretada lógicamente por un computador. Esta interpretación contempla:

- Que el programador pueda analizar de manera lógica la problemática planteada y sepa cómo estructurar la solución.
- Y que cree una solución usando lenguajes de programación.

Existen cuatro principios del pensamiento computacional:

- **Descomposición:** que se refiere a la capacidad para desagregar un problema complejo en pequeñas partes que pueden ser mejor manejadas, analizadas y desarrolladas.
- **Reconocimiento de patrones:** identificar modelos de comportamiento de soluciones ante problemas recurrentes les ayuda a los desarrolladores poder crear algoritmos más eficientes.
- **Abstracción:** es reconocer las características y comportamiento que tienen los objetos del mundo real, para luego poder plasmarlas en un programa.
- **Algoritmo:** es una secuencia de pasos finitos que permiten resolver un problema.



Lo invito a revisar el siguiente video Microaprendizaje: [¿Qué es el pensamiento computacional?](#), que explica qué es el pensamiento computacional, información que será muy importante revisar por el alumno para que pueda asociar los conceptos con la práctica.

1.3. Programación estructurada

El término programación se refiere a la acción crear un proceso sistemático y lógico que permita dar respuesta a la problemática planteada (algoritmo), partiendo desde la identificación de información necesaria que dará respuesta dicha problemática; esta información luego es procesada a través de cálculos, operaciones, procedimientos; para finalmente obtener resultados esperados (salida).

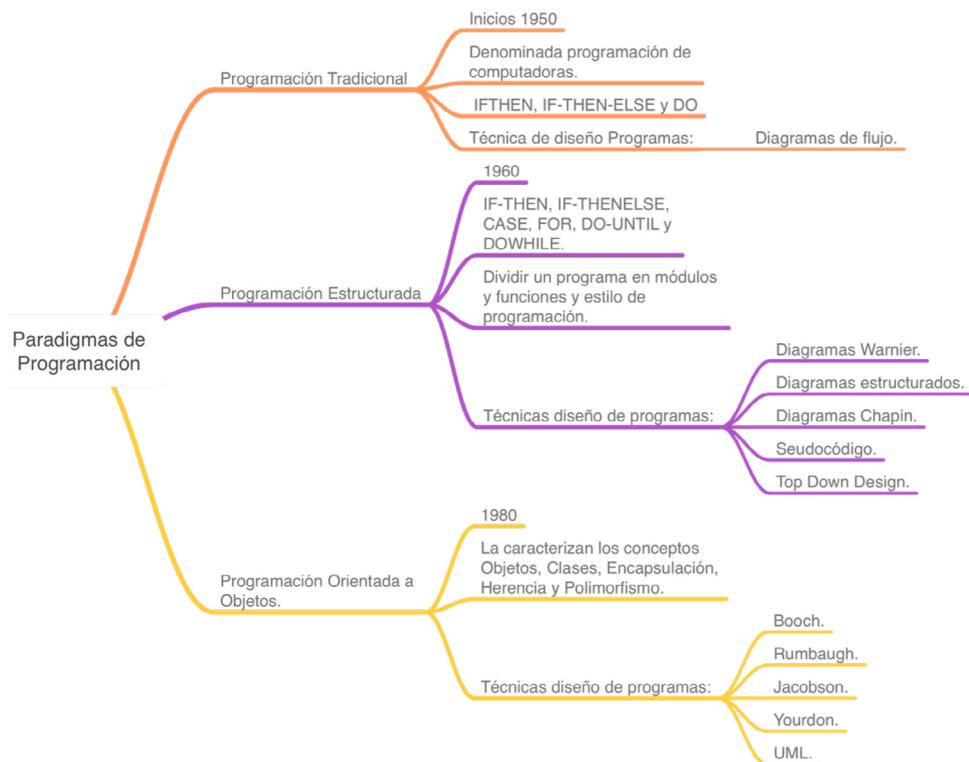
Otro término es la codificación que consiste en el proceso de transformar el algoritmo en código que el computador puede entenderlo.

Para que un programa sea correctamente diseñado, se deben contemplar ciertas características, entre ellas:

- Operatividad.
- Legibilidad.
- Transportabilidad.
- Claridad.
- Modularidad.

En la actualidad existen algunos estilos de usar la programación para resolver problemas, a ello se lo conoce como paradigmas de la programación. La figura 1 muestra una clasificación según (López, 2013).

Figura 1
Paradigmas de programación.



Nota. Adaptado de Metodología de la programación orientada a objetos (p. 9 - 12), por López, L., 2013, México: Alfaomega Grupo Editor.

1.4. Pseudocódigo

Una de las técnicas de diseño de programas en la programación estructurada es el pseudocódigo, es un lenguaje informal que no cumple con sintaxis de un lenguaje de programación y permite dar solución a un problema planteado. En el código 1 se puede apreciar su estructura.

Gráfico 1

Estructura de pseudocódigo

El pseudocódigo tiene la siguiente estructura:
Algoritmo<nombre>
 Acción1;
 Acción2;
 .
 .
 Acción n;
Fin del algoritmo.

En el código 2 muestra el pseudocódigo de un algoritmo que permita a una persona preparar una taza de café usando una cafetera.

Gráfico 2

Pseudocódigo de hacer café

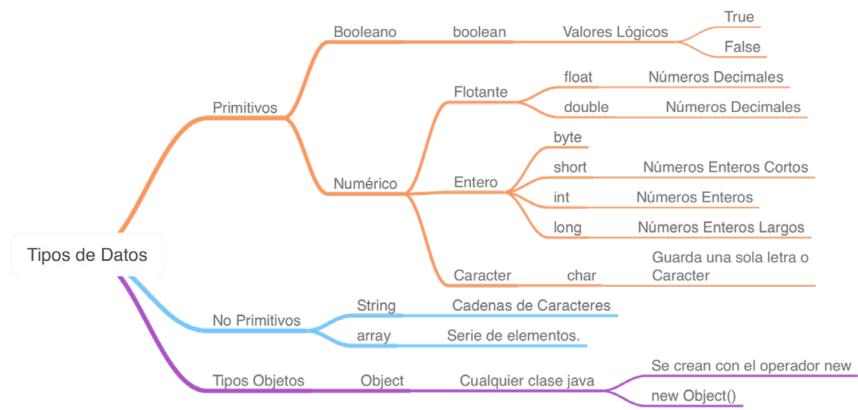
Algoritmo HacerCafe
 1. Colocar agua en la cafetera.
 2. Colocar un filtro en la cafetera.
 3. Colocar 3 cucharadas de café sobre el filtro.
 4. Encender la cafetera.
 5. Esperar que el agua empiece a hervir y pasar por el filtro.
 6. Colocar el café en una taza.
 7. Endulzar al gusto.

Existen muchas formas de solucionar un problema mediante algoritmos, para ello se debe considerar elementos básicos como: estructuras de datos, operaciones primitivas y las estructuras de control.

1.5. Estructuras de datos

Son los tipos de datos que se usan para que una computadora pueda interpretar. En la figura 2, se muestra los tipos de datos usados en el lenguaje de programación java:

Figura 2
Tipos de datos.



Nota. Adaptado de Java - Curso práctico de formación para la preparación del examen de certificación Java SE Programmer I (p. 30 - 36), por Sierra, A., 2018, Madrid, España: Alfaomega Grupo Editor.

Estos datos son almacenados generalmente en variables, quienes guardan valores en memoria dependiendo de tipo de datos que se identificó.

Lectura



Para profundizar los conceptos de tipos de datos se sugiere revisar el capítulo II: Tipos de Datos del libro de (Sierra, 2018) páginas 23-51.

1.5.1. Variables:

Sirven para representar y manejar datos, como su nombre lo indica, estas variables cambian su información ya sea por procesos calculados o generados. Se recomienda dar el nombre de una variable utilizando la primera letra en mayúscula y el resto en minúscula, si el nombre de la variable es compuesto, se recomienda que la primera letra de cada palabra se escriba en mayúscula, a continuación, algunos ejemplos: AlumnoGrado, DocenteInvitado, DocenteTitular.

Además, cada variable está asociada a un tipo de dato, a partir de ello dicha variable deberá ser manejada por el mismo tipo de dato, es decir que, si se

declara una variable de tipo entero, no corresponde usar esta variable con valores del alfabeto o símbolos, ni tampoco número fraccionario.

1.5.2. Constantes

Son valores que no varían y se usa la palabra reservada final antes del tipo de dato. Ejemplo: final $\pi = 3,1416$.

1.5.3. Operaciones primitivas

Son las acciones básicas que el computador puede realizar, a continuación, se detallan las siguientes operaciones:

1.5.4. Declarar constantes y variables

La figura 3, muestra un ejemplo de cómo declarar variables y constantes, además muestra cómo se usan los tipos de datos.

Figura 3

Declarar variables y constantes.

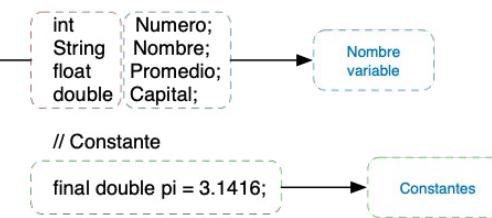
```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/
license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Project/
Maven2/JavaApp/src/main/java/${packagePath}/${mainClassName}.java to edit this template
 */

package com.mycompany.programacionestructurada;

/**
 *
 * @author xavicrip
 */
public class ProgramacionEstructurada {

    public static void main(String[] args) {
        System.out.println("Bienvenidos a Programación!");
        // Declarar Variable
        int Numero;
        String Nombre;
        float Promedio;
        double Capital;
    }

    // Constante
    final double pi = 3.1416;
}
```



Nota. Bustamante, W., 2023.

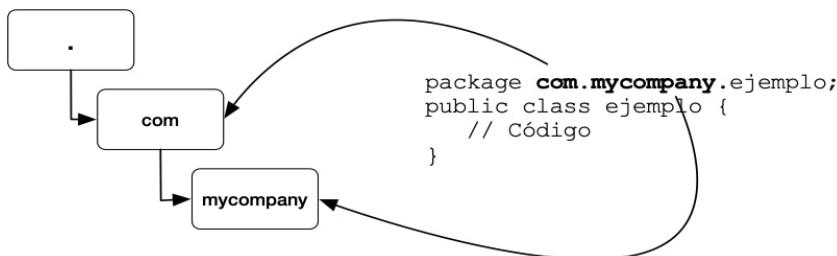
- Lectura de datos:** esta operación nos permite ingresar información al computador a través de sus variables o constantes. Los dispositivos de entrada son: teclados, escáner, mouse, etc.; y permiten ingresar datos.
- Salida de datos:** son los datos que la computadora, guarda, proceso y los muestra mediante periféricos de salida, entre ellas la pantalla e impresora.

Para desarrollar el primer ejemplo que se presenta a continuación, existen unas particularidades que son necesarias conocerlas y que se detallan a continuación:

- **Package:** es la primera sentencia o línea de código del archivo .java.

El nombre del paquete establece el nombre de las carpetas donde se alojan los módulos del código, en la figura 4 se detalla el siguiente sistema de archivos en Maven.

Figura 4
Estructura de paquetes en Maven



Nota. Adaptado de Java - Curso práctico de formación para la preparación del examen de certificación Java SE Programmer I (p. 7), por Sierra, A., 2018, Madrid, España: Alfaomega Grupo Editor.

- La palabra reservada **import**, nos permite importar diferentes clases, librerías, paquetes; que se utilizarán en el programa, además sirve para facilitar la modularidad del código. Se coloca entre el paquete (package) y antes de la clase.
- **Clases:** son la base para la programación orientada a objetos, y es una plantilla o modelo que permite crear objetos. En la unidad II, se profundizará más del tema. Las clases se definen con la palabra reservada **class** seguida del nombre de la clase entre llaves ({.....}).

Por lo general, el nombre de la clase lleva el mismo nombre del archivo .java.

- **Método main():** constituye en el método principal de una clase, permite ejecutar la Máquina Virtual de Java (JVM) cuando se lleve a ejecución una clase. Dentro de esta clase principal se coloca todo el código de un programa en java. La estructura de este método se detalla en el código 3.

Gráfico 3

Método principal.

```
Public class ejercicio1 { // clase
    public static void main (String[] args) { // método main ()
        }
}
```

- La clase **Scanner** es una librería de java. Permite **acceder a leer ficheros o datos ingresados por teclado**. Para lograrlo se debe:
 - Importar la clase Scanner:
- Import java.Util.Scanner.
 - Crear un objeto de la clase Scanner:
- Scanner entrada=new Scanner(System.in).

En la figura 5 se puede observar cómo se usa la clase Scanner para poder ingresar información por teclado.

Figura 5

Uso de clase Scanner.

```
package com.mycompany.programacionestructurada;
import java.util.Scanner; → 1. Uso de Clase Scanner
/*
 *
 * @author xavicrip
 */
public class ProgramacionEstructurada {

    public static void main(String[] args) {
        System.out.println("Bienvenidos a Programación!");

        // Declarar Variable

        int Numero;
        String Nombre;
        float Promedio;
        double Capital;

        // Constante

        final double pi = 3.1416;

        // Ingreso y salida de datos
        Scanner entrada = new Scanner(System.in); → 2. Crear un objeto Scanner
        System.out.println("Ingrese un número");
        Numero = entrada.nextInt(); ← 3. Ingreso Datos
        System.out.println("El numero ingresado es: " + Numero); ← 4. Salida Datos

    }
}
```

Nota. Bustamante, W., 2023.

La figura 6, se muestra el resultado de la ejecución del código anterior, donde un usuario ingresa por teclado un número, este es almacenado en un variable Número , para finalmente ser mostrado por pantalla.

Figura 6

Ejecución uso clase Scanner.

```
| --- exec-maven-plugin:3.0.0:exec (default-cli) @ ProgramacionEstructurada ---
Bienvenidos a Programación!
Ingrese un número
7
El numero ingresado es: 7
-----
BUILD SUCCESS
-----
Total time: 11.560 s
Finished at: 2022-11-27T23:46:00-05:00
-----
```

Nota. Bustamante, W., 2023.

Un archivo .java debe tener una sola clase pública, pero también puede tener muchas más clases de otro tipo (*private, protected, abstract*).

1.5.5. Operaciones aritméticas fundamentales:

Son operaciones que permite manipular o modificar los datos de entrada, se clasifican según el tipo de dato que va a operar. Entre las más comunes tenemos:

Operadores aritméticos:

Tabla 1

Operadores aritméticos.

Operación		Ejemplos:
Suma	+	$x = m + n$ $x = 3 + 7$ $x = 10$
Resta	-	$x = m - n$ $x = 3 - 7$ $x = -4$
Multiplicación	*	$x = m * n$ $x = 3 * 7$ $x = 21$
División real	/	$x = m / n$ $x = 3 / 7$ $x = 0,43$
División Entera	\	$x = m \ n$ $x = 3 \ 7$ $x = 0$
Mod	%	$x = m \% n$ $x = 7 \% 3$ $x = 1$
Asignación	=	$x = 7$
Cambio de signo	-	$-x$
Incremento en uno	++	$x++$
Decremento en uno	--	$x--$

Nota. Adaptado de Java - Curso práctico de formación para la preparación del examen de certificación Java SE Programmer I (p. 51-57), por Sierra, A., 2018, Madrid, España: Alfaomega Grupo Editor.

Operadores Relacional:

Tabla 2*Operadores relationales.*

Operación		Ejemplos:
Igual	<code>==</code>	<code>x == "HOLA";</code>
Distinto	<code>!=</code>	<code>x != "SI";</code>
Menor	<code><</code>	<code>x < y 3 < 7</code>
Menor o igual	<code><=</code>	<code>x <= y 3 <= 7</code>
Mayor	<code>></code>	<code>y > x 7 > 3</code>
Mayor o igual	<code>>=</code>	<code>y >= x 7 >= 3</code>

Nota. Adaptado de Java - Curso práctico de formación para la preparación del examen de certificación Java SE Programmer I (p. 51-57), por Sierra, A., 2018, Madrid, España: Alfaomega Grupo Editor.

Operadores Lógicos:**Tabla 3***Operadores lógicos.*

Operación		Ejemplos:
NOT	<code>!</code>	<code>!x</code>
AND: es true si el valor de ambos operandos es true.	<code>&&</code>	<code>x > 70 && y >= 20 80 > 70 && 35 >= 20 TRUE</code>
OR: es true si uno de los dos operandos es true	<code> </code>	<code>x > 70 y >= 20 80 > 70 1 >= 20 TRUE</code>

Nota. Adaptado de Java - Curso práctico de formación para la preparación del examen de certificación Java SE Programmer I (p. 51-57), por Sierra, A., 2018, Madrid, España: Alfaomega Grupo Editor.

Para más información puede consultar el libro de (Sierra,2018) páginas 51 – 57, tema operadores y estructuras de decisión.

Para precisar lo mencionado en esta primera semana, a continuación, se detalla un ejercicio escrito en la figura 7.

Figura 7

Ejemplo del uso de operadores aritméticos.

```
package com.mycompany.programacionestructurada;  
import java.util.Scanner;  
  
/*  
 * @author xavicrip  
 */  
public class ProgramacionEstructurada {  
  
    public static void main(String[] args) {  
        System.out.println("Bienvenidos a Programación!");  
  
        // Declarar Variable  
  
        int Numero;  
        String Nombre;  
        float Promedio;  
        double Capital;  
  
        // Constante  
  
        final double pi = 3.1416;  
  
        // Ingreso y salida de datos  
  
        Scanner entrada = new Scanner(System.in);  
  
        System.out.println("Ingrese un número");  
        Numero = entrada.nextInt();  
  
        System.out.println("El numero ingresado es: " + Numero);  
  
        // Asignación Nombres  
        Nombre = "Juan";  
  
        // Operaciones  
        Promedio = Numero / 2;  
        System.out.println("El promedio es: " + Promedio);  
    }  
}  
  
1. Asignar Juan a la variable nombre ←  
   ↓  
   // Asignación Nombres  
   Nombre = "Juan";  
  
   // Operaciones  
   ↓  
   Promedio = Numero / 2;  
   System.out.println("El promedio es: " + Promedio); → 2. Operación Aritmética
```

Nota. Bustamante, W., 2023.

El resultado de ejecutar el código anterior se muestra en la figura 8.

Figura 8

Ejecución de ejemplo de operadores aritméticos.

```
- Bienvenidos a Programación!  
- Ingrese un número  
- 7  
- El numero ingresado es: 7  
- El promedio es: 3.0  
-----  
BUILD SUCCESS  
-----  
Total time: 4.283 s  
Finished at: 2022-11-28T00:43:57-05:00  
-----
```

Nota. Bustamante, W., 2023.

1.6. Mini especificaciones:

Son un conjunto de instrucciones secuenciales que nos permiten vislumbrar la solución a un problema planteado. Ejemplo.

Gráfico 4

Estructura de mini especificaciones.

```
Algoritmo <nombre del algoritmo>
Clase <nombre de la clase>
Declarar variables, constantes y tipos de datos.
Acción 1
Acción 2|
.
.
Acción n
Fin Clase
Fin Algoritmo
```

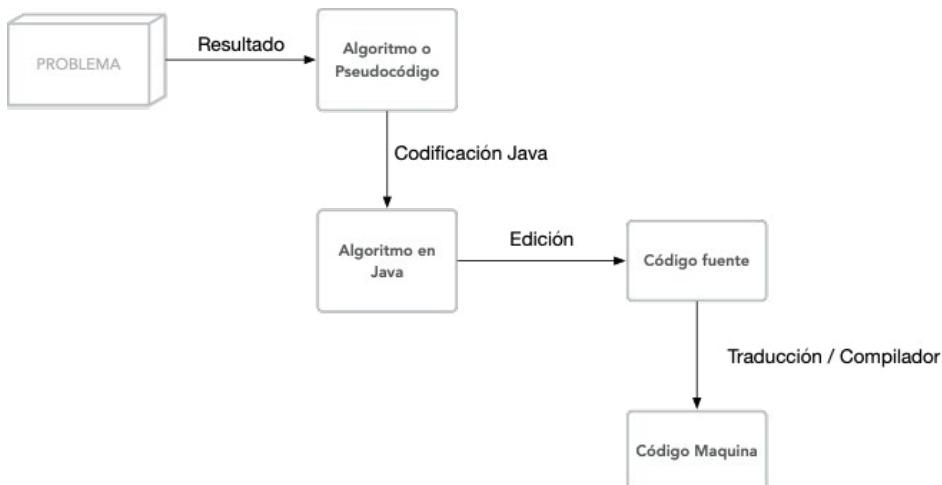
1.7. Algoritmos:

Algoritmos + Estructuras de datos = Programas; “solo se puede llegar a realizar un buen programa con el diseño de un algoritmo y una correcta estructura de datos”.

Para la resolución de un problema se debe seguir la siguiente figura.

Figura 9

Esquema de algoritmos.



Nota. Adaptado de Metodología de la programación orientada a objetos (p. 12 - 14), por López, L., 2013, México: Alfaomega Grupo Editor.

Para el desarrollo de los ejercicios y ejemplos, se va a trabajar con el IDE (Entorno de Desarrollo Integrado) de NetBeans, que se constituye en un software que permite a los desarrolladores crear código, a continuación, se detalla los pasos para su instalación.

1. Para instalar el Lenguaje de programación Java, haga clic en el siguiente enlace [ORACLE](#)
2. Para instalar NetBeans, haga clic en el siguiente enlace [Apache NetBeans 17](#)
3. Luego busque el ejemplo 1 de variables, constantes, m ini especificaciones, en el siguiente enlace [GitHub](#)



Actividades de aprendizaje recomendadas

Continuemos con el aprendizaje mediante su participación en las actividades que se describen a continuación:

1. Para complementar lo aprendido en esta semana, se sugiere revisar el siguiente recurso donde encontrará ejercicios para repasar en López, L. (2013). *Metodología de la Programación Orientada a Objetos*. Ciudad de México, México: Alfaomega, Páginas 21-34.

2. Se recomienda revisar el siguiente [Vídeo #1 - Integrar GIT en NETBEANS y sincronizar repositorios](#), para que conozca como integrar una herramienta llamada GIT al IDE de NetBeans y pueda sincronizar los ejercicios que desarrolle y guardarlos automáticamente en la nube.



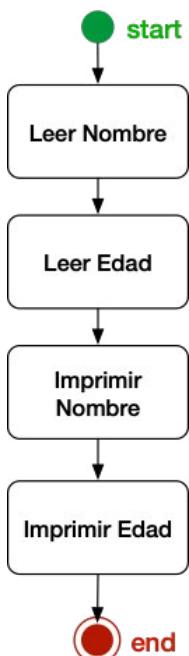
Semana 2

1.8. Estructura secuencial:

Este tipo de estructuras indican que las instrucciones se ejecutan una tras otra, y se las representa de la siguiente manera.

Figura 10

Estructura secuencial.



Nota. Bustamante, W., 2023.

1.9. Estructuras de selección

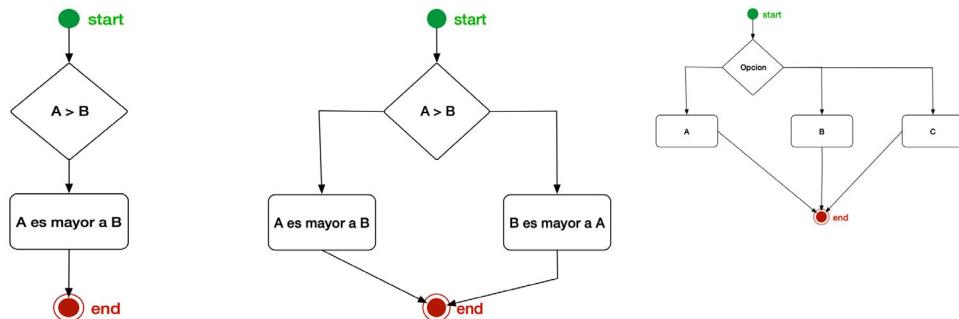
Este tipo de estructuras permiten ejecutar ciertas acciones dependientes de una condición. Indica que las instrucciones de un programa se ejecutan una después de la otra, en el mismo orden en el cual aparecen en el programa.

La estructura de selección tiene tres formas:

- Simple: una alternativa de acción.
- Doble: dos alternativas de acción.
- Múltiple: más de dos alternativas de acción.

La figura 11 muestra los tipos de estructuras de selección:

Figura 11
Diagramas de flujo condicional



Nota. Bustamante, W., 2023.

1.10. *If*

Es una instrucción que permite comprobar una condición booleana, ejecutando un bloque de sentencias cuando el resultado es *TRUE* y otro bloque de sentencias cuando el resultado es *FALSE*, tal como se muestra en la figura 11 literal condicional doble. En código se expresa de la siguiente manera.

Gráfico 5

Estructura de condicional IF.

```
if (condición){  
    // Sentencias cuando el resultado es TRUE  
}else{  
    // Sentencias cuando el resultado es FALSE  
}
```

A continuación, se detalla un ejemplo.

Gráfico 6

Ejemplo condicional IF – Números pares e impares.

```
int a = 7;  
if (a%2==0){  
    System.out.println("Es un número par");  
}else{  
    System.out.println("Es un número impar");  
}
```

El resultado de este ejercicio es:

Gráfico 7

Resultado de la ejecución del código números pares e impares.

Es un número impar

1.11. Switch

Es una instrucción que permite trabajar con múltiples alternativas, que se configuran o programan de manera independiente. Además, de manera opcional cuenta con una alternativa llamada *default*, donde se puede especificar instrucción que queden fuera de cualquiera de las alternativas personalizadas con anterioridad. La sintaxis del *switch* es.

Gráfico 8

Estructura de condicional switch.

```
Switch(variable){  
    case 1:  
        // Código.  
        break; //palabra reservada que da por finalizado el caso 1  
    case 2:  
        // Código.  
        break; //palabra reservada que da por finalizado el caso 2  
    .  
    .  
    .  
    case n:  
        // Código.  
        break; //palabra reservada que da por finalizado el caso n  
    default:  
        // Código.  
}
```

El siguiente ejercicio muestra las opciones que se presentan cuando se hace una llamada telefónica a un *call center*.

Gráfico 9

Ejemplo condicional Switch – Call center.

```
// Ejercicio call center

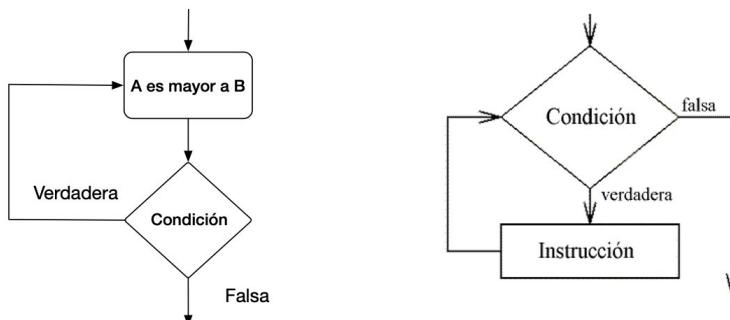
package com.mycompany.acd1;
/**
 *
 * @author xavicrip
 */
import java.util.Scanner;
public class Acd1 {
    public static void main(String[] args) {
        //Declaración de variables
        int opc;
        Scanner entrada = new Scanner(System.in);
        System.out.println("Bienvenidos a la empresa de cárnicos San
Jacinto");
        System.out.println("El asistente automático le ayudará con su
pedido");
        System.out.println("Presione 1: Si desea comprar carne de
aves");
        System.out.println("Presione 2: Si desea comprar carne de
cerdo");
        System.out.println("Presione 3: Si desea comprar carne de
res");
        opc = entrada.nextInt();
        // uso de instrucción switch
        switch(opc){
            case 1:
                System.out.println("Gracias por seleccionar la sección de
carnes de aves");
                break;
            case 2:
                System.out.println("Gracias por seleccionar la sección de
carnes de cerdo");
                break;
            case 3:
                System.out.println("Gracias por seleccionar la sección de
carnes de res");
                break;
            default:
                System.out.println("La opción digitada no es válida,
                presione 1, 2 o 3");
        }
    }
}
```

1.12. Estructuras repetitivas: *Do-While* y *While*

Son estructuras que permiten repetir algunas acciones, estas repeticiones deben ser controladas a través de condiciones, que se encargarán de dar por finalizada el ciclo repetitivo del código. La figura 12 muestra el diagrama de flujo de las estructuras repetitivas *do-while* y *while*.

Figura 12

Diagrama de flujo estructuras repetitivas do-while y while.



Do-While

Nota. Bustamante, W., 2023.

La instrucción Do-While, se ejecuta primero el bucle de sentencias y después se evalúa la condición, en esta instrucción al menos se ejecuta una sola vez las sentencias de código. A continuación, se muestra la sintaxis del código.

Gráfico 10

Estructura repetitiva do-while.

```
do{  
// sentencias o código  
} while(condición);
```

A continuación, un ejemplo que permite calcular el área de un triángulo muchas veces, hasta que el usuario desee cerrar el bucle digitando la letra "N".

Gráfico 11

Ejemplo estructura repetitiva do-while – área de un triángulo.

```
String opc="S";
float a, b, h;
Scanner entrada = new Scanner(System.in);
do{
    System.out.print("Ingrese la base del triángulo:");
    b = entrada.nextFloat();
    System.out.print("Ingrese la altura del triángulo:");
    h = entrada.nextFloat();
    // Calculo del área
    a = (b*h)/2;
    System.out.print("El área del triángulo es: " + a);
    // Condición para cerrar el bucle
    System.out.print("Desea calcular el área de otro triangulo:");
    opc = entrada.nextLine();
} while(opc=="S");
```

Mientras que la instrucción *while*, ejecuta un conjunto de instrucciones mientras cumplan con la condición específica como se muestra en la figura 12. A continuación, se muestra la sintaxis en java.

Gráfico 12

Estructura repetitiva *while*.

```
while (condición) {
// sentencias o código
}
```

A continuación, un ejemplo que permite calcular el área de un triángulo muchas veces, hasta que el usuario desee cerrar el bucle digitando la letra “N”.

Gráfico 13

Ejemplo estructura repetitiva while – área de un triángulo.

```
String opc="S";
float a, b, h;
Scanner entrada = new Scanner(System.in);
while(opc=="S") {
    System.out.print("Ingrese la base del triángulo:");
    b = entrada.nextFloat();
    System.out.print("Ingrese la altura del triángulo:");
    h = entrada.nextFloat();
    // Calculo del área
    a = (b*h)/2;
    System.out.print("El área del triángulo es: " + a);
    //Condición para cerrar el bucle
    System.out.print("Desea calcular el área de otro triangulo:");
    opc = entrada.nextLine();
}
```

Para que estos bucles no se vuelven infinitos, es importante poder cerrarlos o finalizarlos, para ello tanto para las instrucciones *while* y *do-while* es necesario usar un contador que permita cumplir con la condición para que pueda cambiar de estado *TRUE* a *FALSE* o viceversa según lo requiera.

Para el desarrollo de la práctica en la unidad I, se va a trabajar con el siguiente ejercicio: Caso ACD#1, información que la encuentra en los siguientes enlaces:



- Para descargar el enunciado del Caso ACD#1: haga clic en el siguiente [enlace POO_Tec/Caso ACD1/](#).
- Para descargar la Mini especificación del Caso ACD#1: haga clic en el siguiente [enlace POO_Tec/Caso ACD1/](#).
- Para descargar el Programa del Caso ACD#1: haga clic en el siguiente [enlace POO_Tec/Caso ACD1/](#).



Actividades de aprendizaje recomendadas

Estimado estudiante lo invito a desarrollar las siguientes actividades para afianzar sus conocimientos.

1. Para complementar lo aprendido en esta semana, se sugiere revisar el siguiente recurso donde encontrará ejercicios para repasar en Deitel, P. & Deitel, H. (2016). *Java como programar*. Ciudad de México, México: Pearson Education, Páginas 100-190.
2. Se recomienda revisar el siguiente [video Java Básico Si condicional Parte 3](#), para que conozca como se escribe código en java usando la estructura de selección – *IF*.
3. Se recomienda revisar el siguiente [video C++ Curso: 86 Programa de Ejemplo de switch con Rangos Numéricos y QUIZ](#), para que conozca como se escribe código en java usando la estructura de selección – *SWITCH*.
4. Se recomienda revisar el siguiente [video Estructura de control repetitiva con contador y acumulador](#), para que conozca como se escribe código en java usando la estructura de repetición – *DO – WHILE, WHILE, FOR*.
5. Se recomienda revisar el siguiente [video Java Ejercicio: 563 Usar un Ciclo while para Sumar los Dígitos de un Número Entero Positivo](#), para que profundizar en el uso de la estructura de repetición – *WHILE* en java.



Semana 3

1.13. Estructuras repetitivas: for

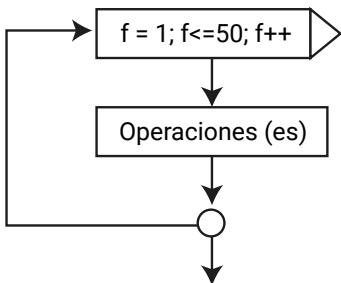
Son estructuras que permiten repetir algunas acciones, estas repeticiones deben ser controladas a través de condiciones, que se encargarán de dar por finalizada el ciclo repetitivo del código. El ciclo repetitivo for se compone de tres elementos:

- Un contador inicializado, ejemplo $i = 1$.
- Una condición, ejemplo $i < n$.
- El mismo contador incrementado con: $i++$, o decrementando con $i-$.

Lo mencionado anteriormente se representa gráficamente en la siguiente figura.

Figura 13

Diagrama de flujo estructuras repetitivas for.



Nota. Bustamante, W., 2023.

Las instrucciones for ejecutan repetidamente un conjunto de sentencias, controladas por un contador, quien delimita dicha cantidad de repeticiones. La sintaxis en java es.

Gráfico 14

Estructura repetitiva for.

```

for (inicialización; condición; incremento) {
// sentencias o código
}
  
```

A continuación, un ejemplo que permite calcular 5 veces el área de un triángulo.

Gráfico 15

Ejemplo estructura repetitiva for – área de un triángulo.

```
int i=0;
float a, b, h;
Scanner entrada = new Scanner(System.in);
for (i=0; i<5; i++) {
    System.out.print("Ingrese la base del triángulo:");
    b = entrada.nextFloat();
    System.out.print("Ingrese la altura del triángulo:");
    h = entrada.nextFloat();
    // Calculo del área
    a = (b*h)/2;
    System.out.print("El área del triángulo es: " + a);
}
```

Para el desarrollo de la práctica en la unidad I, se va a trabajar con el siguiente ejercicio: Caso ACD#1:

De acuerdo con CNNMoney.com, Facebook llegó a los mil millones de usuarios en octubre de 2012. Suponiendo que su base de usuarios crezca con una tasa del 4 % mensual.

- ¿Cuántos meses tardará Facebook en aumentar su base de usuarios a mil quinientos millones?
- ¿Cuántos meses tardará Facebook en aumentar su base de usuarios a dos mil millones?

La información Ud. la encuentra en los siguientes enlaces:



- Para descargar todo el Caso ACD#1, lo puede realizar en el siguiente [enlace POO_Tec/Caso ACD1/](#).
- Para descargar la Mini especificación del Caso ACD#1, lo puede realizar en el siguiente [enlace POO_Tec/Caso ACD1/](#).
- Para descargar el Programa del Caso ACD#1, lo puede realizar en el siguiente [enlace POO_Tec/Caso ACD1/](#).

Gráfico 16. Ejemplo caso [facebook](#).



Actividades de aprendizaje recomendadas

Continuemos con el aprendizaje mediante su participación en las actividades que se describen a continuación:

1. Para complementar lo aprendido en esta semana, se sugiere revisar el siguiente recurso donde encontrará ejercicios para repasar en Deitel, P. & Deitel, H. (2016). *Java como programar*. Ciudad de México, México: Pearson Education, Páginas 100-190.
2. Se recomienda revisar el siguiente [video Estructura de control repetitiva con contador y acumulador](#), para que conozca como se escribe código en java usando la estructura de repetición – *DO – WHILE, WHILE, FOR*.
3. Se recomienda revisar el siguiente [video ciclos en JAVA - JAVA programming](#), para que profundizar en el uso de la estructura de repetición – *FOR* en java.



Semana 4

1.14. Arreglos

Son un tipo de datos estructurados que contienen un conjunto de elementos del mismo tipo, es decir que existen arreglos de tipo entero, cadena, reales, etc.

Existe una clasificación que depende del número de dimensiones; cuando el arreglo tiene una dimensión se llaman arreglos unidimensionales, si el arreglo tiene dos dimensiones se llama arreglos bidimensionales y se llaman multidimensionales si el arreglo tiene más de dos dimensiones. A continuación, se muestra un ejemplo de estos tipos de arreglos.

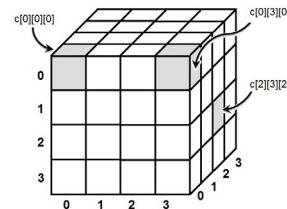
Figura 14

Tipos de arreglos.

Días	0	Lunes
1		Martes
2		Miércoles
3		Jueves
4		Viernes
5		Sábado
6		Domingo

A	0	1	2
B			
C			

Arreglo bidimensional



Arreglos tridimensionales

Arreglo unidimensional

Nota. Bustamante, W., 2023.

En java se usa la palabra reservada `array` y consiste en un conjunto de datos del mismo tipo y que son accedidos a través de índice o posición, cuyo primer elemento es la posición 0.

En la figura 14, para acceder al valor “jueves ” del arreglo Días, se accede de la siguiente manera: Días [3].

Para usar arreglos multidimensionales se usan parejas de corchetes de acuerdo con el número de dimensiones, es así como para arreglos bidimensionales usamos `<nombre_arreglo>[][]`; tridimensionales `<nombre_arreglo>[][][]`; etc.

Finalmente, para recorrer un array se usa las instrucciones `for` por cada una de las dimensiones del arreglo, es decir para arreglos bidimensionales, necesitamos dos instrucciones `for` para recorrer el arreglo por cada uno de los ejes x, y; se usa el mismo criterio para arreglos de más dimensiones.

Restructuramos el código del ejemplo anterior usando arreglos y el resultado es el siguiente.

Gráfico 17

Ejemplo caso facebook con arreglos.

```
package com.mycompany.acdarreglos;

/**
 *
 * @author xavicrip
 */
public class AcdArreglos {
    public static void main(String[] args) {
        // 1. Declaración de variables:
        double UsuariosFB = 100000000; // Inicializada la cantidad de usuarios en Facebook;
        double UsuarioFinales = 1500000000; // Usuarios Objetivo 1
        double UsuarioFinalesDos = 2000000000; // Usuarios Objetivo 2
        int FechaMes = 10;
        int FechaAnio = 2012;
        int ContadorMes = 0;
        int ContadorAnio = 0;
        int CasoMes = 0;
        String Mes = "";
        int anio = 0;
        int j = 1;
        int i = 0;
        String[] arregloMes = {
            "Enero",
            "Febrero",
            "Marzo",
            "Abril",
            "Mayo",
            "Junio",
            "Julio",
            "Agosto",
            "Septiembre",
            "Octubre",
            "Noviembre",
            "Diciembre"
        };
        while (UsuarioFinales > UsuariosFB) {
            UsuariosFB = UsuariosFB * 1.04;
            ContadorMes = ContadorMes + 1;
        }
        System.out.println("El número de meses que tarda Facebook en alcanzar 1500000000 de usuarios es: "
+ ContadorMes);
        CasoMes = FechaMes + ContadorMes;
        if (CasoMes <= 12) {
            for (i = 0; i <= arregloMes.length; i++) {
                if (i == CasoMes) {
                    Mes = arregloMes[i];
                }
            }
        } else {
            anio = CasoMes/12;
            FechaAnio = FechaAnio + anio;
            CasoMes = CasoMes % 12;
            for (i = 0; i <= arregloMes.length; i++) {
                if (i == CasoMes) {
                    Mes = arregloMes[i - 1];
                }
            }
        }
        System.out.println("Resultado del año es:" + FechaAnio);
        System.out.println("Resultado del mes es:" + Mes);
    }
}
```



Actividades de aprendizaje recomendadas

Estimado estudiante lo invito a desarrollar las siguientes actividades con el fin de afianzar sus conocimientos.

1. Para complementar lo aprendido en esta semana, se sugiere revisar el siguiente recurso donde encontrará ejercicios para repasar en

Deitel, P. & Deitel, H. (2016). *Java como programar*. Ciudad de México, México: Pearson Education, Páginas 285-358.

2. Se recomienda revisar el siguiente video [Añadir posición a un arreglo en Java](#), para que conozca como se escribe código en java usando la estructura de arreglos.



Semana 5

1.15. Variables globales, locales y parámetros

Las variables de estos tipos son fundamentales identificarlas y conocerlas, estas variables son usadas tanto en métodos como en funciones, es ahí cuando una variable de tipo global es declarada fuera del método o función y sirve para toda la clase principal, mientras que las variables de tipo local, se declara y opera dentro de la función o método, por tal motivo no puede ser usar fuera. Ejemplo:

Gráfico 18

Variables locales y globales.

```
// Ejemplo de Declaración variables globales

public class AcdArreglos {
    public static void main(String[] args) {
        // Declaración de variables:
        double UsuariosFB = 1000000000; // Inicializada Usuarios en Facebook;
        double UsuarioFinales = 1500000000; // Usuarios Objetivo 1
        double UsuarioFinalesDos = 2000000000; // Usuarios Objetivo 2
        int FechaMes = 10;
        int FechaAnio = 2012;
        int ContadorMes = 0;
        int ContadorAnio = 0;
        int CasoMes = 0;
        String Mes = "";
        int anio = 0;
        int j = 1;
        int i = 0;
    }

// Ejemplo de Declaración variables locales

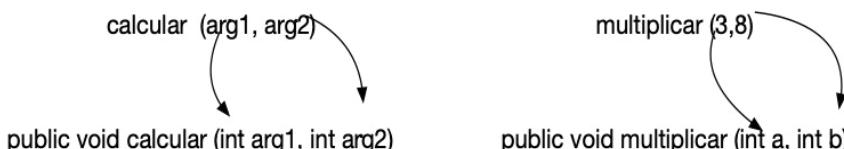
public class AcdArreglos {
    public static void main(String[] args) {
        // 1. Declaración de variables:
        int j = 1;
        int i = 0;
    int suma(){
        int suma = 0;
        suma = j+i;
        return suma;
    }
}
```

Los parámetros conocidos como argumentos sirven para conectar el código desde las variables globales a las que están dentro de un método y función (locales); existen de dos tipos, los parámetros por referencia y por valor.

Los parámetros por referencia son variables locales de un método que se define como parámetro en el encabezado de este y sirve para conectarse con otra variable de otro método mediante el envío de su dirección; mientras que los parámetros por valor son: es una variable local de un método que se define como parámetro en el encabezado de este y sirve para conectarse con otra variable de otro método mediante el envío de su valor (López, 2013).

Los argumentos tienen que coincidir con la lista de parámetros que se definen en un método o función, de esta manera los valores de los parámetros en cada argumento según el orden especificado en el método. A continuación, se detallan ejemplos.

Figura 15
Pasos de parámetros a métodos



Nota. Bustamante, W., 2023.

Como se puede apreciar en el ejemplo anterior, los argumentos de la llamada al método multiplicar deben coincidir con los parámetros definidos en el método multiplicar, es decir, 3 coincide con el parámetro a y 8 coincide con el parámetro b.

1.16. Funciones y métodos:

Los métodos y las funciones en java están en capacidad de realizar las mismas tareas, es decir, son funcionalmente idénticos, pero su diferencia radica en la manera en que hacemos uso de uno u otro (el contexto), el método actúa siempre en función a un objeto, mientras que las funciones pueden existir sin el uso de un objeto.

Para comprender el uso de funciones y métodos es importante conocer la siguiente terminología o palabras claves:

- **Modificadores:** constituye el nivel de acceso o visibilidad que se les otorga tanto a atributos como a métodos de una clase. Esta visibilidad les permite determinar el lugar desde donde los atributos y métodos pueden ser usados. Existen los siguientes: *public*, *private*, *static*, *protected*. En la figura 16 se resume el ámbito de acceso de los modificadores.

Figura 16

Modificadores de los métodos

	public	protected	default	private
Clase	● SI	● NO	● SI	● NO
Atributo	● SI	● SI	● SI	● SI
Método	● SI	● SI	● SI	● SI
Constructor	● SI	● SI	● SI	● SI

Nota. Adaptado de Java - Curso práctico de formación para la preparación del examen de certificación Java SE Programmer I (p. 132), por Sierra, A., 2018, Madrid, España: Alfaomega Grupo Editor.

En las secciones 2.3 y 2.4 de esta guía se puede ampliar más los conceptos e interpretaciones de métodos y modificadores respectivamente.

- **Encapsulación:** es un principio de la programación orientada a objetos que consiste en declarar como privados (*private*) los atributos de una clase y proporcionando acceso a los mismos a través de métodos y constructores. El objetivo de este principio es evitar que los atributos puedan ser accesibles desde otras clases externas e incluso que puedan ser sujetas a modificaciones de sus valores. (Sierra, 2018).

- **Return:** esta palabra reservada se usa para finalizar la ejecución de una función y devuelve un valor a quien llamó o usó dicha función. En java es importante mencionar que cualquier sentencia escrita después *return* no se ejecutará.

Las funciones encapsulan código que puede ser reutilizado, reciben parámetros que son usados para realizar operaciones y por lo general devolver valores (**return**). En java se usa el modificador **static** en funciones. A continuación, se describe como se detalla una función.

Gráfico 19

Ejemplo de una función – suma de dos números.

```
[acceso] [modificador] tipodeDato <nombreFunción> ([TipodeDato
<nombre_argumento1>, [TipodeDato <nombre_argumento2>],..., 
[TipodeDato <nombre_argumento>]) {
    /*
        // Bloque de instrucciones
        int suma = 0; // Variable Local
        return suma;
    */
}
```

El siguiente ejemplo corresponde a la de una función:

Gráfico 20

Estructura de una función.

```
public static int Suma (int a, int b){
    int suma = 0;
    Suma = a + b;
    return suma;
}
```

Es importante mencionar que, para usar una función o método, este necesita ser invocado o llamado usando la siguiente estructura.

Gráfico 21

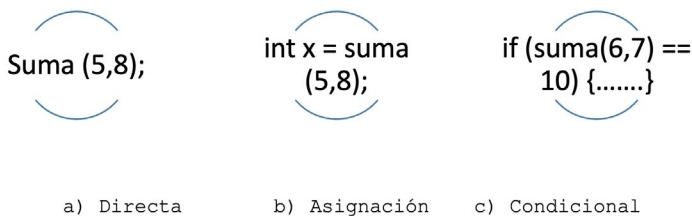
Llamado de funciones.

```
<nombreFuncion> ([valor], [valor],...);
```

La figura 17, muestra las diferentes maneras de invocar a una función o método.

Figura 17

Lamar o invocar una función



Nota. Adaptado de Java - Curso práctico de formación para la preparación del examen de certificación Java SE Programmer I (p. 115), por Sierra, A., 2018, Madrid, España: Alfaomega Grupo Editor.

En la figura 17 se puede apreciar tres formas de invocar o llamar a una función, el literal a) muestra un llamado directo a una función, el literal b) el valor que retorna el uso de la función suma es almacenado en la variable x ($x = 13$) y el literal c) se usa el llamado a la función como condición para evaluar, en este sentido el resultado de sumar 6 y 7 (*return (13)*) es evaluado en la condición para determinar si $13 == 10$.

De igual manera, como se trabajó con arreglos, vamos a usar funciones sobre el ejemplo ACD#1, obteniendo los siguientes resultados.

Gráfico 22 Revise el [Caso Facebook con funciones](#).



Actividades de aprendizaje recomendadas

Continuemos con el aprendizaje mediante su participación en las actividades que se describen a continuación:

1. Para complementar lo aprendido en esta semana, se sugiere revisar el siguiente recurso donde encontrará ejercicios para repasar en Deitel, P. & Deitel, H. (2016). *Java como programar*. Ciudad de México, México: Pearson Education, Páginas 203-205.
2. Se recomienda revisar el siguiente video [Creación de Objetos e Invocación de Métodos en JAVA](#), para que conozca como se escribe código en java usando funciones o métodos.
3. Realice la autoevaluación para comprobar sus conocimientos.



Autoevaluación 1

1. ¿Qué es una función?
 - a. Un bloque de código que realiza una tarea específica y permite reutilizar código.
 - b. Una variable de tipo numérico.
 - c. Una estructura de control de flujo.
 - d. Es una estructura secuencial.
2. ¿Cuál es la diferencia entre una variable global y una variable local en Java?
 - a. Una variable global es accesible desde cualquier parte del código, mientras que una variable local solo es accesible dentro de una función o bloque específico.
 - b. Una variable global tiene un alcance más limitado que una variable local.
 - c. No existe diferencia entre ambas.
 - d. La variable local es accesible desde cualquier parte del código, mientras que la global solo se accede desde el interior de un método.
3. ¿Qué es un bucle en programación estructurada con Java?
 - a. Una estructura de control que permite ejecutar un bloque de código varias veces.
 - b. Una variable de tipo cadena.
 - c. Una función específica.
 - d. Una condicional que limita la ejecución de determinadas sentencias.

4. ¿Cuál es la diferencia entre un *if* y un *switch* en Java?
 - a. Un *if* se utiliza para evaluar una condición y ejecutar un bloque de código si se cumple, mientras que un *switch* se utiliza para elegir entre varias opciones de código según el valor de una variable.
 - b. No existe diferencia entre ambos.
 - c. Un *if* se utiliza para bucles, mientras que un *switch* se utiliza para condicionales.
 - d. El *if* necesita de un iterador para finalizar su ejecución, mientras que un bucle no.
5. ¿Qué es un arreglo en programación estructurada con Java?
 - a. Una estructura de datos que permite almacenar varios valores del mismo tipo de datos.
 - b. Una función específica.
 - c. Una estructura de control de flujo.
 - d. Un arreglo es un bucle que permite iterar con datos de distinto tipo.
6. ¿Qué es un tipo de datos primitivo en Java?
 - a. Un tipo de dato básico, como números o caracteres, que no puede ser dividido en otros tipos de datos.
 - b. Una clase de Java que se utiliza para almacenar y manipular información.
 - c. Una estructura de control de flujo.
 - d. Un objeto en java.

7. ¿Cuál es la diferencia entre un tipo de datos *int* y un tipo de datos *double* en Java?
- Un *int* es un tipo de dato entero, mientras que un *double* es un tipo de dato entero y decimal
 - Un *int* es un tipo de dato de punto flotante, mientras que un *double* es un tipo de dato entero.
 - No existe diferencia entre ambos.
 - Un tipo de datos *int* permite guardar datos textuales, mientras que el tipo de datos *double* solo datos enteros.
8. ¿Qué es una variable de tipo char en Java?
- Una variable que almacena un carácter específico.
 - Una variable que almacena un número entero.
 - Una variable que almacena una cadena de texto.
 - Una variable que almacena un valor como “true”.
9. ¿Qué es una variable de tipo boolean en Java?
- Una variable que almacena un valor lógico, verdadero o falso.
 - Una variable que almacena un número entero.
 - Una variable que almacena un carácter específico.
 - Una variable que puede almacenar el siguiente valor “Hola”.
10. ¿Cuál de los siguientes literales no tiene errores en la clase Escuela?
- ```
import java.util.*;
package mypackage;
public class Escuela{.....};
```
  - ```
package mypackage;
import java.util.*;
public class Escuela;
```
 - ```
class Escuela{
public class Escuela();
}
```
  - ```
import mypackage;
public class Escuela();
```

11. Determine, ¿cuál es el resultado del siguiente código?

```
public class AritmeticOperators {  
    public static void main(String[] args) {  
        int num1 = 10;  
        int num2 = 20;  
        int num3 = 5;  
        double num4 = 4.5;  
        int respuesta = (num1 + num2 / num3)*num4;  
        System.out.println("La respuesta es: " + respuesta);  
    }  
}
```

- a. 63.
- b. 22,5.
- c. 28.
- d. 27.

12. ¿Qué falta para que se ejecute correctamente el siguiente algoritmo de cálculo de un número primo?

```
public class IfExample {  
    public static void main(String[] args) {  
        int num = 10; // número a evaluar  
        if (num % 2 = 0) {  
            System.out.println(num + " es un número par.");  
        } else {  
            System.out.println(num + " es un número impar.");  
        }  
    }  
}
```

- a. Operador de comparación de valores.
- b. Condición que indique num / 2.
- c. Un contador i++ para cerra el if.
- d. Está correcto el código.

13. Identifique, ¿cuál es el error en el siguiente código usando for para imprimir los primeros 10 números?

```
public class ForExample {  
    public static void main(String[] args) {  
        for (int i = 0; i <= 10; i++) {  
            System.out.println(i);  
        }  
    }  
}
```

- a. Que la condición sea $i < 10$.
b. Usar decremento con el iterador $i--$
c. Que se coloque dentro del for una condición que indique que cuando sea $= 10$ finalice el código.
d. Está correcto el código.
14. ¿Cuál es el resultado de ejecutar el siguiente ejercicio usando arreglos?

```
public class ArrayExample {  
    public static void main(String[] args) {  
        int[] numbers = {1, 2, 3, 4, 5};  
        System.out.println("El elemento seleccionado es: " + numbers[3])  
    }  
}
```

- a. El elemento seleccionado es 4.
b. El elemento seleccionado es 3.
c. El elemento seleccionado es 5.
d. El elemento seleccionado es 2.

15. ¿Qué sentencia falta para que el siguiente código funcione?

```
import java.util.Scanner;

public class DoWhileExample {
    public static void main(String[] args) {
        Scanner entrada = new Scanner(System.in);
        int num;
        do {
            System.out.print("Ingrese un número entero positivo: ");
        } while (num <= 0);
        System.out.println("Número válido ingresado: " + num);
        sc.close();
    }
}
```

- a. num = entrada.nextInt();
- b. ;
- c. do while(num <= 0);
- d. while (num >10);

[Ir a solucionario](#)



Unidad 2. Programación orientada a objetos

Este nuevo paradigma de la programación nos va a permitir trabajar tanto conceptualmente como de manera práctica, llevándonos por los conceptos de programación orientada a objetos, definiendo términos como objetos, clases, atributos, métodos, constructores y modificadores.

Los pilares de la programación orientada a objetos se muestran en la figura 18 y se especifica en que momentos vamos a trabajar con ellos.

Figura 18

Pilares de la programación orientada a objetos.



Nota. Bustamante, W., 2023.

2.1. Introducción a programación orientada a objetos

La programación orientada a objetos, del acrónico POO, revoluciona la programación estructurada para convertirla en una programación más modular y abstracta, se caracteriza por trabajar con propiedades como: Objetos, Clases, Encapsulamiento, Herencia y Polimorfismo (Sierra, 2018).

2.2. Tipos de datos objetos

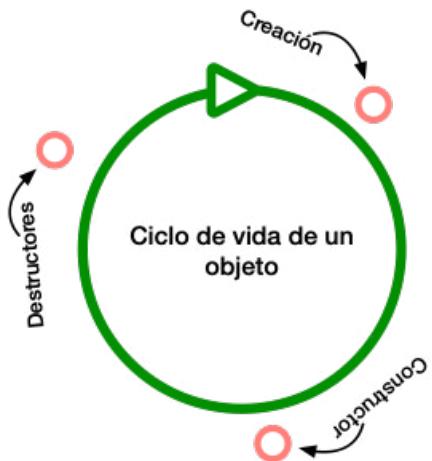
Un objeto es una entidad que representa o modelan objetos del mundo real, en programación se conoce a un objeto como una instancia de una clase. En java existen tipos de datos, objetos, que consisten en cualquier clase de

java, como se pudo apreciar en el gráfico de estructura de datos apartado 1.4.1.

El ciclo de vida de un objeto se detalla en la figura 19.

Figura 19

Ciclo de vida de los objetos.



Nota. Bustamante, W., 2023.

Para la creación de un objeto en java se utiliza el operado ***new*** seguido del nombre de la clase. Este objeto es creado en memoria generando una referencia a dicho objeto. Ejemplos.

Gráfico 23

Creación de objetos

```
ClaseUno c = new ClaseUno();
String obj = new String ("Hola");
Object obj = new Object();
```

Los constructores son bloques de código que se ejecutan durante la creación de un objeto y se les nombre con el mismo nombre de la clase. Ejemplo.

Gráfico 24

Constructor

```
class Programa{  
    public Programa(){  
        // Código del constructor  
    }  
}
```

En cambio, la destrucción de objetos sirve para liberar la memoria de la Máquina Virtual Java (MVJ), proceso que se conoce como *Garbage Collector* (GC).

2.3. Atributos

Son los datos que representan la estructura de un objeto. Existen datos que se los puede calcular a partir de la operación entre otros datos. Entre ellas tenemos, por ejemplo: cédula, nombres, apellidos, edad, etc.

Gráfico 25

Declaración de variables

```
// 1. Declarar Variables  
  
private String Cedula; //variable privada de tipo cadena  
public String Nombres;  
protected String Apellidos;  
static int Edad;
```

2.4. Métodos

Contemplan las funciones que manejan el comportamiento del objeto, en la figura 20 el objeto persona se comporta de acciones como: caminar, correr, nadar, estudiar; comportamientos que por lo general se describen usando verbos que representa la acción del objeto.

Los métodos son acciones que implementan el comportamiento o las funciones de un objeto.

Toda clase tiene métodos como: *get* y *set*, los métodos *set* sirven para establecer los valores para cada uno de los datos de un objeto, se requiere definir y diseñar un método perteneciente al objeto que permita colocar

(setter) el valor a cada dato o también puede ser calculado. Mientras que los métodos get sirven para dar salida al valor de un dato, se requiere definir y diseñar un método que permita acceder (getter) al valor del dato para darlo como salida, esto es, un método para obtener cada dato.

A continuación, se muestra un ejemplo, un esquema de una clase con los atributos y métodos que se mencionan en este apartado.

Figura 20

Esquema de una clase

Class Jugador
Cédula
Nombre
Apellido
Edad
caminar()
correr()
viajar()

Nota. Bustamante, W., 2023.

Además, en el siguiente código se puede apreciar la sintaxis de un método.

Gráfico 26

Sintaxis de un método

```
private void caminar() {  
    System.out.println("La persona camina: ");  
}
```

2.5. Modificadores

Se usa tanto para atributos como para métodos, y contempla el uso de símbolos que representan el nivel de accesibilidad de los datos y los métodos frente a la misma clase u otras clases. En la figura 21 se muestra a continuación se puede apreciar su uso.

Figura 21

Tipos de modificadores de acceso



Nota. Bustamante, W., 2023.

La siguiente infografía muestra los modificadores de acceso tanto para atributos como para métodos.

Modificadores de acceso.



Actividades de aprendizaje recomendadas

Estimado estudiante lo invito a desarrollar las siguientes actividades para afianzar sus conocimientos.

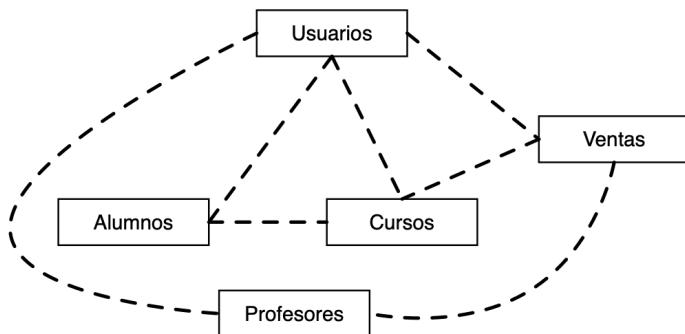
1. Para complementar lo aprendido en esta semana, se sugiere revisar el siguiente recurso donde encontrará ejercicios para repasar en Deitel, P. & Deitel, H. (2016). *Java como programar*. Ciudad de México, México: Pearson Education, Páginas 232-258.
2. Se recomienda revisar el siguiente video [Encapsular atributos de una clase con NetBeans](#), para que conozca como se escribe código en java usando el principio de programación orientada a objetos denominado encapsulamiento.



Semana 7

El paradigma de POO organiza las funciones en entidades denominadas objetos. Para resolver un problema, la POO interactúa con un conjunto de objetos que se relacionan, como se puede apreciar en la siguiente figura.

Figura 22
Objetos



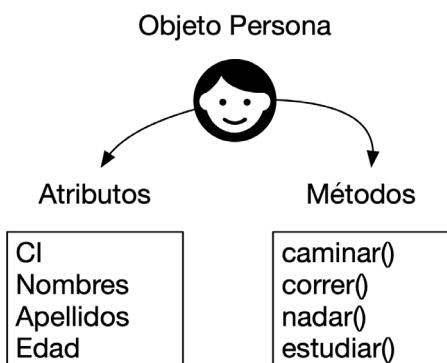
Nota. Bustamante, W., 2023.

2.6. Objetos

Los objetos son entes, entidades, sujetos o cosas que encontramos en el dominio del problema de nuestra realidad, entiéndase: en situaciones o problemas de nuestro mundo cotidiano empresarial, organizacional o institucional, que se requiere manejar o solucionar mediante la computadora (López, 2013).

Estos objetos se componen de datos denominados Atributos y de funcionalidades denominados Métodos, como se aprecia en la siguiente figura.

Figura 23
Ejemplo de objeto persona



Nota. Bustamante, W., 2023.

También los objetos se pueden representar gráficamente a través de un rectángulo, con el conjunto de datos y funcionalidades como se aprecia en la siguiente manera:

Figura 24
Representación gráfica de objetos personas

Objeto Persona 1	Objeto Persona 2	Objeto Persona N
1103456899	1145678902	1293833377
Pablo	María	Juan.
Jiménez.	Pérez.	Loja.
28	43	18
caminar()	caminar()	caminar()
correr()	correr()	correr()
nadar()	nadar()	nadar()
estudiar()	estudiar()	estudiar()

Nota. Bustamante, W., 2023.

2.7. Clases

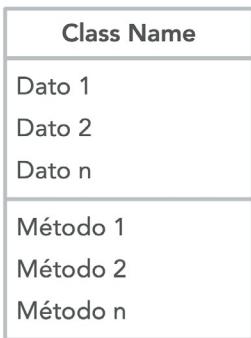
La clase es una representación abstracta que describe a un conjunto de objetos; en otras palabras, es un tipo de dato abstracto que representa a un conjunto de objetos que tienen en común una misma estructura (de datos) y un mismo comportamiento (las mismas funciones) (López, 2013).

Las clases es considerada como una plantilla que permite crear varios objetos, que contengan los mismos datos y comportamientos. Se puede representar gráficamente una clase usando diagramas que contemplen las siguientes características:

- Uso de un rectángulo con tres partes: 1. Nombre de clase, 2. Lista de Atributos y 3. Lista de métodos.
- Además, los métodos deben ser representados con () .
- Tanto atributos como métodos tienen sus propios Modificadores de Acceso.

Figura 25

Esquema de una clase

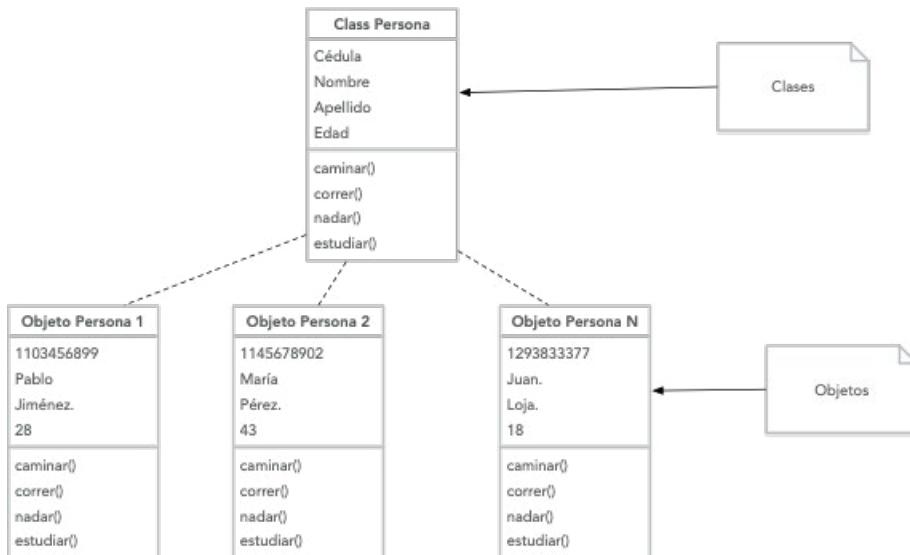


Nota. Bustamante, W., 2023.

Relacionando las clases con los objetos se puede tener un esquema de la siguiente manera.

Figura 26

Clases y objetos



Nota. Bustamante, W., 2023.

2.8. Constructor:

El método constructor es ejecutado con *new* y su función es asignar espacio y crear el objeto en la memoria dinámica.

A continuación, se presenta un ejemplo general de una clase con sus respectivos atributos, métodos, modificadores y constructor.

Gráfico 27

Ejemplo de constructor – clase persona

```
// Ejemplo de constructor
package com.mycompany.ejemploclases;
/**
 * @author xavicrip
 */
public abstract class Persona {
    // 1. Declarar Variables
    private String Cedula;
    public String Nombres;
    protected String Apellidos;
    static int Edad;
    // 2. Constructor
    public Persona(String Cedula, String Nombres, String Apellidos) {
        this.Cedula = Cedula;
        this.Nombres = Nombres;
        this.Apellidos = Apellidos;
    }
    // 3. Métodos
    private void caminar() {
        System.out.println("La persona camina: ");
    }
    public void correr() {
        System.out.println("La persona corre: ");
    }
    protected void nadar () {
        System.out.println("La persona nada: ");
    }
    static void estudiar () {
        System.out.println("La persona estudia: ");
    }
    abstract void bailar ();
}
```

Para el desarrollo de la práctica en la unidad II, se va a trabajar con el siguiente ejercicio: Caso ACD#2:

Sistema para cargar las horas de los empleados:

Se necesita manejar la carga de horas de los empleados de una empresa. La idea es que haya varios grupos/módulos y que vayan cargando las horas semanales para cada una de las tareas que hayan realizado.

De esta manera, a fin de mes, se podrá hacer un cierre y calcular el total de estas, y con base en esos totales, calcular los honorarios para cada empleado.



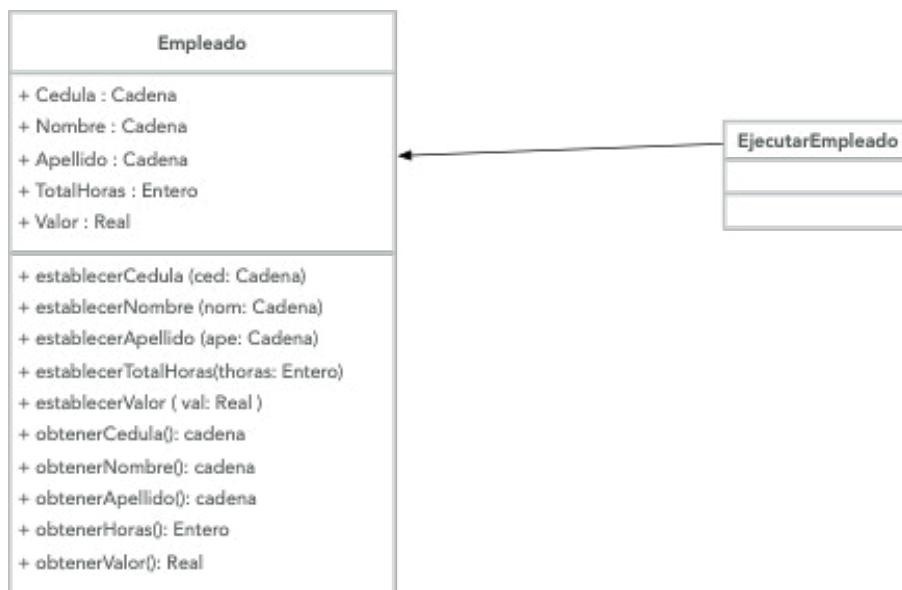
A continuación Ud. podrá encontrar toda la información del ejercicio en los siguientes enlaces:

- Para descargar el Enunciado del ejercicio Caso ACD#2, realícelo del siguiente enlace [POO_Tec/Caso ACD2/Problema.rtf](#).
- Para descargar la mini especificación, diagrama UML y código del ejercicio Caso ACD#2, realícelo del siguiente enlace [POO_Tec/Caso ACD2/](#).

La figura 27, muestra el diagrama de clases del caso horas empleado, y a continuación se muestra el desarrollo de la solución en el código 28 y 29.

Figura 27

Caso horas empleado



Nota. Bustamante, W., 2023.

Código de la clase Empleado.

Gráfico 28

Caso horas empleado – Clase empleado.

```
| // Clase Empleado
| package com.mycompany.ejecutaremppleado;
| /**
| *
| * @author xavicrip
| */
| public class Empleado {
|     // 1. Declarar Variables
|     String Cedula;
|     String Nombre;
|     String Apellido;
|     int TotalHoras;
|     double Valor;
|     //2. Métodos Establecer
|     public void establecerCedula(String ced) {
|         Cedula = ced;
|     }
|     public void establecerNombre(String nom) {
|         Nombre = nom;
|     }
|     public void establecerApellido(String ape) {
|         Apellido = ape;
|     }
|     public void establecerTotalHoras(int thoras) {
|         TotalHoras = thoras;
|     }
|     public void establecerValor(double val) {
|         Valor = val;
|     }
|     //3. Métodos Obtener
|     public String obtenerCedula() {
|         return Cedula;
|     }
|     public String obtenerNombre() {
|         return Nombre;
|     }
|     public String obtenerApellido() {
|         return Apellido;
|     }
|     public int obtenerHoras() {
|         return TotalHoras;
|     }
|     public double obtenerValor() {
|         return Valor;
|     }
| }
```

Código de la clase principal: Ejecutar Empleado.

Gráfico 29

Caso horas empleado – Clase principal.

```
package com.mycompany.ejecutarempleado;
import java.util.Scanner;
/**
 *
 * @author xavicrip
 */
public class Ejecutarempleado {
    public static void main (String[] args) {
        System.out.println("Cálculo de Honorarios de los Empleados");
        // 1. Declarar Variables
        String ced;
        String nom;
        String ape;
        int horas;
        double valor;
        // 2. Uso de la librería en lenguaje Java para lectura de datos
        desde
            Scanner entrada = new Scanner (System.in); // importante, se
        debe importar
            // 3. Declarar, crear e iniciar objeto
            Empleado empleado = new Empleado ();
            // 4. Ingresar datos por teclado
            System.out.println("Ingrese la cedula del empleado:");
            ced = entrada.nextLine();
            System.out.println("Ingrese el nombre del empleado:");
            nom = entrada.nextLine();
            System.out.println("Ingrese el apellido del empleado:");
            ape = entrada.nextLine();
            System.out.println("Ingrese el valor que gana el empleado por
        hora:");
            valor = entrada.nextDouble();
            // 4. Establecer valores
            empleado.establecerCedula(ced);
            empleado.establecerNombre(nom);
            empleado.establecerApellido(ape);
            empleado.establecerValor(valor);
            // 5. Ingresar 5 actividades para calcular el tiempo que ha
            trabajado en una semana
            int i = 1;
            int aux = 0;
            int suma = 0;
            System.out.println("A continuación se detallan la cantidad de
            horas trabajadas en cinco actividades");
            do {
                System.out.println("Ingresa la cantidad de horas trabajadas
                en la Actividad #" + i);
                aux = entrada.nextInt();
                suma = suma + aux;
                i = i + 1;
            } while (i <= 5);
            horas = suma;
            // 6. Establecer valores de horas totales trabajadas
            empleado.establecerTotalHoras(horas);
            // 7. Cálculo de honorarios
            double salario;
            salario = empleado.obtenerHoras() * empleado.obtenerValor();
            // 7. Presentar los resultados
            System.out.println("El empleado " + empleado.obtenerNombre() +
            " " + empleado.obtenerApellido() + " con # cédula: " +
            empleado.obtenerCedula() + " ha trabajado " +
            empleado.obtenerHoras() + " horas a la semana y le corresponde
            " + salario + " dólares");
    }
}
```

Ejecución del Código: la figura 28 muestra la ejecución por consola del ejercicio caso horas empleado.

Figura 28

Ejecución caso horas empleado

```
--- exec-maven-plugin:3.0.0:exec (default-cli) @ ejecutarempelado ---
Cálculo de Honorarios de los Empleados
Ingrese la cedula del empleado:
1234567
Ingrese el nombre del empleado:
Xavier
Ingrese el apellido del empleado:
Bustamante
Ingrese el valor que gana el empleado por hora:
5
A continuación se detallan la cantidad de horas trabajadas en cinco actividades
Ingresa la cantidad de horas trabajadas en la Actividad #1
10
Ingresa la cantidad de horas trabajadas en la Actividad #2
4
Ingresa la cantidad de horas trabajadas en la Actividad #3
7
Ingresa la cantidad de horas trabajadas en la Actividad #4
9
Ingresa la cantidad de horas trabajadas en la Actividad #5
7
El empleado Xavier Bustamante con # cédula: 1234567 a trabajado 37 horas a la semana y le corresponde 185.0 dolares
BUILD SUCCESS
-----
Total time: 18.162 s
Finished at: 2022-11-30T12:19:57-05:00
```

Nota. (Bustamante, 2023)



Actividades de aprendizaje recomendadas

Continuemos con el aprendizaje mediante su participación en las actividades que se describen a continuación:

1. Para complementar lo aprendido en esta semana, se sugiere revisar el siguiente recurso donde encontrará ejercicios para repasar en Deitel, P. & Deitel, H. (2016). *Java como programar*. Ciudad de México, México: Pearson Education, Páginas 232-258.
2. Se recomienda revisar el siguiente video [Java Orientado a Objetos: 43 Ejemplo Mundo Real Paciente y Doctor Entender Ocultamiento de Datos](#), para que conozca como se escribe objetos en java.
3. Se recomienda revisar el siguiente video [Programación Orientada a Objetos \(POO\) - Objetos y Clases](#), para que conozca como se escribe objetos y clases en java.

4. Le invito a reforzar sus conocimientos, participando en la siguiente autoevaluación:



Autoevaluación 2

1. ¿Cuál de los siguientes ítems le corresponde el enunciado: “Usa modificadores de acceso para proteger variables y ocultarlos dentro de su propia clase”?
 - a. Encapsulamiento.
 - b. Abstracción.
 - c. Polimorfismo.
 - d. Herencia.

2. ¿Qué es una clase en programación orientada a objetos?
 - a. Un conjunto de instrucciones que realizan una tarea específica.
 - b. Una plantilla para crear objetos con atributos y métodos específicos.
 - c. Una función que se puede reutilizar en varias partes de un programa.
 - d. Un método que usa solo variables locales para su implementación.

3. ¿Qué es un objeto en programación orientada a objetos?
 - a. Una instancia de una clase con valores específicos para sus atributos.
 - b. Un conjunto de instrucciones que realizan una tarea específica.
 - c. Una función que se puede reutilizar en varias partes de un programa.
 - d. Un método que se puede llamar desde otras clases.

4. ¿Qué es un método en programación orientada a objetos?
 - a. Una función dentro de una clase que realiza una tarea específica y puede ser llamada por un objeto de esa clase.
 - b. Un atributo de una clase que almacena un valor específico.
 - c. Una relación entre clases donde una clase hereda atributos y método.
 - d. Un tipo de dato que puede ser usado en cualquier momento dentro de la clase.

5. ¿Qué hay que modificar en el siguiente ejercicio para que cumpla el concepto de encapsulamiento?

```
public class EncapsulationExample {  
    public int num;  
    public void setNum(int num) {  
        this.num = num;  
    }  
    public int getNum() {  
        return num;  
    }  
}
```

- a. *Private int num.*
 - b. *Private void setNum(int num) y private int getNum().*
 - c. *Protected void setNum(int num) y protected int getNum().*
 - d. El código está correcto.
6. ¿Qué modificadores de acceso se pueden utilizar en Java para controlar el acceso a un atributo o método de una clase?
- a. *Public , protected, private.*
 - b. *Global , local, protected.*
 - c. *Public , private, package-private (sin modificador).*
 - d. *Class private.*
7. Seleccione, ¿cuál es el constructor correcto para el siguiente ejemplo?

```
public class ConstructorExample {  
    private int num1;  
    private int num2;  
    public void printNumbers() {  
        System.out.println("num1: " + num1);  
        System.out.println("num2: " + num2);  
    }  
}
```

- a. *public ConstructorExample(int num1, int num2) {this.num1 = num1; this.num2 = num2;}*
- b. *private constructor(int num1, int num2);*
- c. *private ConstructorExample (int num1) {this.num1 = num1;}*
- d. *private ConstructorExample(int num2) {this.num1 = num1;}*

8. ¿Los métodos *public* de una clase se conocen también cómo?
- a. Servicios *public*.
 - b. Atributos *public*.
 - c. Características *public*.
 - d. *Public*.
9. ¿Qué clase se debe usar para hacer cálculos monetarios de precios?
- a. BigDecimal.
 - b. Decimal.
 - c. Integer.
 - d. Boolean.
10. Supongamos que tienes una clase “Rectángulo” que tiene las variables “ancho” y “altura”, y los métodos “calcularArea()” y “calcularPerímetro()”. ¿Cuál de las siguientes opciones instanciaría un objeto de la clase “Rectángulo” con un ancho de 3.5 y una altura de 8.2?
- a. Rectángulo miRectángulo = new Rectángulo(); miRectángulo. Ancho = 3.5. miRectángulo. Altura = 8,2.
 - b. Rectángulo miRectángulo = Rectángulo(3.5, 8.2).
 - c. Rectángulo miRectángulo = new Rectángulo(3.5, 8.2).
 - d. miRectángulo. Ancho = 3.5. miRectángulo. Altura = 8.2.
Rectángulo miRectángulo = new Rectángulo().

[Ir a solucionario](#)



Semana 8



Actividades finales del bimestre



Actividades de aprendizaje recomendadas

Estimado estudiante lo invito a desarrollar las siguientes actividades con el fin de reforzar sus conocimientos.

1. La semana 8 es un espacio de tiempo que permite a los estudiantes reforzar temáticas no comprendidas en las siete semanas de clases anteriores, con el objetivo de poder presentarse a la evaluación del primer bimestre; por lo cual se propone revisar:

- Unidad 1: Programación Estructurada.
- Unidad 2. Programación Orientada a Objetos.

Esta revisión consiste en las siguientes actividades.

- Revisar conceptos desarrollados en las siete semanas, entre ellos: programación estructurada, estructuras secuenciales, estructuras de control, estructuras repetitivas, arreglos, funciones, métodos, modificadores y programación orientada a objetos principalmente.
- Desarrollar ejemplos prácticos de programación estructurada y programación orientada a objetos, mismos que los encuentra en el repositorio compartido en GitHub de los dos casos: *Facebook* y horas de empleado, dichos ejemplos le permitirán afianzar sus destrezas en programación.

2. Con el objetivo de reforzar los contenidos de la unidad I y II, se recomienda ingresar al **texto básico** Deitel, P. & Deitel, H. (2016). *Java como programar*. Ciudad de México, México: Pearson Education; y trabajar con los siguientes apartados:

- **Resumen** de los siguientes capítulos II, III, IV, V, VI, VII y VIII.
- **Ejercicios de autoevaluación** de los siguientes capítulos II, III, IV, V, VI, VII y VIII.

Procedimiento

El **texto básico** tiene apartados denominados **Resumen** y **Ejercicios de autoevaluación**, en esta actividad se recomienda revisar cada uno de esos apartados de los capítulos señalados anteriormente, con la finalidad que Ud. pueda:

- En el apartado **Resumen**, repasar el contenido revisado en el primer bimestre.
 - En el apartado **Ejercicios de Autoevaluación**, ejercitarse resolviendo ejercicios prácticos y a su vez tener una pronta y oportuna retroalimentación.
3. Con el objetivo de reforzar los contenidos de la unidad I y II, se recomienda ingresar al **texto básico** Deitel, P. & Deitel, H. (2016). *Java como programar*. Ciudad de México, México: Pearson Education; y desarrollar los siguientes ejercicios.
- **Calculadora de ahorro por viajes compartidos en automóvil.**

El enunciado lo encuentra en el ejercicio 2.35 del **texto básico**, en la página 68.

- **Crecimiento de la población mundial**

El enunciado lo encuentra en el ejercicio 4.39 del **texto básico**, en la página 151.

Procedimiento

Para el desarrollo de los ejercicios propuesto del texto básico, se recomienda realizar las siguientes actividades:

- Escriba una mini especificación por cada uno de los ejercicios.
- Diagrama el modelo de la solución usando diagramas UML.
- Cree el código usando el IDE de NetBeans.



Segundo bimestre

Resultado de aprendizaje 3

- Desarrolla aplicaciones de complejidad media integrando los principios de POO.

Continuado con programación orientada a objetos, en el segundo bimestre se van a abordar conceptos con mayor grado de complejidad y con una nueva visión de desarrollo en la nube, es por ello el nombre de la unidad III: Programación Orientada a Objetos Avanzada.

Con la finalidad de abordar adecuadamente el resultado de aprendizaje propuesto, es importante contemplar los siguientes conceptos: herencia, polimorfismo, parámetros en los métodos, arrayList de objetos y finalizando con una revisión general de programación en la nube usando el IDE de AWS Cloud9.

Contenidos, recursos y actividades de aprendizaje



Semana 9

En el segundo bimestre de la asignatura de Programación Orientada a Objetos, se continúa desarrollando habilidades que le permitan al estudiante adquirir conocimientos conceptuales y prácticos con mayor complejidad en la programación orientada a objetos, entre ellos: herencia, polimorfismo, interfaces, paquetes.

Unidad 3. Programación orientada a objetos avanzada

3.1. Programación avanzada

En esta unidad se pretende aplicar y usar conceptos avanzados de programación orientada a objetos, entre ellos lista de arreglos, métodos y sus tipos, herencia, polimorfismo y programación en la nube con AWS Cloud9.

3.2. Herencia

El objetivo de la presente semana es que el alumno pueda elaborar algoritmos usando unos de los principales conceptos de programación orientada a objetos que es la herencia.

Conceptualmente, la herencia consiste en que los objetos hijos o descendiente) heredan características (atributos) y comportamientos (métodos) de los objetos padres.

Las ventajas de usar herencia son:

- Permite a los desarrolladores reutilizar código.
- Permite poder asociar e identificar características comunes entre los objetos.
- Trabaja de manera jerárquica con conceptos de subclases (clases hijo) y superclases (clases padres).

El siguiente ejemplo permite reconocer como se interpreta la identificación del concepto herencia en programación orientada a objetos. Iniciamos planteando tres clases que simulan la problemática de los actores de un equipo básquet, como se muestra en la siguiente figura.

Figura 29

Caso equipo de básquet

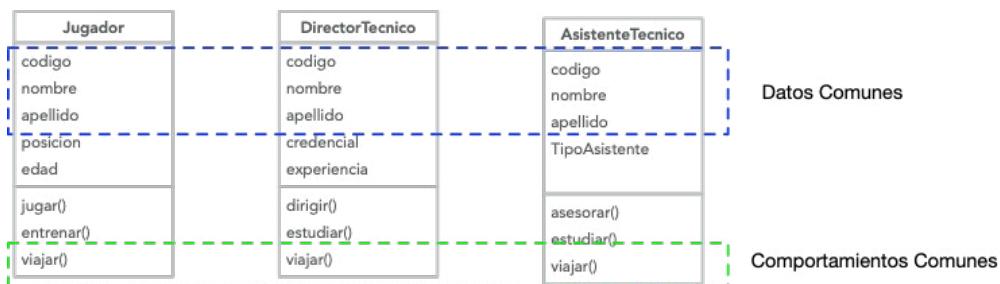
Jugador	DirectorTecnico	AsistenteTecnico
codigo		
nombre		
apellido		
posicion	credencial	
edad	experiencia	
jugar()	dirigir()	
entrenar()	estudiar()	
viajar()	viajar()	

Nota. Bustamante, W., 2023.

En la figura 30 se puede apreciar un conjunto de características y comportamientos que son comunes para las tres clases, como se muestra a continuación.

Figura 30

Identificando atributos y métodos comunes.



Nota. Bustamante, W., 2023.

Con la finalidad de no duplicar código con características y comportamientos que se comparten entre las clases, se procede a crear una superclase (Equipo Básquet) que tome esos atributos y comportamientos comunes. Y dejando los atributos y comportamientos específicos en las clases correspondientes y que pasan a ser denominadas subclases. A continuación, se puede visualizar gráficamente mediante diagramas de clases.



Actividades de aprendizaje recomendadas

Continuemos con el aprendizaje mediante su participación en las actividades que se describen a continuación:

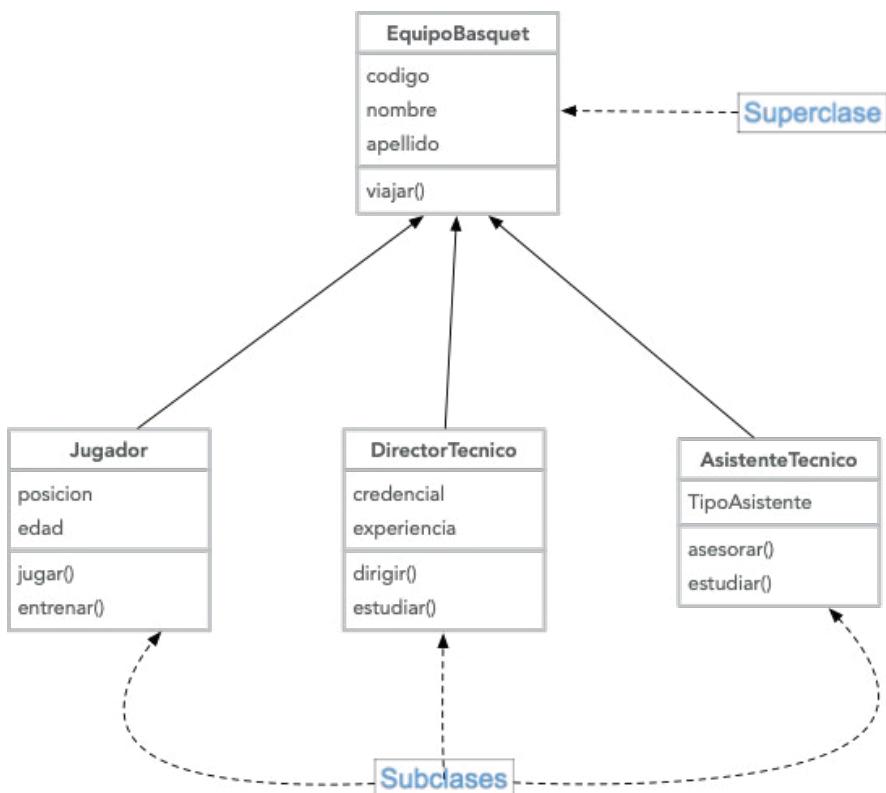
1. Para complementar lo aprendido en esta semana, se sugiere revisar el siguiente recurso donde encontrará ejercicios para repasar en Deitel, P. & Deitel, H. (2016). *Java como programar*. Ciudad de México, México: Pearson Education, Páginas 360-391.
2. Para complementar lo aprendido en esta semana, se sugiere revisar el siguiente recurso donde encontrará ejercicios para repasar en Sierra, A. (2018). *Java - Curso práctico de formación para la preparación del examen de certificación Java SE Programmer I: IZO-808*. Alfaomega, Páginas 149 – 191.
3. Se recomienda revisar el siguiente video [10. Herencia en java](#), para que conozca como se escribe código en java usando el pilar de programación orientado a objetos denominado herencia.



3.3. Ejercicios con herencia:

A continuación, se va a desarrollar el siguiente ejercicio detallado en el siguiente diagrama de clases.

Figura 31
Caso equipo de básquet - Herencia



Nota. Bustamante, W., 2023.

Explicación:

Las subclases **Jugador**, **director Técnico** y **Asistente Técnico** heredan atributos y comportamientos de la superclase **Equipo Básquet**, es decir; que todas las subclases heredan atributos, código, nombre y apellidos de la superclase; de igual manera heredan su comportamiento de **viajar()**.

También es importante señalar que las subclases pueden tener sus propias características y comportamientos que no se heredan a la superclase y que son independientes y propias de otras subclases.

A continuación, se representa en código en java el ejemplo planteado anteriormente mediante diagrama de clases, donde se encuentran los códigos de [superclase y subclase](#).



Actividades de aprendizaje recomendadas

Estimado estudiante lo invito a desarrollar las siguientes actividades que le servirán como refuerzo.

1. Para complementar lo aprendido en esta semana, se sugiere revisar el siguiente recurso donde encontrará ejercicios para repasar en Deitel, P. & Deitel, H. (2016). *Java como programar*. Ciudad de México, México: Pearson Education, Páginas 360-391.
2. Para complementar lo aprendido en esta semana, se sugiere revisar el siguiente recurso donde encontrará ejercicios para repasar en Sierra, A. (2018). *Java - Curso práctico de formación para la preparación del examen de certificación Java SE Programmer I: IZO-808*. Alfaomega, Páginas 149 – 191.
3. Se recomienda revisar el siguiente video [10.- \[Curso java básico\] Herencia en java \(Extends\)](#), para que conozca como se escribe código en java usando el pilar de programación orientado a objetos denominado herencia.



Semana 11

3.4. Polimorfismo:

El estudiante con el estudio de este tema podrá crear algoritmos usando polimorfismo, iniciaremos explicando cómo diseñar algoritmos apoyados en los diagramas de clases y continuamos con ejemplos con código en java del desarrollo de una problemática planteada.

Según (Deitel & Deitel, 2016), “el polimorfismo nos permite “programar en forma general”, en vez de “programar en forma específica”. Permite escribir programas que procesen objetos que comparten la misma superclase como si todos fueran objetos de la superclase”.

Los lenguajes de programación orientados a objetos proporcionan mecanismos que trabajan de manera estática y dinámicas, el concepto de polimorfismo trabaja con mecanismos dinámicos donde las referencias de los métodos se resuelven en tiempo de ejecución y no en tiempo de compilación como lo hacen los métodos estáticos.

El polimorfismo se implementa a través de métodos abstractos, permitiendo crear métodos de comportamiento dinámico, es decir, este concepto de polimorfismo permite tomar código y agregar nuevos comportamientos sin modificar el código fuente, esto se conoce con el término de extensibilidad.

Según (Sierra, 2018), la extensibilidad es una forma de crecer más, dada por la herencia; se hereda todo lo que el tipo ancestro tiene y se añaden las nuevas funciones que se necesiten.

En una clase abstracta no se pueden crear objetos, pero las clases hijas pueden heredar sus métodos y atributos. Para que una clase sea abstracta se debe usar la palabra reservada (*abstract*) en la definición de la clase y dicha clase debe tener al menos un método abstracto que será implementado en las clases hijas.

(Sierra, 2018), menciona algunas consideraciones de una clase abstracta, entre ellas:

- No es posible crear objetos de una clase abstracta.
- Además de métodos abstractos, las clases abstractas pueden incluir atributos, constructores y métodos estándares.
- Una clase que herede una clase abstracta está obligada a sobrescribir los métodos abstractos heredados o declararlos también como abstractos.
- Una clase puede declararse como *abstract* aunque no tenga métodos abstractos.

Un método abstracto en java es un método que tiene una particularidad, la cual consiste en que es un método que no tiene una implementación específica, y deben ser implementados en las clases hijas que heredan siempre de una clase padre.



Actividades de aprendizaje recomendadas

Continuemos con el aprendizaje mediante su participación en las actividades que se describen a continuación:

1. Para complementar lo aprendido en esta semana, se sugiere revisar el siguiente recurso donde encontrará ejercicios para repasar en Deitel, P. & Deitel, H. (2016). *Java como programar*. Ciudad de México, México: Pearson Education, Páginas 395-420.
2. Para complementar lo aprendido en esta semana, se sugiere revisar el siguiente recurso donde encontrará ejercicios para repasar en Sierra, A. (2018). *Java - Curso práctico de formación para la preparación del examen de certificación Java SE Programmer I: IZO-808*. Alfaomega, Páginas 149 – 191.
3. Se recomienda revisar el siguiente video [Polimorfismo Alquiler de Vehículos](#), para que conozca como se escribe código en java usando el pilar de programación orientada a objetos denominado polimorfismo, con el caso práctico “[Alquiler de Vehículos](#)”.



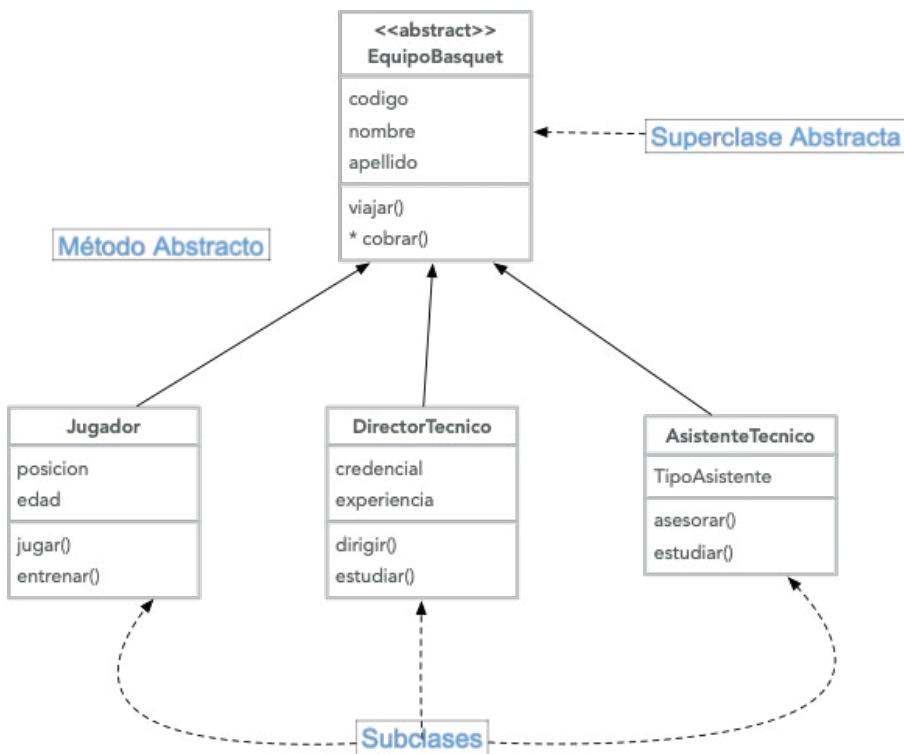
Semana 12

3.5. Herencia y polimorfismo:

Basandonos en el ejemplo planteado en Herencia, se traslada el ejemplo usando conceptos de polimorfismo para desarrollar el siguiente ejercicio, según diagrama de clases.

Figura 32

Caso equipo de básquet - Polimorfismo



Nota. Bustamante, W., 2023.

A continuación, se representa en código en java el ejemplo planteado anteriormente mediante diagrama de clases, donde se detalla: la clase principal, superclase, subclase.

Ejecución : se puede apreciar cómo se ejecuta los métodos, viajar y sueldos de los integrantes del equipo de básquet.

Figura 33

Ejecutar caso equipo de básquet - Polimorfismo

```
Scanning for projects...

-----< com.mycompany:ejecutarBasquet >-----
[INFO] Building ejecutarBasquet 1.0-SNAPSHOT
[INFO]   [ jar ]

[INFO] --- exec-maven-plugin:3.0.0:exec (default-cli) @ ejecutarBasquet ---
Todos los integrantes viajan
Juan Jiménez
Viajar a Guayaquil
Byron Bustamante
Viajar a Guayaquil
Duval Galarza
Viajar a Guayaquil
Sueldos de integrantes
Juan Jiménez
Su salario es: $1500 dolares
Byron Bustamante
Su salario es: $3000 dolares.
Duval Galarza
Su salario es: $9000 dolares.

BUILD SUCCESS
-----
Total time: 0.311 s
Finished at: 2022-12-01T21:03:00-05:00
-----
```

Nota. Bustamante, W., 2023.



Actividades de aprendizaje recomendadas

Estimado estudiante lo invito a desarrollar las siguientes actividades que le servirán como refuerzo.

1. Para complementar lo aprendido en esta semana, se sugiere revisar el siguiente recurso donde encontrará ejercicios para repasar en Deitel, P. & Deitel, H. (2016). *Java como programar*. Ciudad de México, México: Pearson Education, Páginas 395-420.
2. Para complementar lo aprendido en esta semana, se sugiere revisar el siguiente recurso donde encontrará ejercicios para repasar en Sierra, A. (2018). *Java - Curso práctico de formación para la preparación del examen de certificación Java SE Programmer I: IZO-808*. Alfaomega, Páginas 149 – 191.

3. Se recomienda revisar el siguiente video [Polimorfismo Alquiler de Vehículos](#), para que conozca como se escribe código en java usando el pilar de programación orientado a objetos denominado polimorfismo, con el caso práctico “Alquiler de Vehículos”.



Semana 13

3.6. Métodos y sus tipos:

En este apartado es importante analizar a detalle la creación y uso de métodos. En este sentido, los métodos pueden ser usados para recibir parámetros y para devolver resultados. El siguiente código muestra la estructura de un método.

Gráfico 30

Estructura de los métodos

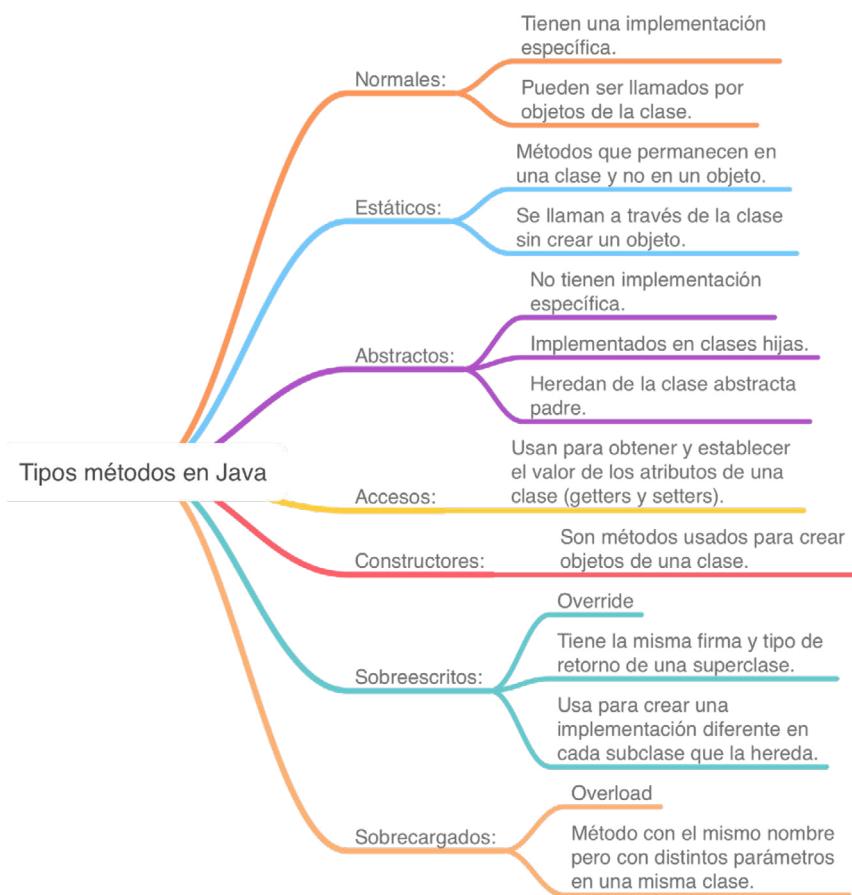
```
Modificador tipo_devolución nombre_método(parámetros) {  
}
```

Es importante mencionar que la estructura del método, los parámetros deben ser separados por comas (,).

Los métodos de tipo *void*, se caracterizan por no devolver ningún resultado, por lo tanto, no usan la palabra reservada *return*, mientras que los otros métodos que no son *void* retornan obligadamente un resultado, para ello usan la palabra reservada *return*; y el resultado debe ser acorde al tipo_ devolución del método, es decir, si el tipo_devolución de un método es *int*, el resultado también debe ser del mismo tipo de dato (un resultado entero).

La figura 34, muestra la clasificación de los métodos.

Figura 34
Clasificación de los métodos



Nota. Adaptado de Java - Curso práctico de formación para la preparación del examen de certificación Java SE Programmer I (p. 158 - 164), por Sierra, A., 2018, Madrid, España: Alfaomega Grupo Editor.

En este apartado haremos referencia especial a los métodos sobrecargar y sobreescritos porque el resto ya se los ha ido revisando en lo que va de esta guía.

Los tipos de métodos sobrecargados definen en una misma clase varios métodos con el mismo nombre, pero con diferentes parámetros (Sierra, 2018). Esta característica le permite al programador realizar una misma función a la que podemos llamar de la misma forma, pero incluso hasta con distintos tipos de devolución. A continuación, se presentan algunos ejemplos.

Gráfico 31

Sobrecarga de métodos

```
class Multiplicacion {  
    public int Multiplicacion (int x, int y) {  
        return x * y;  
    }  
    public int Multiplicacion (int x, int y, int z) {  
        return x * y * z;  
    }  
    public double Multiplicacion (double x, double y) {  
        return x * y;  
    }  
    public void Multiplicacion (doble x, doble y){  
        System.out.println("Multiplicación de dos números");  
    }  
}
```

Los tipos de métodos sobrescritos tienen la misma firma y tipo de retorno de una superclase y permite tener una implementación personalizada en cada subclase que lo hereda. La nomenclatura (@Override) indica al compilador que el método de la subclase sobrescribirá un método de la superclase.

A continuación, se presenta un ejemplo donde el método sobrescrito del ejemplo propuesto en el código 41 anula la información que se imprime de la superclase Animal que indica que “Los animales no hablan”; cambiando por “Excepto el perico que repite ciertas palabras”.

Gráfico 32

Sobreescritura de métodos

```
class Animal {  
    public void hablar() {  
        System.out.println("Los animales no hablan.");  
    }  
}  
  
class Perico extends Animal {  
    @Override  
    public void hablar() {  
        System.out.println("Excepto el perico que repite ciertas  
palabras");  
    }  
}
```

Finalmente (Sierra, 2018), menciona algunas reglas que se deben considerar para sobrescribir correctamente un método:

- El nombre del método y lista de parámetros deben ser idénticos al del original.
- El ámbito del método debe ser igual o menos restrictivo que el del original.
- El tipo de devolución debe ser igual o un subtipo del original.
- La nueva versión del método no debe propagar excepciones tipo *checked* que no están definidas en el original.

3.7. Parámetros en los métodos

Hasta el momento hemos aprendido a pasar parámetros de tipos de datos primitivos, pero en el siguiente apartado nos vamos a centrar en definir y ejemplificar el paso de parámetros de tipo objeto.

Según (Sierra, 2018), cuando se pasa un objeto en la llamada a un método, se está pasando una copia de la referencia al objeto. Es decir que la variable argumento y parámetro apuntan al mismo objeto, a continuación, se presenta un ejemplo.

Gráfico 33

Ejemplo de paso de parámetros tipo objetos

```
class Objeto {  
    private int valor;  
    public Objeto (int valor) {  
        this.valor = valor;  
    }  
    public int obtenerValor() {  
        return valor;  
    }  
    public void establecerValor(int valor) {  
        this.valor = valor;  
    }  
}  
  
class Ejemplo {  
    public void cambiarValor(Objeto obj) {  
        obj.establecerValor(5);  
    }  
}  
  
public class Main {  
    public static void main(String[] args) {  
        Objeto obj = new Objeto(1);  
        Ejemplo ej = new Ejemplo();  
        ej.cambiarValor(obj);  
        System.out.println(obj.obtenerValor()); // imprime 5  
    }  
}
```

Existe un concepto denominado “Casting entre tipos de objetos”; el cual consiste en una conversión explícita de un tipo de objeto a otro tipo. Se realiza utilizando el operador “*instanceof*” o el operador de *casting* (“*TipoDeDatos*”), a continuación, se presenta un ejemplo donde se crea un objeto de la clase Perro y se asigna a una variable de la superclase Animal. Antes de realizar el *casting*, se utiliza el operador “*instanceof*” para comprobar si el objeto es una instancia de la clase Perro. Si es así, se realiza el *casting* utilizando el operador de *casting* “Perro” .

Gráfico 34

Casting objetos

```
class Animal {}  
class Perro extends Animal {}  
public class Main {  
    public static void main(String[] args) {  
        Animal animal = new Perro();  
        if (animal instanceof Perro) {  
            Perro p = (Perro) animal;  
        }  
    }  
}
```

Si el objeto no es de la clase o subclase especificada, se lanzará una excepción de tipo “*ClassCastException*”.

En el libro de (Sierra, 2018), en la sección de paso de parámetros a métodos en la página 119 a 128 se puede ampliar esta información.

3.8. ArrayList con objetos

Los ArrayList son estructuras de datos dinámicas utilizadas para almacenar colecciones de objetos y que se ajustan automáticamente al número de elementos que las componen. Permite agregar o eliminar elementos en cualquier momento y su tamaño se vuelve a ajustar. También se pueden almacenar una colección de objetos de distintos tipos.

El ArrayList puede ser usado por cualquier tipo de dato, es decir un ArrayList<Integer> es un ArrayList de enteros, un ArrayList<String> es un ArrayList de cadenas, un ArrayList<Objetos> es un ArrayList de objetos; a continuación, se presenta un ejemplo de cómo crear ArrayList de objetos Casa.

Gráfico 35
Uso de ArrayList

```
class Casa {  
    private int valor;  
    public Casa(int valor) {  
        this.valor= valor;  
    }  
    public int ObtenerValor() {  
        return valor;  
    }  
}  
  
public class Main {  
    public static void main(String[] args) {  
  
        // Crear una ArrayList de objetos - Casa  
        ArrayList<Casa> list = new ArrayList<Casa>();  
        list.add(new Casa(120000));  
        list.add(new Casa(250000));  
        list.add(new Casa(38500));  
  
        // Acceder a los elementos de la ArrayList  
        System.out.println(list.get(0).ObtenerValor()); // imprime  
120000  
        System.out.println(list.get(1).ObtenerValor()); // imprime  
250000  
        System.out.println(list.get(2).ObtenerValor()); // imprime  
38500  
    }  
}
```

(Sierra,2018), menciona una lista de métodos usados con ArrayList, entre ellos tenemos:

- **Boolean add(E elemento):** añade el elemento “E” al final de la colección.
- **Boolean add(int index, E elemento):** añade el elemento “E” en la posición indicada en el índice.
- **E get (int index):** devuelve el elemento que ocupa la posición indicada.
- **E remove (int index):** elimina el elemento que ocupa la posición indicada y devuelve el elemento eliminado. Devuelve true si se eliminó correctamente.
- **Boolean remove(Object ob):** elimina la primera ocurrencia del objeto indicado.
- **E set (int index, E elemento):** reemplaza el elemento existente en esa posición por el nuevo elemento, devuelve el elemento sustituido.
- **Int size():** devuelve el tamaño de la colección.
- **String toString():** método heredado de Object, ArrayList lo sobrescribe para devolver una cadena con cada uno de los elementos del ArrayList.
- **T[] toArray(T[] a):** Devuelve un array con los elementos de la colección.



Actividades de aprendizaje recomendadas

Continuemos con el aprendizaje mediante su participación en las actividades que se describen a continuación:

1. Para complementar lo aprendido en esta semana, se sugiere revisar el siguiente recurso donde encontrará ejercicios para repasar en Sierra, A. (2018). Java - Curso práctico de formación para la preparación del examen de certificación Java SE Programmer I: IZO-808. Alfaomega, Páginas 221 – 227.

2. Se recomienda revisar el siguiente video [Métodos en Java](#), para que conozca como se escribe código en java usando métodos y sus tipos.
3. Se recomienda revisar el siguiente video [ArrayList De Objetos en Java](#), para que conozca como se escribe código en java usando arrayList de objetos.



Semana 14

3.9. Ambientes de desarrollo *cloud*:

Trabajar en el mundo de desarrollo de *software* ofrece muchas ventajas, una de ellas es poder trabajar de manera remota a través de entornos de desarrollo en la nube, en la actualidad este campo está muy bien visto y valorado por empresas multinacionales.

Cloud computing es un ecosistema completo y complejo que ofrece servicios a gran escala usando conexión a *Internet*. Existen varios tipos, entre ellos:

- Nube privada.
- Nuble pública.
- Nube híbrida.
- Nube comunitaria.

Y ofrece las siguientes ventajas:

- Flexibilidad a la demanda.
- Actualizaciones automáticas de *software*.
- Se puede recuperar fácilmente ante cualquier tipo de desastres.
- Facilita el trabajo remoto.
- Trabajo colaborativo para los equipos de la organización.
- Seguridad en los datos.

Además, se conforma por varios modelos:

- SaaS: Software como servicio, consiste en ofrecer aplicativos por medio de un navegador, ejemplo Google Docs .

- IaaS: Infraestructura como servicio, consiste en ofrecer recursos para los servidores remotos. Ejemplo memoria, procesador, etc.
- PaaS: Plataforma como servicio, consiste en ofrecer a plataformas que se ejecutan en servidores virtuales y se accede y administra por Internet. Además, ofrece los recursos para desarrollar aplicaciones en la nube, modelo que se desarrollará en esta unidad.

Información sobre *Cloud computing* la puede ampliar en el siguiente enlace [¿Qué es cloud computing?](#).

3.10. AWS *cloud* 9

Amazon Web Services (AWS) es una plataforma integral de computación en la nube que incluye ofertas de infraestructura como servicio (IaaS) y de plataforma como servicio (PaaS). Los servicios de AWS ofrecen soluciones escalables para la computación, el almacenamiento, las bases de datos, el análisis y mucho más. (Amazon Web Services, Inc., 2022).

Amazon Web Services, ofrece una guía compuesta de cuatro pasos para configurar el entorno de AWS de desarrollo, en el siguiente enlace [Configuración de su entorno de AWS](#).

Pasos para configurar:

- Creación de cuenta en AWS, en el siguiente enlace [Configuración de su entorno de AWS](#).

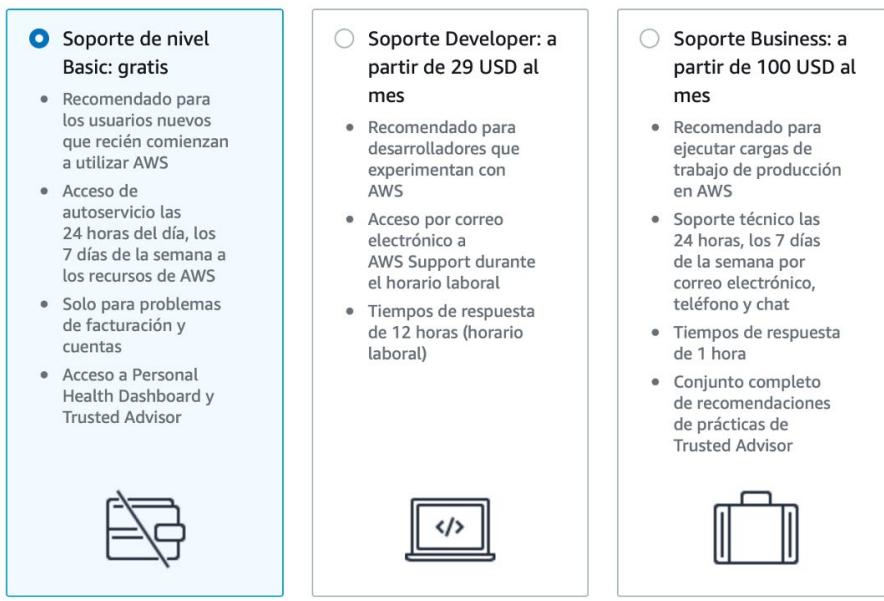
El registro finaliza seleccionando el plan: “Soporte de Nivel Básico – Gratis”, como se puede apreciar a continuación.

Figura 35
Plan de AWS

Registrarse en AWS

Seleccionar un plan de soporte

Elija un plan de soporte para su cuenta personal o empresarial. [Compare planes y ejemplos de precio](#). Puede cambiar su plan en cualquier momento desde la consola de administración de AWS.



Nota. Adaptado de Entorno de desarrollo en la nube con AWS Cloud9 (p. 7 - 8), por Cunha, A., 2020.

- Protección de cuenta: en el siguiente enlace [Configuración de su entorno de AWS](#).
- En este paso es importante crear usuario, raíz para la administración completa de los servicios, se recomienda usar autentificación por doble factor.
- Configuración de la AWS CLI: en el siguiente enlace [Module 3: Set Up the AWS CLI](#).

Para instalar el Comand Line Interface (AWS CLI): en el siguiente enlace [Installing or updating the latest version of the AWS CLI](#).

- Y para configuración del entorno de desarrollo integrado (IDE) de AWS *Cloud9*: en el siguiente enlace [Configuración de su entorno de AWS](#).

AWS *Cloud 9* es un IDE alojado en la nube, el mismo permite trabajar a través de un navegador desde cualquier parte del mundo mediante una conexión a *Internet*. Permite escribir, ejecutar códigos en distintos lenguajes de programación, entre ellos: Java, Python, PHP, etc., una de las ventajas principales es que no se necesita instalar ni configurar sus dispositivos, porque es un entorno que está listo para usarse.

[Pasos para configuración AWS Cloud9.](#)



Actividades de aprendizaje recomendadas

Estimado estudiante lo invito a desarrollar las siguientes actividades con el fin de reforzar sus conocimientos.

1. Para complementar lo aprendido en esta semana, se sugiere revisar el siguiente recurso donde encontrará información oficial sobre el Entorno de desarrollo en la nube con AWS *Cloud9*, en el siguiente enlace [Entorno de desarrollo en la nube con AWS Cloud9](#).
2. Se recomienda revisar el siguiente video [What is AWS? | Amazon Web Services](#), para que conozca sobre que es Amazon Web Services – AWS.
3. Se recomienda revisar el siguiente video [IDE en la nube cloud9 tutorial \(programadores\)](#), para que conozca sobre el IDE de Amazon Web Services, denominado AWS *Cloud9*.

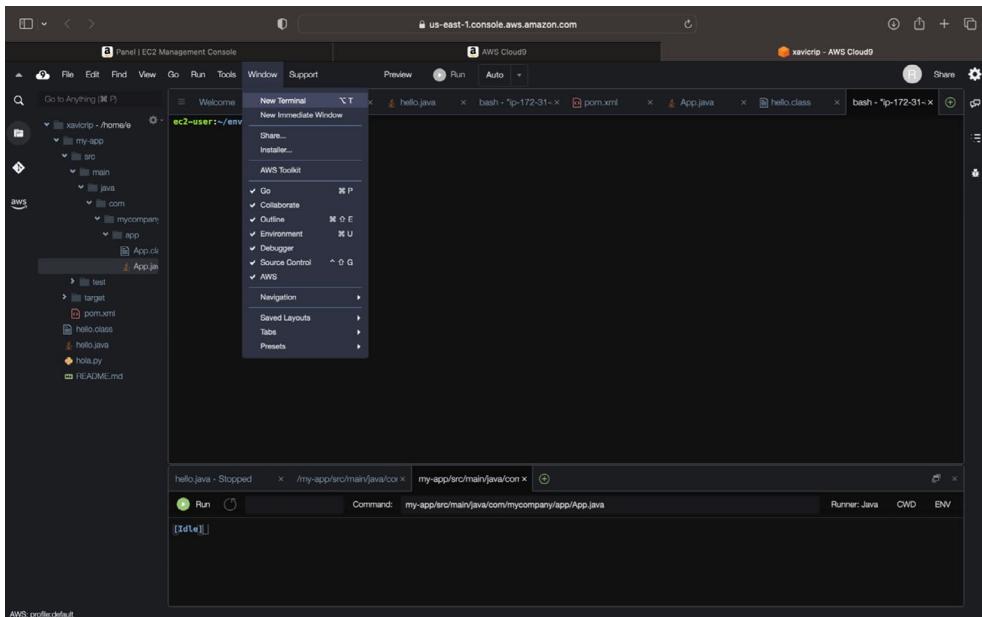


Semana 15

En la presente unidad se va a configurar el IDE AWS *Cloud9* para que se puedan ejecutar código en java. Para realizar las configuraciones, use Terminal de AWS para ejecutar las siguientes líneas de comandos, como se aprecia en la siguiente figura.

Figura 36

Uso de terminal en AWS Cloud9



Nota. Adaptado de Entorno de desarrollo en la nube con AWS Cloud9, por Cunha, A., 2020.

3.11. Gestores de paquetes java: Maven:

En esta sección se va a usar el IDE de AWS Cloud9 para configurar y poner en marcha gestores de paquetes con MAVEN, a continuación, se detallan los pasos a seguir:

Configuración:

- Como primer paso se debe instalar actualizaciones de IDE, con el siguiente comando:

Gráfico 36

Comando de actualizaciones en AWS

```
sudo yum -y update
```

Nota. (AWS, 2023)

- Luego instale OpenJDK versión 8.

Gráfico 37

Comando de instalación de OpenJDK 8

```
sudo yum -y install java-1.8.0-openjdk-devel
```

Nota. (AWS, 2023)

- Cambie java y javac a la versión 1.8

Gráfico 38

Cambiar y actualizar versión 1.8

```
sudo update-alternatives --config java-1.8  
sudo update-alternatives --config javac-1.8
```

Nota. (AWS, 2023)

- Verificar versiones de java y javac:

Gráfico 39

Comando verificación versiones de java y javac

```
java -version  
javac -version
```

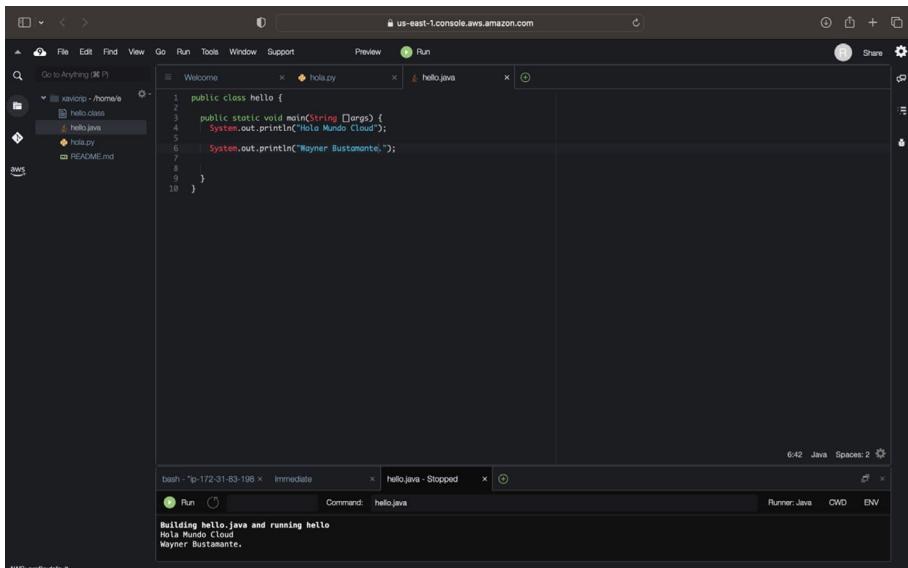
Nota. (AWS, 2023)

Código Hola Mundo:

En AWS Cloud9, cree un archivo y guárdelo con el nombre hola.java.

Figura 37

Interfaz de AWS Cloud9.

A screenshot of the AWS Cloud9 IDE interface. The top navigation bar includes File, Edit, Find, View, Go, Run, Tools, Window, Support, Preview, and Run buttons. The main workspace shows a file tree on the left with a project named 'xavcrip - home' containing files 'Hello.java', 'hola.py', and 'README.md'. In the center, there are two code editors: one for 'hola.py' and another for 'Hello.java'. The 'Hello.java' editor contains the following Java code:

```
public class Hello {
    public static void main(String []args) {
        System.out.println("Hola Mundo Cloud");
        System.out.println("Wayner Bustamante.");
    }
}
```

Below the editors is a terminal window titled 'bash - *ip-172-31-83-198 x Immediate' with the command 'hello.java' entered. The terminal output shows:

```
Building hello.java and running hello
Hola Mundo Cloud
Wayner Bustamante.
```

The bottom status bar indicates the time as 6:42 and shows Java, Spaces: 2, and other system information.

Nota. Bustamante, W., 2023.

A continuación, se muestra la ejecución del siguiente código en java.

Gráfico 40

Ejemplo de java en AWS Cloud9

```
import java.util.Scanner;
public class hello {
    public static void main(String []args) {
        System.out.println("Hola Mundo Cloud");
        System.out.println("Wayner Bustamante bienvenido a un ambiente
en la nube (AWS Cloud9)");
        int edad=0;
        String nombre="";
        String apellido="";
        Scanner entrada = new Scanner(System.in);
        System.out.println("Ingrese el nombre:");
        nombre = entrada.nextLine();
        System.out.println("Ingrese el apellido:");
        apellido = entrada.nextLine();
        System.out.println("Ingrese la edad:");
        edad = entrada.nextInt();
        // Mostrar datos:
        System.out.println("El nombre del usuario es: " + nombre + " "
+ apellido + " tiene " + edad + " años");
    }
}
```

Nota. (Bustamante, 2023)

En la figura 38 se observa la ejecución del código anterior.

Figura 38

Consola de AWS Cloud9.



A screenshot of a terminal window titled "Hello.java - Stopped". The terminal shows the following output:

```
hello.java - Stopped      x /my-app/src/main/java/cor x my-app/src/main/java/con x java - auto      x | +  
d_socket\,server=r\,\suspend=y\,address=localhost:38725 -cp /home/ec2-user/environment/.c9/xavicrip/redhat.java/jdt_ws/jdt.ls-java-project/bin hello  
Hola Mundo Cloud  
Wayner Bustamante bienvenido a un ambiente en la nube (AWS Cloud9)  
Ingrese el nombre:  
Juan Pablo  
Ingrese el apellido:  
Urrutia Jimbo  
Ingrese el edad:  
45  
El nombre del usuario es: Juan Pablo Urrutia Jimbo tiene 45 años  
ec2-user:~/environment $
```

Nota. Bustamante, W., 2023.

3.12. Configurar el uso del AWS SDK para Java

Sirve para utilizar el AWS SDK para Java, para ello es necesario instalar Apache Maven o Gradle en su entorno de desarrollo.

Maven o Gradle son sistemas de automatización de compilación comunes que se pueden utilizar con proyectos Java. Esta configuración es la básica que recomienda el sitio oficial del [AWS Cloud9](#) (Guía de Usuario).

Configurar Apache Maven.

- Para la configuración se inicia desde terminal, abriendo un nuevo terminal.

Windows – Nuevo Terminal

- Es necesario verificar la instalación de Maven.

Gráfico 41

Comando para verificar versión de Maven

```
mvn -version
```

Nota. (AWS, 2023)

- Use las siguientes líneas de comandos para instalar Maven:

Gráfico 42

Comando para instalar Maven

```
sudo wget http://repos.fedorapeople.org/repos/dchen/apache-maven/epel-apache-maven.repo -O /etc/yum.repos.d/epel-apache-maven.repo
sudo sed -i s/\$releasever/6/g /etc/yum.repos.d/epel-apache-maven.repo
sudo yum install -y apache-maven
```

Nota. (AWS, 2023)

- Confirme la instalación:

```
mvn -version
```

- Con las siguientes líneas de comandos genere un nuevo proyecto Java:

Gráfico 43

Comando crear un proyecto en Maven

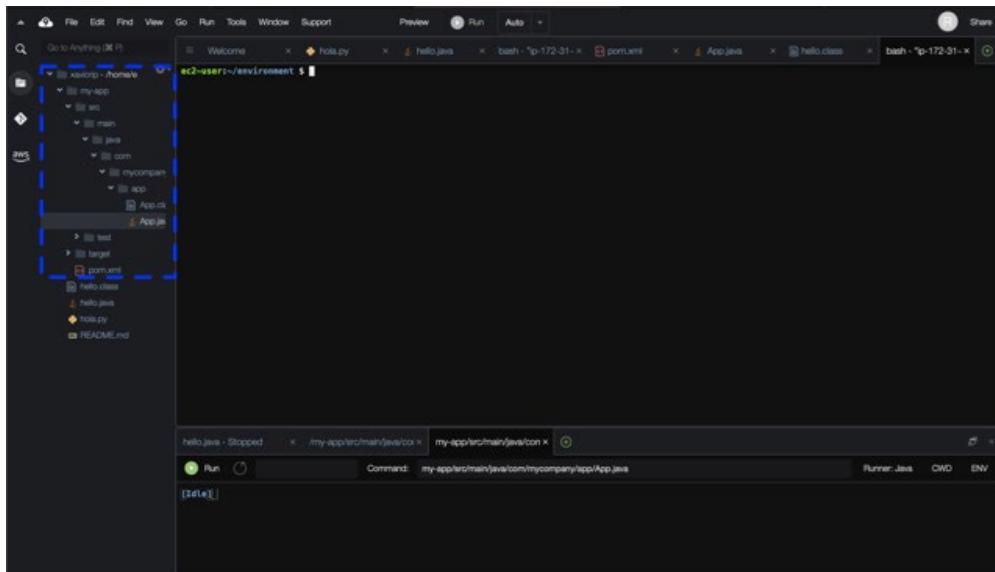
```
mvn archetype:generate -DgroupId=com.mycompany.app -DartifactId=my-app -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
```

Nota. (AWS, 2023)

La estructura de Maven, que se puede apreciar en el IDE AWS Cloud9 de la siguiente manera.

Figura 39

Estructura de Maven en AWS Cloud9.



Nota. Bustamante, W., 2023.

- A continuación, modifique el archivo Project Object Model (POM) del proyecto, usando el código 53.

Gráfico 44

Modificar POM

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.mycompany.app</groupId>
  <artifactId>my-app</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-assembly-plugin</artifactId>
        <version>3.0.0</version>
        <configuration>
          <descriptorRefs>
            <descriptorRef>jar-with-dependencies</descriptorRef>
          </descriptorRefs>
          <archive>
            <manifest>
              <mainClass>com.mycompany.app.App</mainClass>
            </manifest>
          </archive>
        </configuration>
        <executions>
          <execution>
            <phase>package</phase>
            <goals>
              <goal>single</goal>
            </goals>
          </execution>
        </executions>
      </plugin>
    </plugins>
  </build>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>com.amazonaws</groupId>
      <artifactId>aws-java-sdk</artifactId>
      <version>1.11.330</version>
    </dependency>
  </dependencies>
</project>
```

Nota. (AWS, 2023)

3.13. Gestores de paquetes java: Gradle

De igual manera, como se configuró un proyecto en Maven, se debe realizar para Gradle, a través de líneas de comandos usando terminal.

A continuación, se detallan los pasos a seguir tomados de la guía de (AWS,2023):

- Verifique si Gradle está instalado.

Gráfico 45

Comando para verificar instalación de gradle.

```
gradle -version
```

Nota. (AWS, 2023)

- A continuación, instale Gradle usando las siguientes líneas de comandos:

Gráfico 46

Comando para instalar gradle.

```
curl -s "https://get.sdkman.io" | bash  
source "$HOME/.sdkman/bin/sdkman-init.sh"  
sdk install gradle
```

Nota. (AWS, 2023)

- Confirmar la instalación:

Gráfico 47

Comando para verificar instalación de Gradle

```
gradle -version
```

Nota. (AWS, 2023)

- Cree una nueva carpeta (*my-app*) e ingrese a dicha carpeta:

Gráfico 48

Comando para crear carpeta my-app

```
mkdir my-app  
cd my-app
```

Nota. (AWS, 2023)

- Crear un nuevo proyecto con Gradle:

Gráfico 49

Comando para crear un proyecto en gradle.

```
gradle init --type java-application
```

Nota. (AWS, 2023)

- El código 59 genera un proyecto en Gradle con la siguiente estructura:

Gráfico 50

Estructura de un proyecto en gradle.

```
my-app
|- .gradle
|   `- (various supporting project folders and files)
|- gradle
|   `- (various supporting project folders and files)
|- src
|   |- main
|   |   `- java
|   |       `- App.java
|   `- test
|       `- java
|           `- AppTest.java
|- build.gradle
|- gradlew
|- gradlew.bat
`- settings.gradle
```

Nota. (AWS, 2023)

- Modifique AppTest.java, ingresando el código 60:

Gráfico 51

Comando para modificar App Test.java

```
import org.junit.Test;
import static org.junit.Assert.*;

public class AppTest {
    @Test public void testAppExists () {
        try {
            Class.forName("com.mycompany.app.App");
        } catch (ClassNotFoundException e) {
            fail("Should have a class named App.");
        }
    }
}
```

Nota. (AWS, 2023)

- También modifique el archivo build.gradle, ingresando el código 61:

Gráfico 52

Comando para modificar build.gradle

```
apply plugin: 'java'
apply plugin: 'application'
repositories {
    jcenter()
    mavenCentral()
}
buildscript {
    repositories {
        mavenCentral()
    }
    dependencies {
        classpath "io.spring.gradle:dependency-management-
plugin:1.0.3.RELEASE"
    }
}
apply plugin: "io.spring.dependency-management"

dependencyManagement {
    imports {
        mavenBom 'com.amazonaws:aws-java-sdk-bom:1.11.330'
    }
}
dependencies {
    compile 'com.amazonaws:aws-java-sdk-s3'
    testCompile group: 'junit', name: 'junit', version: '4.12'
}
run {
    if (project.hasProperty("appArgs")) {
        args Eval.me(appArgs)
    }
}
mainClassName = 'App'
```

Nota. (AWS, 2023)

3.14. Compile y ejecute el código:

Para probar la ejecución de los proyectos en Maven y Gradle, use los siguientes comandos desde el terminal. Este proceso crea un archivo JAR ejecutable para el proyecto. Use las siguientes líneas de comando para Maven y para Gradle.

- Para Maven, ejecute los siguientes comandos:

Gráfico 53

Comando para ejecutar proyecto Maven

```
cd my-app  
mvn package  
java -cp target/my-app-1.0-SNAPSHOT-jar-with-dependencies.jar  
com.mycompany.app.App my-test-bucket us-east-2
```

Nota. (AWS, 2023)

- Para Gradle, ejecute los siguientes comandos.

Gráfico 54

Comando para ejecutar proyecto gradle

```
gradle build  
gradle run -PappArgs="['my-test-bucket', 'us-east-2']"
```

Nota. (AWS, 2023)

Finalmente, el resultado se puede ejecutar por terminar mediante líneas de comando, y se observa como en el código 64.

Gráfico 55

Resultados de la ejecución de los proyectos en Maven y gradle.

```
My buckets now are:  
Creating a new bucket named 'my-test-bucket'...  
My buckets now are:  
my-test-bucket  
Deleting the bucket named 'my-test-bucket'...  
My buckets now are:
```

Nota. (AWS, 2023)



Actividades de aprendizaje recomendadas

Continuemos con el aprendizaje mediante su participación en las actividades que se describen a continuación:

1. Para complementar lo aprendido en esta semana, se sugiere revisar el siguiente recurso donde encontrará información oficial de la Guía AWS Cloud y gestor de paquetes con MAVEN .
2. Se recomienda revisar el siguiente video [AWS Cloud 9 practical on Java+Maven Example](#), para que conozca sobre que es Amazon Web Services – AWS, gestor de paquetes con MAVEN.
3. Se recomienda revisar el siguiente video [Spring boot + Gradle en AWS CLOUD9](#), para que conozca sobre el IDE de Amazon Web Services – AWS, gestor de paquetes con GRADLE.
4. Realice la autoevaluación para comprobar sus conocimientos.



Autoevaluación 3

1. ¿Qué es una herencia en programación orientada a objetos?
 - a. Una relación entre clases donde una clase hereda atributos y métodos de otra clase.
 - b. Una relación entre objetos donde un objeto tiene acceso a los atributos y métodos de otro objeto.
 - c. Una relación entre funciones donde una función tiene acceso a las variables de otra función.
 - d. Una relación entre atributos de diferentes clases.
2. El siguiente ejemplo reptiles usando herencia, presenta un error, identifique, ¿cuál es el error?

Superclase	Subclase
<pre>class Reptiles { protected String nombre; protected String tipo; public Reptiles (String nombre, String tipo) { this.nombre = nombre; this.tipo = tipo; } public void mover() { System.out.println("El reptil se mueve."); } public void comer() { System.out.println("El reptile come alfalfa."); } }</pre>	<pre>class iguana Reptiles { public iguana (String nombre) { super(iguana, escamosos); } @Override public void mover() { System.out.println("La iguana se mueve lento"); } }</pre>

- a. class iguana extends Reptiles.
- b. class Reptiles extends iguana.
- c. El código es correcto.
- d. El método mover no debe ser sobreescrito.

3. El siguiente ejemplo de formas geográficas usa polimorfismo, identifique, ¿qué le falta al código para que sea correcto?

```
abstract class Figuras {
    public double ObtenerArea();
}
class Cuadrado extends Figuras {
    private double lado;
    public Figuras (double lado) {
        this.lado = lado;
    }
    @Override
    public double ObtenerArea () {
        return lado * lado;
    }
}
class Circulo extends Figuras {
    private double radio;
    public Circulo(double radio) {
        this. radio = radio;
    }
    @Override
    public double ObtenerArea () {
        return Math.PI * radio * radio;
    }
}
class EjecutarFiguras {
    public static void main(String[] args) {
        Figuras Cuadrado = new Figuras (5);
        Figuras Circulo = new Circulo (2);
        System.out.println("El área de un cuadrado es: " +
Cuadrado.ObtenerArea ());
        System.out.println("Área del Círculo: " + Circulo.ObtenerArea
());
    }
}
```

- a. La clase padre deba tener un método abstracto.
- b. Qué los métodos de las clases hijos no sobrescriban el método `ObtenerArea()`.
- c. Los cálculos de las áreas son incorrectos.
- d. La clase principal no crea correctamente los objetos.

4. ¿Qué es AWS Cloud9?

- a. Un servicio de almacenamiento en la nube.
- b. Un entorno de desarrollo en la nube.
- c. Un servicio de escalabilidad en la nube.
- d. Es una base de datos como servicio alojado en la nube.

5. ¿Cuál de las siguientes no es una ventaja de crear un proyecto Maven en AWS Cloud9?
- Cuando crea un proyecto Maven en AWS Cloud9, le permite gestionar dependencias automáticamente.
 - Crear un proyecto en Maven le permite tener una estructura estandarizada que le permite mantener fácilmente el código.
 - Crear un proyecto en Maven con AWS Cloud9 le permite automatizar tareas como compilación, empaquetado y despliegue del código.
 - La automatización de tareas de un proyecto en Maven se logra usando su interfaz gráfica.
6. Se tiene una clase "Vehículo" en Java que tiene los métodos "acelerar()" y "frenar()", y una clase "Coche" que hereda de la clase "Vehículo". ¿Cuál de las siguientes opciones de código Java llamaría al método "acelerar()" de un objeto "Coche"?
- Vehículo miCoche = new Vehículo(); miCoche. Acelerar ();
 - Vehículo miCoche = new Coche(); miCoche. Acelerar ();
 - Coche miCoche = new Vehículo(); miCoche. Acelerar ();
 - Coche miCoche = new Coche(); miCoche. Acelerar ();
7. Se tiene una clase "Animal" en Java que tiene un método "hacerSonido()" y una clase "Perro" que hereda de la clase "Animal". Además, tienes una interfaz "Doméstico" que tiene un método "entrenar()". ¿Cuál de las siguientes opciones crearía un objeto "Perro" y llamaría al método "entrenar()" de la interfaz "Doméstico"?
- Perro miPerro = new Perro(); miPerro. Entrenar (); Vehículo miCoche = new Coche(); miCoche. Acelerar ();
 - Animal miPerro = new Perro(); miPerro. Entrenar ();
 - Domestico miPerro = new Perro(); miPerro.entrenar();
 - Perro miPerro = new Animal(); miPerro. Entrenar ();

8. Se tiene un ArrayList de números enteros en Java. ¿Cuál de las siguientes opciones elimina el primer elemento de la lista?
- a. myList.removeFirst();
 - b. myList.remove(0);
 - c. myList.removeLast();
 - d. myList.removeElementAt(0);
9. Se tiene una clase “Persona” que tiene los atributos “nombre” y “edad”. Además, tienes un ArrayList de objetos “Persona”. ¿Cuál de las siguientes opciones eliminará el objeto “Persona” con el nombre “Jorge” de la lista?
- a.

```
for (int i = 0; i < myList.size(); i++) {  
    if (myList.get(i).getNombre().equals("Jorge ")) {  
        myList.remove(i);  
        break;  
    }  
}
```
 - b.

```
for (Persona p : myList) {  
    if (p.getNombre().equals("Jorge ")) {  
        myList.remove(p);  
        break;  
    }  
}
```
 - c. myList.remove(new Persona("Jorge ", 0));
 - d. myList.remove("Jorge ");
10. ¿Cuál de las siguientes opciones es el archivo de configuración de AWS Cloud9 que se utiliza para configurar un entorno de desarrollo para Java?
- a. c9sdk.config.
 - b. Project.properties.
 - c. Build.gradle.
 - d. Pom.xml.

[Ir a solucionario](#)



Semana 16



Actividades finales del bimestre



Actividades de aprendizaje recomendadas

Estimado estudiante lo invito a desarrollar las siguientes actividades con el fin de reforzar sus conocimientos.

1. La semana 16 es un espacio de tiempo que permite a los estudiantes reforzar temáticas no comprendidas en las siete semanas de clases anteriores correspondientes al segundo bimestre, con el objetivo de poder presentarse a la evaluación; por lo cual se propone revisar:

- Unidad 3. Programación Orientada a Objetos Avanzada.

Esta revisión consiste en las siguientes actividades.

- Revisar conceptos de programación orientada a objetos avanzada, entre ellos herencia, polimorfismo y desarrollo en la nube con AWS.
- Desarrollar ejemplos prácticos de programación orientada a objetos compartidos en GitHub, entre ellos caso de estudio Equipo Básquet y Ejemplo en AWS Cloud9.

2. Con el objetivo de reforzar los contenidos de la unidad III, se recomienda ingresar al **texto básico** Deitel, P. & Deitel, H. (2016). *Java como programar*. Ciudad de México, México: Pearson Education; y trabajar con los siguientes apartados:

- **Resumen** de los siguientes capítulos VII, IX, X.
- **Ejercicios de autoevaluación** de los siguientes capítulos VII, IX, X.

Procedimiento

El **texto básico** tiene apartados denominados **Resumen** y **Ejercicios de Autoevaluación**, en esta actividad se recomienda revisar cada uno de esos apartados de los capítulos señalados anteriormente, con la finalidad que Ud. pueda:

- En el apartado **Resumen**, repasar en contenido revisado en el segundo bimestre.
 - En el apartado **Ejercicios de Autoevaluación**, ejercitarse resolviendo ejercicios prácticos y a su vez tener una pronta y oportuna retroalimentación de herencia, polimorfismo y `arrayList` en java.
3. Con el objetivo de reforzar los contenidos de la unidad III, se recomienda ingresar al **texto básico** Deitel, P. & Deitel, H. (2016). *Java como programar*. Ciudad de México, México: Pearson Education; y desarrollar los siguientes ejercicios.
- **(Sistema de reservaciones de una aerolínea).**

El enunciado lo encuentra en el ejercicio 7.19 del texto básico, en la página 301.

- **Jerarquía empleado**

El enunciado lo encuentra en el ejercicio 9.14 del texto básico, en la página 394.

- **Modificación al sistema de nómina**

El enunciado lo encuentra en el ejercicio 10.12 del texto básico, en la página 439.

Procedimiento

Para el desarrollo de los ejercicios propuesto del **texto básico**, se recomienda realizar las siguientes actividades:

- Escriba una mini especificación por cada uno de los ejercicios.
- Diagramé el modelo de la solución usando diagramas UML.
- Cree el código usando el IDE de NetBeans.



4. Glosario

Abstracción sirve para reducir la complejidad de un programa, a través del uso de clases e interfaces abstractas, que permite la reutilización de código y la simplificación a través de la herencia.

Algoritmos: conjunto ordenado de operaciones sistemáticas que permite hacer un cálculo y hallar la solución de un tipo de problemas.

Cloud: la computación en la nube, conocida también como servicios en la nube, informática en la nube, nube de cómputo o simplemente «la nube», es el uso de una red de servidores remotos conectados a Internet para almacenar, administrar y procesar datos, servidores, bases de datos, redes y software.

Encapsulamiento consiste en ocultar atributos de un objeto de manera que solo se pueda cambiar mediante operaciones definidas en ese objeto.

Herencia es el mecanismo por el cual una clase permite heredar las características (atributos y métodos) de otra clase.

Polimorfismo es una palabra de origen griego que significa “**muchas formas**” .

Programación e structurada es una forma de escribir programas de computadora, utilizando únicamente tres estructuras: secuencia, selección e iteración.

La programación orientada a objetos se basa en objetos que son capaces de interactuar y modificar los valores contenidos en sus atributos a través de sus métodos.



5. Solucionario

Autoevaluación 1		
Pregunta	Respuesta	Retroalimentación
1	A	Una función es un bloque de código que realiza una tarea específica y puede ser llamado en varias partes de un programa. Las funciones permiten organizar el código de una manera lógica y reutilizable, lo que facilita la lectura y el mantenimiento del código.
2	A	Una variable global como su nombre lo indica es una variable declarada en la clase principal y se usa en todo el código de dicha clase, mientras que las variables locales se declaran dentro de un método y función y solo sirven para operar dentro de dicha función.
3	A	Un bucle es ciclo repetitivo, una estructura de control que ejecuta n veces un conjunto de sentencias y que por lo general finaliza cuando se cumple una condición.
4	A	Un if se utiliza para evaluar una condición y ejecutar un bloque de código si se cumple, mientras que un switch se utiliza para elegir entre varias opciones de código según el valor de una variable.
5	A	Una estructura de datos que permite almacenar varios valores del mismo tipo de datos.
6	A	Un tipo de dato básico, como números o caracteres, que no puede ser dividido en otros tipos de datos.
7	A	Un int es un tipo de dato entero, mientras que un double es un tipo de dato entero y decimal.
8	A	Una variable que almacena un carácter específico no almacena cadenas ni números.
9	A	Una variable que almacena un valor lógico, verdadero o falso.
10	B	<p>El literal b, porque tiene el orden y secuencia que se necesita para su ejecución, es decir:</p> <ol style="list-style-type: none">1. Se llama al paquete a usar.2. Luego se importar cualquier clase en java.3. Declara la clase Escuela.

Autoevalación 1		
Pregunta	Respuesta	Retroalimentación
11	A	<p>63 es el resultado, para obtener este resultado se debe aplicar el orden de operaciones:</p> <ol style="list-style-type: none"> 1. El resultado de la expresión $(10 + 20 / 5) * 4.5$ sería: 2. Primero se realiza la operación de división $20 / 5 = 4$ 3. Luego, se realiza la operación de suma $10 + 4 = 14$ 4. Finalmente, se multiplica el resultado anterior por $4.5 = 63$ <p>Entonces $(10 + 20 / 5) * 4.5 = 63$</p>
12	A	<p>Le falta en la condición usar el doble <code>=</code>, es decir <code>==</code> para que la condición pueda ser comparada y no asignada como es en el caso de solo usar un signo igual <code>=</code>.</p>
13	A	<p>La condición debe ser <code>i<10</code> porque el elemento iterador empieza su ejecución desde el número 0, y para que se puedan imprimir los 10 primeros números el iterador debe llegar hasta la posición 9, es decir: 0,1,2,3,4,5,6,7,8,9; que contando resultan 10 ejecuciones.</p>
14	A	<p>El elemento seleccionado es 4; porque la posición del iterador es tres, como se muestra a continuación:</p> <pre>numbers[0]= 1 numbers[1]= 2 numbers[2]= 3 numbers[3]= 4</pre>
15	A	<p>Falta guardar el número que es ingresado por teclado en la variable num:</p> <pre>num = entrada.nextInt();</pre>

Ir a la
autoevaluación

Autoevaluación 2		
Pregunta	Respuesta	Retroalimentación
1	A	El concepto de encapsulamiento sirve para ocultar los detalles de implementación de una clase y proporcionar una interfaz pública para acceder a los atributos y métodos de la clase.
2	B	Una clase es una plantilla para crear objetos con atributos y métodos específicos.
3	A	Un objeto es una instancia de una clase con valores específicos para sus atributos.
4	A	Un método es una función dentro de una clase que realiza una tarea específica y puede ser llamada por un objeto de esa clase.
5	A	Hay que hacer privado la variable num para proteger la información almacenada en dicha clase, para que no pueda ser accedida desde otras clases.
6	A	Los modificadores de acceso son: public, private, protected
7	A	El constructor correcto es: public ConstructorExample(int num1, int num2) { this.num1 = num1; this.num2 = num2; } Porque lleva el mismo nombre de la clase y reserva memoria para cada uno de los atributos de la clase.
8	A	A los métodos se los conocen también como servicios, por lo tanto, un método public es un servicio public.
9	A	BigDecimal representar números con coma flotante, de una forma más precisa, se usa comúnmente para operaciones como suma, resta, multiplicaciones y divisiones.
10	C	Crea un objeto “Rectángulo” y utiliza el constructor que acepta los parámetros de ancho y altura para establecer los valores de las variables de instancia.

[Ir a la
autoevaluación](#)

Autoevaluación 3		
Pregunta	Respuesta	Retroalimentación
1	A	Herencia en la relación entre clases donde una subclase hereda atributos y métodos de una superclase.
2	A	Falta usar la palabra reservada extends en la subclase para que pueda heredar atributos y comportamientos de la superclase.
3	A	En polimorfismo para permitir que un objeto de una clase padre sea tratado como un objeto de una clase hija de manera diferente, se debe crear un método abstracto en la superclase.
4	B	Un entorno de desarrollo en la nube
5	D	La automatización de tareas de un proyecto en Maven se logra usando líneas de comando y con ayuda del terminal.
6	D	Crea un objeto "Coche" y llama al método "acelerar()" del objeto "Coche". Como "Coche" hereda de "Vehículo", puede utilizar los métodos de la clase "Vehículo".
7	C	Crea un objeto "Perro" y lo asigna a una variable de tipo "Domestico", que es la interfaz que tiene el método "entrenar()". Como "Perro" implementa la interfaz "Domestico", se puede llamar al método "entrenar()" del objeto "Perro".
8	B	La opción B llama al método "remove()" del objeto ArrayList y le pasa el índice 0 como argumento. Como los índices en Java comienzan en 0, esto eliminará el primer elemento de la lista.
9	A	Recorre la lista utilizando un bucle "for" y comprueba si cada objeto "Persona" en la lista tiene el nombre "Jorge". Si encuentra un objeto con el nombre "Jorge", utiliza el método "remove()" del objeto ArrayList para eliminarlo de la lista.
10	D	El archivo de configuración de AWS Cloud9 para proyectos de Java se llama "pom.xml", que es un archivo de configuración para proyectos de Maven, una herramienta de gestión de proyectos y construcción de software en Java.

Ir a la
autoevaluación



6. Referencias bibliográficas

- Blasco, F. (2019). Programación orientada a objetos en Java. Madrid, España: RA-MA.
- Cunha, A. (2020). Entorno de desarrollo en la nube con AWS Cloud9. Recuperado el 15 de diciembre de 2022, de <https://aws.amazon.com/es/blogs/aws-spanish/entorno-de-desarrollo-en-la-nube-con-aws-cloud9/>
- Deitel, H. M., y Deitel, P. J. (2016). Cómo programar en Java. México: Pearson Educación.
- Elizalde, R. (2018). Guía didáctica de Programación Orientada a Objetos. Loja, Ecuador: Universidad Técnica Particular de Loja.
- Iam, D. (2014). Java Cookbook. Sebastopol, EE. UU. : O REILLY & ASSOCIATES.
- López, L. (2013). Metodología de la programación orientada a objetos. México: Alfaomega Grupo Editor.
- Sierra, A. (2018). Java - Curso práctico de formación para la preparación del examen de certificación Java SE Programmer I: IZO-808. Madrid, España: Alfaomega Grupo Editor.



7. Anexos

Código 16. Ejemplo caso facebook

```
package com.mycompany.acd1;
/**
*
* @author xavicrip
*/
public class Acd1 {
    public static void main(String[] args) {
        // 1. Declaración de variables:
        double UsuariosFB = 1000000000; // Inicializada la cantidad de usuarios en
Facebook;
        double UsuarioFinales = 1500000000; // Usuarios Objetivo 1
        double UsuarioFinalesDos = 2000000000; // Usuarios Objetivo 2
        int FechaMes = 10;
        int FechaAnio = 2012;
        int ContadorMes = 0;
        int ContadorAnio = 0;
        int CasoMes = 0;
        String Mes = "";
        int anio = 0;
        int j = 1;
        // 2. Procedimientos:
        // Objetivo 1: Cuántos meses tardará Facebook en aumentar su base de usuarios a
mil quinientos millones?
        while (UsuarioFinales > UsuariosFB) {
            UsuariosFB = UsuariosFB * 1.04;
            ContadorMes = ContadorMes + 1;
        }
        System.out.println("El número de meses que tarda Facebook en alcanzar
1500000000 de usuarios es: " + ContadorMes);
        CasoMes = FechaMes + ContadorMes;
        if (CasoMes <= 12) {
            switch (CasoMes) {
                case 1:
                    Mes = "Enero";
                    break;
                case 2:
                    Mes = "Febrero";
                    break;
            }
        }
    }
}
```

```

        case 3:
            Mes = "Marzo";
            break;
        case 4:
            Mes = "Abril";
            break;
        case 5:
            Mes = "Mayo"; break;
        case 6:
            Mes = "Junio";
            break;
        case 7:
            Mes = "Julio";
            break;
        case 8:
            Mes = "Agosto";
            break;
        case 9:
            Mes = "Septiembre";
            break;
        case 10:
            Mes = "Octubre";
            break;
        case 11:
            Mes = "Noviembre";
            break;
        case 12:
            Mes = "Diciembre";
            break;
    }
} else {
    anio = CasoMes / 12;
    FechaAnio = FechaAnio + anio;
    CasoMes = CasoMes % 12;
    switch (CasoMes) {
        case 1:
            Mes = "Enero";
            break;
        case 2:
            Mes = "Febrero";
            break;
        case 3:
            Mes = "Marzo";
            break;
        case 4:
            Mes = "Abril";
            break;
        case 5:
            Mes = "Mayo";

```

```

        break;
    case 6:
        Mes = "Junio";
        break;
    case 7:
        Mes = "Julio";
        break;
    case 8:
        Mes = "Agosto";
        break;
    case 9:
        Mes = "Septiembre";
        break;
    case 10:
        Mes = "Octubre";
        break;
    case 11:
        Mes = "Noviembre";
        break;
    case 12:
        Mes = "Diciembre";
        break;
    }
}
System.out.println("Resultado del año es:" + FechaAnio);
System.out.println("Resultado del mes es:" + Mes);
// Objetivo dos: ¿Cuántos meses tardará Facebook en aumentar su base de usuarios
a dos mil millones?
UsuariosFB = 1000000000;
for (j = 1; UsuarioFinalesDos > UsuariosFB; j++) {
    UsuariosFB = UsuariosFB * 1.04;
}
System.out.println("El número de meses que tarda Facebook en alcanzar
2000000000 de usuarios es: " + j);
CasoMes = FechaMes + j;
if (CasoMes <= 12) {
    switch (CasoMes) {
case 1:
    Mes = "Enero";
    break;
case 2:
    Mes = "Febrero";
    break;
case 3:
    Mes = "Marzo";
    break;
case 4:
    Mes = "Abril";

```

```

        break;
case 5:
    Mes = "Mayo";
    break;
case 6:
    Mes = "Junio";
    break;
case 7:
    Mes = "Julio";
    break;
case 8:
    Mes = "Agosto";
    break;
case 9:
    Mes = "Septiembre";
    break;
case 10:
    Mes = "Octubre";
    break;
case 11:
    Mes = "Noviembre";
    break;
case 12:
    Mes = "Diciembre";
    break;
}
} else {
anio = CasoMes / 12;
FechaAnio = 2012;
FechaAnio = FechaAnio + anio;
CasoMes = CasoMes % 12;
switch (CasoMes) {
case 1:
    Mes = "Enero";
    break;
case 2:
    Mes = "Febrero";
    break;
case 3:
    Mes = "Marzo";
    break;
case 4:
    Mes = "Abril";
    break;
case 5:
    Mes = "Mayo";
    break;
case 6:

```

```
Mes = "Junio";
break;
case 7:
    Mes = "Julio";
    break;
case 8:
    Mes = "Agosto";
    break;
case 9:
    Mes = "Septiembre";
    break;
case 10:
    Mes = "Octubre";
    break;
case 11:
    Mes = "Noviembre";
    break;
case 12:
    Mes = "Diciembre";
    break;
}
}
System.out.println("Resultado del año es." + FechaAnio);
System.out.println("Resultado del mes es." + Mes);
}
```

Código 22. Caso facebook con funciones.

```
// Uso de funciones.
package com.mycompany.acdfunciones;
/**
 *
 * @author xavicrip
 */
public class AcdFunciones {
    public static void main(String[] args) {
        // 1. Declaración de variables:
        double UsuariosFB = 1000000000; // Inicializada la cantidad de usuarios en
Facebook;
        double UsuarioFinales = 1500000000; // Usuarios Objetivo 1
        double UsuarioFinalesDos = 2000000000; // Usuarios Objetivo 2
        int FechaMes = 10;
        int FechaAnio = 2012;
        int ContadorMes = 0;
        int ContadorAnio = 0;
        int CasoMes = 0;
        String Mes = "";
        int anio = 0;
        int j = 1;
        int i = 0;
        String[] arregloMes = {
            "Enero",
            "Febrero",
            "Marzo",
            "Abril",
            "Mayo",
            "Junio",
            "Julio",
            "Agosto",
            "Septiembre",
            "Octubre",
            "Noviembre",
            "Diciembre"
        };
        //CalculoMes calculo= new CalculoMes();
        while (UsuarioFinales > UsuariosFB) {
            UsuariosFB = UsuariosFB * 1.04;
            ContadorMes = ContadorMes + 1;
        }
        System.out.println("El número de meses que tarda Facebook en alcanzar
1500000000 de usuarios es: " + ContadorMes);
        CasoMes = FechaMes + ContadorMes;
        if (CasoMes <= 12) {
            CalculoMes(CasoMes);
        }
    }
}
```

```

} else {
    anio = CasoMes / 12;
    FechaAnio = FechaAnio + anio;
    CasoMes = CasoMes % 12;
    CalculoMes(CasoMes);
}
System.out.println("Resultado del año es." + FechaAnio);
System.out.println("Resultado del mes es." + Mes);
}

private static String CalculoMes(int CasoMes) {
    int x = CasoMes;
    String Mes = "";
    switch (x) {
        case 1:
            Mes = "Enero";
            break;
        case 2:
            Mes = "Febrero";
            break;
        case 3:
            Mes = "Marzo";
            break;
        case 4:
            Mes = "Abril";
            break;
        case 5:
            Mes = "Mayo";
            break;
        case 6:
            Mes = "Junio";
            break;
        case 7:
            Mes = "Julio";
            break;
        case 8:
            Mes = "Agosto";
            break;
        case 9:
            Mes = "Septiembre";
            break;
        case 10:
            Mes = "Octubre";
            break;
        case 11:
            Mes = "Noviembre";
            break;
        case 12:
            Mes = "Diciembre";
    }
}

```

```

        break;
    }
    return Mes;
}
}

```

1. Superclase:

Código 30.

Caso equipo de básquet – Superclase EquipoBasquet

```

package com.mycompany.ejecutarequipo;
/**
*
* @author xavicrip
*/
public class EquipoBasquet {
    protected int codigo;
    protected String nombre;
    protected String apellido;
    public EquipoBasquet(int codigo, String nombre, String apellido) {
        this.codigo = codigo;
        this.nombre = nombre;
        this.apellido = apellido;
    }
    public int getCodigo() {
        return codigo;
    }
    public void setCodigo(int codigo) {
        this.codigo = codigo;
    }
    public String getNombre() {
        return nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    public String getApellido() {
        return apellido;
    }
    public void setApellido(String apellido) {
        this.apellido = apellido;
    }
    // comportamiento común
    public void viajar(){
        System.out.println("Viaje a Guayaquil");
    }
}

```

2. Subclase Jugador:

Código 31.

Caso equipo de básquet – Subclase Jugador

```
package com.mycompany.ejecutarequipo;  
/**  
 *  
 * @author xavicrip  
 */  
public class jugador extends EquipoBasquet {  
    private String posicion;  
    private int edad;  
    public jugador(String posicion, int edad, int codigo, String nombre, String apellido) {  
        super(codigo, nombre, apellido);  
        this.posicion = posicion;  
        this.edad = edad;  
    }  
    public String getPosicion() {  
        return posicion;  
    }  
    public void setPosicion(String posicion) {  
        this.posicion = posicion;  
    }  
    public int getEdad() {  
        return edad;  
    }  
    public void setEdad(int edad) {  
        this.edad = edad;  
    }  
    public int getCodigo() {  
        return codigo;  
    }  
    public void setCodigo(int codigo) {  
        this.codigo = codigo;  
    }  
    public String getNombre() {  
        return nombre;  
    }  
    public void setNombre(String nombre) {  
        this.nombre = nombre;  
    }  
    public String getApellido() {  
        return apellido;  
    }  
    public void setApellido(String apellido) {  
        this.apellido = apellido;  
    }  
}
```

```

// comportamientos propios
public void jugar(){
    System.out.println("Ecuador vs Argentina");
}
public void entrenar(){
    System.out.println("Entrenamiento 08:00 a 11:00 lunes");
}
}

```

3. Subclase DirectorTecnico:

Código 32.

Caso equipo de básquet – Subclase DirectorTecnico

```

package com.mycompany.ejecutarequipo;
/**
 *
 * @author xavicrip
 */
public class DirectorTecnico extends EquipoBasquet {
    public int credencial;
    public int experiencia;
    public DirectorTecnico(int credencial, int experiencia, int codigo, String nombre,
String apellido) {
        super(codigo, nombre, apellido);
        this.credencial = credencial;
        this.experiencia = experiencia;
    }
    public int getCredencial() {
        return credencial;
    }
    public void setCredencial(int credencial) {
        this.credencial = credencial;
    }
    public int getExperiencia() {
        return experiencia;
    }
    public void setExperiencia(int experiencia) {
        this.experiencia = experiencia;
    }
    public int getCodigo() {
        return codigo;
    }
    public void setCodigo(int codigo) {
        this.codigo = codigo;
    }
    public String getNombre() {
        return nombre;
    }
}

```

```

}

public void setNombre(String nombre) {
    this.nombre = nombre;
}

public String getApellido() {
    return apellido;
}

public void setApellido(String apellido) {
    this.apellido = apellido;
}

// Comportamientos propios
public void dirigir(){
    System.out.println("Táctica Maestra");
}

public void estudiar(){
    System.out.println("Nuevos modelos de entrenamiento ofensivo");
}

}

```

4. Subclase AsistenteTecnico:

Código 33.

Caso equipo de básquet – Subclase AsistenteTecnico

```

package com.mycompany.ejecutarequipo;
/**
 *
 * @author xavicrip
 */
public class AsistenteTecnico extends EquipoBasquet {
    public String tipoAsistente;
    public AsistenteTecnico(String tipoAsistente, int codigo, String nombre, String
apellido) {
        super(codigo, nombre, apellido);
        this.tipoAsistente = tipoAsistente;
    }
    public String getTipoAsistente() {
        return tipoAsistente;
    }
    public void setTipoAsistente(String tipoAsistente) {
        this.tipoAsistente = tipoAsistente;
    }
    public int getCodigo() {
        return codigo;
    }
    public void setCodigo(int codigo) {
        this.codigo = codigo;
    }
}

```

```
public String getNombre() {
    return nombre;
}
public void setNombre(String nombre) {
    this.nombre = nombre;
}
public String getApellido() {
    return apellido;
}
public void setApellido(String apellido) {
    this.apellido = apellido;
}
// comportamiento común
public void asesorar(){
    System.out.println("conceptos de encestar 3 puntos");
}
public void estudiar(){
    System.out.println("Nuevos modelos de entrenamiento ofensivo");
}
}
```

1. Clase Principal:

Código 34.

Caso equipo de básquet – Clase principal EjecutarBasquet

```
package com.mycompany.ejecutarbasquet;
import java.util.ArrayList;
/**
*
* @author xavicrip
*/
public class EjecutarBasquet {
    public static ArrayList<EquipoBasquet> integrantes = new ArrayList<EquipoBasquet>();
    public static void main(String[] args) {
        EquipoBasquet JorgeGuzman = new Jugador(1,"Juan","Jiménez","Defensa",28);
        EquipoBasquet Duval = new DirectorTecnico(7,"Duval","Galarza",12345,2);
        EquipoBasquet Byron = new AsistenteTecnico(9,"Byron","Bustamante","Ofensivo");
        integrantes.add(JorgeGuzman);
        integrantes.add(Byron);
        integrantes.add(Duval);
        // Viajar
        System.out.println("Todos los integrantes viajan");
        for(EquipoBasquet integrante : integrantes){
            System.out.println(integrante.getNombre()+" "+integrante.getApellido());
            integrante.viajar();
        }
    }
}
```

2. Superclase Abstracta:

Código 35.

Caso equipo de básquet – Superclase abstracta EquipoBasquet

```
package com.mycompany.ejecutarbasquet;
/**
*
* @author xavicrip
*/
public abstract class EquipoBasquet {
    protected int codigo;
    protected String nombre;
    protected String apellido;
    // constructor
    public EquipoBasquet(int codigo, String nombre, String apellido) {
        this.codigo = codigo;
        this.nombre = nombre;
```

```

        this.apellido = apellido;
    }
    // métodos get y set
    public int getCodigo() {
        return codigo;
    }
    public void setCodigo(int codigo) {
        this.codigo = codigo;
    }
    public String getNombre() {
        return nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    public String getApellido() {
        return apellido;
    }
    public void setApellido(String apellido) {
        this.apellido = apellido;
    }
    // métodos propios
    public void viajar(){
        System.out.println("Viajar a Guayaquil");
    }
    // Método Abstracto
    public abstract void cobrar();
}

```

3. Subclase Jugador:

Código 36.

Caso equipo de básquet – Subclase Jugador

```

package com.mycompany.ejecutarbasquet;
/**
 *
 * @author xavicrip
 */
public class Jugador extends EquipoBasquet {
    private String posicion;
    private int edad;
    public Jugador(int codigo, String nombre, String apellido, String posicion, int edad) {
        super(codigo, nombre, apellido);
        this.posicion = posicion;
        this.edad = edad;
    }
    public String getPosition() {

```

```

        return posicion;
    }
    public void setPosicion(String posicion) {
        this.posicion = posicion;
    }
    public int getEdad() {
        return edad;
    }
    public void setEdad(int edad) {
        this.edad = edad;
    }
    public int getCodigo() {
        return codigo;
    }
    public void setCodigo(int codigo) {
        this.codigo = codigo;
    }
    public String getNombre() {
        return nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    public String getApellido() {
        return apellido;
    }
    public void setApellido(String apellido) {
        this.apellido = apellido;
    }
    // comportamientos propios
    public void jugar(){
        System.out.println("Ecuador vs Argentina");
    }
    public void entrenar(){
        System.out.println("Entrenamiento 08:00 a 11:00 lunes");
    }
    @Override
    public void cobrar() {
        System.out.println("Su salario es: $1500 dólares");
    }
}

```

4. Subclase DirectorTecnico:

Código 37.

Caso equipo de básquet – Subclase DirectorTecnico

```
package com.mycompany.ejecutarbasquet;
```

```

/**
 *
 * @author xavicrip
 */
public class DirectorTecnico extends EquipoBasquet {
    public int credencial;
    public int experiencia;
    public DirectorTecnico(int codigo, String nombre, String apellido, int credencial, int
experiencia) {
        super(codigo, nombre, apellido);
        this.credencial = credencial;
        this.experiencia = experiencia;
    }
    public int getCredencial() {
        return credencial;
    }
    public void setCredencial(int credencial) {
        this.credencial = credencial;
    }
    public int getExperiencia() {
        return experiencia;
    }
    public void setExperiencia(int experiencia) {
        this.experiencia = experiencia;
    }
    public int getCodigo() {
        return codigo;
    }
    public void setCodigo(int codigo) {
        this.codigo = codigo;
    }
    public String getNombre() {
        return nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    public String getApellido() {
        return apellido;
    }
    public void setApellido(String apellido) {
        this.apellido = apellido;
    }
    // Comportamientos común
    public void dirigir(){
        System.out.println("Táctica Maestra");
    }
    public void estudiar(){

```

```

        System.out.println("Nuevos modelos de entrenamiento ofensivo");
    }
    @Override
    public void cobrar() {
        System.out.println("Su salario es: $9000 dólares.");
    }
}

```

5. Subclase AsistenteTecnico:

Código 38.

Caso equipo de básquet – Subclase AsistenteTecnico

```

package com.mycompany.ejecutarbasquet;
/**
 *
 * @author xavicrip
 */
public class AsistenteTecnico extends EquipoBasquet {
    public String tipoAsistente;
    public AsistenteTecnico( int codigo, String nombre, String apellido, String
    tipoAsistente) {
        super(codigo, nombre, apellido);
        this.tipoAsistente = tipoAsistente;
    }
    public String getTipoAsistente() {
        return tipoAsistente;
    }
    public void setTipoAsistente(String tipoAsistente) {
        this.tipoAsistente = tipoAsistente;
    }
    public int getCodigo() {
        return codigo;
    }
    public void setCodigo(int codigo) {
        this.codigo = codigo;
    }
    public String getNombre() {
        return nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    public String getApellido() {
        return apellido;
    }
    public void setApellido(String apellido) {
        this.apellido = apellido;
    }
}

```

```
// comportamiento común
public void asesorar(){
    System.out.println("conceptos de encestar 3 puntos");
}
public void estudiar(){
    System.out.println("Nuevos modelos de entrenamiento ofensivo");
}
// Método Abstracto de la Superclase
@Override
public void cobrar() {
    System.out.println("Su salario es: $3000 dólares.");
}
```