Universidad Continental

Programación II

Henry Joe Wong Urquiza

INNOVADOR · INTERACTIVO · INTEGRAL · 3i

Datos de Catalogación Bibliográfica

Asignatura Programación II. Manual Autoformativo /
Henry Joe Wong Urquiza
- Huancayo:
Universidad Continental. 2017. - 82 p.

Datos de catalogación del CENDOC UC

Asignatura Programación II. Manual Autoformativo
Henry Joe Wong Urquiza

Primera edición

Huancayo, mayo de 2017

Av. San Carlos 1980, Huancayo-Perú
Teléfono: (51 64) 481-430 anexo 7361
Correo electrónico: recursosucvirtual@continental.edu.pe
http://www.continental.edu.pe/

Versión e-book
Disponible en http://repositorio.continental.edu.pe/
ISBN electrónico n.° 978-612-4196-

Dirección: Emma Barrios Ipenza
Edición: Miguel Angel Cordova Solis
Asistente de edición: Andrid Kary Poma Acevedo
Asesor didáctico: Mary Karina Calle Loayza
Corrección de textos: Sara Maricruz Bravo Montenegro
Diseño y diagramación: Alexander Frank Vivanco Matos

ÍNDICE

INTRODUCCIÓN

El curso de especialidad de Programación en la carrera de Ingeniería de Sistemas de Información de modalidad a distancia, de carácter teórico-práctico, está dirigido a los estudiantes del XXI ciclo. Busca desarrollar como competencia general el pensamiento crítico y como competencia específica la participación en equipos multidisciplinarios, lo cual les permitirá trabajar con profesionales de diferentes especialidades o dominios de aplicación.

En general, los contenidos propuestos en el manual autoformativo se dividen en cuatro unidades: Fundamentos de programación, Programación orientada a objetos I, Programación orientada a objetos II y Diseño orientado a objeto, que incluyen temas como la introducción a la plataforma Java EE, creación de objetos, herencia y características de las aplicaciones orientadas a objetos en modo consola.

Es recomendable que el estudiante desarrolle una permanente lectura de estudio esey de los textos seleccionados que amplían o profundizan el tratamiento de la información, junto a la elaboración de resúmenes y una minuciosa investigación vía Internet. El desarrollo del manual se complementa con autoevaluaciones, que son una preparación para la prueba final de la asignatura de Programación II, que se basará en el desarrollo de una aplicación orientada a objetos en modo consola que cumpla los requerimientos funcionales de un usuario.

Organice su tiempo para que obtenga buenos resultados; la clave está en encontrar el equilibrio entre sus actividades personales y las actividades que asume como estudiante. El estudio a distancia requiere constancia, por ello es necesario encontrar la motivación que le impulse a ser mejor profesional cada día.

El Autor

ORGANIZACIÓN DE LA ASIGNATURA

Resultado de aprendizaje de la asignatura

Al término de la asignatura el estudiante será capaz de desarrollar aplicaciones usando la tecnología J2EE para implementar requerimientos funcionales de los usuarios.

Unidades didácticas

UNIDAD I	UNIDAD II	UNIDAD III	UNIDAD IV
Fundamentos de programación	Programación orientada a objetos I	Programación orientada a objetos II	Diseño orientado a objetos
Resultado de aprendizaje	Resultado de aprendizaje	Resultado de aprendizaje	Resultado de aprendizaje
Al finalizar la unidad, el estudiante diseña e implementa aplicaciones considerando los fundamentos de programación orientada a objetos.	Al finalizar la unidad el estudiante diseña e implementa aplicaciones con interfaz gráfica utilizando la programación orientada a objetos.	Al finalizar la unidad el estudiante diseña e implementa aplicaciones con interfaz gráfica utilizando la programación orientada a objetos y el uso de colecciones.	Al finalizar la unidad el estudiante utiliza la metodología orientada a objetos para el diseño de una aplicación centrada en el usuario.

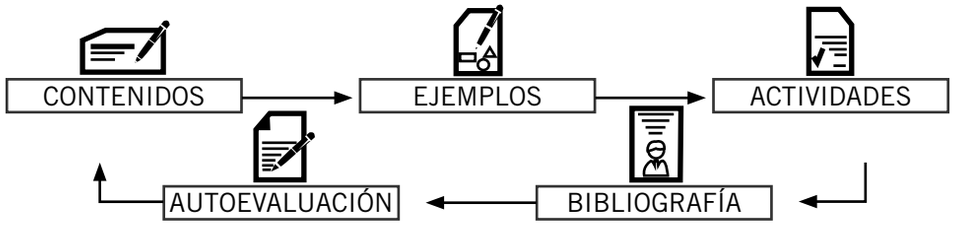
Tiempo mínimo de estudio

UNIDAD I	UNIDAD II	UNIDAD III	UNIDAD IV
Semana 1 y 2	Semana 3 y 4	Semana 5 y 6	Semana 7 y 8
24 horas	24 horas	24 horas	24 horas

UNIDAD I

FUNDAMENTO DE PROGRAMACIÓN

DIAGRAMA DE ORGANIZACIÓN DE LA UNIDAD I



ORGANIZACIÓN DE LOS APRENDIZAJES

Resultados del aprendizaje de la Unidad I: Al finalizar la unidad el estudiante diseña e implementa aplicaciones considerando los fundamentos de programación orientada a objetos.

CONOCIMIENTOS	HABILIDADES	ACTITUDES
Tema n.º 1: Antecedentes sobre java 1. ¿Qué es Java? 2. Los primeros programas en Java **Tema n.°: Estructura de un programa en java** 1. Sintaxis de un programa en Java (variables y Cálculos) 2. Parámetros y métodos **Lectura seleccionada n.º 1:** La plataforma de programación Java. **Lectura seleccionada n.º 2:** Programas en Java **Autoevaluación de la Unidad I**	1. Identifica las características de la arquitectura de la tecnología Java. 2. Diferencia los términos que se utilizan para el desarrollo de programas en Java. **ACTIVIDAD N.º 1** Configurando variables de entorno **ACTIVIDAD N.º 2** Programa "Hola Mundo" **CONTROL DE LECTURA N.º 1** Evaluación de la lectura seleccionada y los temas n.º 1 y 2	Valora la importancia de desarrollar un sistema en multiplataforma.

Antecedentes sobre Java

Tema n.° 1

A partir de la información teórica proporcionada en el curso de Programación I, durante este tema se explicarán los temas correspondientes a los antecedentes de Java para que sea capaz de reconocer las características principales del lenguaje de programación y cuál es la arquitectura tecnológica que usa.

1. ¿Qué es Java?

Java es una plataforma y lenguaje orientado a objetos. Fue diseñado originalmente por Sun Microsystems para aparatos electrodomésticos y actualmente pertenece a Oracle. Contiene una librería de clases Base para el desarrollo de sus aplicaciones y que sea de fácil uso para los programadores de aplicaciones de escritorio, web y móviles.

Las características principales de Java son las siguientes (Oracle University, 2016):

- Simple
- Orientado a objetos
- Distribuido
- Multihilos
- Dinámicos
- Arquitectura neutral
- Portable
- Robusto
- Seguro

1.1. Java Virtual Machine (JVM)

Java utiliza una máquina virtual para la ejecución de un programa. Existe una máquina virtual por cada sistema operativo, lo cual permite que una aplicación desarrollada en Java se pueda ejecutar en diferentes sistemas operativos.

Lo explicado anteriormente se puede expresar en el siguiente gráfico, donde una aplicación web o consola necesita el JVM (Java Virtual Machine) para poder ejecutarse y una aplicación del tipo Applet, que es una aplicación de modo escritorio, necesita también un navegador para poder ejecutarse.

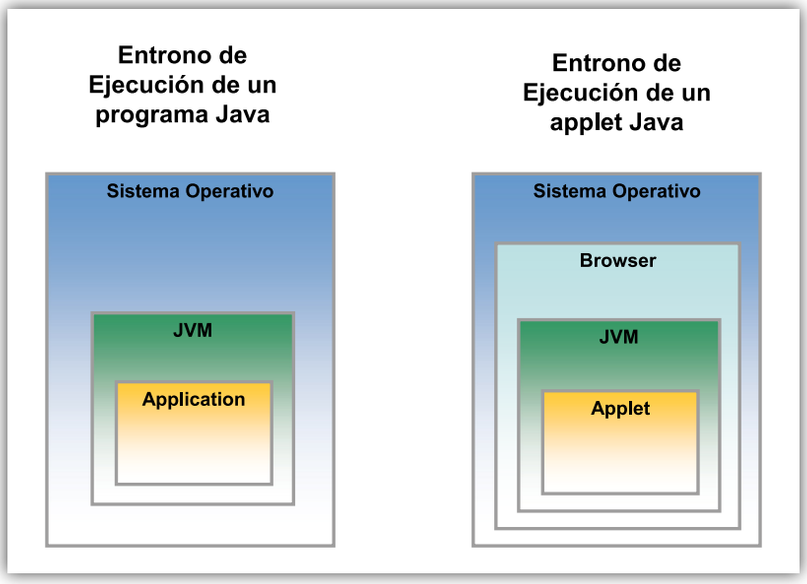


Figura 1. JVM. Fuente:
http://es.slideshare.net/nano_trujillo/mdulo-01-introduccin-a-la-tecnologa-java

1.2. Java Developer Kit y Java Runtime Edition

1.2.1. Java Developer Kit (JDK)

Es un programa que se instala en cualquier sistema operativo y permite el desarrollo y ejecución de aplicaciones en Java.

1.2.2. Java Runtime Enviroment (JRE)

Es un programa que se instala en cualquier sistema operativo y permite la ejecución de aplicaciones en Java.

La JVM se configura en el sistema operativo, instalando el Java Developer Kit (JDK) o el Java Runtime Enviroment (JRE); el instalador se encuentra en el siguiente enlace:

http://www.oracle.com/technetwork/es/java/javase/downloads/index.html

Al descargar podrá ver una página como la que se muestra en la Figura 2 y debe de hacer clic en cualquier cuadro indicado, según la necesidad que se tenga.

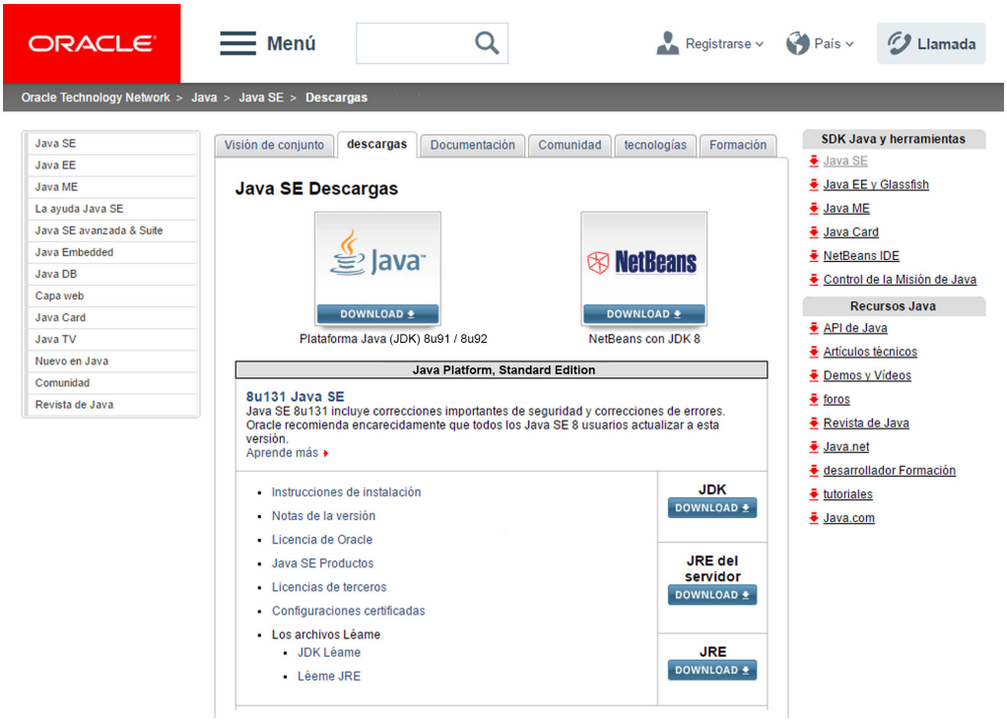


Figura 2. Página para descargar el JDK o JRE. Fuente: Tomada de Oracle, 2016.

2. Los primeros programas en Java

2.1. Compiladores en Java

En el lenguaje de programación, se puede codificar en cualquier editor de texto y dicho archivo debe terminar con la extensión .java. Luego, para saber que la codificación sea la adecuada, es necesario que dicho archivo sea compilado y si todo es correcto, generará un archivo con extensión .class y este archivo es el que necesita el JVM para poder ejecutar la aplicación.

Todo lo explicado anteriormente se puede representar en el siguiente gráfico:

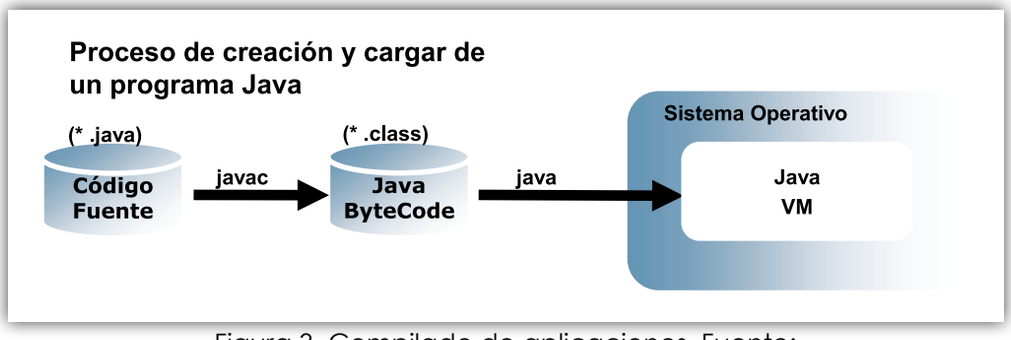


Figura 3. Compilado de aplicaciones. Fuente: http://es.slideshare.net/nano_trujillo/mdulo-01-introduccin-a-la-tecnologa-java

Como se puede apreciar en el gráfico anterior, es necesario ejecutar ciertos comandos en consola, como los siguientes:

- javac : Comando que es necesario para compilar archivos *.java
- java : Comando que es necesario para ejecutar archivos *.class

Usualmente, un lenguaje —no todos los lenguajes, pero sí la mayoría— suele ejecutarse solamente en una plataforma para la cual fue diseñado. Se han normalizado algunos lenguajes de programación (por ejemplo, ANSI C) con lo que se mantiene cierta compatibilidad; sin embargo, siempre es necesaria la recompilación del código fuente porque cada fabricante agrega funcionalidad no estándar a un lenguaje de programación y cada sistema operativo restringe a los programas en los servicios y capacidades de los mismos.

Java se diferencia a los otros lenguajes porque es multiplataforma, porque fue diseñado para que se ejecute en sistemas heterogéneos para que no dependa de la plataforma de *hardware* y *software* en la que se ejecuta. También provee las mismas librerías (API) para cada sistema operativo que se ejecuta.

Para eso, cuando se compila un código, se generan los bytecodes (*.class) independientes de la máquina. Todos los sistemas operativos principales tienen los entornos de ejecución necesarios (JVM).

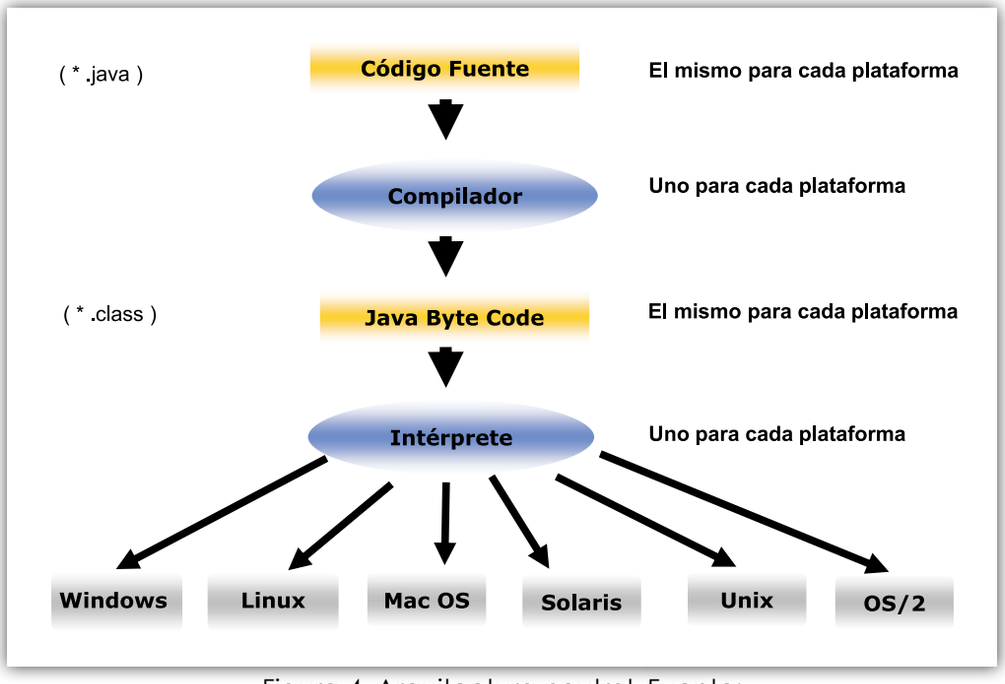


Figura 4. Arquitectura neutral. Fuente:
http://es.slideshare.net/nano_trujillo/mdulo-01-introduccin-a-la-tecnologa-java

2.2. Variables de entorno JAVA_HOME y PATH

Para que se pueda ejecutar los programas en Java, se tiene que configurar ciertas variables de sistema en el sistema operativo donde se desee ejecutarlos.

En el caso de Java se tiene que configurar las siguientes variables de entorno:

- JAVA_HOME: Ruta en donde se instaló el JDK o el JRE
- PATH: Ruta en donde se encuentra los ejecutables de Java, que usualmente se encuentran en la carpeta BIN

Estructura de un programa en Java

Tema n.° 2

En el tema n.º 1, se trató todo lo referente a la arquitectura de Java, en este punto se desarrollará cuál es la estructura de un programa en Java y cuáles son sus principales componentes.

1. Sintaxis de un programa en Java (variables y cálculos)

Java se encuentra conformado por los siguientes componentes léxicos:

- **Token**
  Es el componente léxico de un lenguaje de programación.
- **Palabra reservada**
  Es la palabra que tiene un significado y uso concreto en el programa.
- **Identificador**
  Son las palabras que se pueden usar para definir algo en el programa.
- **Literal**
  Es la especificación de un valor concreto de un tipo de dato.
- **Operadores**
  Realizan una acción específica en el programa.
  - Suelen estar definidos en el núcleo del compilador.
  - Suelen representarse con tokens formados por símbolos.
  - Suelen utilizar notación infija.
  - Pueden aplicarse a uno o varios operadores (argumentos).
  - Suelen devolver un valor.
- **Delimitadores**
  Son los símbolos utilizados como separadores de las distintas construcciones de un lenguaje de programación; es decir, son los signos de puntuación del lenguaje de programación.
- **Comentarios**
  Es la aclaración que el programador incluye en el texto del programa para mejorar su inteligibilidad.

1.1. Palabras reservadas

Las palabras reservadas, como se explicó anteriormente, no se pueden usar como un identificador en nuestros programas porque tiene un uso específico y son las siguientes:

abstract	continue	for	new	switch
assert***	default	goto*	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum****	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp**	volatile
const*	float	native	super	while

Figura 5. Palabras reservadas. Fuente: Tomada de Oracle University, 2016.

Es recomendable tener mucho cuidado, pues no se deben usar como variables o nombre de métodos.

1.2. Identificadores

Los identificadores son tokens que representan nombres asignables a variables, métodos y clases para identificarlos de forma única ante el compilador y darles nombres con sentido para el programador. Todos los identificadores de Java diferencian entre mayúsculas y minúsculas (Java es Case Sensitive o Sensible a mayúsculas) (Alvarellos & Pastor, 2012, p.5)

Los identificadores, como explica Alvarellos & Pastor (2012), sirven para nombrar variables, funciones, clases y objetos. Los identificadores deben de cumplir con un estándar de codificación cuyo uso no es necesariamente obligatorio, pero se considera como una buena práctica de programación. A continuación, nombraremos algunos:

- El primer símbolo del identificador será un carácter alfabético (a, …, z, A, …, Z, '_', '$'), pero no un dígito.
- Después de ese primer carácter, podremos poner caracteres alfanuméricos (a, …, z) y (0, 1, …, 9), signos de dólar '$' o guiones de subrayado '_'.
- Los identificadores no pueden coincidir con las palabras reservadas.
- Las mayúsculas y las minúsculas se consideran diferentes.
- El signo de dólar y el guion de subrayado se interpretan como una letra más.

Para ver este estándar de programación, puede entrar a la siguiente ruta:

http://www.oracle.com/technetwork/java/codeconventions-150003.pdf

En esta ruta se encuentra el estándar de programación en Java, que ha sido normada por Oracle para que todos los programadores lo utilicen.

1.3. Variables

Java es un lenguaje fuertemente tipado, lo que significa que cada variable debe tener un tipo declarado. Para declarar una variable se coloca en primer lugar el tipo, seguido del nombre que la identificará. Por ejemplo:

- int valorEntero
- float valorDecimal
- char ch1,ch2

1.4. Literales

Los literales sirven para almacenar valores en las variables y existen para los siguientes casos:

- **Números enteros**
  21 (int), 21L (long), 077 (en octal), 0xDC00 (en hexadecimal)

- **Números reales**
  3.14 (double), 3.14f (float), 3.14d (double), 2e12, 3.1E12

- **Valores booleanos**
  true (verdadero), false (falso)

- **Caracteres**
  'a'

- **Cadenas de caracteres**
  "mensaje"

1.5. Operadores

Los operadores en Java, como en las matemáticas, tienen una función específica en diferentes operandos y luego devuelven un resultado. Los operadores de las tablas siguientes se enumeran según el orden de precedencia. Cuanto más cerca de la parte superior de la tabla aparezca un operador, mayor será su prioridad, y cuando en la misma expresión aparezcan operadores de la misma procedencia, se evaluarán los que están en primer lugar.

Las tablas siguientes han sido extraídas de la página oficial de Oracle (Oracle University, 2016):

1.5.1. Aritméticos

Tabla 1.
Operadores aritméticos

Operador	Operación
+ - * /	Adición, sustracción, multiplicación y división.
%	Módulo
++	Incremento en 1
--	Decremento en 1
-	Cambio de signo
~	Complemento al bit
&	Y binario (AND al bit)
|	O inclusivo binario (OR al bit)
^	O exclusivo binario (XOR al bit)
<<	Desplazamiento de bits a la izquierda
>>	Desplazamiento de bits a la derecha
>>>	Desplazamiento de bits a la derecha (sin considerar signo)

Fuente: Oracle University, 2016

1.5.2. De comparación

Tabla 2.
Operadores de comparación

Operador	Operación
==	Igual
!=	Diferente
<	Menor que
<=	Menor o igual que
>	Mayor que
>=	Mayor o igual que

Fuente: Oracle University, 2016.

1.5.3. Lógicos

Tabla 3.
Operadores lógicos

Operador	Operación
&	Y lógico (AND) de evaluación completa
|	O lógico inclusivo (OR) de evaluación completa
^	O lógico exclusivo (XOR)
&&	Y lógico (AND) de evaluación "suficiente"
||	O lógico inclusivo (OR) de evaluación "suficiente"
!	Negación lógica
==	Igual
!=	Diferente
?:	Condicional ternario (Sintaxis: ExpBooleana? ExpSiTrue : ExpSiFalse)

Fuente: Oracle University, 2016.

1.5.4. De conversión

Tabla 4.
Operadores de conversión

Operador	Operación
=	Asignación
+=	Suma y asigna
-=	Resta y asigna
*=	Multiplica y asigna
/=	Divide y asigna
%=	Módulo y asigna
&=	Y lógico (AND) y asigna
|=	O lógico inclusivo (OR) y asigna
^=	O lógico exclusivo (XOR) y asigna

Fuente. Oracle University, 2016.

1.6. Delimitadores

Los delimitadores sirven para separar una línea código o un bloque de código y tenemos los siguientes:

- ( ) Paréntesis: Listas de parámetros en la definición y llamada a métodos, precedencia en expresiones, expresiones para control de flujo y conversiones de tipo.
- { } Llaves: Inicialización de *arrays*, bloques de código, clases, métodos y ámbitos locales.
- [ ] Corchetes: *Arrays*.
- ; Punto y coma: Separador de sentencias.
- , Coma: Identificadores consecutivos en una declaración de variables y sentencias encadenadas dentro de una sentencia *for*.

- . Punto: Separador de nombres de paquetes, subpaquetes y clases; separador entre variables y métodos/miembros.

1.7. Comentarios

Los comentarios son sentencias de código que sirven para documentar un programa que desarrollemos. En Java existen tres tipos de comentarios:

- Comentarios de una sola línea (//)
- Comentarios de una o más líneas (/* */)
- Comentarios de documentación (/** */)

2. Parámetros y métodos

Los métodos son líneas de código que entiende y manipula el estado de un programa en Java y sirve para invocar a operaciones definidas por nosotros. Estos métodos pueden ser llamados dentro de la clase o por otras clases. Además, es obligatorio indicar el tipo de retorno o void.

La forma de declaración de un método es la siguiente:

```
[acceso][static]tipoRetorno nombreMetodo ([parametros]){
   //cuerpo del metodo
   [return valorRetorno;]
}
```

Como se ve en la declaración anterior, un método puede recibir varios parámetros. Un parámetro puede ser usado dentro de un método para poder realizar cualquier acción con su valor, por ejemplo:

```
public int calcularSuma(int numero01, int numero02){
   return numero01 + numero02;
}
```

Lectura seleccionada n.º 1: Introducción a la programación en Java

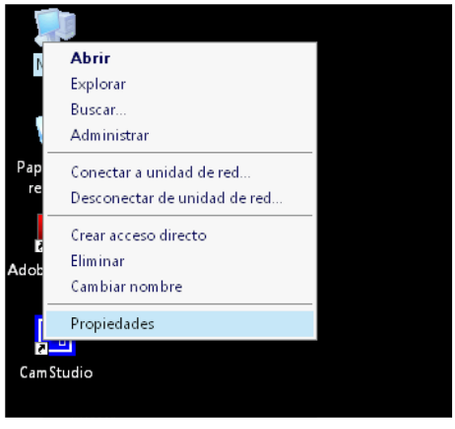
Berzal, F. (2011). *Introducción a la programación Java*. Disponible en https://goo.gl/E3xjfc

Actividad n.º 1

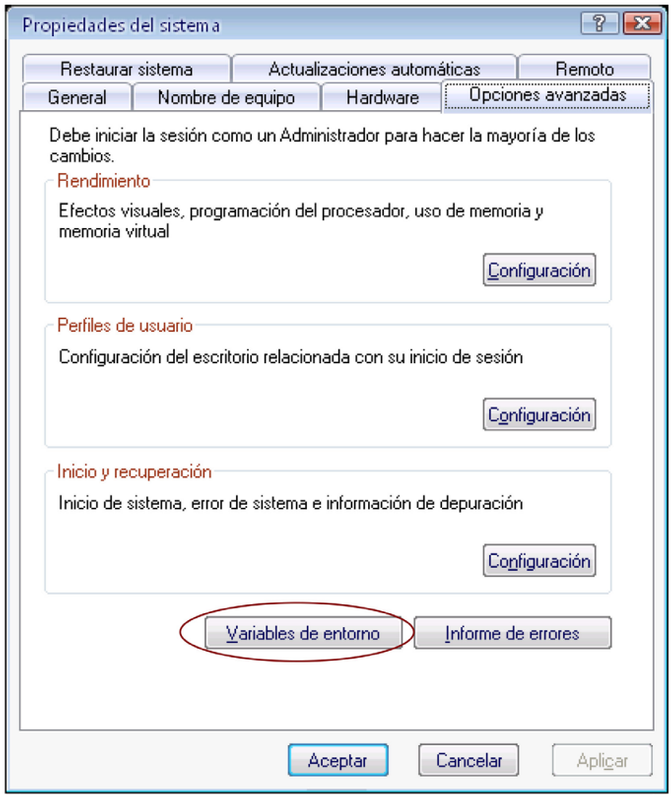
El objetivo de la práctica es usar el kit de desarrollo de Java (JDK) y aprender a configurar las variables del sistema que utiliza el lenguaje Java. Para eso debe de haber instalado el JDK, indicado en el punto 1.2.: Java Developer Kit Y Java Runtime Edition"

1. Antes de hacer cualquier aplicación en Java se debe configurar las variables del sistema, que en este caso son las tres más importantes que utiliza el lenguaje Java como son CLASSPATH, PATH Y JAVA_HOME.

   1.1. Clic derecho en el ícono "Mi PC" y clic en "Propiedades".



   1.2. Nos mostrará la ventana de "Propiedades del sistema". Luego, nos debemos dirigir a la pestaña "Opciones avanzadas" y hacer clic en el botón "Variables de entorno".

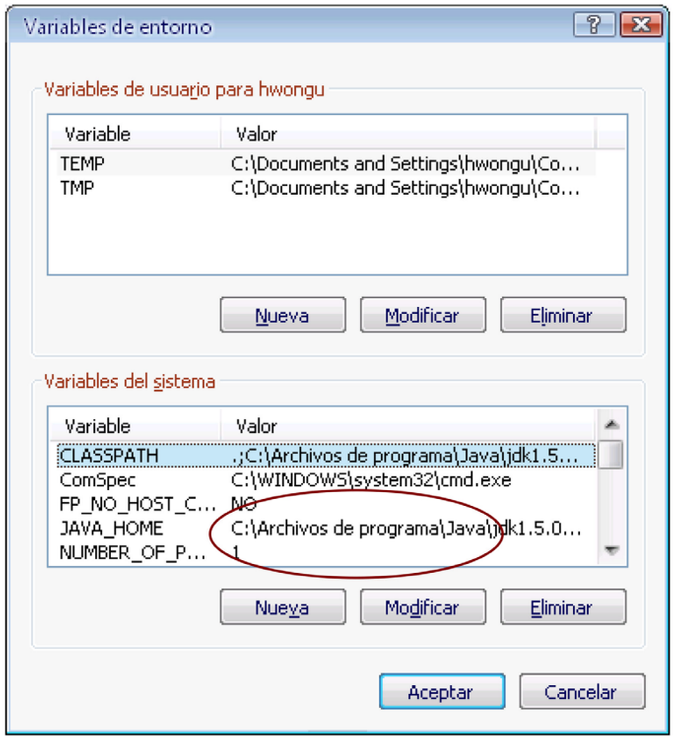


1.3. Luego, hacemos clic en "Nueva" o "Modificar" en los rectángulos de "Variables del sistema" según sea el caso. Si es que las variables Classpath, Path y Java_Home ya existieran, se recomienda agregarles un punto y coma (;) al final de la dirección que tienen, para luego agregarle la nueva dirección a la que se desea que apunte y no alterar cualquier funcionamiento de la computadora.

PATH:
C:\Archivos de programa\Java\jdk<version>\bin

CLASSPATH
C:\Archivos de programa\Java\jdk<version>\src.zip

JAVA_HOME
C:\Archivos de programa\Java\jdk<version>



Nota: Si no existiera la variable Classpath, se tiene que poner antes de la dirección un punto seguido de un punto y coma (.;). Por ejemplo:

.;C:\Archivos de programa\Java\jdk<version>src.zip

Lectura seleccionada n.º 2

Berzal, F. (2011). Programas. En *Introducción a la programación Java* (pp. 54–66). Disponible en https://goo.gl/MjiWhX

Actividad n.º 2

El objetivo de la práctica es que pueda crear su primer programa en Java.

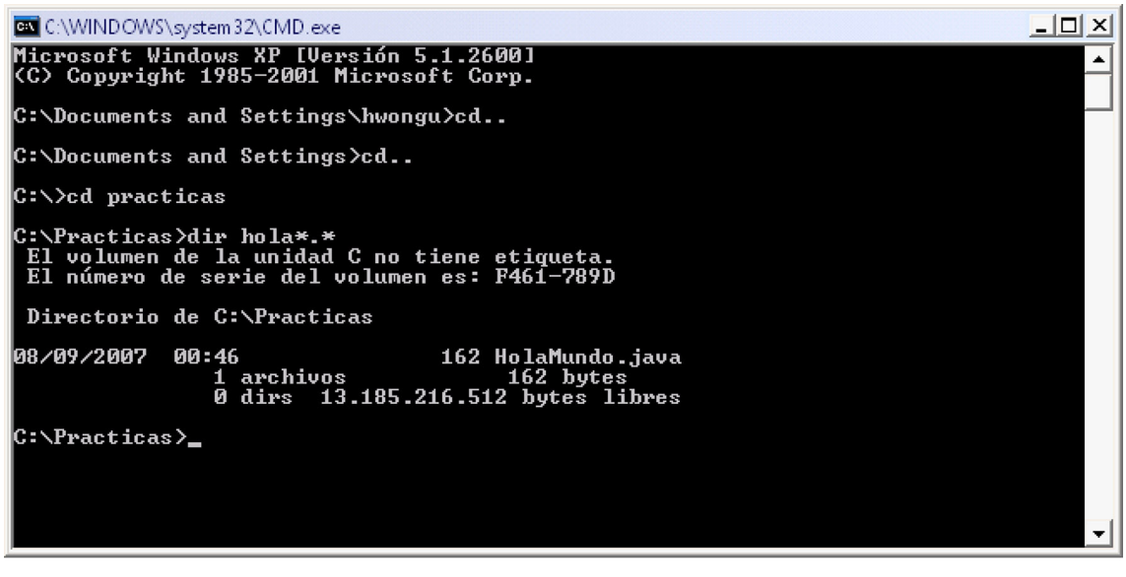
1. Cree una carpeta dentro del disco "C" que se llame "Prácticas". Luego, abra el bloc de notas y copie el siguiente código:

```
//Archivo HolaMundo.java
//Autor <<Tu Nombre>>

public class HolaMundo{
 public static void main(String args[]){
  System.out.println("Hola mundo");
 }
}
```

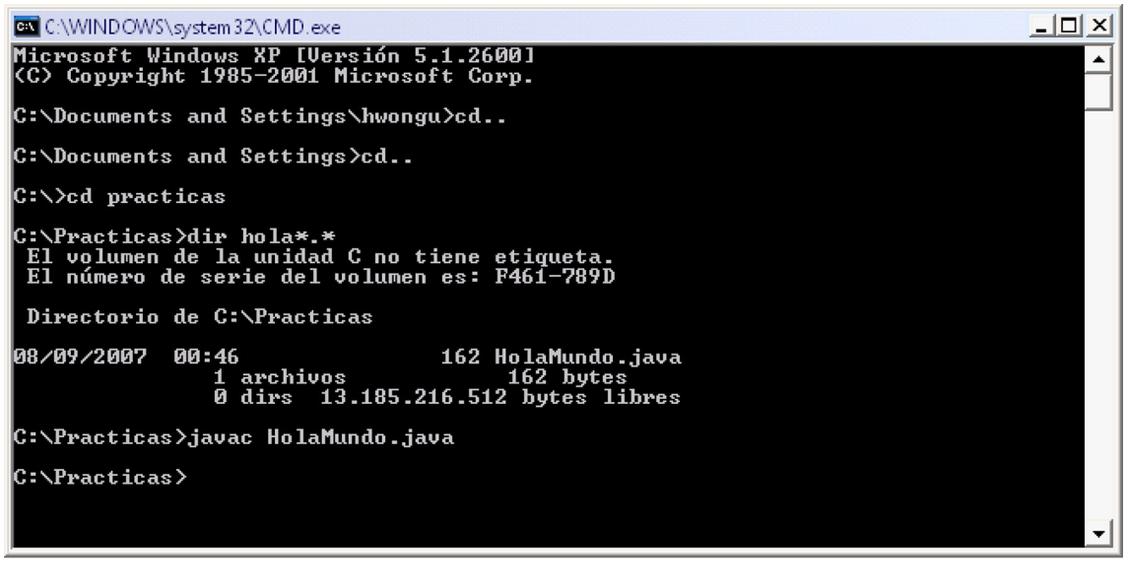
2. Guarde el archivo con el nombre "HolaMundo.java". No cierre el bloc de notas para verificar los errores.

3. Abra una ventana de DOS y navegue hasta la carpeta "Prácticas" y verifique si el archivo existe.



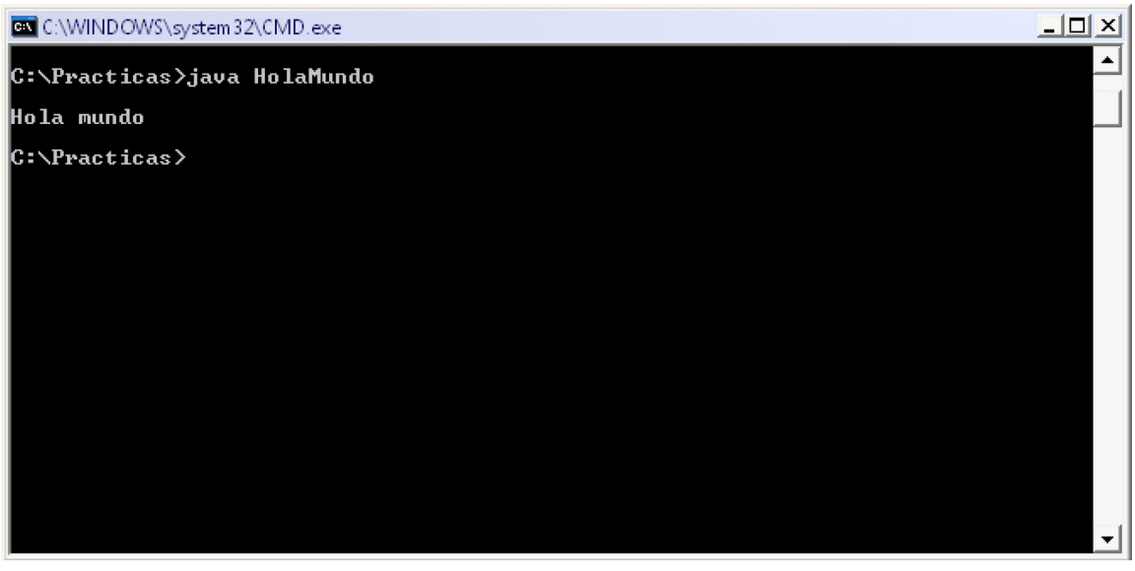
4. Compile el archivo "HolaMundo.java", con la sentencia:

```
javac HolaMundo.java
```



5. Si el código estuviera correcto, no debería salir ningún mensaje de advertencia, más bien se generará un archivo "HolaMundo.class" junto con nuestro archivo "HolaMundo.java". Para probar el código, escriba la siguiente sentencia:

```
java HolaMundo
```



Nota: Cuando se va usar la sentencia "java", después de esa sentencia se pone el nombre del archivo con extensión ".class" que se generó. Recuerde que debe tener en cuenta las mayúsculas y minúsculas del nombre del archivo.

Glosario de la Unidad I

1. **Delimitadores**

   Sirven para separar una línea código o un bloque de código.

2. **JAVA_HOME**

   Es la variable de entorno donde se especifica la ruta de instalación del JDK y JRE.

3. **JDK**

   Son las niciales de Java Devolper Kit que es un conjunto de librerías que nos permite desarrollar y ejecutar programas en Java.

4. **JRE**

   Son las iniciales de Java Runtime Edition que es un conjunto de librerías que solo nos permite ejecutar programas en Java.

5. **JVM**

   Son las iniciales de Java Virtual Machine que es un programa que se instala en el sistema operativo para poder ejecutar programas en Java.

6. **ORACLE**

   Es la empresa dueña del lenguaje de programación Java.

7. **PATH**

   Es la variable de entorno donde se especifica la ruta donde se encuentran los ejecutables de Java.

8. **Token**

   Representan nombres asignables a variables, métodos y clases para identificarlos de forma única.

Bibliografía de la Unidad I

Alvarellos, D., & Pastor, G. (2012). Guía Java para docentes. Uruguay: Consejo de Educación Técnico Profesional (CETP). Recuperado de files.itsp-informatica.webnode.com.uy/200000117.../GuiaJavaparaDocentes2012.pdf

Bell, D., & Parr, M. (2003). Java para estudiantes [en línea]. Recuperado de https://books.google.com.pe/

Deitel, P., & Deitel, H. (2003). Cómo Programar en C++ (4.ª ed.). Naucalpan de Juárez, México: Pearson Educación.

Oracle (13 de 07 de 2016). Java SE Donwloads [en línea]. Disponible en http://www.oracle.com/technetwork/es/java/javase/downloads/index.html

Oracle University (13 de 07 de 2016). About the Java Technology [en línea]. Disponible en http://docs.oracle.com/javase/tutorial/getStarted/intro/definition.htm l

Rodríguez, A. (s.f.). Aprender programación Java desde cero [en línea]. Recuperado de http://www.aprenderaprogramar.com/

Autoevaluación n.º I

1. **¿Qué comando sirve para ejecutar un programana en Java?**

   a) javac
   b) java
   c) javadoc
   d) JAVA_HOME

2. **¿Qué comando sirve para compilar un programa en Java?**

   a) javac
   b) java
   c) javadoc
   d) JAVA_HOME

3. **¿Qué variable de entorno es necesario para poder ejecutar un programa en Java?**

   a) javac
   b) java
   c) javadoc
   d) JAVA_HOME

4. **¿En qué sistemas operativos se puede ejecutar Java?**

   a) Windows
   b) Linux
   c) Unix
   d) Todos los anteriores

5. **¿Cuál no es una caracteristica de Java?**

   a) Simple
   b) Orientado a Objetos
   c) Monohilo
   d) Distribuido

6. **Realizan una acción específica en el programa.**

   a) Token
   b) Palabra reservada
   c) Identificador
   d) Operadores

7. **Palabras que se pueden usar para definir algo en el programa.**

   a) Token
   b) Palabra reservada
   c) Identificador
   d) Operadores

8. Palabra que tiene un significado y uso concreto en el programa.

a) Token
b) Palabra reservada
c) Identificador
d) Operadores

9. ¿Qué se necesita instalar para poder desarrollar y ejecutar un programa en Java?

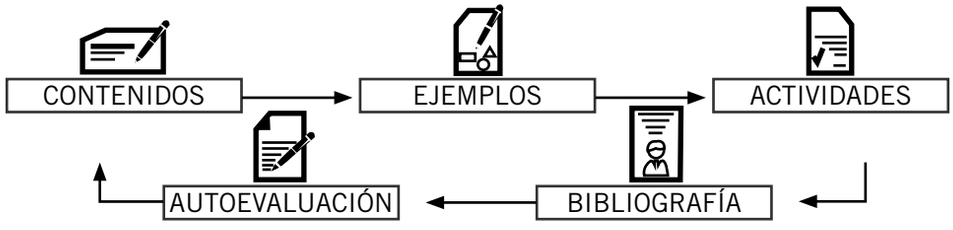
a) JDK
b) JRE
c) JAVA_HOME
d) PATH

10. ¿Qué se necesita instalar para poder desarrollar y ejecutar un programa en Java?

a) JDK
b) JRE
c) JAVA_HOME
d) PATH

UNIDAD II
PROGRAMACIÓN ORIENTADA A OBJETOS I

DIAGRAMA DE ORGANIZACIÓN DE LA UNIDAD II



ORGANIZACIÓN DE LOS APRENDIZAJES

Resultados del aprendizaje de la Unidad II: Al finalizar la unidad el estudiante diseña e implementa aplicaciones con interfaz gráfica utilizando la programación orientada a objetos.

CONOCIMIENTOS	HABILIDADES	ACTITUDES
Tema n.º 1: ¿Cómo usar objetos? 1. ¿Qué es la programación orientada a objetos? 2. ¿Qué es objeto? 3. ¿Qué es una clase? **Tema n.º 2: Estructuras de selección y repetitivas** 1. Estructuras de selección 2. Estructuras repetitivas **Lectura seleccionada n.º 3:** Estructuras repetitivas y selección **Lectura seleccionada n.º 4:** Clases y objetos **Lectura seleccionada n.º 5:** Clases abstractas e interfaces **Autoevaluación de la unidad II**	1. Utiliza las clases de la biblioteca de Java en la programación. 2. Utiliza las instrucciones *if* y *switch* para llevar a cabo evaluaciones. 3. Realiza repeticiones mediante *while* y *for*. **Actividad n.º 1** Uso de objetos y estructuras condicionales **Control de lectura n.º 2** Evaluación de la lectura seleccionada y de los temas n.º 1 y 2	Aprecia la importancia de la programación en la solución de problemas. Asume una actitud ética y de respeto a los demás.

¿Cómo usar objetos?

Tema n.° 1

En la Unidad n.° 1, usted conoció todo lo referente a cómo crear un programa en Java, en este punto aprenderá cómo utilizar la programación orientada a objetos para la creación de sus programas.

1. ¿Qué es la programación orientada a objetos?

La programación orientada a objetos surgió como una manera de mejorar la calidad interna de los programas, debido a que con la programación estructura resultaba muy difícil el mantenimiento de los sistemas. Por ejemplo, para el tema de la reutilización de componentes se tenía que crear varias rutinas en diferentes archivos de programación y el mantenimiento de los programas se hacía dificultoso.

Por tanto, a la programación orientada a objetos se le considera como una técnica, que permite la reutilización y extensibilidad de líneas código simulando un entorno real. Se puede definir reutilización y extensibilidad de la siguiente manera:

- Reutilización
  Es la capacidad de un producto *software* de ser utilizado en la construcción de diferentes aplicaciones, de modo que permite no reinventar soluciones para problemas ya resueltos. Entonces, se escribe menos *software* y se puede dedicar más tiempo a mejorar otros factores.

- Extensibilidad
  Es la facilidad de adaptación de los productos *software* a los cambios en la especificación que puede dar un usuario que utilizará el sistema. Los cambios son frecuentes, puesto que nuestro entorno es muy cambiante. Por ejemplo, si ahora el IGV cambiara de 18% a 15%, todos los sistemas se deberían de actualizar de manera rápida y sin afectar mucho los programas. Los principios esenciales para facilitar la extensibilidad son los siguientes:

  - Simplicidad de la arquitectura del *software*
  - La descentralización, pues crea módulos autónomos

1.1. Tecnología de objetos

La tecnología de objetos nos indica que todos los programas que creamos se tienen que basar en objetos; por ende, todos deben poder ser reutilizables, porque "no debemos de crear aplicaciones desde cero".

Además, la tecnología de objetos nos permite el diseño de programas basados en interfaces y arquitecturas estándares de programación como, por ejemplo, los patrones MVC, singleton, entre otros.

El éxito de la tecnología de objetos frente a la programación estructura se debe a lo siguiente (Pavón, s.f):

- Avances en arquitectura de computadores
- Avances en lenguajes de programación (C++, Smalltalk, Ada, Java, C# ...)
- Ingeniería del *software* (modularidad, encapsulado de la información, proceso de desarrollo incremental)
- Los límites de la capacidad de gestionar la complejidad

Asimismo, las ventajas de utilizar la tecnología de objetos son las siguientes (Pavón, s.f):

- Mejoras significativas de la productividad y calidad del código

- Estabilidad de los modelos respecto a entidades del mundo real
- Construcción iterativa
- Promueve la reutilización de *software* y de diseños (componentes, *frameworks*).
- Los sistemas OO son generalmente más pequeños que su equivalente no OO: menos código y más reutilización.
- Permite desarrollar sistemas más preparados para el cambio.
- Vale para aplicaciones de pequeño y gran tamaño.

2. ¿Qué es un objeto?

Los objetos son cosas del mundo real, por ejemplo, podemos decir que tenemos los objetos persona, animal, alumno, aula, entre otros. Fernando Berzal (s.f., 3) los define de la siguiente manera:

- Un objeto puede caracterizar una entidad física (un teléfono, un interruptor, un cliente) o una entidad abstracta (un número, una fecha, una ecuación matemática).
- Todos los objetos son instancias de una clase: Los objetos se crean por instanciación de las clases.
- Todos los objetos de una misma clase (p.ej. automóviles) comparten ciertas características: sus atributos (tamaño, peso, color, potencia del motor) y el comportamiento que exhiben (aceleran, frenan, curvan).

Como se puede apreciar, un objeto simula una entidad del mundo real tanto en sus atributos y acepciones que puede realizar.

3. ¿Qué es una clase?

Los objetos con estados similares y con el mismo comportamiento se pueden agrupar en clases. Las clases representan una plantilla (*template*) de varios objetos y describe cómo estos están estructurados internamente.

Las clases se encuentran conformados por dos elementos:

- Atributos: son las características de las clases como, por ejemplo, tamaño, peso, etc.
- Métodos: son los comportamientos que se pueden hacer con las clases como, por ejemplo, acelerar, frenar, etc.

3.1. Instanciar clases

Una instancia es un objeto creado a partir de una clase. La clase describe la estructura de la instancia (información y comportamiento), mientras que el estado actual de la instancia es definido por las operaciones ejecutadas. El estado interno de cada instancia es propia de cada una y se compone de los atributos establecidos en la clase (Joyanes, s.f.).

Para explicar lo anterior, lo graficaremos de la siguiente manera:

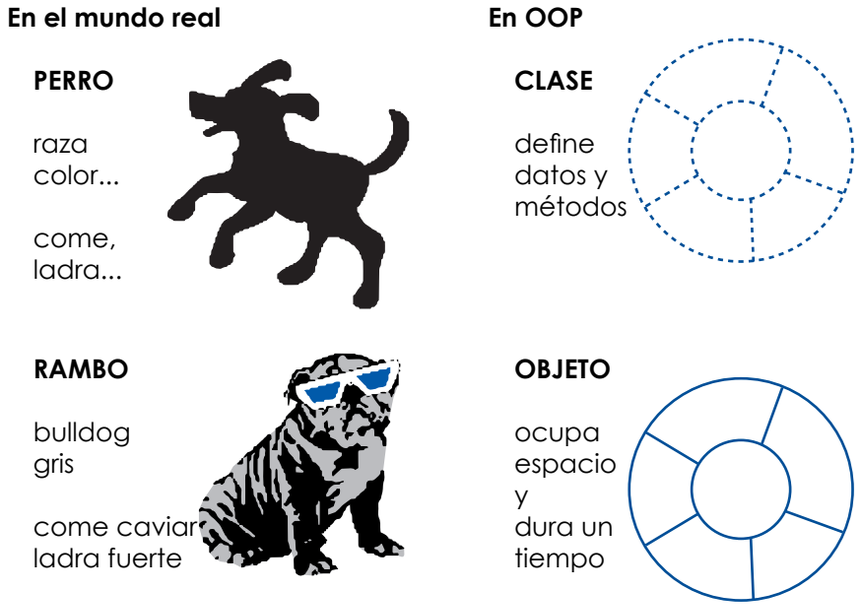


Figura 6. Diferencia entre objeto y clase
Fuente: Elaboración propia

Como se aprecia en el Figura 6, la clase por sí sola no ocupa un espacio y no tiene asignado los valores para los atributos o métodos. Ya cuando se instancia la clase se convierte en objeto, ocupa un especio y dura un tiempo. Además, los atributos de la clase ya tienen un valor asignado y puede realizar las acciones definidas.

3.2. Clases, UML y Java

UML (Unified Modeling Language) es el lenguaje que usamos para proporcionar una forma visual y utilizable para el desarrollo de aplicaciones basadas en clases y sirve para fomentar el uso de la programación orientada a objetos en lugar de la programación estructurada.

Por ejemplo, la clase perro en UML y su respectivo código en Java se representarían de la siguiente manera:

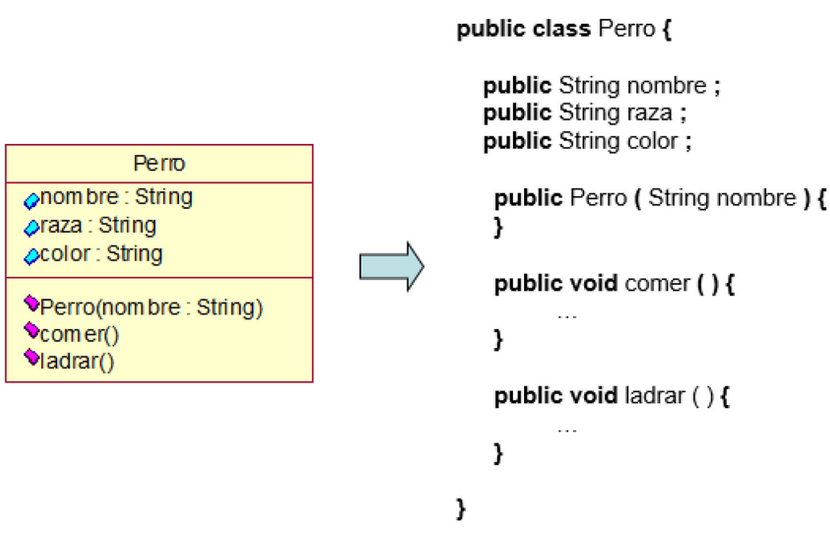


Figura 7. UML y Java. Fuente: Elaboración propia

Donde la clase perro está conformado por lo siguiente:

- Atributos de la clase perro:
  - Nombre
  - Raza
  - Color
- Métodos de la clase perro:
  - Comer
  - Ladrar

3.3. Estructura de una clase en Java

A continuación, se presentará la estructura de cómo usted debe declarar una clase en Java:

```
package nombrePaquete ; //define el nombre del paquete contenedor de la clase
[lista de importaciones]
[ public ]  class NombreClase //Define la clase
{ //Inicio del cuerpo de la Clase
      //Define una constante
      public static final constante = valorInicial ;
      //Define un atributo de instancia
      alcance tipoDato atributoInstancia [ = valorInicial ] ;
      //Define un atributo de clase
      alcance static tipoDato atributoClase [ = valorInicial ] ;
      //Define un constructor
      public NombreClase ( [ listaArgumentos ] ) {
            listaSentencias ;
      }
      //Define un método de instancia
      alcance void|tipoRetorno metodoInstancia( [ listaArgumentos ] ) {
            listaDeclaraciones ;
      listaSentencias ;
            [ return expresión ; ] //si tipo de retorno diferente de void
      }
      //Define un método de clase
      alcance static void|tipoRetorno metodoClase( [ listaArgumentos ] ) {
            listaDeclaraciones ;
      listaSentencias ;
            [ return expresión ; ] //si tipo de retorno diferente de void
      }
} //Fin de la clase
```

Figura 8. Estructura de una clase Java. Fuente: Elaboración propia

Una clase debe estar definida dentro de un archivo de texto con extensión *.java de la siguiente forma: *NombreClase.java,* y como primera sentencia debe especificar el paquete al cual pertenece.

A continuación, pasaremos a analizar cada una de las partes de una clase en Java:

3.3.1. Atributos

Los atributos representan los estados o características de un objeto o conjunto de objetos, los cuales se pueden dividir de la siguiente manera:

- Atributos de clases:

  Son aquellos que son comunes para todos los objetos de una clase como, por ejemplo, la cantidad de alumnos matriculados en un aula.

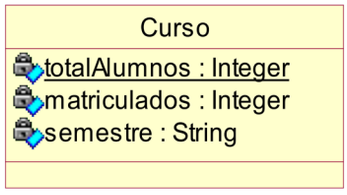


Figura 9. Atributos de clases. Fuente: Elaboración propia

Código Java:

```
static int totalAlumnos = 0;
```

Atributos de instancia

Son aquellos que hacen que un objeto sea único y se diferencie de otros como, por ejemplo, el nombre de un alumno, la fecha de nacimiento, entre otros.

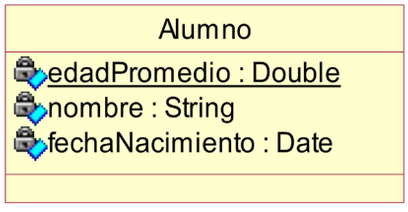


Figura 10. Atributos de instancia. Fuente: Elaboración propia

Código Java:

```
String nombre;
Date fechaNacimiento;
```

3.3.2. Métodos

Los métodos representan el comportamiento de un objeto o conjunto de objetos, que pueden ser llamados dentro de la misma clase u otras clases. Siempre es obligatorio indicar el tipo de retorno o si no retorna ningún valor, indicar que es *void*.

La forma para declarar un método es la siguiente (Oracle University, 2016):

```
[acceso][static]tipoRetorno nombreMetodo ([parametros]){
   //cuerpo del metodo
   [return valorRetorno;]
}
```

Por ejemplo, si deseamos indicar que tenemos una clase calculadora y queremos realizar la suma de dos números, se implementaría como se aprecia a continuación:

- Diagrama de clases:

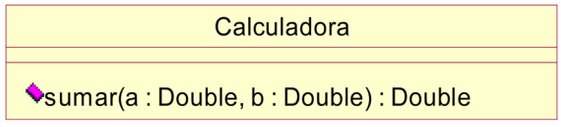


Figura 11. Clase calculadora. Fuente: Elaboración propia

- Código Java:

```
static double sumar ( double a , double b ) {
          double suma = 0.0 ;
          suma = a + b ;
          return suma ;
}
```

Ahora bien, la forma para invocar al método desde otra clase sería de la siguiente manera:

- Código Java:

```
Calculadora.sumar( 12.2 , 13.6 ) ;
```

3.3.3. Constructores

Los constructores son procedimientos con el mismo nombre de la clase usados para inicializar ciertos valores de los atributos de la clase y que se invocan automáticamente cada vez que instanciamos una clase. Los constructores no pueden especificar valores de retorno y si no se define uno, el compilador crea uno por defecto sin argumentos.

3.4. Creación de un objeto

Antes de poder utilizar una clase se debe crear o instanciar, lo cual se logra utilizando el operador *new*. Cuando utilizamos el operador *new*, lo que estamos haciendo es crear el objeto para que ocupe un espacio en la JVM.

- Código Java:

```
Tipo identificador = new Tipo( [argumentos] );
```

3.5. Control de acceso

Los controles de acceso en Java son los siguientes (Joyanes, Sánchez, & Zahonero, 2007, pp. 49-51):

- Public
  Los miembros que se declaran como *public* son accesibles en cualquier parte donde la clase sea accesible.
- Protected
  Los miembros declarados como *protected* solo son accesibles en la propia clase y sus subclases.
- Private
  Los miembros declarados como *private* solo son accesibles en la propia clase.

Oracle recomienda que todos los atributos de una clase sean del *private* y que se utilicen métodos de acceso para cambiar sus estados.

Estructuras de selección y repetitivas

Tema n.° 2

En el tema anterior, se explicó todo lo referente sobre cómo instanciar una clase y las diferentes partes que la conforman; en la sección actual, se revisarán los temas correspondientes a las estructuras de selección y las estructuras repetitivas.

1. Estructuras de selección

1.1. Sentencia If-Else

La sentencia If–Else es el condicional más básico de los de control de flujo. Lo que hace el programa cuando encuentra la sentencia *If* es evaluar la condición y si se cumple que es verdadera, se ejecutan los pasos siguientes. En caso contrario, se ejecuta lo que se encuentra en el bloque *Else*.

La sintaxis en Java sería la siguiente:

```
if( expresión-booleana )
{
   sentencias;
}
 [else {
   sentencias;
}]
```

La representación en diagrama de flujo corresponde a la que se muestra a continuación:

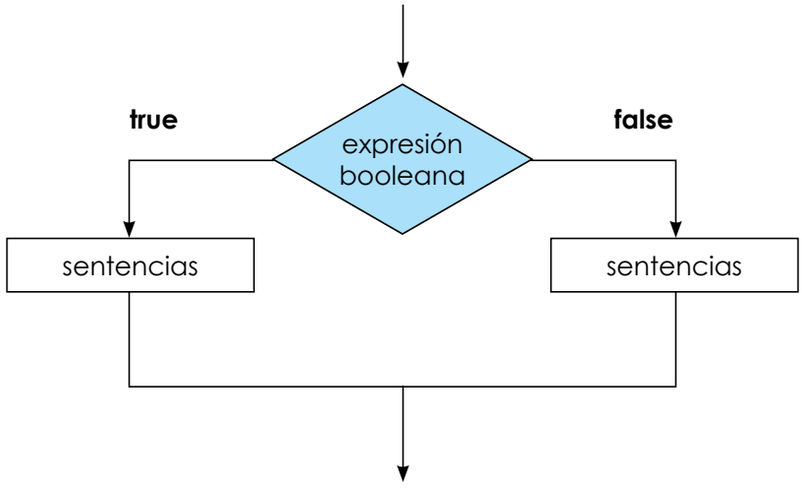


Figura 12. Diagrama de flujo If–Else. Fuente: Elaboración propia.

Y, finalmente, la representación en diagrama N/S se expresaría de la siguiente manera:

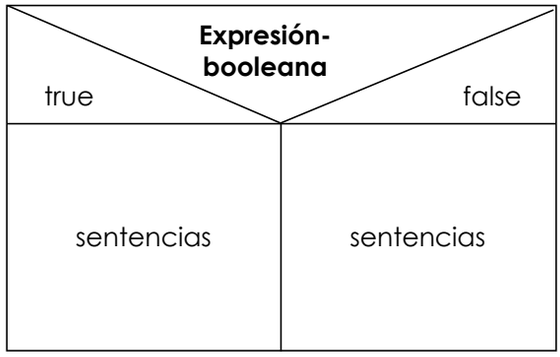


Figura 13. Diagrama N/S If–Else. Fuente: Elaboración propia.

1.2. Sentencia Switch

A diferencia de la sentencia *If–Elfse*, esta sirve para evaluar un número posibles de casos o caminos de ejecución. Los valores de la condición se establecen en la sentencia *Case* y cada sentencia tiene que terminar con la sentencia *Break* para que no siga evaluando condiciones.

La sintaxis en Java sería la siguiente:

```
switch(expresión) {
   case valor1:
         sentencias;
         break;
   case valor2:
         sentencias;
         break;
   [default:
         sentencias;]
}
```

La representación en diagrama de flujo se aprecia a continuación:

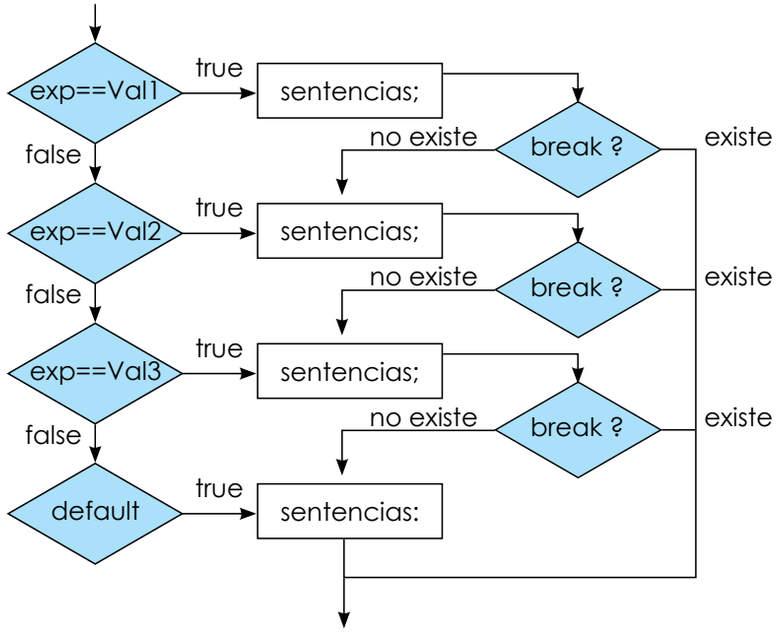


Figura 14. Diagrama de flujo *Case*. Fuente: Elaboración propia

Y la representación en diagrama N/S se representa tal como se observa en la siguiente tabla:

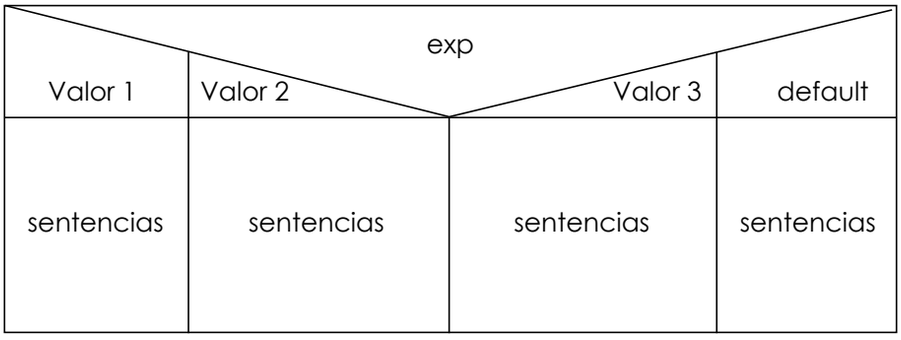


Figura 15. Diagrama N/S *Case*. Fuente: Elaboración propia.

2. Estructuras repetitivas

2.1. Sentencia While

La sentencia While ejecuta continuamente un bloque de instrucciones, mientras que una condición particular es verdadera.

La sintaxis en Java sería la siguiente:

```
while( expresión-booleana ) {
   sentencias;
}
```

La representación en diagrama de flujo sería como se muestra en la siguiente imagen:

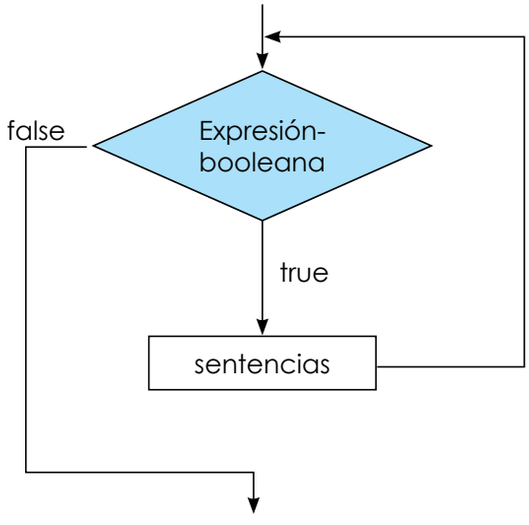


Figura 16: Diagrama de flujo *While*. Fuente: Elaboración propia

Y la representación en diagrama N/S tal como aparece a continuación:

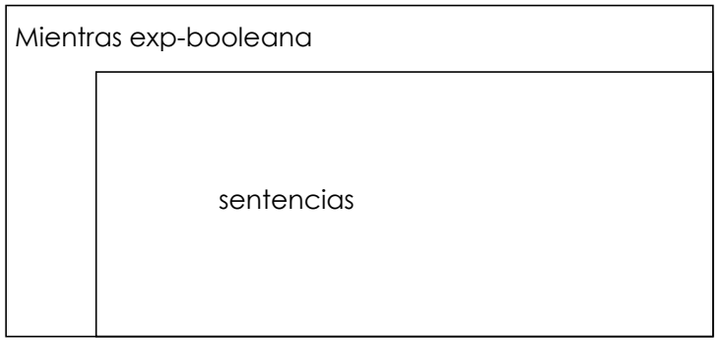


Figura 17. Diagrama N/S While. Fuente: Elaboración propia

2.2. Sentencia Do-While

La diferencia entre el While y Do–While es que el último ejecuta el bucle y, luego, evalúa la condición.

La sintaxis en Java sería la siguiente:

```
do {
   sentencias;
}while(expresión-booleana);
```

La representación en diagrama de flujo sería la siguiente:

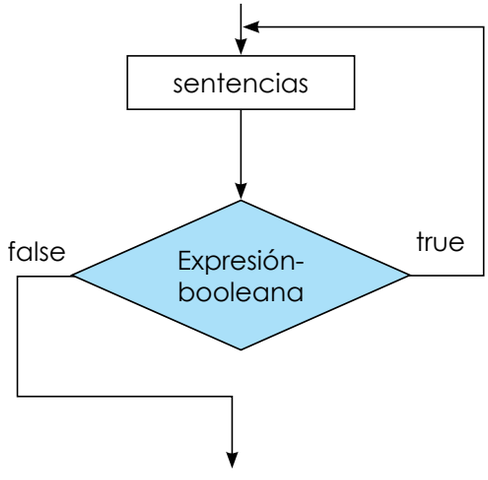


Figura 18. Diagrama de flujo Do–While
Fuente: Elaboración propia

Y la representación en diagrama N/S sería la siguiente:

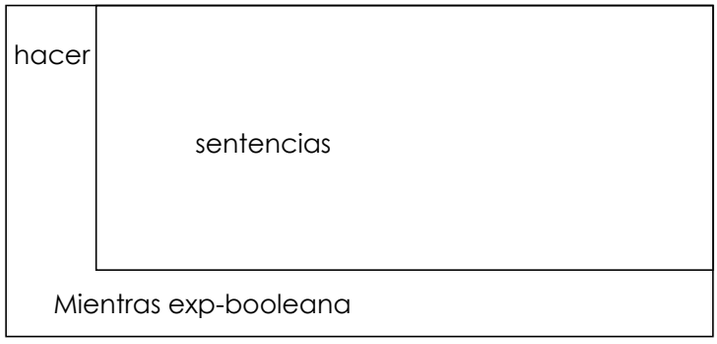


Figura 19. Diagrama N/S Do–While. Fuente: Elaboración propia

2.3. Sentencia For

La sentencia For proporciona una forma compacta para iterar sobre un rango de valores. Los programadores a menudo se refieren a él como el *for loop,* debido a la forma en la que se realiza un bucle varias veces hasta que una condición particular esté satisfecha.

La sintaxis en Java sería la siguiente:

```
for( inicialización; exp-booleana; iteración ) {
   sentencias;
}
```

La representación en diagrama de flujo se presenta a continuación:

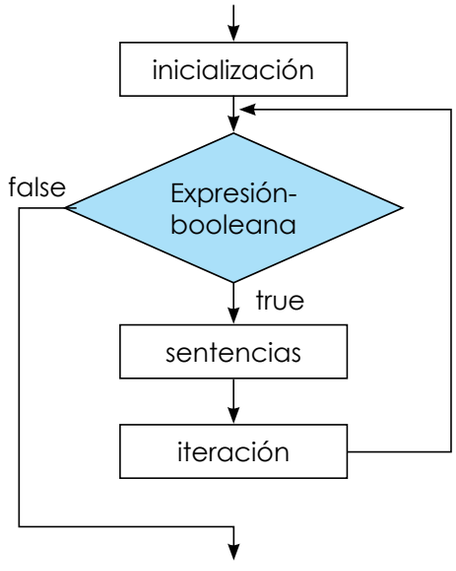


Figura 20: Diagrama de flujo *For*. Fuente: Elaboración propia

Y la representación en diagrama N/S sería la siguiente:

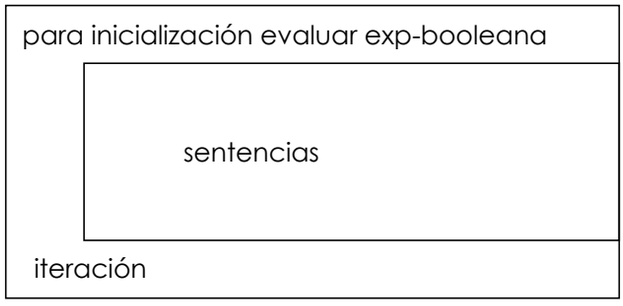


Figura 21. Diagrama N/S For. Fuente: Elaboración propia

Lectura seleccionada n.° 3

Berzal, F. (2011). Estructuras de control repetitivas / iterativas. En *Introducción a la programación orientada a objetos* (pp. 19–32). Disponible en http://bit.ly/2bE39BS

Lectura seleccionada n.º 4

Berzal, F. (2011). Clases y objetos. En *Introducción a la programación orientada a objetos* (pp. 3–22). Disponible en http://bit.ly/2b4aUDg

Actividad n.º 3

Se le pide que realice lo siguiente:

- Cree una clase "operación", que tenga un atributo "operación Matemática" que almacenará qué operación se realizará (+,-,/,*) y un método "calcular", que pide que se ingrese dos números decimales y que devuelva el resultado de la operación.
- Luego, cree una aplicación consola, que pida que se ingrese la operación a realizar y dos números, y que muestre el resultado de la operación.

Todo lo anterior mencionado lo tiene que realizar en el IDE de Desarrollo Netbeans, que lo puede encontrar en la siguiente ruta:

https://netbeans.org/downloads/

de donde deberá descargar la versión completa del programa, tal cual como indica la Figura.

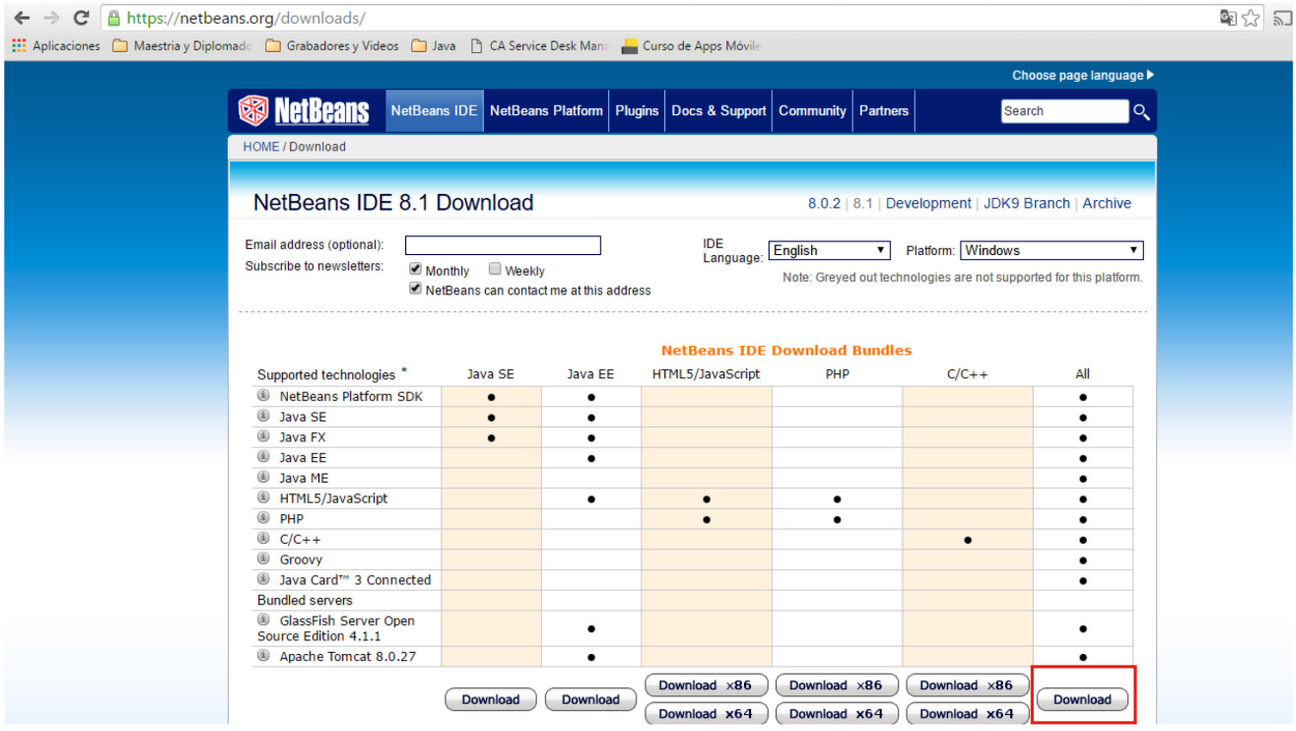


Figura 22: Descargar Netbeans. Fuente: https://netbeans.org/

Si necesita ayuda de cómo desarrollar la Actividad n.º 1, puede visualizar el siguiente video:

https://youtu.be/xk1avFa1AZA

Lectura seleccionada n.º 5

Berzal, F. (2011). Clases abstractas e interfaces. Disponible en http://bit.ly/1UGeTo0

Actividad n.º 4

Foro de discusión sobre clases y objetos

Instrucciones:

- Debe de ingresar al foro y participar con comentarios críticos y analíticos sobre el tema.
- Lea la lectura selecciona n.º 2.
- Responda en el foro las siguientes preguntas:
  - ¿Cuándo se utiliza el método finalize()?
  - ¿Para qué sirve la sentencia *this*?
  - ¿Cuándo se utilizan las variables *static*?
  - ¿Para qué sirven los métodos *static*?
  - Defina qué es la clase *object*.

Glosario de la Unidad II

1. **Atributos**
   Son las características de las clases.

2. **Clases**
   Los objetos con estados similares y con el mismo comportamiento se pueden agrupar en clases.

3. **Extensibilidad**
   Es la facilidad de adaptación de los productos *software* a los cambios en la especificación que puede dar un usuario que va a usar el sistema.

4. **Reutilización**
   Es la capacidad de un producto *software* de ser utilizado en la construcción de diferentes aplicaciones, permitiendo no reinventar soluciones para problemas ya resueltos.

5. **Métodos**
   Son los comportamientos que se puede hacer con las clases.

6. **MVC**
   Son las iniciales que hacen referencia al patrón de desarrollo modelo, vista y controlador.

7. **Objeto**
   Los objetos son cosas del mundo real. Por ejemplo, podemos decir que tenemos los objetos persona, animal, alumno, aula, entre otros.

8. **OO**
   Son las iniciales que hacen referencia a la programación orientada a objetos.

9. **UML**
   Son las iniciales de Unified Modeling Language.

Bibliografía de la Unidad II

Berzal, F. (s.f.). Clases y objetos [en línea]. Recuperado de http://elvex.ugr.es/dec sai/java/pdf/3B-Clases.pdf

Joyanes Aguilar, L., Sánchez García, L., & Zahonero Martínez, I. (2007). *Estructura de datos en C++*. (C. Sánchez González, Ed.). Madrid, España: McGRAW-HILL/Interamericana de España.

Oracle (13 de 07 de 2016). Java SE Donwloads [en línea]. Recuperado de http://www.oracle.com/technetwork/es/java/javase/downloads/index.html

Oracle (13 de 07 de 2016). Java SE Donwloads [en línea]. Recuperado de http:// www.oracle.com/technetwork/es/java/javase/downloads/index.html

Oracle University (13 de 07 de 2016). About the Java Technology [en línea]. Disponible en http://docs.oracle.com/javase/tutorial/getStarted/intro/definition.ht ml

Pavón, J. (s.f.). Conceptos de POO [en línea]. Recuperado de http://www.fdi.ucm. es/profesor/jpavon/poo/01ConceptosOO.pdf

Rodríguez, A. (s.f.). Aprender programación Java desde cero [en línea]. Recuperado de http://www.aprenderaprogramar.com/

Autoevaluación n.º 2

1. ¿Para qué sirven los *packages*?

a) Por estándar de Java se debe de crear un package antes de una clase.
b) Para ordenar clases en común.
c) No existen los *packages* en Java.
d) Se debe de evitar su uso.

2. Cuando se instancia una clase se está creando un(a) …

a) Variable
b) Clase
c) Objeto
d) Constructor

3. ¿Qué se utiliza para inicializar una clase?

a) Atributo
b) Parámetro
c) Constructor
d) Método

4. ¿Cuál es la caracteristica de una clase?

a) Atributo
b) Parámetro
c) Constructor
d) Método

5. ¿Cuál es el comportamiento de una clase?

a) Atributo
b) Parámetro
c) Constructor
d) Método

6. ¿Para qué sirve el operador *instanceof*?

a) Para saber de qué tipo es una clase.
b) Para crear una instancia de una clase.
c) Para inicializar una clase.
d) Para eliminar una clase.

7. ¿Cuál no es la afirmación correcta sobre la clase Object?

a) Es la superclase de todas las clases en Java.
b) Es un atributo de las clases en Java.
c) Toda clase en Java lo hereda.
d) Toda clase se puede convertir automáticamente en ella.

8. ¿Cuál no es una afirmación correcta del uso de la palabra reservada *this*?

a) Se utiliza para hacer referencia a un atributo propio de una clase.
b) Para diferenciar variables con el mismo nombre.
c) Para llamar a variables heredadas.
d) Su uso es opcional si no existe variables con el mismo nombre.

9. ¿Cuándo se ejecuta el metodo *finalize()*?

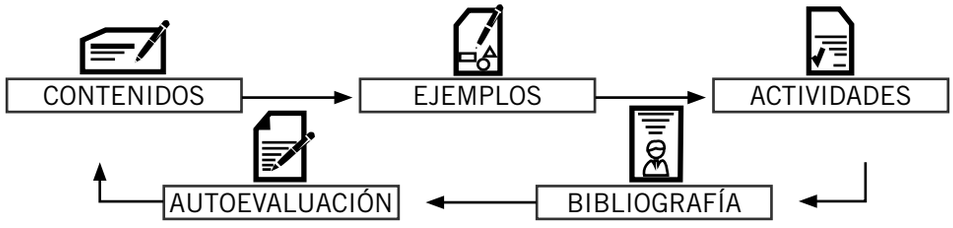
a) Al inicio de una clase.
b) Después de instanciar una clase.
c) Antes de liberar información de la memoria virtual de un objeto.
d) Cuando se elimina un objeto.

10. ¿Con qué palabra reservada se puede instanciar una clase?

a) new
b) instanceof
c) create
d) var

UNIDAD III
PROGRAMACIÓN ORIENTADA A OBJETOS II

DIAGRAMA DE ORGANIZACIÓN DE LA UNIDAD III



ORGANIZACIÓN DE LOS APRENDIZAJES

Resultados del aprendizaje de la Unidad III: Al finalizar la unidad el estudiante diseña e implementa aplicaciones con interfaz gráfica utilizando la programación orientada a objetos y el uso de colecciones.

CONOCIMIENTOS	HABILIDADES	ACTITUDES
Tema n.º 1: ¿Cómo usar objetos? 1. Colecciones 1.1 Estructura de datos 1.2 Arreglo (Array) 1.2.1 Declaración de arreglos 1.2.2 Matrices 1.2.3 Consideraciones para trabajar con arreglos 1.3 ArrayList **Tema n.º 2: Herencia de objetos** 1.1 Herencia 1.2 Polimorfismo 1.3 Abstract 1.4 Interface **Tema n.º 2: Manipulación de cadenas de texto** 1. Entrada y salida de datos 2. Clase String **Lectura seleccionada n.º 6:** Herencia de clases **Lectura seleccionada n.º 7:** UML **Autoevaluación de la unidad III**	1. Crea una nueva clase a partir de una existente mediante la herencia. 2. Utiliza el objeto ArrayList en la implementación de programas. 3. Implementa aplicaciones utilizando arreglos y matrices. 4. Utiliza herramientas para trabajar con cadenas de texto. **Actividad n.º 5** Uso de herencia simple **Actividad n.º 6** Uso de interface **Control de lectura n.º 3** Evaluación de la lectura seleccionada y los temas n.º 1, 2 y 3	Aprecia la importancia de la programación en la solución de problemas. Asume una actitud ética y de respeto a los demás.

¿Cómo usar objetos?

Tema n.° 1

En la siguiente unidad, usted conocerá todo lo referente al uso de arreglos y colecciones, así como el uso de la herencia para el desarrollo de programas.

1. Colecciones

Para el desarrollo de programas orientados a objetos, muchas veces es necesario el uso de un mecanismo para almacenar varios datos del mismo tipo. En otras palabras, es un mecanismo de almacenaje de información de manera temporal. Por ejemplo, es posible almacenar la información de las personas que están haciendo cola para que sean atendidos, los datos se deberían de almacenar como un conjunto de personas.

1.1. Estructura de datos

Una estructura de datos es un conjunto de datos del mismo tipo almacenados en un variable. Esta estructura viene a ser un *array* o arreglo que está conformado por una secuencia de ítems del mismo tipo y que son accesibles por una posición o índice dentro del *array*.

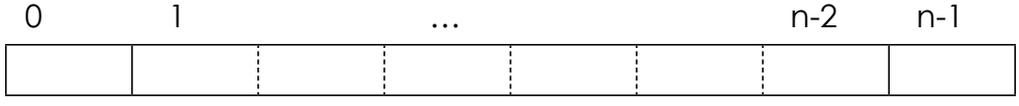


Figura 23. Estructura de datos. Fuente: Elaboración propia

1.2. Arreglo (array)

Un arreglo siempre está definido por su longitud (n) y su tipo, es decir, tiene un tope definido. Por ejemplo, la cantidad de alumnos que pueden entrar en un aula de clases tiene un tope definido (n alumnos) y eso se representaría usando un arreglo de alumnos.

La gestión de información con el uso de arreglos es eficiente. De hecho, la memoria del ordenador se gestiona como un arreglo en los lenguajes de bajo nivel (http://www.tecn.upf.es/~vlopez/docencia/castellano/todoshtml/PIIt2cas/node3.html).

1.2.1. Declaración de arreglos

Al declarar e inicializar un arreglo, el valor por defecto de todos sus componentes es *null* para el caso de objetos (*string*, Alumno, etc...), *0* para números enteros (byte, short, int, long), *0.0* para números reales (*float* y *double*), *'\0'* para caracteres (char) y *false* para booleanos (boolean).

Para iniciar un arreglo se debe declarar indicando la cantidad de elementos que tendrá el arreglo. A continuación, se ejemplifica así:

```
int miArreglo[ ] = new int[10] ;
```

Entonces, la capacidad del arreglo anterior es de 10 elementos, donde el primer elemento tiene índice 0 y el último índice 9, donde podríamos asignar el valor 5 a la primera posición tal como se aprecia en la siguiente expresión:

```
miArreglo[ 0 ] = 5 ;
```

Y si deseamos recuperar el valor de la segunda posición y almacenarla en otra variable, se haría de la siguiente manera:

```
int miVariable = miArreglo[ 1 ] ;
```

Cada arreglo tiene una variable *length* que indica la cantidad de elementos que puede almacenar el arreglo y se usaría tal como se puede observar.

```
int capacidad = miArreglo.length ;
```

1.2.2. Matrices

Las matrices o arreglos bidimensionales son un tipo de estructura que sirve para que un arreglo pueda almacenar otro, de modo que se puede almacenar la información como estructura de una tabla. De forma gráfica, el arreglo se conceptualizaría de la siguiente manera:

Si deseamos declarar un arreglo de 3 filas con 3 columnas, se haría de la siguiente manera:

```
int miArreglo[ ][ ] = new int[3] [3]  ;
```

Y su representación del arreglo sería tal como se muestra a continuación:

	Columnas		
Filas	(0,0)	(0,1)	(0,2)
	(1,0)	(1,1)	(1,2)
	(2,0)	(2,1)	(2,2)

Figura 24. Matrices. Fuente: Elaboración propia

Si se desea almacenar un valor, se tendría que indicar en qué fila y en qué columna se desea almacenar dicho dato.

Si deseamos recuperar el valor de la segunda fila y la primera columna y almacenarla en otra variable, se haría de la siguiente manera:

```
int miVariable = miArreglo[1][0] ;
```

1.2.3. Consideraciones para trabajar con arreglos

Para trabajar con arreglos se debe tener en consideración lo siguiente:

- El tamaño de un arreglo no puede cambiar, ya que tiene un tamaño fijo al inicializarlo y no se le puede cambiar ni para aumentar su capacidad ni para disminuirlo.
- Es una buena práctica recorrer el arreglo haciendo uso de la variable *length*.
- Los elementos que almacena un arreglo deben ser del mismo tipo.
- Si tratamos de acceder a una posición del arreglo que no existe, el programa en Java nos mandará una excepción del tipo *ArrayIndexOutOfBoundsException*.

1.3. ArrayList

Un *arrayLis*t es un tipo de colección que permite almacenar información del mismo tipo de datos, pero sin tener un tope fijo. Al igual que el arreglo, se trabaja con la posición.

La forma de declarar un *arrayList* sería de la siguiente manera:

```
ArrayList miarray = new ArrayList() ;
```

Para almacenar elementos de un *arrayList* se emplea la siguiente forma:

```
for( int i=0 ; i < 10 ; i++ )
     miarray.add( i );
```

Los métodos que tiene un ArrayList son los siguientes (Oracle, s.f.)

- *get* : Permite obtener un elemento dado un índice.
- *set* : Permite cambiar el valor del elemento dado un índice.
- *add* : Inserta un nuevo elemento en el índice específico.
- *remove* : Permite quitar un elemento de la colección dado un índice.
- *indexOf* : Retorna el índice de la primera aparición del elemento buscado si existe, sino retorna a -1.
- *lastIndexOf* : Retorna el índice de la última aparición del elemento buscado si existe, sino retorna a -1.

Herencia de objetos

Tema n.° 2

En este apartado, se abordará el tema de la herencia entre clases, la cual no permite la reutilización de código para el desarrollo de nuestros programas en Java.

1. Herencia

La herencia es una metodología que permite la definición de una clase a partir de la definición de otra ya existente. Además, facilita el relacionar un conjunto de clase bajo las mismas características, de manera que puedan ser consideradas y manejadas colectivamente. La herencia se emplea para la reutilización de código y es una característica importante de la programación orientada a objetos.

Por ejemplo, en el mundo real existen diversos tipos de aviones que podrían ser un avión comercial y otra sería un avión militar. Su representación UML se muestra a continuación:

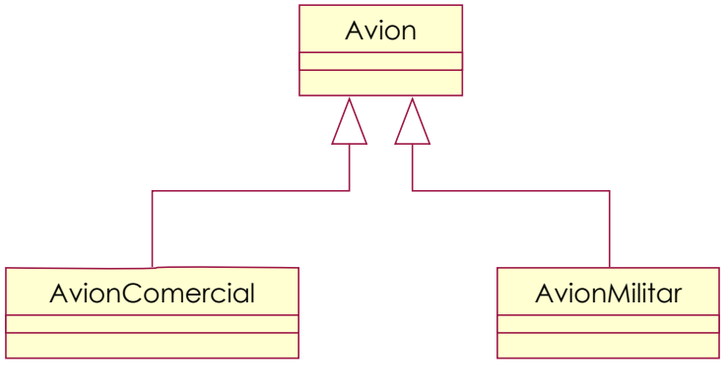


Figura 25. Herencia UML. Fuente: Tomada de Herencia y polimorfismo, por Quezada, 2010.

Y el código en Java se escribiría de la siguiente manera:

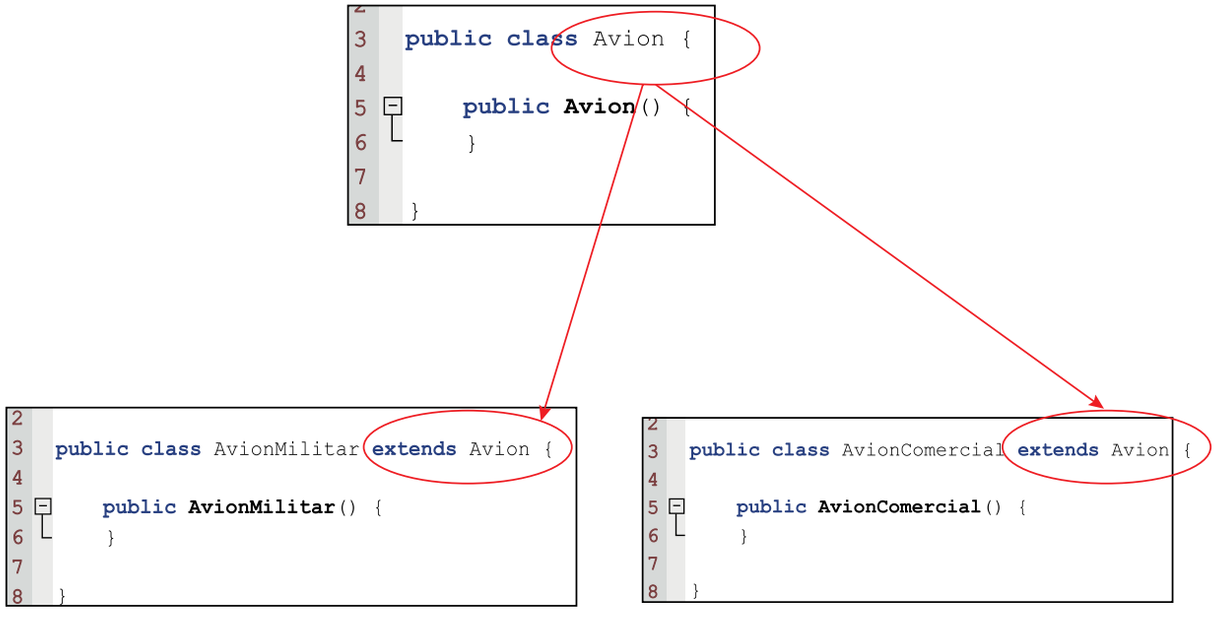


Figura 26. Herencia Java. Fuente: Tomada de Herencia y polimorfismo, por Quezada, 2010.

2. Polimorfismo

El polimorfismo es la habilidad de tener varias formas; por ejemplo, la clase avión comercial tiene acceso a los métodos de la clase avión. Y se podría declarar el código de la siguiente manera (Quezada, 2010):

```
Avion miavion = new AvionMilitar( ) ;
```

Solo se puede acceder a las partes del objeto que pertenecen a la clase avión; las partes específicas de la clase avión militar no se ven. Este efecto se consigue porque, para el compilador, mi avión es solo una variable de tipo avión, no avión militar.

3. Clases abstractas

Una clase padre puede estar compuesta de uno o más métodos abstractos, los que son declarados, pero no implementados: No tienen cuerpo, pues las implementaciones se encuentran en (son responsabilidad de) las subclases. Puede tener también métodos no abstractos, cuyas implementaciones serían comunes para sus subclases (Quezada, 2010).

Bastará que una clase contenga un método abstracto para que tenga que ser declarada también abstracta. Generalmente, esto se requiere por conveniencia en el diseño de aplicaciones.

Por ejemplo, para todas las figuras geométricas se puede calcular su área, pero el cálculo de esta va a depender del tipo de figura; entonces, se representaría en un UML de la siguiente manera:

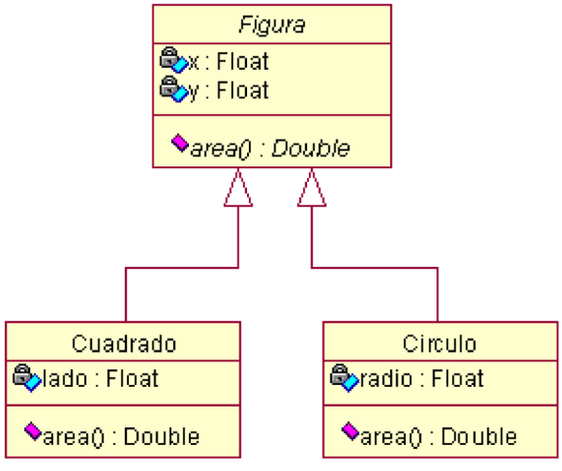


Figura 27. Abstract UML. Fuente: Tomada de Herencia y polimorfismo, por Quezada, 2010.

La implementación del diagrama anterior en Java quedaría como se observa a continuación:

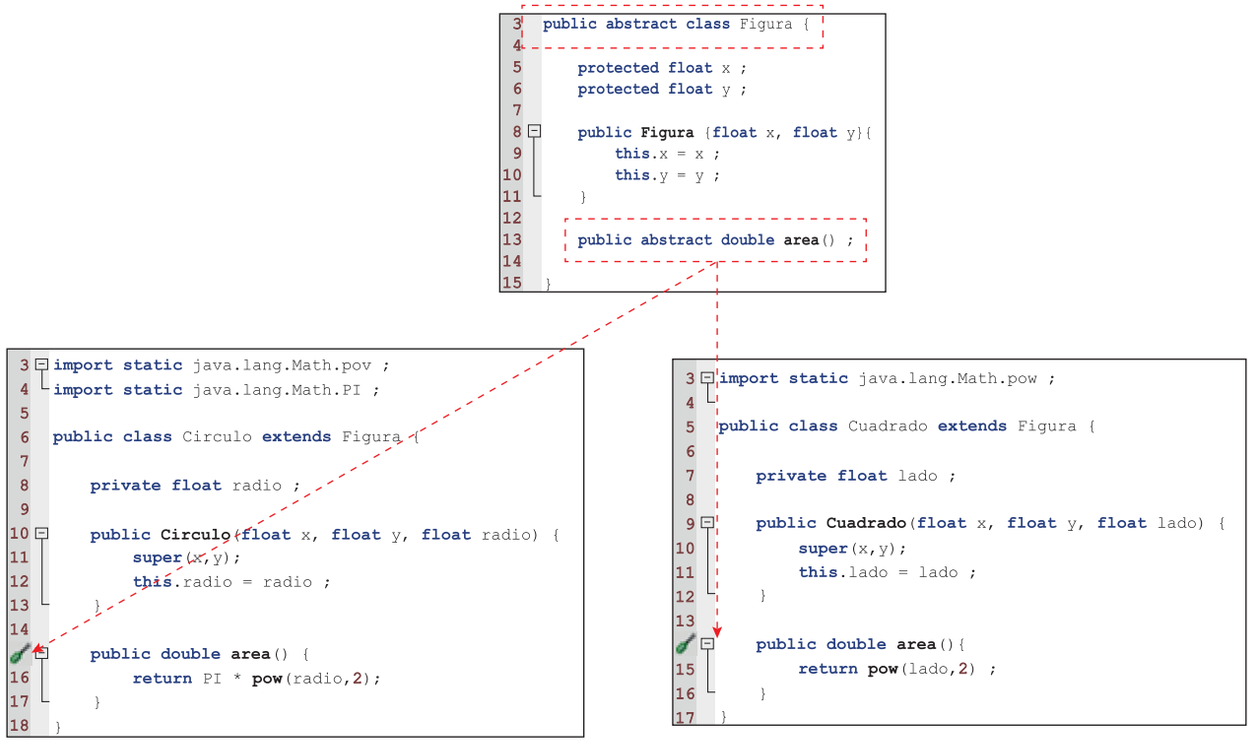


Figura 28. Abstract Java. Fuente: Tomada de Herencia y polimorfismo, por Quezada, 2010.

4. Interface

Interface es una clase totalmente abstracta, es decir, todos sus métodos son abstractos (*public abstract*) y todas sus variables son constantes (*public static final*). Una interface puede extender múltiples interfaces. Por lo tanto, se tiene herencia múltiple de interfaces.

Una clase puede implementar más de una interface y, de esta manera, provee un mecanismo similar a la herencia múltiple.

Por ejemplo, si se deseara implementar el ejemplo de figura usando Interface, la representación en UML sería la siguiente:

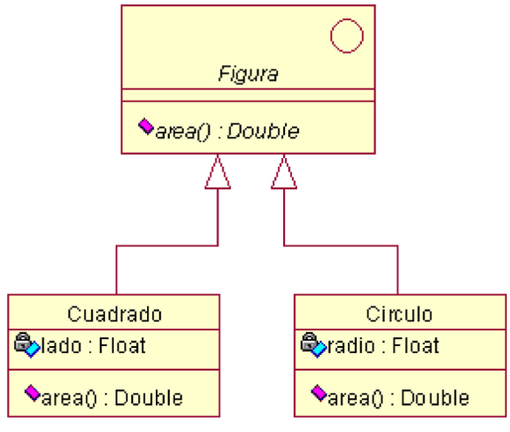


Figura 29. Interface UML. Tomada de Herencia y polimorfismo, por Quezada, 2010.

La implementación del diagrama anterior en Java se haría de la siguiente manera:

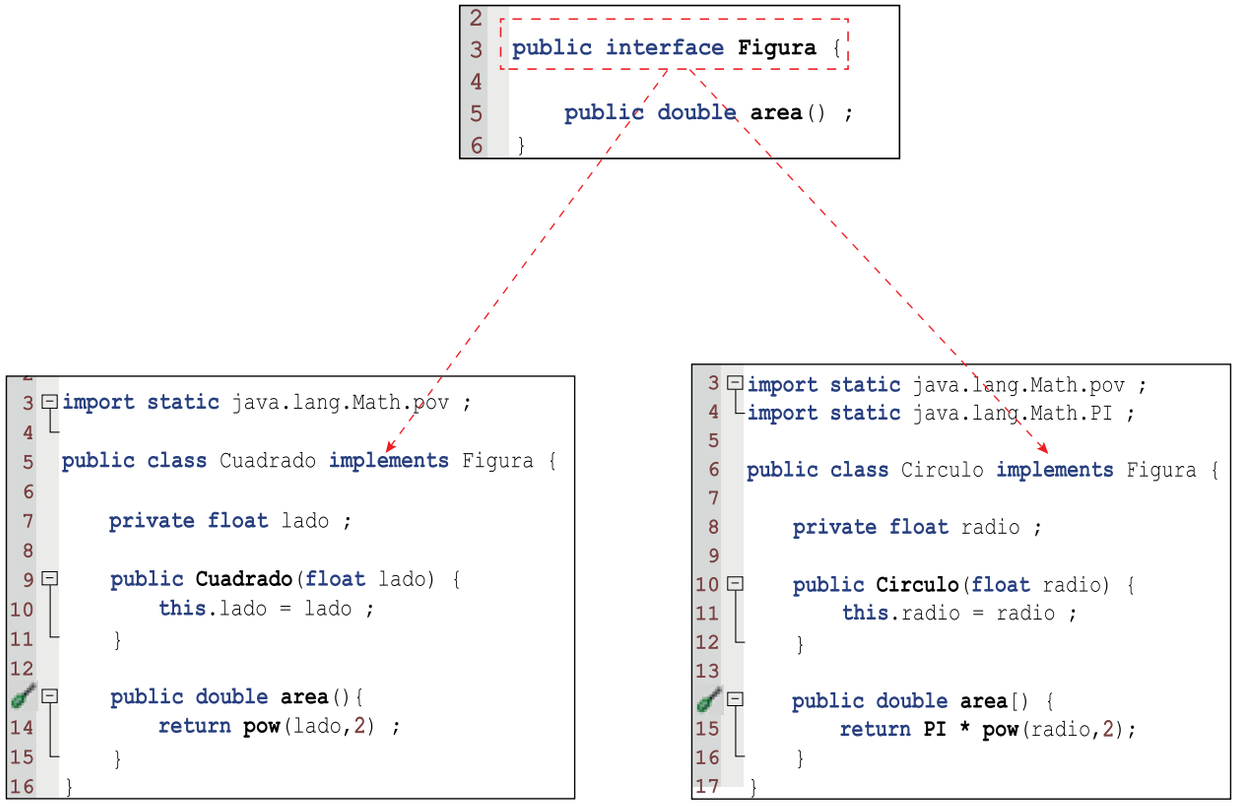


Figura 30. Interface UML. Fuente: Tomada de Herencia y polimorfismo, por Quezada, 2010.

Manipulación de cadenas de texto

Tema n.° 3

A continuación, se explicará cómo manejar cadenas de texto para el desarrollo de nuestros programas en Java.

1. Entrada / Salida de datos

Java tiene acceso a la entrada/salida estándar a través de la clase *System*, lo cual nos permite mostrar información y pedir el ingreso de información.

1.1. Entrada de datos

Leer datos desde la consola de Java, se haría por medio de la clase *BufferedReader* que obtiene los datos de la consola por medio de la clase *System*.

La forma de trabajo de la clase *BufferedReader* es la de solicitar los datos por medio de una clase *InputStreamReder* que recibe datos de la clase *System* y se podría representar de la siguiente manera:

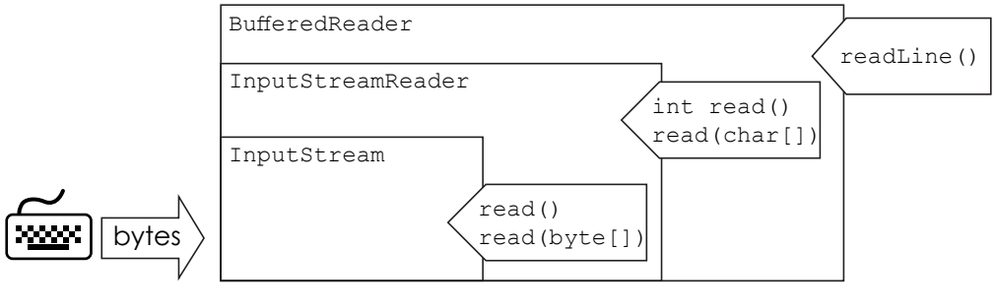


Figura 31. Clase BufferedReader
Fuente: Quezada, 2010

En Java si se quisiera solicitar que se ingrese el nombre de una persona, se tendría que declarar la variable *BufferedReader* de la siguiente manera:

```
BufferedReader lector = new BufferedReader(new InputStreamReader(System.in));
```

Por medio de dicha variable *(lector)* se podría solicitar que una persona ingrese un dato con la siguiente sentencia de código Java:

```
lector.readLine();
```

Donde la consola se pondrá en modo de espera para que se digite un nombre y que cuando presione la tecla *Enter* el dato será leído y almacenado en alguna variable si fuera necesario.

Un ejemplo completo de la implementación sería el siguiente:

```
public static void main(String[] args) {
    String nombre = "" ;

BufferedReader lector = new BufferedReader( new InputStreamReader( System.in ) );
    try{
        nombre = lector.readLine() ;
        System.out.println( nombre ) ;
    }catch(IOException ioe) {
        System.out.println("Error al leer datos"+ioe.getMessage());
    }

}
```

1.2. Salida de datos

Para mostrar datos por medio de la consola existen los siguientes métodos:

- *System.out.print()* : Imprimir datos en consola sin realizar salto de línea.
- *System.out.println()* : Imprimir datos en consola y realizar salto de línea.

2. La clase String

Dentro de un objeto de la clase *String* o *StringBuffer*, Java crea un arreglo de caracteres de una forma similar a como lo hace el lenguaje C++. A este arreglo se accede a través de las funciones miembros de la clase.

Los *strings* u objetos de la clase *String* se pueden crear explícitamente o implícitamente. Para crear un *String* implícitamente basta colocar una cadena de caracteres entre comillas dobles (http://www.sc.ehu.es/sbweb/fisica/cursoJava/fundamentos/clases1/string.htm). Por ejemplo, cuando se escribe lo siguiente:

```
System.out.println("Hola");
```

Para crear un *String* explícitamente colocamos la siguiente expresión:

```
String variable = new String( "Hola mundo" ) ;
```

O también se podría declarar de la siguiente manera:

```
String variable = "Hola mundo";
```

La clase *String* tiene los siguientes métodos que podría utilizar (Oracle, 2016)

- *length* : Permite obtener la longitud de la cadena.
- *startsWith* : *true* si la cadena empieza con el parámetro indicado.
- *endsWith* : *true* si la cadena termina con el parámetro indicado.
- *indexOf* : retorna la posición de la primera aparición del valor buscado (carácter o sub-cadena). Se puede especificar un índice inicial para la búsqueda.

Lectura seleccionada n.º 6

Berzal, F. (2011). Herencia. En *Clases y objetos* (pp. 2–13). Disponible en http://bit.ly/2bHaeDa

Berzal, F. (2011). Clases abstractas e interfaces. Disponible en http://bit.ly/1UGeTo0

Actividad n.º 5

Implemente el siguiente diagrama de clases, teniendo en cuenta que un empleado "Contratado" gana $10 por día trabajado y el empleado "Estable" gana $50 por día trabajado, pero si trabaja más de 30 días ganará siempre $2000. Realice su respectivo programa en consola.

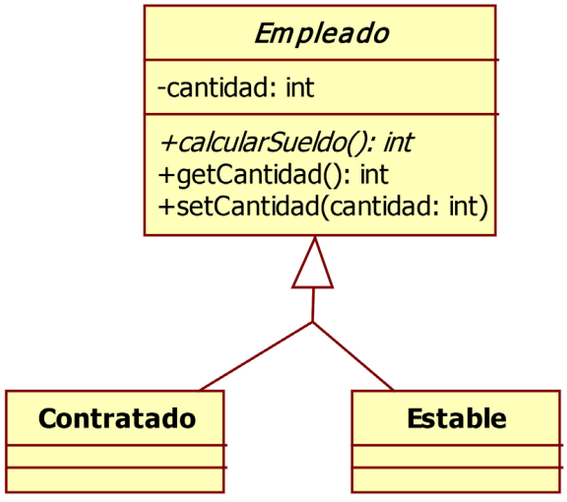


Figura 32. Ejercicio de herencia 01. Fuente: Elaboración propia

Lectura seleccionada n.º 7

Berzal, F. (2011). El lenguaje unificado de modelado. En *Introducción a la programación orientada a objetos* (pp. 38–47). Disponible en http://bit.ly/2bGndUK

Actividad n.º 6

Implemente el siguiente diagrama de clases y realice su respectivo programa en consola, que diga qué figura se desea implementar y que indique cuál es la respectiva área.

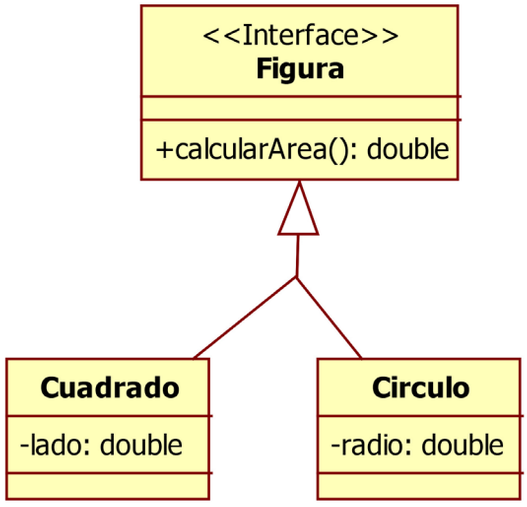


Figura 33. Ejercicio de herencia 02. Fuente: Elaboración propia

Glosario de la Unidad III

1. **Array**
   Es una estructura de datos que sirve para almacenar en una variable objetos del mismo tipo con un tope definido.

2. **ArrayList**
   Es una estructura de datos que sirve para almacenar en una variable objetos del mismo tipo sin un tope definido.

3. **Atributos**
   Son las características de las clases.

4. **Clase Abstract**
   Es una clase que por lo menos tiene un método abstracto que será implementado por la clase.

5. **Clases**
   Los objetos con estados similares y con el mismo comportamiento se pueden agrupar en clases.

6. **Interface**
   Es una clase donde todos los métodos son abstractatos.

7. **Matrices**
   Es un arreglo bidimensional que sirve para definir un formato de filas con columnas.

8. **Métodos**
   Son los comportamientos que se puede hacer con las clases.

9. **UML**
   Son las iniciales de Unified Modeling Language.

Bibliografía de la Unidad III

Oracle (13 de 07 de 2016). Java SE Donwloads [en línea]. Recuperado de http:// www.oracle.com/technetwork/es/java/javase/downloads/index.html

Oracle University (13 de 07 de 2016). About the Java Technology [en línea]. Disponible en http://docs.oracle.com/javase/tutorial/getStarted/intro/definition.html

Quezada, R. (04 de 01 de 2010). Herencia y polimorfismo [en línea]. Recuperado de http://es.slideshare.net/nano_trujillo/herencia-y-polimorfismo-2827363

Autoevaluación n.º 3

1. Se encuentra solo conformado por métodos abstractos. ¿Qué tipo de herencia se está utilizando?

   a) Clase Abstract
   b) Interface
   c) Clase
   d) Herencia Simple

2. Puede estar conformado por métodos abstractatos y métodos implementados. ¿Qué tipo de herencia se está utilizando?

   a) Clase Abstract
   b) Interface
   c) Clase
   d) Herencia simple

3. Si una clase B hereda de una clase A y la clase A se encuentra conformado por puros métodos implementados, ¿qué tipo de herencia se está utilizando?

   a) Clase Abstract
   b) Interface
   c) Clase
   d) Herencia simple

4. Cuando queremos llamar a un método o atributo de la clase padre por medio de una clase hija, ¿qué palabara reservada debemos usar?

   a) super
   b) this
   c) extends
   d) implements

5. Cuando queremos llamar a un método o atributo propio de la clase, ¿qué palabara reservada debemos usar?

   a) super
   b) this
   c) extends
   d) implements

6. ¿Qué tipo de estructura de datos utilizaria para guardar la cantidad de personas que puede ingresar en un banco?

   a) ArrayList
   b) Array
   c) Matrices
   d) Ninguna de las anteriores

7. ¿Qué tipo de estructura de datos utilizaria para guardar la cantidad de personas que nacen en el Perú?

   a) ArrayList
   b) Array
   c) Matrices
   d) Ninguna de las anteriores

8. Para obtener el tamaño de un *array*, ¿qué variable utilizamos?

   a) length
   b) size
   c) tamaño
   d) top

9. ¿Con qué linea de código completaría la siguiente sentencia?
   _____.out.println("Hola Mundo");

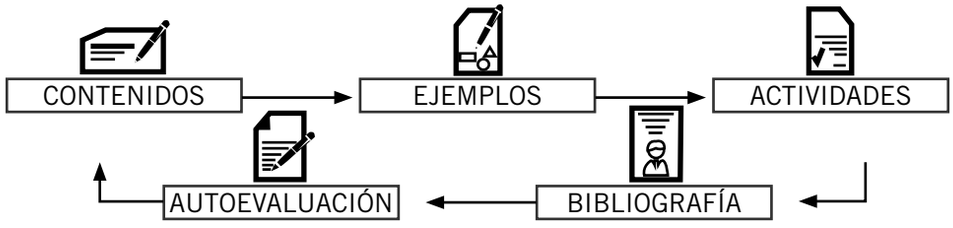
   a) system
   b) System
   c) Print
   d) print

10. ¿Qué método utilizaría para hacer que una palabra empiece con un término específico?

    a) indexOf
    b) startsWith
    c) endsWith
    d) length

UNIDAD IV
DISEÑO ORIENTADO A OBJETOS

DIAGRAMA DE ORGANIZACIÓN DE LA UNIDAD IV



ORGANIZACIÓN DE LOS APRENDIZAJES

Resultados del aprendizaje de la Unidad IV: Al finalizar la unidad el estudiante será capaz de utilizar la metodología orientada a objetos para el diseño de una aplicación centrada en el usuario.

CONOCIMIENTOS	HABILIDADES	ACTITUDES
Tema n.º 1: Excepciones 1.1 Excepción 1.2 Propagación de excepciones 1.3 Lanzamiento de excepciones 1.4 Excepciones personalizadas **Tema n.º 2: Diseño orientado a objetos** 1.1 Diagrama de clases 1.2 Relaciones entre clases 1.3 Asociaciones 1.3.1 Asociación binaria 1.3.2 Asociación reflexiva 1.3.3 Asociación N-aria 1.3.4 Asociación de generalización/especialización **Lectura seleccionada n.º7** Relaciones entre clases **Lectura seleccionada n.º 8:** Uso de excepciones en Java **Autoevaluación de la unidad IV**	1. Identifica las clases que se necesitan para el desarrollo de un programa. 2. Utiliza un estilo de programación para mejorar la legibilidad del mismo. 3. Realiza procesos de prueba y depuración de una aplicación. **Actividad n.º 7** **Actividad n.º 8** **Control de lectura n.º 4** Evaluación de la lectura seleccionada y los temas n.º 1, 2 y 3	Aprecia la importancia de la programación en la solución de problemas. Asume una actitud ética y de respeto a los demás.

Excepciones

Tema n.° 1

En la Unidad N.° 2, usted conoció todo lo referente a la programación orientada a objetos, mientras en esta unidad usted conocerá todo lo referente al uso de arreglos y colecciones y el uso de la herencia para el desarrollo de programas.

1. Excepción

Una excepción viene a ser un flujo anormal dentro del desarrollo de una aplicación. Por ejemplo, si queremos dividir un número entre cero, la aplicación se "caería", eso quiere decir que podría dejar de funcionar correctamente.

Una excepción siempre corta el flujo de un programa, por eso se debe controlar, para que no afecte de manera brusca al programa. Por ejemplo, en las siguientes líneas de código no se está implementando un control de excepciones.

```
3  public class Main {
4
5      public static void main(String[] args) {
6          String alumnos[] = new String[5] ;
7          alumnos[0] = "Rolando" ;
8          alumnos[1] = "Jorge" ;
9          alumnos[2] = "Elmer" ;
10         alumnos[3] = "José" ;
11         alumnos[4] = "Iván" ;
12
13         for(int i=0; i<= alumnos.length; i++ ){
14             System.out.println( alumnos[i] ) ;
15         }
16
17     }
18
19 }
```

```
Rolando
Jorge
Elmer
José
Iván
Exception in thread  main  java.lang.ArrayIndexOutOfBoundsException: 5
        at Excepciones.Main.main(Main.java:14)
Java Result: 1
```

Figura 34. Herencia simple. Fuente: Tomada de Herencia y polimorfismo, por Quezada, 2010.

Además, la aplicación está dejando de funcionar porque se quiere acceder a una posición que no existe. Y para controlar las excepciones se tiene que usar el bloque *try-catch*.

1.1. Bloque try-catch

El bloque *try-catch* se utiliza para controlar las excepciones de codificación. Dentro del bloque de la sentencia *try* se codifica el código fuente que podría generar una excepción y en el bloque *catch* se implementa el control de excepciones. El código anterior se debería implementar de la siguiente manera para efectuar un correcto control de excepciones.

```java
package excepciones;

public class Main {

    public static void main(String[] args) {
        String alumnos[] = new String[5] ;
        alumnos[0] = "Rolando" ;
        alumnos[1] = "Jorge" ;
        alumnos[2] = "Elmer" ;
        alumnos[3] = "José" ;
        alumnos[4] = "Iván" ;

        try{
            for(int i=0; i<= alumnos.length; i++ ){
                System.out.println( alumnos[i] ) ;
            }
        }catch(ArrayIndexOutOfBoundsException ex){
            System.out.println("No hay mas elementos en el arreglo") ;
        }

    }

}
```

```
run-simgle:
Rolando
Jorge
Elmer
José
Iván
No hay mas elementos en el arreglo
```

Figura 35. Bloque try-catch. Fuente: Tomada de Herencia y polimorfismo, por Quezada, 2010.

1.2. Captura de excepciones

En el siguiente punto se mostrará una serie de ejemplos de cómo se lleva el control de excepciones.

1.2.1. Solo una sola excepción

Dada la siguiente línea de código:

```java
// Bloque 1
try{
     // Bloque 2
}catch(Exception e){
     // Bloque 3
}
// Bloque 4
```

El flujo de ejecución del programa sería el siguiente:

A. Si no ocurre ninguna excepción, el flujo sería:
   a. Bloque 1
   b. Bloque 2
   c. Bloque 4

B. Si ocurre una excepción en el Bloque 2, el flujo sería:
   a. Bloque 1
   b. Bloque 2
   c. Bloque 3
   d. Bloque 4

C. Si ocurre una excepción en el Bloque 1, el flujo sería:
   a. Bloque 1

Tener en consideración que la clase *Exception* es la clase de la que heredan los demás tipos de excepciones.

1.2.2. Captura selectiva de excepciones

También se puede capturar excepciones según el tipo de excepción que deseamos controlar. Por ejemplo, si queremos controlar excepciones del tipo aritmético (por ejemplo, división entre cero) y excepciones del tipo *NullPointer* (cuando se quiere usar una variable sin haberla instanciado), el código sería el siguiente:

```
// Bloque 1
try{
     // Bloque 2
}catch(ArithmeticException ae){
     // Bloque 3
}catch(NullPointerException ne){
     // Bloque 4
}
// Bloque 5
```

El flujo de ejecución del programa sería el siguiente:

A. Si no ocurre ninguna excepción, el flujo sería el siguiente:
   a. Bloque 1
   b. Bloque 2
   c. Bloque 5

B. Si ocurre una excepción en el Bloque 2 del tipo aritmético, el flujo sería el siguiente:
   a. Bloque 1
   b. Bloque 2
   c. Bloque 3
   d. Bloque 5

C. Si ocurre una excepción en el Bloque 2 del tipo *NullPointer*, el flujo seria el siguiente:
   a. Bloque 1
   b. Bloque 2
   c. Bloque 4
   d. Bloque 5

D. Si ocurre una excepción en el Bloque 2 diferente a *NullPointer* o aritmético, el flujo sería el siguiente:
   a. Bloque 1
   b. Bloque 2

E. Si ocurre una excepción en el Bloque 1, el flujo sería el siguiente:
   a. Bloque 1

1.2.3. Bloque finally

Todo lo que se ponga dentro del bloque *finally* se ejecutará así exista o no exista una excepción.

Por ejemplo, dadas las siguientes líneas de código:

```
// Bloque 1
try{
     // Bloque 2
}catch(ArithmeticException ae) {
     // Bloque 3
}finally {
     // Bloque 4
}
// Bloque 5
```

El flujo de ejecución del programa sería como presentamos a continuación.

A. Si no ocurre ninguna excepción, el flujo sería el siguiente:
   a. Bloque 1
   b. Bloque 2
   c. Bloque 4
   d. Bloque 5

B. Si ocurre una excepción en el Bloque 2 del tipo aritmético, el flujo sería el siguiente:
   a. Bloque 1
   b. Bloque 2
   c. Bloque 3
   d. Bloque 4
   e. Bloque 5

C. Si ocurre una excepción en el Bloque 2 distinto al tipo aritmético, el flujo sería el siguiente:
   a. Bloque 1
   b. Bloque 2
   c. Bloque 4

2. Propagación de excepción

La propagación de excepciones se utiliza para propagar una excepción y que otro método sea el responsable de implementar el control de la misma. La estructura del código fuente debería ser la siguiente:

```
type NombreMetodo(argumentos)throws excepcionX, excepcionY,… { }
```

La palabra reservada *throws* indica que otro método debe implementar el control de excepciones usando un bloque *try-catch* o seguir propagando la excepción. Por ejemplo, tenemos las siguientes líneas de código:

```
static void unMetodo() throws I0Exception {
      throw new I0Exception("Error de entrada/salida");
}
```

3. Lanzamiento de excepción

En muchas ocasiones se necesita generar una excepción y para eso se utiliza la sentencia *throw*. Por ejemplo, en las siguientes líneas de código:

```
throw new RuntimeException("Mensaje de Error...") ;
```

```
public class Clase {

    static void unMetodo() {
        try {
            throw new NullPointerException( "demo" );
        } catch( NullPointerException e ) {
            System.out.println( "Carturada la excepcion en unMetodo" );
            throw e;
        }
    }

    public static void main( String args[] ) {
        try {
            unMetodo();
        } catch( NullPointerException e ) {
            System.out.println( "Carturada de nuevo: " + e );
        }
    }
}
```

```
run-single:
Capturada la excepcion en unMetodo
Capturada de nuevo: java.lang.NullPointerException: demo
```

Figura 36. Sentencia *throws*. Fuente: Tomada de Herencia y polimorfismo, por Quezada (2010)

Se está generando una excepción manual del tipo *NullPointeException* que está siendo controlada por el método "*unMetodo()*" y también en el método "*main()*".

4. Excepciones personalizadas

En ocasiones el programador puede definir excepciones personalizadas, no las que vienen por defecto en Java, y son usadas para controlar excepciones que son del tipo de lógica de negocio. Por ejemplo, cuando se tiene saldo insuficiente de una tarjeta de crédito, se debe de controlar dicho código creando una excepción personalizada y para eso se tendría que crear una clase que herede de la clase excepción. El diagrama UML del programa sería el siguiente:

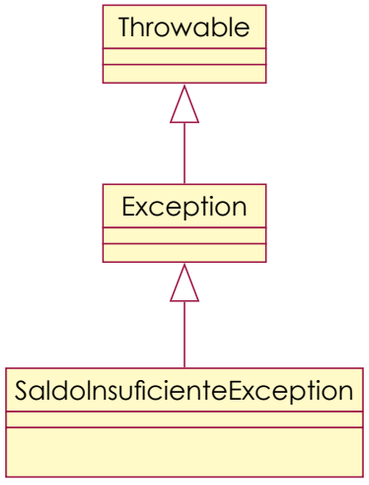


Figura 37. Excepción personalizada. Fuente: Tomada de Herencia y polimorfismo, por Quezada, 2010.

El código fuente de su implementación sería el siguiente:

```
package excepciones;

public class SaldoInsuficienteException extends java.lang.Exception {

    public SaldoInsuficienteException() {
    }

    public SaldoInsuficienteException(String msg) {
        super(msg);
    }
}
```

Diseño orientado a objetos

Tema n.° 2

En el siguiente punto usted comprenderá la importancia de hacer un diseño orientado a objetos que permita implementar programas orientados a objetos.

1. Diagrama de clases

El diagrama de clases es un diagrama UML que permite definir como un conjunto de clases se interrelacionan entre sí y es el principal diagrama para el desarrollo de aplicaciones orientado a objetos. Un ejemplo de diagrama de clases podría ser el siguiente:

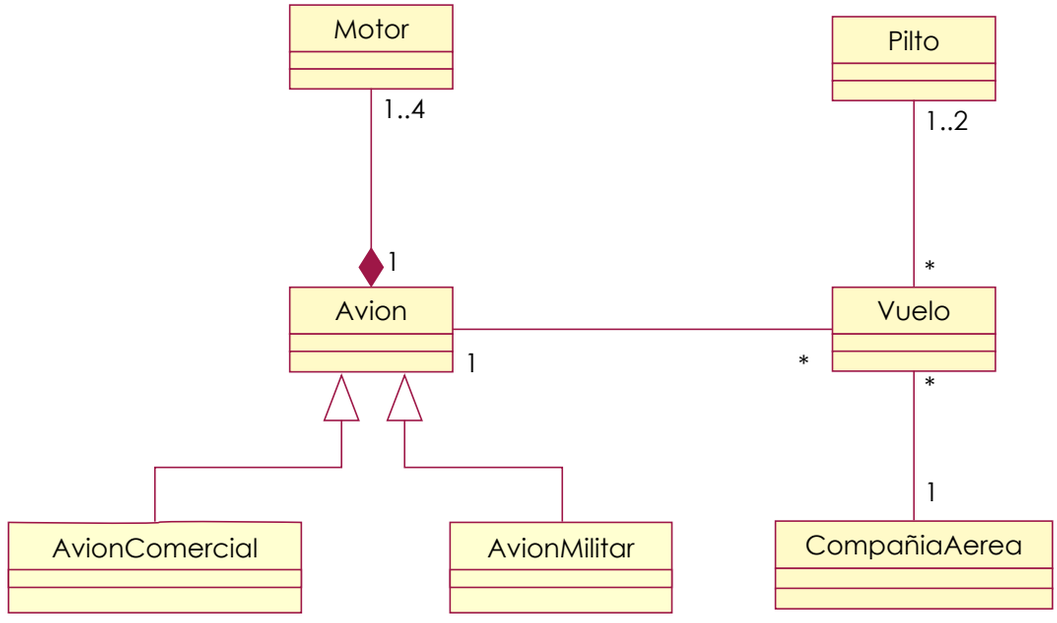


Figura 38. Diagrama de clases. Fuente: Tomada de Herencia y polimorfismo, por Quezada, 2010.

2. Relaciones entre clases

Las relaciones se realizan entre clases para representar como se comunican los objetos de las clases entre sí. La línea puede ir acompañada de diferentes tipos de adornos que definen su semántica y características. Los elementos opcionales que pueden existir en un diagrama de clases son los siguientes:

- Rol
- Relación
- Multiplicidad

Por ejemplo, si queremos indicar que una empresa tiene varias personas que trabajan para ellos y que una persona trabaja solamente para una empresa, se tendría que hacer el siguiente diagrama:

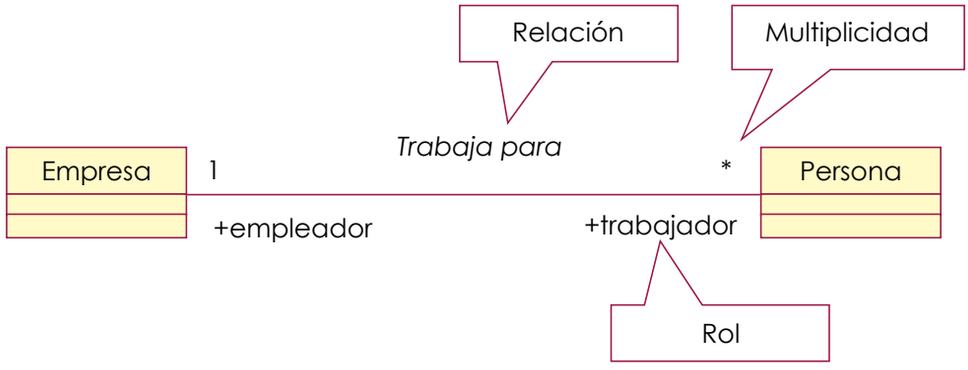


Figura 39. Relación entre clases. Fuente: Tomada de Herencia y polimorfismo, por Quezada, 2010.

3. Asociaciones

La asociación expresa una conexión bidireccional entre objetos. Una asociación es una abstracción de la relación existente en los enlaces entre los objetos (Quezada, 2010). Tenemos los siguientes tipos de asociaciones:

3.1. Asociación binaria

Una asociación binaria se representa mediante una línea sólida que une dos clases. Se trata de una relación entre las dos clases no muy fuerte; es decir, no se exige dependencia existencial ni encapsulamiento (Quezada, 2010).

Su representación y código fuente de ejemplo sería de la siguiente manera:

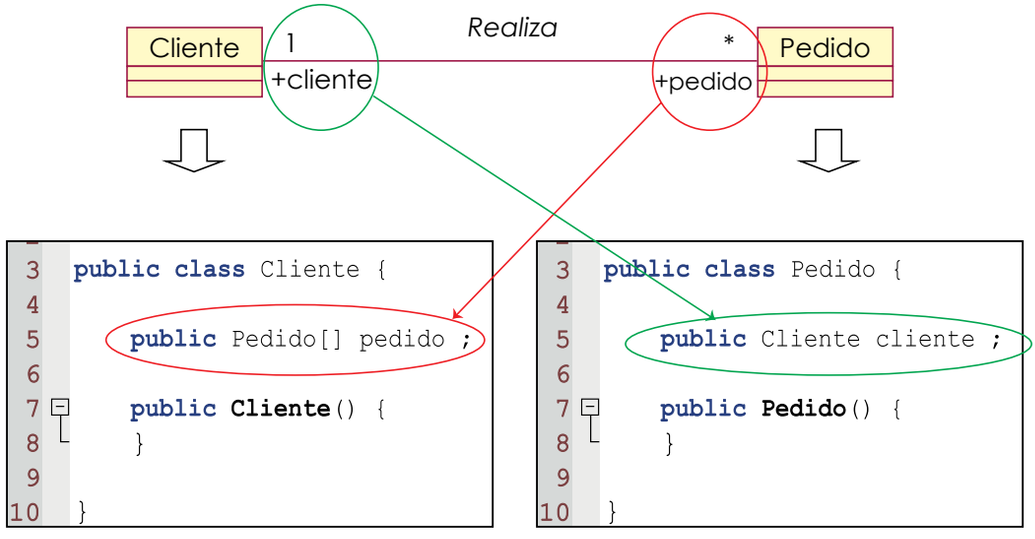


Figura 40. Relación binaria. Fuente: Tomada de Herencia y polimorfismo, por Quezada, 2010.

3.2. Asociación reflexiva

Una asociación reflexiva se representa mediante una línea sólida que recae sobre la misma clase de forma recursiva, se trata de una relación no muy fuerte, es decir, no se exige dependencia existencial ni encapsulamiento (Quezada, 2010).

Su representación y código fuente de ejemplo sería de la siguiente manera:

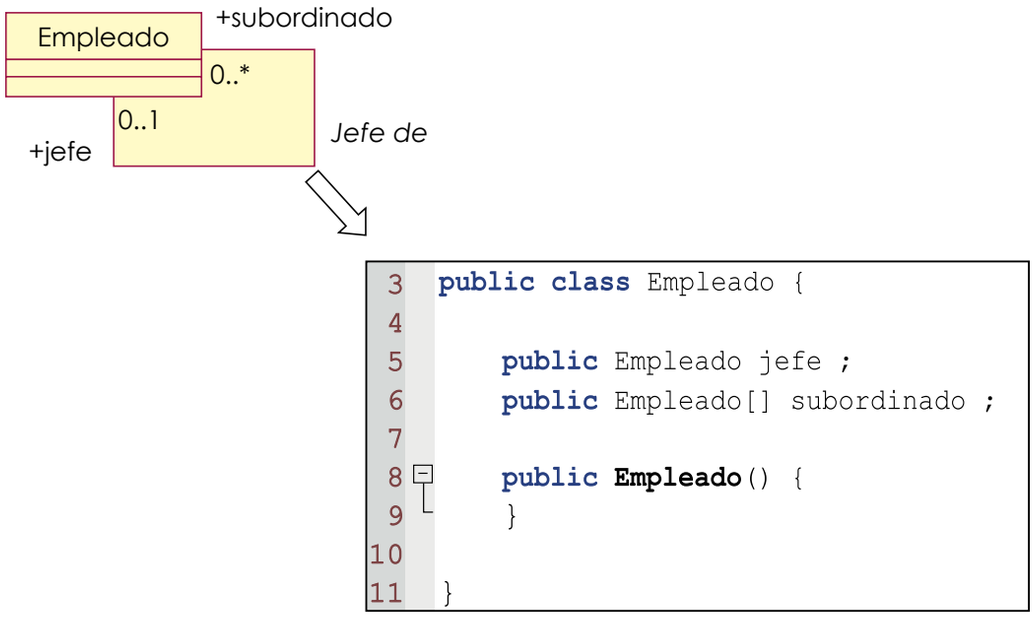


Figura 41. Relación reflexiva. Fuente: Tomada de Herencia y polimorfismo, por Quezada, 2010.

3.3. Asociación N-Aria

"Es una forma de expresar una relación entre tres o más clases. La clase de asociación es dependiente en existencia de las otras clases" (Quezada, 2010).

Su representación y código fuente de ejemplo sería de la siguiente manera:

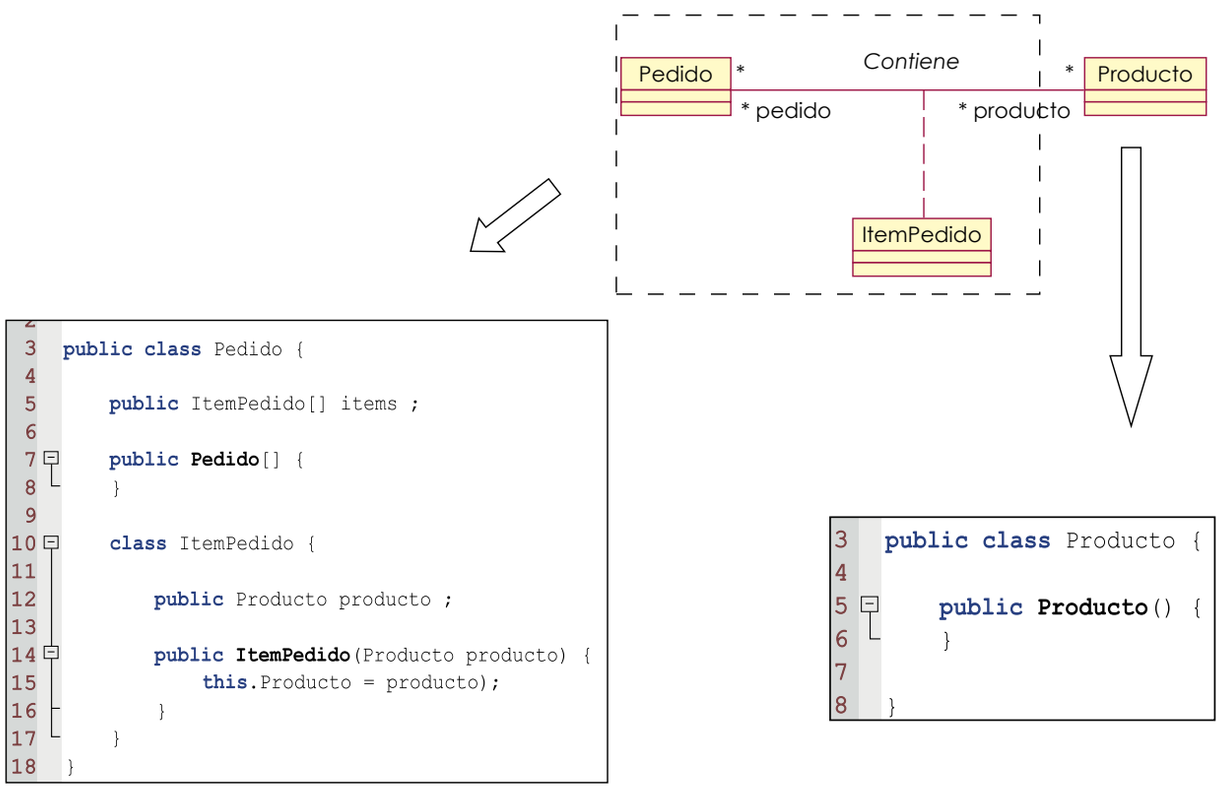


Figura 42. Relación N-Aria. Fuente: Tomada de Herencia y polimorfismo, por Quezada, 2010.

3.4. Asociación de generalización / especialización

Son el tipo de dependencia que representa la herencia entre clases, que vimos en la Unidad III.

Lectura seleccionada n.º 8

Berzal, F. (2011). Relaciones entre clases: Diagramas de clases UML. En *Introducción a la programación orientada a objetos* (pp. 23–33). Disponible en http://bit.ly/2boV02v

Actividad n.º 7

Realice un programa en Java donde se solicite qué operación matemática queremos realizar e ingresemos dos números enteros. Además, debe usted controlar la excepción de la división entre cero.

Lectura seleccionada n.º 9

Berzal, F. (2011). Uso de excepciones en Java. En *Manejo de excepciones* (pp. 6–14). Disponible en http://bit.ly/2bP0Han

Actividad n.º 8

A continuación, se muestra el diagrama de clases resultante del análisis y diseño del área de Gestión Académica de un centro de educación superior.

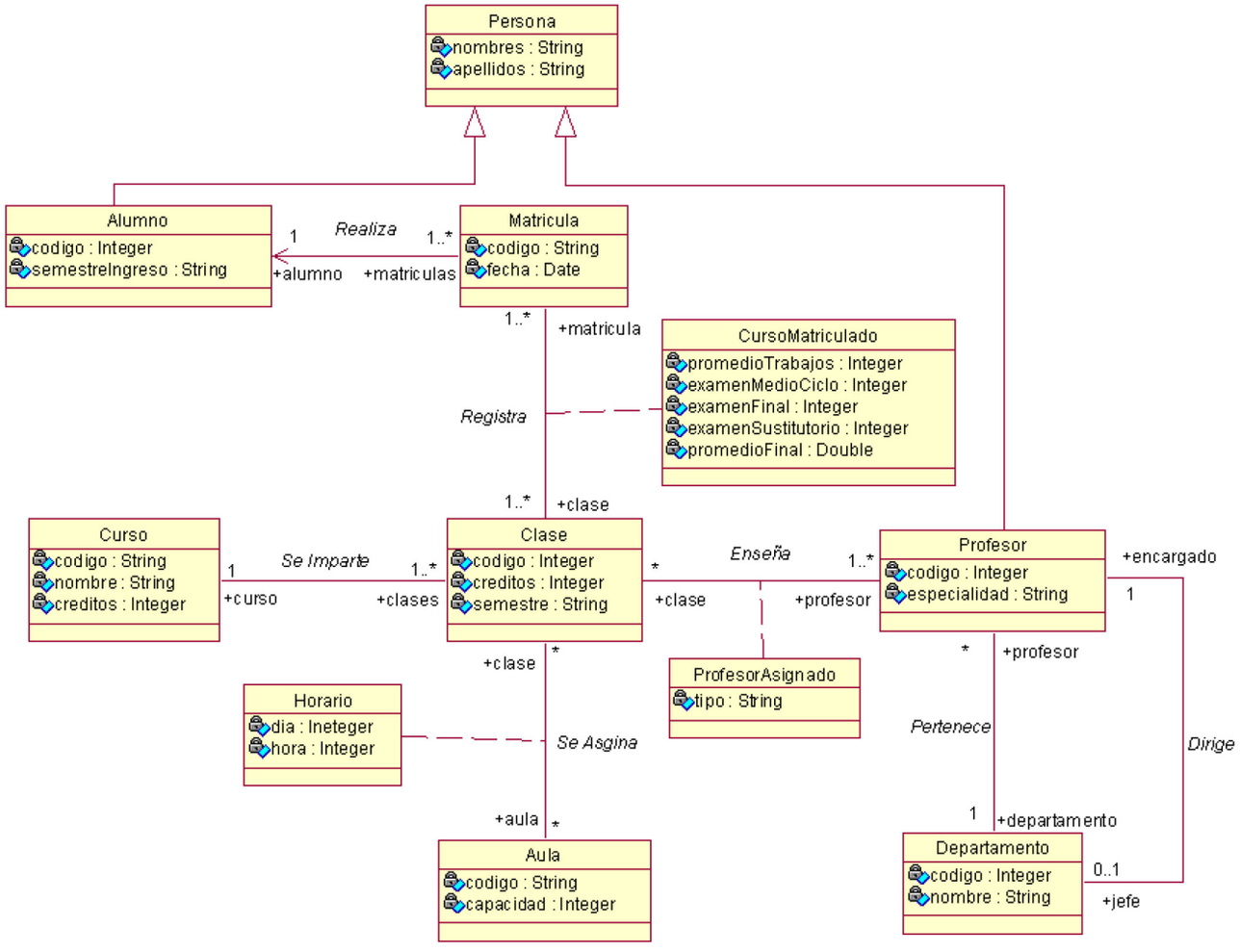


Figura 43. Diseño orientado a objetos. Fuente: Elaboración propia

Escriba las clases necesarias en Java para su implementación.

Glosario de la Unidad IV

1. **Asociación**
   Es la conexión entre clases.

2. **Bloque catch**
   Dentro de este bloque se insertan las líneas de código para controlar la excepción.

3. **Bloque finally**
   Dentro de este bloque se insertan las líneas de código que se necesita ejecutar así exista o no un error.

4. **Bloque try**
   Dentro de este bloque se insertan las líneas de código en las que se considera que puede existir una excepción.

5. **Dependencia**
   Es la relación de uso entre clases.

6. **Diagrama de clases**
   Es el diagrama donde se representa la relación entre clases.

7. **Generalización/dependencia**
   Es la relación de herencia entre clases.

8. **Multiplicidad de asociaciones**
   Es el número de instancias de una clase que se relacionan con una instancia de la otra clase.

9. **UML**
   Son las iniciales de Unified Modeling Language.

Bibliografía de la Unidad IV

Oracle (13 de julio de 2016). Java SE Donwloads [en línea]. Recuperado de http:// www.oracle.com/technetwork/es/java/javase/downloads/index.html

Oracle University (13 de julio de 2016). About the Java Technology [en línea]. Disponible en http://docs.oracle.com/javase/tutorial/getStarted/intro/definition. html

Quezada, R. (04 de enero de 2010). Herencia y polimorfismos [en línea]. Recuperado de http://es.slideshare. http://es.slideshare.net/nano_trujillo/herencia-y-polimorfismo-2827363

Autoevaluación n.º 4

1. ¿Cúal es el bloque siguiente que se ejecuta si ocurre una excepción en el bloque 1?

```
// Bloque 1
try{
     // Bloque 2
}catch(ArithmeticException ae) {
     // Bloque 3
}finally {
     // Bloque 4
}
// Bloque 5
```

a) Bloque 1
b) Bloque 2
c) Bloque 3
d) Bloque 4
e) Bloque 5
f) Ninguno

2. ¿Cúal es el bloque siguiente que se ejecuta si ocurre una excepción del tipo aritmético en el bloque 2?

```
// Bloque 1
try{
     // Bloque 2
}catch(ArithmeticException ae) {
     // Bloque 3
}finally {
     // Bloque 4
}
// Bloque 5
```

a) Bloque 1
b) Bloque 2
c) Bloque 3
d) Bloque 4
e) Bloque 5
f) Ninguno

3. ¿Cúal es el bloque siguiente que se ejecuta si ocurre una excepción que no sea del tipo aritmético en el bloque 2?

```
// Bloque 1
try{
     // Bloque 2
}catch(ArithmeticException ae) {
     // Bloque 3
}finally {
     // Bloque 4
}
// Bloque 5
```

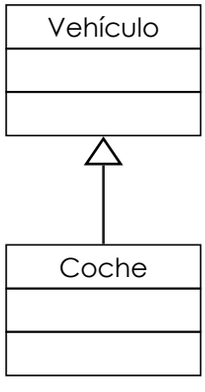
a) Bloque 1
b) Bloque 2
c) Bloque 3
d) Bloque 4
e) Bloque 5
f) Ninguno

4. ¿Cúal es el bloque siguiente que se ejecuta si ocurre una excepción que sea del tipo aritmético en el bloque 4?

```
// Bloque 1
try{
     // Bloque 2
}catch(ArithmeticException ae) {
     // Bloque 3
}finally {
     // Bloque 4
}
// Bloque 5
```

a) Bloque 1
b) Bloque 2
c) Bloque 3
d) Bloque 4
e) Bloque 5
f) Ninguno

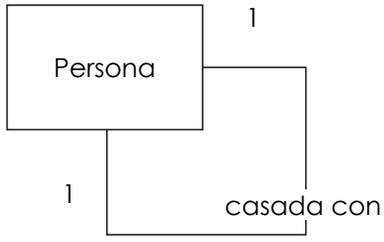
5. ¿Qué tipo de relación o asociación representa el siguiente gráfico?



a) Asociación binaria

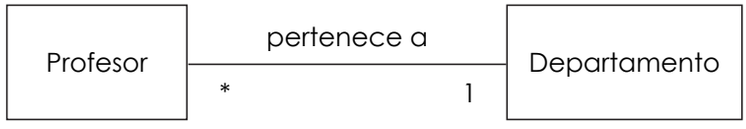
b) Asociación reflexiva
c) Asociación N-Aria
d) Generalización

6. ¿Qué tipo de relación o asociación representa el siguiente gráfico?



a) Asociación binaria
b) Asociación reflexiva
c) Asociación N-Aria
d) Generalización

7. ¿Qué tipo de relación o asociación representa el siguiente gráfico?



a) Asociación binaria
b) Asociación reflexiva
c) Asociación N-Aria
d) Generalización

8. ¿En qué bloque se tiene que colocar las lineas de código donde existirá una excepción?

a) try
b) catch
c) finally
d) simple

9. ¿En qué bloque se tiene que colocar las líneas de código para trabajar con la excepción "excepción"?

a) try
b) catch
c) finally
d) simple

10. ¿En qué bloque se tiene que colocar las lineas de código que se desea ejecutar así exista o no exista una excepción?

a) try
b) catch
c) finally
d) simple

Anexos

UNIDAD I

Número	Respuestas
1	B
2	A
3	D
4	D
5	C
6	D
7	C
8	B
9	A
10	B

UNIDAD II

Número	Respuestas
1	B
2	C
3	C
4	A
5	D
6	A
7	B
8	C
9	D
10	A

UNIDAD III

Número	Respuestas
1	B
2	A
3	D
4	A
5	B
6	B
7	A
8	A
9	A
10	B

UNIDAD IV

Número	Respuestas
1	F
2	C
3	D
4	F
5	D
6	B
7	B
8	A
9	B
10	C

Huancayo
Av. San Carlos 1980 - Huancayo
Teléfono: 064 - 481430

Lima
Jr. Junín 355 - Miraflores
Teléfono: 01 - 2132760

Cusco
Av. Collasuyo S/N Urb. Manuel Prado - Cusco
Teléfono: 084 - 480070

Arequipa
Calle Alfonso Ugarte 607 - Yanahuara
Oficina administrativa: Calle San José 308 2° piso - Cercado
Teléfono: 054 - 412030