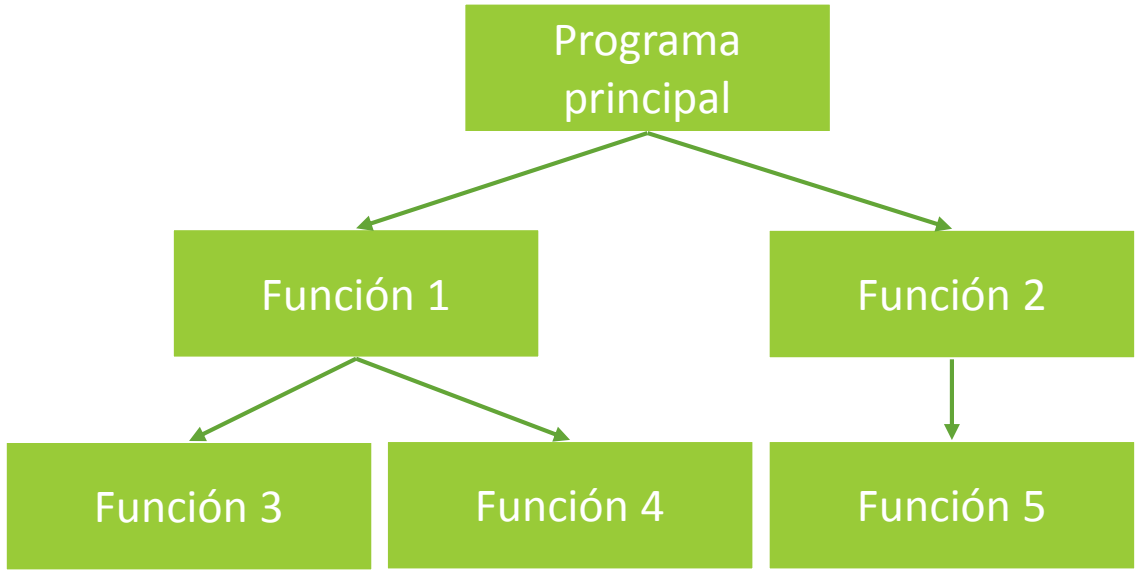
Introducción a Java

¿Qué es Java?

- Es un lenguaje de programación orientado a objetos desarrollado por Sun Microsystems
- Es un lenguaje de programación fuertemente tipado
  - Las variables se asocian por definición a **un** solo tipo de dato
- Concurrente y basado en clases
- Multiplataforma
- Se puede utilizar para aplicaciones de escritorio, web o móviles

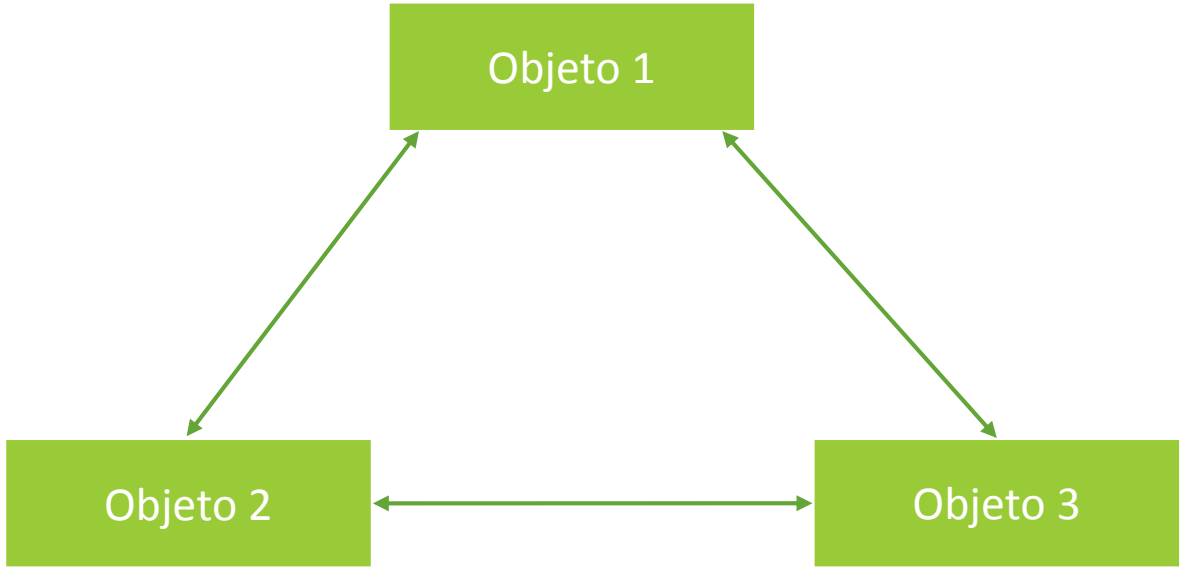
Programación Procedural

- Rutinas o funciones que contienen una serie de pasos a ser ejecutados en secuencia



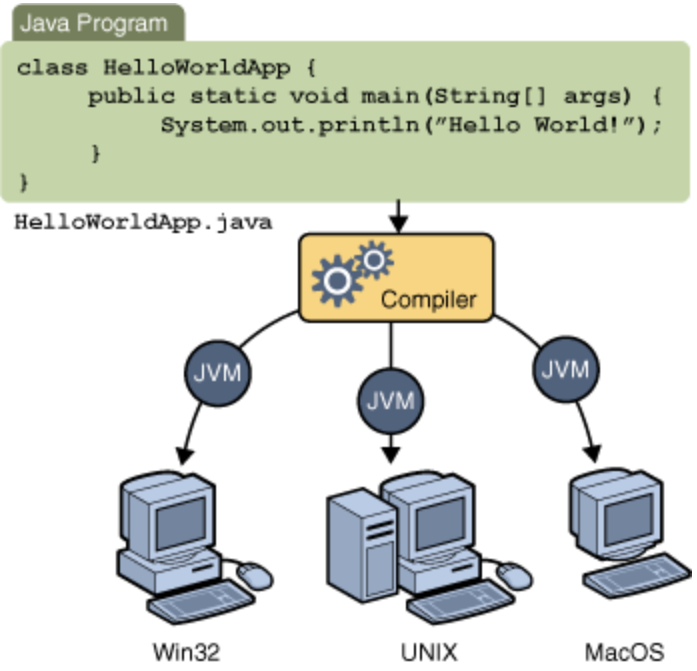
Programación Orientada a Objetos

- Objetos encapsulados que se comunican entre ellos a través de mensajes



El compilador Java

- Al programar para la plataforma Java, se escribe el código fuente en un archivo con extensión '*.java*' y luego se lo compila. El compilador hace un checkeo de las reglas de sintaxis del lenguaje y luego genera un archivo con extensión '*.class*' en lo que se conoce como *bytecodes*. Bytecodes son instrucciones estándares diseñadas para correr en la Java virtual machine (JVM)
- Al añadir esta capa de abstracción, el compilador de Java difiere de los compiladores de otros lenguajes, los cuales generan instrucciones que pueden ser directamente interpretadas por un CPU



JVM – Java Virtual Machine

- Al momento de correr el programa, la JVM lee e interpreta los archivos '.class' y ejecuta las instrucciones del programa en el hardware nativo de la plataforma donde la JVM está corriendo.
- La JVM es el corazón detrás de la filosofía "write-once, run-anywhere" (WORA) del lenguaje Java. Un código puedo correr en cualquier chipset que posea una implementación de Java.
- La JVM esta disponible para la mayor parte de las plataformas, como Linux, Windows y Mac además de implementaciones para teléfonos móviles.

Java - Componentes

- JDK – Java Development Kit
  - Al instalar el JDK se obtienen, además de el compilador y otras herramientas, una gran cantidad de librerías pre-compiladas con utilidades que nos ayudaran a completar las tareas de desarrollo.
- JRE – Java Runtime Environment
  - El JRE es lo mínimo necesario para ejecutar una aplicación Java.
  - Incluye la JVM, librerías y componentes que son necesarios para correr programas escritos en Java. Esta disponible para múltiples plataformas.
  - El JRE esta incluido en el JDK

Java – Paquetes

- Se utilizan en java para prevenir conflictos de nombres, manejar accesos y simplificar el uso de las distintas clases.
- Se pueden definir como una manera de agrupar tipos relacionados
- Se suelen nombrar de la siguiente manera:
  - `com.empresa.proyecto[.funcionalidad]`

Tipos de Datos en Java

Parte 1 – Datos primitivos

Tipos de Datos Primitivos

- También llamados ‘Value Types’ ya que tienen un largo fijo por lo cual ocupan siempre la misma cantidad de memoria
- Se traducen a un tipo de dato del procesador

Tipos de Datos Primitivos

Numéricos enteros

Tipo	Bytes	Rango
`sbyte`	1	-128 a 127
`byte`	1	0 a 255
`short`	2	-32.768 a 32.767
`ushort`	2	0 a 65.535
`int`	3	-2 billones a 2 billones
`uint`	3	0 a 4 billones
`long`	8	-10^{20} a 10^{20}
`ulong`	8	0 a $2x10^{20}$

Tipos de Datos Primitivos

Numéricos de punto flotantes

Tipo	Bytes	Rango	Precisión*
`float`	4	$-3.4*10^{38}$ a $3.4*10^{38}$	6 a 7
`double`	8	$\pm5.0*10^{-324}$ a $\pm1.7*10^{308}$	15 a 16
`decimal`	16	$-7.9*10^{28}$ a $7.9*10^{28}$	28 a 29

* Número de dígitos significativos que puede representar

Tipos de Datos Primitivos

Caracteres

- Sólo existe el tipo char
- Se utiliza para guardar un solo carácter
- Ocupa 2 bytes en memoria

Tipos de Datos Primitivos

Lógico

- Sólo existe el tipo `bool`
- Sólo puede tomar el valor `true` ó `false`
- Ocupa 1 byte en memoria

Strings

- El tipo de dato `string` es el utilizado cuando queremos trabajar con cadenas de texto que contengan más de un carácter
  - `string nombre = “Mariano”;`
- Se pueden aplicar muchas de las operaciones que hacemos con tipos de datos primitivos

Variables y constantes en Java

Variables

- Se utilizan para guardar un valor de un tipo determinado en un espacio de la memoria de la computadora
- El tipo con el cual la variable sea declarada, restringirá los valores que la misma pueda tomar

Nombrando Variables

- Los nombres de variables comenzaran con una letra minúscula
- Los nombres que contengan varias palabras, deben tener en mayúsculas los comienzos de palabras subsecuentes (Camel Casing)
- Utilizar nombres significativos para las variables que ayuden a identificar su uso
- Los nombres no pueden contener puntuación, espacios o barras

Declarando Variables

- Sintaxis básica
  - `tipo nombreVariable;`
  - `tipo nombreOtraVariable = valor;`
- Java contiene distintos modificadores que pueden ser aplicados a la declaración de una variable
  - `[modificadores] tipo nombreTercerVariable;`
  - `[modificadores] tipo nombreCuartaVariable = valor;`

Asignando valores a las variables

- Se conoce al operador ‘=‘ como el operador de Asignación, el cual le da un valor a la variable que declaremos
- Se pueden asignar valores literales
  - `int i = 15;`
  - `char primeraLetra = ‘A’;`
- Se pueden asignar resultados de una operación
  - `float resultado = 15,0 / 2,0;`

Asignando valores a las variables

- Se puede asignar el retorno de una función
  - `int resultado = sumaDosNumeros(numero1,numero2);`
- Se puede asignar una variable a otra
  - `float precio = 15,50;`
  - `float otroPrecio = precio;`

Promoción Automática de Variables

- Promoción Automática:
  - Se llama Promoción a asignar un tipo de tamaño mas chico a uno mas grande (ej: int a long)
  - Ejemplo:

```
int numero = 5;
long numeroGrande = numero;
```

Casteo de Variables

- Cuando no se puede pasar de un tipo de variable a otra sin riesgo de perder información, se necesita explicitar la conversión, a eso se le llama ‘cast’.
- Ejemplo:

```
double x = 1234.7;
int a;
a = (int)x; //’a’ tendrá por valor 1234
```

Constantes

- Se utilizan para guardar valores que no cambiaran durante la ejecución de nuestro programa
- Tendrán tipo y serán inicializadas al declararlas

Nombrando Constantes

- Las constantes se nombraran totalmente en mayúscula y separadas por ‘_’ (guiones bajos)
- Utilizar nombres significativos para las constantes que ayuden a identificar el valor contenido

Declarando constantes

- `final int MESES_DEL_AÑO = 12;`
- `final string NOMBRE;` → Será inicializada una vez y luego no podrá modificarse más

CAPTTON
IT CONSULTING

Tipos de Datos en Java

Parte 2

DateTime

- Contiene muchas operaciones para trabajar con fechas y horas
- Ej: Declaramos una variable del tipo y le asignamos la fecha y hora del sistema
  - `DateTime date = DateTime.Now;`

Vectores - *Arrays*

- Son una serie de elementos del mismo tipo. Estos tipos pueden ser tipos primitivos o clases
- Pueden ser multidimensionales - Matrices
- Sus elementos son rápidamente accesibles
- Tienen tamaño fijo
- Son iterables

Declarando vectores

- Declaración
  - `tipo[]  nombreVector;`
- Ejemplo
  - `string[] nombreAlumnos;`
  - `int[] listaPrecios;`

Inicializando vector

- Luego de declarar la variable, tendremos que instanciar el vector
  - `nombreArray = new tipo[tamaño];`
- `Ejemplo`
  - `nombreAlumnos = new string[15];`
  - `listaPrecios = new int[10];`

Inicializando un vector

- Asignamos un valor a una posición dentro del vector
  - `nombreAlumnos[0] = “Mariano Rodriguez”;`
  - `nombreAlumnos[1] = “Juliana Perez”`
- Otra manera de declarar e inicializar al mismo tiempo
  - `int[] listadoPrecios = { 1, 15, 26, 44}`

Accediendo a los vectores

- Accederemos a los valores guardados en los vectores a partir del índice en el cual se encuentra el valor buscado
  - `string alumno4 = listadoAlumnos[4];`
  - `int precios = listadoPrecios[3];`
- Podemos conocer el largo del vector utilizando la propiedad '*Length*'
- El último valor accesible del vector será su longitud menos uno

Vectores en dos dimensiones - Matrices

- Se declaran
  - `tipo[row,col] nombreMatriz = new tipo[row,col];`
- Se inicializan
  - `int[4,2] matrizNumeros = new int[4,2];`
  - `matrizNumeros[0,0] = 1`
  - `matrizNumeros[0,1] = 3`

Vectores en dos dimensiones - Matrices

- Otra forma de inicializar matrices
  - `int[,] matriz = new int[,] { { 1, 2 }, { 3, 4 }, { 5, 6 }, { 7, 8 } };`
  - `int[,] matriz2 = new int[4,2] { { 1, 2 }, { 3, 4 }, { 5, 6 }, { 7, 8 } };`

CAPTTON
IT CONSULTING

CAPTTON
IT CONSULTING

Operadores en Java

Operadores matemáticos estándar

Operación	Operador	Ejemplo
Suma	+	`int resultado = 3 + 5;`
Resta	-	`int  resultado2 = 25 – 4;`
Multiplicación	*	`int resultado3 = 10 * 3;`
División*	/	`int resultado4 = 10 / 3;`

*La división en este caso es entera, no devuelve decimales

Operadores Racionales

- Los operadores racionales evalúan si una condición se cumple o no y devuelven un valor booleano acordemente

Condición	Operador	Ejemplo
Es igual a	==	Int a = 1; a == 1
No es igual a	!=	Int a = 2; a != 1
Es menor a	<	Int a = 0; a < 1
Es menor o igual a	<=	Int a = 1; a <= 1
Es mayor a	>	Int a = 2; a > 1
Es mayor o igual a	>=	Int a = 1; a >= 1

Operadores Condicionales Comunes

Condición	Operador	Ejemplo
Devolverá `true` si se cumplen ambas condiciones (AND)	&&	Int a = 2; Int b = 9; ((a < 1) && (b > 6))
Devolverá `true` si se cumple una u otra condición (OR)	||	Int a = 2; Int b = 9; ((a < 1) || (b > 10))
Devolverá `true` si NO se cumple la condición (NOT)	!	Int a = 5; (!(a < 3))

Operadores de incremento y decremento

Operación	Operador	Ejemplo	Comentario

Incremento precedido	++	`int i = 6;`
		`int j = ++i;`
		`i es 7 / j es 7`
			Primero incrementa, luego asigna

Incremento procedido		`int i = 6;`
		`int j = i++;`
		`i es 7 / j es 6`
			Primero asigna, luego incrementa

Decremento precedido	--	`int i = 6;`
		`int j = --i;`
		`i es 5 / j es 5`
			Primero hace el decremento, luego asigna

Decremento procedido		`int i = 6;`
		`int j = i--;`
		`i es 5 / j es 6`
			Primero asigna, luego hace el decremento

- Ejemplo:

```
int numero=15;
int a, b, c, d;
a = numero++;
b = numero;
c = ++numero;
d = numero;
```

- ¿Que valores tomarán a, b, c y d luego de ejecutadas estas sentencias?

Precedencia de operaciones

- Reglas de precedencia:
  - Operadores entre paréntesis
  - Operadores de incremento o decremento
  - Operadores de división y multiplicación, evaluados de izquierda a derecha.
  - Operadores de suma y resta, evaluados de izquierda a derecha.
- Ejemplo:
  - `c = 25 – 5 * 4 / 2 – 10 + 4`
  - ¿Qué valor tiene ‘c’ luego de ejecutada la operación según las reglas?

Estructuras de Control

Estructuras Condicionales

- Se utilizan la sentencias ‘if’ y ‘else’ para delimitar comportamiento condicional dentro de nuestro programa

```
if ( expresion_booleana ){
  //Código si expresión_booleana es True
} // fin del if
else {
  //Código si expresión_booleana NO es True
} // fin del else
```

Estructuras Condicionales

- Se puede utilizar la sentencia ‘if’ sin declarar una sentencia ‘else’
- Se pueden anidar
- Se puede utilizar de la forma ‘if’, ‘else if’, ‘else’

```
if( expresion_booleana) {
  if( expresion_booleana) {
          bloque_Codigo;
  } // fin del if interno
} // fin del primer if
else if( expresion_booleana) {
  if( expresion_booleana ) {
          bloque_Codigo;
  } // fin del if interno
  else {
          bloque_Codigo;
  } // fin del else del if interno
} // fin del else if
```

Estructuras Condicionales – ‘Switch’

- Se utiliza cuando se desean realizar pruebas de igualdad.
- Se utiliza cuando se desea preguntar por valores simples
- Se utiliza cuando se desea preguntar por valores posibles en variables del tipo `int`, `char` o `string`.

Estructuras Condicionales – 'Switch'

- Sintaxis

```
switch( variable ) {
        case valor:
             bloque_Codigo;
             [break;]
        case otro_valor:
             bloque_Codigo;
             [break;]
        [default:]
             bloque_Codigo;
}
```

Iteraciones - While

- Ejecutará continuamente el código contenido hasta que la expresión condicional sea False
- Pueden anidarse
- Sintaxis:

```
while(expresion_booleana){
    bloque_Codigo;
} // fin del bloque while
```

Iteraciones – Do/While

- Se ejecutará el bloque de código una vez y luego se evaluará la condición. De ser True, repetirá el código contenido hasta que la condición deje de ser verdadera
- Sintaxis

```
do{
        bloque_Codigo;
}
while( expresion_booleana );
```

Iteraciones - For

- Se utilizan para repetir un número contado de veces el código contenido
- Pueden anidarse
- Sintaxis

```
for(inicializacion [, inicializacion]; expresion_booleana;
update [, update] ) {
        bloque_Codigo;
}
```

Comparativa de Iteraciones

- El While ejecutará el código contenido **0 o más veces**
- El Do/While ejecutará el código contenido **1 o más veces**
- El For ejecutará el código contenido **una cantidad de veces definidas**

CAPTTON
IT CONSULTING