**Instituto Politécnico Nacional**

**Escuela Superior de Cómputo**

Sistemas Distribuidos

**Tarea 1: Fundamentos de Java**

7CM1

Alumno: Franco Olvera Demian Oder

Boleta: 2021630278

Profesor: Ukranio Coronilla Contreras

Fecha: 18 de febrero de 2024

**Contenido**

[**Preámbulo** 2](#_Toc159104275)

[**JDK, JRE & JVM** 2](#_Toc159104276)

[**JDK** 2](#_Toc159104277)

[**JRE** 2](#_Toc159104278)

[**JVM** 3](#_Toc159104279)

[**Datos primitivos en Java** 3](#_Toc159104280)

[**Casting** 3](#_Toc159104281)

[**Clases y objetos** 3](#_Toc159104282)

[**Clases Wrapper** 4](#_Toc159104283)

[**Almacenar en memoria variable local y de tipo objeto** 5](#_Toc159104284)

[**Arreglos** 5](#_Toc159104285)

[**Arreglo String[] args** 6](#_Toc159104286)

[**Packages** 6](#_Toc159104287)

# **Preámbulo**

Java es un lenguaje de programación del paradigma Orientado a Objetos de uso popular que fue creado desde 1990 y ha permanecido como una opción viable al momento de desarrollar proyectos informáticos. El pertenecer al POO le permite a Java tener una opción viable para reutilizar código y es conocido por ser de propósito general, lo que significa que es capaz de ejecutarse en cualquier máquina sin la necesidad de realizar modificaciones importantes dentro del código fuente.

A continuación, se presenta una serie de apuntes que explican términos importantes con respecto a este lenguaje de programación.

# **JDK, JRE & JVM**

El desarrollo en Java parte del uso de múltiples herramientas clave, como es el caso del *Java Development Kit* (JDK), *Java Runtime Environment* (JRE) y la *Java Virtual Machine* (JVM). Se explican a continuación:

## **JDK**

El *Java Development Kit* es un software que incluye al intérprete de Java, clases y otras herramientas de desarrollo como un compilador, depurador, desensamblador, generador de archivos y documentación, entre otras cosas. Permite generar aplicaciones que de ejecución única (sin modificación o segunda compilación del código). Parte importante del JDK son también los paquetes y herramientas de Java.

## **JRE**

Se lo entiende como una capa de software ejecutado sobre el Sistema de Operativo para proporcionar las bibliotecas, clases y demás recursos para que un programa Java se pueda ejecutar. El JRE realiza una combinación entre el código Java que es creado gracias a las herramientas proporcionadas por el JDK para ejecutarlo en una instancia de la JVM.

## **JVM**

La *Java Virtual Machine* es un entorno de ejecución que puede colocarse dentro de cualquier Sistema Operativo. Tiene como función ejecutar las instrucciones generadas por un compilador de Java; tiene un intérprete de *Bytecode* y un entorno de tiempo de ejecución para ejecutar archivos de clase Java en cualquier plataforma. Esto es lo que permite que Java sea un lenguaje de propósito general.

# **Datos primitivos en Java**

Dentro de Java, el lenguaje como tal es considerado de tipado estático, por lo que cada variable es declarada junto con el tipo de dato al que pertenece. Existen ocho tipos de datos primitivos, los cuales son descritos a continuación junto con sus rangos y la memoria que utilizan:

* byte: utiliza 8 bits en memoria y su rango es -128 a 127.
* short: utiliza 16 bits en memoria y su rango es -32,768 a 32,767.
* int: utiliza 32 bits en memoria y su rango es -231 a 231.
* long: utiliza 64 bits en memoria y su rango es -263 a 263.
* float: utiliza 32 bits en memoria y su rango es -231 a 231.
* double: utiliza 64 bits en memoria y su rango es -263 a 263.
* boolean: utiliza sólo un bit de información.
* char: utiliza 16 bits en memoria y su rango es -32,768 a 32,767.

# **Casting**

Hay ciertas situaciones donde es necesario convertir un dato a otro tipo de dato de manera cruda y directa. A esta acción se le conoce como Casting o conversión y es muy utilizada entre datos primitivos. Suele hacerse directamente cuando los dos tipos de datos guardan una relación concreta.

Puede realizarse de forma explícita o implícita. Cuando se realiza un Cast implícito se iguala el valor de una variable con el de otra de distinto tipo siempre y cuando el tipo que va a almacenarla posea un rango más grande, mientras que los Cast explícitos declaran el tipo de variable específico al que se quiere convertir y puede almacenar datos grandes en primitivos con rangos más pequeños.

# **Clases y objetos**

Las clases y objetos son quizá el concepto más importante y general de Java. Las clases son la base de la POO. Una clase es una plantilla para la creación de objetos; dentro de una clase se colocan los datos y el código que se va a utilizar para operar dichos datos; las clases periten construir objetos, que vendrían a ser simplemente instancias de esa misma clase con datos específicos propios del objeto.

Al definir una clase nueva en Java, se declara su forma y naturaleza exacta. Esto quiere decir que se especifican los atributos que van a utilizar en instancia y los métodos que van a operar dentro de dichas clases.

En resumen, una clase es un prototipo o plantilla que define a las variables y a los métodos de todos los objetos comunes que van a representar a una entidad en particular. Los objetos, por otro lado, es una instancia o espécimen de una clase en particular. Los objetos suelen modelar o representar objetos del mundo real que suelen encontrarse en la vida cotidiana. A continuación, se presenta un ejemplo:

|  |  |
| --- | --- |
| |  | | --- | | 1. // Se declara una clase pública y se nombra 2. **public** **class** ClasePrueba 3. { 4. // Se declaran los atributos que tendrá la clase 5. **private** String atributo1; 6. **private** **long** atributo2; 7. **private** fouble atributo3; 9. // Se declara un constructor para la clase 10. **public** ClasePrueba(){} 12. // Se declaran los métodos que tendrá la clase 13. // Método vacío 14. **public** **void** metodo1() 15. { 16. System.out.println("Metodo vacio."); 17. } 18. // Método con retorno 19. **public** String metodo2() 20. { 21. **return** "Metodo 2.";     // Regresamos la String "Metodo 2." 22. } 23. } | |

# **Clases Wrapper**

Como ya se sabe, Java posee tipos de datos primitivos. Tomando en consideración las características POO inherentes del lenguaje, se define a los datos primitivos como aquellos que no tienen métodos, mientras que los no primitivos son todos aquellos que tienen métodos (objetos).

Para todos los datos primitivos existen unas clases conocidas como *Wrapper* o de envoltorio, las cuales tienen una serie de mecanismos que envuelven datos primitivos para tratarlos como objetos. Cada dato primitivo tiene asociado un *Wrapper*, los cuales se mencionan a continuación:

* *Byte.*
* *Short.*
* *Integer.*
* *Long.*
* *Float.*
* *Double.*
* *Boolean.*
* *Character.*

Todos los *Wrapper* provienen de la clase *Object*. La forma en que se utilizan es muy sencilla ya que lo único que se tiene que hacer es tratar a los datos a los que se les quiere hacer *Wrapper* como la creación de cualquier otro objeto. A continuación, un ejemplo con la clase *Integer*:

|  |  |
| --- | --- |
| |  | | --- | | 1. // Se declara una clase pública y se nombra 2. **public** **class** ClasePrueba 3. { 4. // Se declara una función main 5. **public** **static** **void** main(String args[]){ 6. // Declaramos un entero con su clase Wrapper 7. Integer i = **new** Integer(5); 8. **int** x = i.intValue(); 9. } 10. } | |

Cada objeto de tipo *Wrapper* tiene su conjunto de métodos particulares.

# **Almacenar en memoria variable local y de tipo objeto**

Hay que explicar primero la diferencia entre una variable local y otra de tipo objeto.

Las variables locales se declaran dentro de un método o un constructor y que sólo pueden accederse bajo ese contexto, por lo que sólo existen durante la ejecución de dicho método o bloque. Por otro lado, las variables de tipo objeto almacenan referencias y suelen estar a nivel de clase o método, por lo que su vida útil realmente no está determinada.

* Las variables locales se almacenan en el *Stack* de la memoria, por lo que se eliminan por completo cuando el bloque de código donde se encuentran termina su ejecución.
* Las variables de objeto almacenan una referencia a dicho objeto en el *Heap* de la memoria. El objeto se crea en el *Heap* y la variable sólo contiene una referencia. Cuando una variable de objeto deja de tener referencias hacia ella, puede eliminarse por el *Garbage Collector*.

Ahora, el Garbage Collector es una parte del JRE que gestiona la memoria liberándola de objetos que dejan de ser referenciados para evitar fugas o desbordamientos de memoria. Al eliminar dichos objetos, da lugar a que objetos nuevos ocupen su lugar. De esta forma, esta herramienta permite que no sea necesario gestionar la memoria directamente.

# **Arreglos**

No se trata realmente de un concepto exclusivo de Java, pero los arreglos son una estructura de datos que se utiliza para almacenar datos del mismo tipo; dichos elementos se encentran almacenados en ubicaciones de memoria juntos unos de otros. Pese a que no son exclusivos de Java, Java trata a los arreglos como objetos, por lo que los es adecuado decir que los arreglos tienen métodos.

Sus funciones son muy útiles: permiten almacenar múltiples valores del mismo tipo dentro de una única variable, así como permitir el acceso a cada uno de dichos valores gracias al hecho de que cada uno de ellos posee un índice numérico. Cabe aclarar que los arreglos son unidimensionales.

# **Arreglo String[] args**

Es normal que todas las aplicaciones Java tengan un método principal o main y cada uno de ellos suele tener declarado dicho arreglo dentro de los argumentos que pueden recibir.

Dicho arreglo es especial dado que recibe comandos directamente desde la línea de comandos (que podría ser el CMD de Windows o la terminal de Linux); de esta manera, es posible enviar información inmediata al programa gracias a los argumentos que se colocan dentro de la línea de comandos. Un caso de uso muy simple puede verse a continuación:

|  |  |
| --- | --- |
| |  | | --- | | 1. // Se declara una clase pública y se nombra 2. **public** **class** ClasePrueba 3. { 4. // Se declara una función main 5. **public** **static** **void** main(String args[]){ 6. // Tomamos los argumentos desde consola para realizar una suma 7. **int** suma = 0; 8. **for** (String arg : args) { 9. **int** num = Integer.parseInt(arg); 10. suma = suma + num; 11. } 12. System.out.println("El resultado es: " + suma); 13. } 14. } | |

# **Packages**

Los paquetes o dentro de Java es el medio que ofrece Java para ordenar los componentes de una aplicación. Permiten colocar en su interior cualquier cosa, como vendría a ser el caso de clases, archivos y demás código. Sirven principalmente para otorgarle un orden particular a alguna aplicación más compleja que incorpore muchos elementos para su funcionamiento. Permite ordenar o incluso categorizar de forma lógica los componentes de una aplicación.

Los paquetes dentro de Java, se generan y se importan; a continuación, se muestra un ejemplo práctico de creación e importación de un paquete dentro de Java:

Se crea un paquete con la clase Suma, para realizar dicha operación a través de un método que toma dos elementos y regresa un int con el resultado de la suma entre ambos:

|  |  |
| --- | --- |
| |  | | --- | | 1. // Nombramos el paquete 2. **package** adicion; 4. // Generamos la clase Suma 5. **public** **class** Suma { 6. // Establecemos el método que va a sumar los dos números enteros 7. **public** **static** **int** sumar(**int** a, **int** b) { 8. **return** a + b; 9. } 10. } | |

Tras ello, dicho paquete es importado en otro archivo que contiene una función main que importará el paquete y utilizará la clase para ejecutar una suma simple:

|  |  |
| --- | --- |
| |  | | --- | | 1. // Importamos el paquete 2. **import** adicion.Suma; 4. // Se declara una clase pública y se nombra 5. **public** **class** ClasePrueba 6. { 7. // Se declara una función main 8. **public** **static** **void** main(String args[]){ 9. **int** resultado = Suma.sumar(5, 3); 10. System.out.println("La suma es: " + resultado); 11. } 12. } | |