JAVA

Autor:

Kevin Cárdenas.

2023

Índice

1. introducción

Java es un lenguaje de programación orientado a objetos diseñado para ser portable, seguro y fácil de entender. La programación orientada a objetos (POO) es una técnica de programación que se centra en la organización del código en torno a objetos, que son entidades que contienen datos y métodos para manipular esos datos.

Java es un lenguaje compilado e interpretado. Cuando se escribe un programa en Java, se escribe en un archivo fuente que luego se compila en un archivo ejecutable. Este archivo ejecutable se puede ejecutar en cualquier plataforma que tenga una máquina virtual Java (JVM) instalada. La JVM es un software que permite que los programas Java se ejecuten en diferentes sistemas operativos sin necesidad de recompilar el código fuente.

El proceso de compilación de Java convierte el código fuente en un formato binario conocido como bytecode, que es un conjunto de instrucciones que la JVM puede interpretar y ejecutar en tiempo de ejecución. La JVM interpreta el bytecode y lo convierte en código de máquina nativo para la plataforma específica en la que se está ejecutando.

En resumen, Java es un lenguaje orientado a objetos que se compila en bytecode y se interpreta en tiempo de ejecución por la JVM, lo que lo hace altamente portable y seguro.

1.1. Instalación

Te proporciono las instrucciones para instalar Java en Linux utilizando la terminal:

1. Abre una terminal: Abre una ventana de terminal en Linux. En la mayoría de las distribuciones de Linux, puedes abrir una terminal haciendo clic en el botón de aplicaciones en la barra de tareas y buscando "terminal".

2. Actualiza el sistema: Antes de instalar Java, asegúrate de actualizar el sistema. Para hacerlo, ejecuta el siguiente comando en la terminal:

```
sudo apt-get update
```

   Este comando actualizará la lista de paquetes disponibles en el sistema.

3. Instala Java: Para instalar Java en Linux, puedes utilizar el comando apt-get. Ejecuta el siguiente comando en la terminal para instalar la versión más reciente de Java:

```
sudo apt-get install default-jdk
```

   Si deseas instalar una versión específica de Java, puedes reemplazar "default-jdk" con el nombre de la versión que deseas instalar (por ejemplo, "openjdk-11-jdk")

4. Verifica la instalación: Una vez que se haya completado la instalación, verifica que Java se haya instalado correctamente. Para hacerlo, ejecuta el siguiente comando en la terminal:

```
java -version
```

Si deseas instalar inteligent IDEA sólo ejecuta el comando `sudo snap install intellij-idea-community --classic`

Y ahora puedes escribir tu primer "Hola Mundo", abriendo un nuevo proyecto en nuestro IDE Creando un archivo llamado "HolaMundo.java" y escribiendo en el lo siguiente:

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

comienza con la definición de la clase "HelloWorld". Todas las clases de Java deben tener una definición como esta. La clase principal del programa debe tener el mismo nombre que el archivo de código fuente (en este caso, "HolaMundo.java").

Dentro de la clase "HelloWorld", se define un método llamado "main". Este método es el punto de entrada del programa. Todos los programas de Java deben tener un método "main" como este.

Dentro del método "main", se utiliza la función "println" de la clase "System" para imprimir el mensaje "Hola Mundo!" en la consola. El mensaje debe estar entre comillas dobles para indicar que es una cadena de texto.

1.2. Tipos de Variables

En Java, existen diferentes tipos de variables que se pueden utilizar para almacenar diferentes tipos de datos. Algunos de los tipos de variables más comunes en Java son:

- **int**: este tipo de variable se utiliza para almacenar valores enteros. Por ejemplo: `int edad = 30;`
- **double**: este tipo de variable se utiliza para almacenar valores decimales. Por ejemplo: `double altura = 1.75;`
- **boolean**: este tipo de variable se utiliza para almacenar valores booleanos (verdadero o falso). Por ejemplo: `boolean esMayorDeEdad = true;`
- **char**: este tipo de variable se utiliza para almacenar caracteres individuales. Por ejemplo: `char inicial = 'J';`
- **String**: este tipo de variable se utiliza para almacenar cadenas de texto. A diferencia de los otros tipos, la sintaxis de una variable de tipo String se escribe con mayúscula al inicio. Por ejemplo: `String nombre = "Juan";`

Además de estos tipos de variables básicos, existen otros tipos más complejos en Java, como arreglos, listas, objetos, etc.

Es importante tener en cuenta que las variables en Java son de tipado estático, lo que significa que una vez que se ha declarado el tipo de una variable, no se puede cambiar. Esto ayuda a prevenir errores comunes en tiempo de ejecución.

Por ejemplo:

```
public class EjemploVariables {

    public static void main(String[] args) {

```

```
        int edad = 30;
        double altura = 1.75;
        boolean esMayorDeEdad = true;
        char inicial = 'J';
        String nombre = "Juan";

        System.out.println("Nombre: " + nombre);
        System.out.println("Edad: " + edad);
        System.out.println("Altura: " + altura);
        System.out.println("Es mayor de edad?: " + esMayorDeEdad);
        System.out.println("Inicial: " + inicial);
    }
```

En este ejemplo, se declaran variables de diferentes tipos y se les asignan valores. Luego, se imprimen los valores de estas variables en la consola utilizando el método "println" de la clase "System". La salida del programa sería la siguiente:

```
Nombre: Juan
Edad: 30
Altura: 1.75
Es mayor de edad?: true
Inicial: J
```

En este caso, el programa imprime los valores de las variables "nombre", "edad", "altura", "esMayorDeEdad" e "inicial". Cada valor se concatena con una cadena de texto que describe qué valor se está imprimiendo.

En programación orientada a objetos (POO), además de los tipos de variables primitivos como `int`, `double`, `boolean` y `char`, existen también los tipos de variables de objeto, que se definen a partir de clases.

Los tipos de variables de objeto se utilizan para representar objetos de una clase. Por ejemplo, si tenemos una clase `Persona`, podemos crear un objeto `persona1` de esa clase y utilizar una variable para hacer referencia a ese objeto:

```
Persona persona1 = new Persona();
```

En este caso, `persona1` es una variable de tipo `Persona` que hace referencia a un objeto de la clase `Persona`. Podemos utilizar métodos y propiedades de la clase `Persona` para interactuar con el objeto al que hace referencia la variable `persona1`.

Además, en POO también existe el concepto de variables de instancia y variables de clase. Las variables de instancia son variables que se definen en una clase y que pertenecen a cada objeto creado a partir de esa clase. Por ejemplo, si tenemos una clase `Persona` con una variable de instancia `nombre`, cada objeto `persona1`, `persona2`, etc. tendría su propio valor de la variable `nombre`.

Por otro lado, las variables de clase se definen en una clase y son compartidas por todos los objetos creados a partir de esa clase. Para definir una variable de clase, se utiliza la palabra clave `static`. Por ejemplo, si queremos tener una variable `contador` que cuente la cantidad de objetos `Persona` creados, podemos definirla como una variable de clase:

```
public class Persona {
```

```
    static int contador = 0;
    String nombre;
    public Persona(String nombre) {
        this.nombre = nombre;
        contador++;
    }
}
```

En este caso, la variable `contador` se incrementa cada vez que se crea un objeto `Persona`. Todas las instancias de la clase `Persona` comparten la misma variable `contador`.

Longitud de un número

El tipo de dato "int" en Java es un tipo primitivo que representa números enteros. Este tipo de dato es utilizado para almacenar valores numéricos que no tienen decimales, y ocupa 32 bits (4 bytes) en la memoria.

Esto significa que un valor de tipo "int" puede representar cualquier número entero en el rango de "-2,147,483,648" a "2,147,483,647". Si intentamos almacenar un valor fuera de este rango, se producirá un error en tiempo de ejecución.

Es importante tener en cuenta que el tipo de dato "int" es de tamaño fijo, lo que significa que siempre ocupa la misma cantidad de memoria, independientemente del valor que esté almacenando. Esto lo hace más eficiente en términos de memoria que otros tipos de datos que permiten una mayor precisión, como el tipo "double" o "float".

Si necesitamos almacenar números enteros fuera del rango de un "int", podemos utilizar tipos de datos de mayor tamaño, como "long", que ocupa 64 bits (8 bytes) en la memoria y puede almacenar valores en el rango de "-9,223,372,036,854,775,808" a "9,223,372,036,854,775,807".

Diferencia entre float y double

En Java, existen dos tipos de datos de punto flotante: **float** y **double**. Ambos se utilizan para representar números decimales, pero difieren en su precisión y tamaño en memoria.

El tipo de dato **float** ocupa 32 bits en memoria y tiene una precisión de aproximadamente 6-7 dígitos decimales significativos. Por otro lado, el tipo de dato **double** ocupa 64 bits en memoria y tiene una precisión de aproximadamente 15-16 dígitos decimales significativos.

Por lo tanto, se recomienda utilizar el tipo de dato **float** cuando se necesita una precisión menor y se quiere ahorrar memoria. Por ejemplo, cuando se trabaja con grandes cantidades de datos en una aplicación. Por otro lado, se recomienda utilizar el tipo de dato **double** cuando se necesita una mayor precisión y no se tiene restricción en cuanto al uso de memoria.

Es importante tener en cuenta que al asignar valores a variables de tipo **float** o **double**, se deben incluir los sufijos **f** o **d**, respectivamente. Por ejemplo:

```
float numeroFloat = 3.14159f;
double numeroDouble = 3.14159d;
```

De lo contrario, el compilador de Java asumirá que se trata de un valor de tipo **double**.

Tipo de dato char

El tipo de dato `char` en Java es utilizado para almacenar un único carácter alfanumérico o un símbolo en Unicode. La sintaxis para declarar una variable de tipo `char` es la siguiente:

```
char variableChar = 'a';
```

Aquí, se declara una variable llamada `variableChar` de tipo `char`, y se le asigna el valor de la letra 'a'. Es importante notar que los valores de tipo `char` siempre deben estar encerrados en comillas simples.

Además, el tipo de dato `char` también puede representar valores numéricos utilizando su representación en la tabla ASCII. Por ejemplo:

```
char variableNum = 65;
```

Aquí, la variable `variableNum` de tipo `char` representa el valor numérico 65 en la tabla ASCII, que es la letra 'A'. Es importante mencionar que la representación en ASCII está limitada a los caracteres del conjunto ASCII, por lo que algunos caracteres especiales pueden no tener una representación ASCII válida.

Es importante tener en cuenta que una variable `char` ocupa 2 bytes de memoria en Java. Esto significa que puede representar valores de caracteres Unicode que se encuentran en el rango de 0 a 65,535.

Tipo de dato boolean

El tipo de dato **boolean** en Java es un tipo de dato que puede tener dos valores: **true** o **false**. Este tipo de dato es útil cuando se desea representar una condición lógica en el programa.

A diferencia de otros tipos de datos como int o double, que pueden tener una amplia gama de valores, un boolean solo puede tener dos posibles valores. Estos valores son útiles para representar situaciones donde una condición es verdadera o falsa, como por ejemplo en una sentencia de control de flujo como un **if** o un **while**.

Es importante recordar que los valores true y false en Java son sensibles a mayúsculas y minúsculas. Además, el tipo de dato boolean solo ocupa un bit en la memoria, lo que lo convierte en uno de los tipos de datos más eficientes en términos de espacio de almacenamiento.

Tipo de dato var

A partir de Java 10, se introdujo el tipo de dato `var`, que permite al compilador inferir el tipo de la variable en tiempo de compilación.

En lugar de declarar una variable como:

```
String nombre = "Juan";
```

Podemos usar `var`:

```
var nombre = "Juan";
```

En este caso, el compilador inferirá que el tipo de la variable `nombre` es `String`. También podemos usar `var` para tipos de datos más complejos, como colecciones:

```
var lista = new ArrayList<String>();
```

En este caso, el compilador infiere que el tipo de la variable `lista` es `ArrayList<String>`. Sin embargo, es importante tener en cuenta que el tipo de la variable no es dinámico, una vez que se ha inferido en tiempo de compilación, el tipo de la variable no puede cambiar.

El uso de `var` puede hacer que el código sea más conciso y fácil de leer, especialmente en casos donde el tipo de dato es complejo o largo de escribir. Sin embargo, también puede hacer que el código sea menos legible si se abusa de su uso o si no se usa con cuidado.

Variables constantes

En Java, también se pueden declarar variables que no pueden cambiar su valor después de haber sido inicializadas. Estas variables se llaman **variables constantes** o **constantes** y se declaran utilizando la palabra clave `final`. Por convención, el nombre de las constantes se escribe en mayúsculas y se separan las palabras con guiones bajos (`CONSTANTE_EJEMPLO`). Por ejemplo, podemos declarar una constante para el valor de PI de la siguiente manera:

```
final double PI = 3.14159265358979323846;
```

Una vez que se ha asignado un valor a una constante, no se puede cambiar. Si intentamos hacerlo, el compilador nos dará un error.

El uso de constantes puede ayudar a que nuestro código sea más fácil de leer y entender, ya que nos permite asignar un significado semántico a valores que de otra manera podrían ser difíciles de interpretar. Además, el hecho de que las constantes no puedan ser cambiadas accidentalmente nos ayuda a evitar errores en nuestro código.

1.2.1. Uso de variables

Para utilizar una variable en Java, primero se debe declarar, lo que significa reservar un espacio en memoria para la variable. La declaración de una variable se realiza especificando el tipo y el nombre de la variable, como se muestra en el siguiente ejemplo:

```
int edad;
```

En este caso, se está declarando una variable de tipo `int` llamada `edad`. Después de la declaración, se puede asignar un valor a la variable utilizando el operador de asignación `=`, como se muestra a continuación:

```
edad = 25;
```

En este caso, se está asignando el valor `25` a la variable `edad`.

También se puede declarar y asignar un valor a una variable en la misma línea, como se muestra en el siguiente ejemplo:

```
double precio = 10.99;
```

En este caso, se está declarando una variable de tipo `double` llamada `precio` y se le está asignando el valor `10.99`.

Una vez que se ha declarado y asignado un valor a una variable, se puede utilizar en el código para hacer cálculos y tomar decisiones. Por ejemplo:

```
int edad = 25;
if (edad >= 18) {
System.out.println("Eres mayor de edad");
} else {
System.out.println("Eres menor de edad");
}
```

En este caso, se está utilizando la variable `edad` para determinar si una persona es mayor o menor de edad.

También se puede actualizar el valor de una variable utilizando el operador de asignación. Por ejemplo:

```
int contador = 0;
contador = contador + 1;
```

En este caso, se está incrementando el valor de la variable `contador` en uno.

Es importante tener en cuenta que las variables en Java ocupan espacio en memoria y que este espacio es limitado. Por lo tanto, es importante utilizar variables de manera eficiente e intentar no crear más variables de las necesarias. Además, las variables que ya no se necesitan podriamos eliminarlas para liberar espacio en memoria y evitar fugas de memoria.

1.2.2. Convención de nombres

La convención de nombres en programación es importante para escribir código legible y fácilmente comprensible. En Java, existen algunas reglas de convención de nombres que se deben seguir para escribir código legible y coherente.

Primero, es importante recordar que Java es sensible a mayúsculas y minúsculas, lo que significa que una letra mayúscula se considera diferente de su contraparte en minúscula. Por ejemplo, la variable “nombre” es diferente de la variable “Nombre” en Java.

Además, existen algunas convenciones de nomenclatura comunes que se utilizan en Java. Por ejemplo, las variables generalmente se nombran usando una convención llamada “camelCase”. En camelCase, el primer carácter de la primera palabra es en minúscula, y la primera letra de cada palabra subsiguiente se escribe en mayúscula. Por ejemplo, “nombreDeUsuario” es un buen nombre de variable en camelCase.

En cuanto a los caracteres permitidos en los nombres de variables en Java, estos pueden contener letras, números y guiones bajos. Sin embargo, el primer carácter de un nombre de variable no puede ser un número. Además, los nombres de variables no pueden incluir espacios ni caracteres especiales como signos de puntuación o símbolos matemáticos.

Es importante seguir estas convenciones de nomenclatura para que el código sea fácil de leer y entender. Además, seguir las convenciones de nomenclatura también puede ayudar a prevenir errores en el código y hacer que sea más fácil de mantener y actualizar en el futuro.

El libro “Clean Code” de Robert C. Martin es una gran referencia para escribir código limpio y legible. Aquí hay un resumen de algunos de los conceptos clave:

- Nombres descriptivos: Elige nombres que describan claramente lo que hace una variable, método o clase. Nombres descriptivos hacen que el código sea más fácil de entender y de mantener.
- Funciones y métodos cortos: Las funciones y los métodos deben ser lo más cortos posible, idealmente menos de 20 líneas de código. Esto hace que el código sea más fácil de entender y de probar.
- Mantener un estilo consistente: Es importante mantener un estilo consistente en todo el código, desde la indentación hasta la nomenclatura de las variables. Esto hace que el código sea más fácil de leer y de mantener.
- Eliminar duplicación: La duplicación de código puede ser un problema importante en el código, ya que hace que el código sea más difícil de mantener y actualizar. Es importante identificar y eliminar la duplicación siempre que sea posible.
- Comentarios: Los comentarios deben ser utilizados para explicar el por qué del código, no el qué. El código debe ser lo suficientemente claro para entender lo que está haciendo sin necesidad de comentarios. Los comentarios pueden ser útiles para explicar decisiones arquitectónicas o decisiones de diseño.
- Pruebas unitarias: Las pruebas unitarias son esenciales para escribir código limpio y de calidad. Las pruebas unitarias aseguran que el código funciona correctamente y que los cambios en el código no introducen nuevos errores.

Estos son sólo algunos de los conceptos clave que se discuten en “Clean Code”. Siguiendo estos principios, podemos escribir código limpio y fácil de entender que es más fácil de mantener y actualizar en el futuro.

Convención de nombres: Upper Camel Case y Lower Camel Case

En Java, se utilizan dos convenciones de nombres comunes: Upper Camel Case y Lower Camel Case.

Upper Camel Case, también conocido como **PascalCase**, se utiliza para nombrar clases y tipos de datos. En Upper Camel Case, la primera letra de cada palabra se escribe en mayúscula, y no se utilizan espacios ni guiones bajos para separar las palabras. Por ejemplo:

```
public class MiClaseEjemplo {
// class code
}
```

Lower Camel Case, también conocido como **camelCase**, se utiliza para nombrar variables, métodos y funciones. En Lower Camel Case, la primera letra de la primera palabra se escribe en minúscula, y la primera letra de cada palabra siguiente se escribe en mayúscula. No se utilizan espacios ni guiones bajos para separar las palabras. Por ejemplo:

```
int edadDelUsuario = 25;
public void miMetodoEjemplo() {
// method code
}
```

1.3. Operadores

En programación, los operadores son símbolos o palabras reservadas que se utilizan para realizar operaciones matemáticas o lógicas sobre variables o valores. En Java, existen diferentes tipos de operadores que se utilizan en distintas situaciones.

Operadores aritméticos

Los operadores aritméticos se utilizan para realizar operaciones matemáticas básicas entre variables numéricas. Los operadores aritméticos en Java son los siguientes:

- Suma (+): se utiliza para sumar dos valores.
- Resta (-): se utiliza para restar un valor de otro.
- Multiplicación (*): se utiliza para multiplicar dos valores.
- División (/): se utiliza para dividir un valor por otro.
- Módulo (%): se utiliza para obtener el resto de una división.

Operadores de asignación

Los operadores de asignación se utilizan para asignar un valor a una variable. El operador de asignación en Java es el signo igual (=). También existen operadores de asignación compuestos, que realizan una operación y luego asignan el resultado a la variable. Algunos ejemplos son:

- +=: suma el valor de la variable y el valor especificado y asigna el resultado a la variable.
- -=: resta el valor especificado de la variable y asigna el resultado a la variable.
- *=: multiplica el valor de la variable y el valor especificado y asigna el resultado a la variable.
- /=: divide el valor de la variable por el valor especificado y asigna el resultado a la variable.
- %=: obtiene el resto de la división entre la variable y el valor especificado y asigna el resultado a la variable.

Operadores de comparación

Los operadores de comparación se utilizan para comparar dos valores y devuelven un valor booleano (verdadero o falso) como resultado. Los operadores de comparación en Java son los siguientes:

- Igual que (==): devuelve verdadero si los dos valores son iguales.
- Distinto que (!=): devuelve verdadero si los dos valores son distintos.
- Mayor que (>): devuelve verdadero si el primer valor es mayor que el segundo valor.

- Menor que (<): devuelve verdadero si el primer valor es menor que el segundo valor.
- Mayor o igual que (>=): devuelve verdadero si el primer valor es mayor o igual que el segundo valor.
- Menor o igual que (<=): devuelve verdadero si el primer valor es menor o igual que el segundo valor.

Operadores lógicos

Los operadores lógicos son aquellos que se utilizan para realizar operaciones booleanas entre dos expresiones, cuyo resultado será verdadero o falso. Los operadores lógicos en Java son los siguientes:

- **AND lógico (&&)**: Este operador devuelve verdadero si ambas expresiones son verdaderas.
- **OR lógico (||)**: Este operador devuelve verdadero si al menos una de las expresiones es verdadera.
- **NOT lógico (!)**: Este operador se utiliza para negar el resultado de una expresión booleana. Si la expresión original es verdadera, el resultado será falso, y viceversa.

A continuación, se muestran algunos ejemplos de uso de operadores lógicos:

```
boolean a = true;
boolean b = false;

boolean c = a && b; // false

boolean d = a || b; // true

boolean e = !a; // false
```

Podemos hacer un ejemplo sencillo que combine varios conceptos que hemos visto. Por ejemplo, podemos escribir un programa que le pregunte al usuario su edad y, dependiendo de si es mayor o menor de edad, le dé la bienvenida o le indique que debe esperar un poco más para poder acceder al contenido.

Vamos a crear un programa que permita al usuario ingresar su edad y verifique si es mayor de edad o no. Para ello, utilizaremos los conceptos vistos anteriormente.

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    System.out.print("Ingrese su edad: ");
    int edad = input.nextInt();

    boolean esMayorDeEdad = edad >= 18;

```

```
    System.out.println("Usted tiene " + edad + " a~nos.");
    System.out.println("Es mayor de edad?: " + esMayorDeEdad);
}
}
```

En este programa, utilizamos la clase `Scanner` para leer la entrada del usuario desde la consola. Luego, guardamos la edad ingresada en una variable de tipo `int` llamada `edad`.

Después, utilizamos una variable de tipo `boolean` llamada `esMayorDeEdad` para verificar si la edad ingresada es mayor o igual a 18 años.

Finalmente, imprimimos la edad ingresada y si el usuario es mayor de edad o no utilizando la función `println` de la clase `System` y concatenando las variables correspondientes.

1.3.1. Operaciones matemáticas con Math

En Java, la clase Math proporciona una serie de métodos estáticos para realizar operaciones matemáticas. A continuación se muestran algunos ejemplos:

- "**max**": El método "max" de Math toma dos números como argumentos y devuelve el mayor de los dos. Por ejemplo:

```
    int a = 10;
    int b = 20;
    int maximo = Math.max(a, b); // devuelve 20

```

- "**min**": El método "min" de Math toma dos números como argumentos y devuelve el menor de los dos. Por ejemplo:

```
int a = 10;
int b = 20;
int minimo = Math.min(a, b); // devuelve 10
```

- "**abs**": El método "abs" de Math devuelve el valor absoluto de un número. Por ejemplo:

```
int a = -10;
int valorAbsoluto = Math.abs(a); // devuelve 10
```

- "**sqrt**": El método "sqrt" de Math devuelve la raíz cuadrada de un número. Por ejemplo:

```
int a = 25;
double raizCuadrada = Math.sqrt(a); // devuelve 5.0
```

- "**pow**": El método "pow" de Math toma dos números como argumentos y devuelve el resultado de elevar el primer número a la potencia del segundo número. Por ejemplo:

```
int base = 2;
int exponente = 3;
double potencia = Math.pow(base, exponente); // devuelve 8.0
```

- "**sin, cos, tan**": Los métodos "sin", "cos" y "tan" de Math calculan el seno, coseno y tangente de un ángulo en radianes, respectivamente. Por ejemplo:

```
double angulo = Math.PI / 4; // angulo de 45 grados en radianes
double seno = Math.sin(angulo); // devuelve 0.7071067811865475
double coseno = Math.cos(angulo); // devuelve 0.7071067811865476
double tangente = Math.tan(angulo); // devuelve 0.9999999999999999
```

Es importante tener en cuenta que los métodos trigonométricos de Math toman como argumento un ángulo en radianes, no en grados. Para convertir de grados a radianes, se puede utilizar la siguiente fórmula:

$$\theta_{rad} = \frac{\theta_{grados} \times \pi}{180}$$

donde θ_{rad} es el ángulo en radianes y θ_{grados} es el ángulo en grados.

Por ejemplo, si se quisiera calcular el seno de 45 grados utilizando el método sin() de Math, se debería convertir primero a radianes utilizando la fórmula anterior:

$$45_{grados} = \frac{\pi}{4}_{radianes}$$

Entonces, el código para calcular el seno de 45 grados utilizando el método sin() sería el siguiente:

```
double anguloEnGrados = 45;
double anguloEnRadianes = anguloEnGrados * Math.PI / 180;
double seno = Math.sin(anguloEnRadianes);
System.out.println("El seno de " + anguloEnGrados + " grados es: " + seno);
```

El resultado sería: `El seno de 45.0 grados es: 0.7071067811865475`

1.4. Casting en Variables

El casting en Java es la conversión explícita de una variable de un tipo de dato a otro. Esto se hace poniendo entre paréntesis el tipo de dato al que se desea convertir la variable, seguido del nombre de la variable.

1.4.1. Casting Manual

En Java, algunos tipos de datos se pueden convertir manualmente en otros tipos de datos. Un ejemplo común es la conversión de un `double` a un `int`. Para hacer esto, simplemente se escribe el tipo de dato al que se desea convertir la variable entre paréntesis, seguido del nombre de la variable:

```
double d = 10.5;
int i = (int) d;
System.out.println(i); // salida: 10
```

También se puede hacer lo contrario, es decir, convertir un `int` a un `double`:

```
int i = 10;
double d = (double) i;
System.out.println(d); // salida: 10.0
```

1.4.2. Casting Automático

En algunos casos, Java realiza la conversión de un tipo de dato a otro de forma automática. Esto se conoce como casting automático y sucede cuando se asigna una variable de un tipo de dato a una variable de otro tipo de dato compatible. Por ejemplo, cuando se asigna una variable `int` a una variable `double`, Java convierte automáticamente el `int` a un `double`:

```
int i = 10;
double d = i;
System.out.println(d); // salida: 10.0
```

Sin embargo, no todos los tipos de datos son compatibles entre sí. En estos casos, es necesario hacer un casting manual para convertir la variable.

1.5. Casting en otros tipos de datos

El casting manual también se puede utilizar en otros tipos de datos, como en el caso de los tipos de datos `char` y `boolean`. Por ejemplo, para convertir un `char` a un `int`, se puede hacer lo siguiente:

```
char c = 'a';
int i = (int) c;
System.out.println(i); // salida: 97
```

Y para convertir un `boolean` a un `int`, se puede hacer lo siguiente:

```
boolean b = true;
int i = b ? 1 : 0;
System.out.println(i); // salida: 1
```

Es importante tener en cuenta que los tipos de datos deben ser compatibles entre sí para poder hacer un casting automático. En caso contrario, es necesario hacer un casting manual para convertir la variable al tipo de dato deseado.

1.6. Versiones de Java y JDK

Java es un lenguaje de programación popular utilizado en una variedad de aplicaciones, desde aplicaciones web hasta aplicaciones móviles. A lo largo de los años, ha habido varias versiones de Java, cada una con sus propias características y mejoras.

El Java Development Kit (JDK) es un kit de herramientas de desarrollo para Java que incluye el compilador Java, la biblioteca de clases Java y otras herramientas de desarrollo. El JDK es necesario para desarrollar y ejecutar programas en Java.

A continuación, se muestran algunas de las versiones de Java y JDK más populares:

1.6.1. Java SE 8

Java SE 8 es una de las versiones más populares de Java. Fue lanzada en marzo de 2014 y es conocida por sus mejoras en el rendimiento y la seguridad. Java SE 8 también introdujo la programación funcional en Java con la adición de las expresiones lambda.

1.6.2. Java SE 11

Java SE 11 es otra versión popular de Java. Fue lanzada en septiembre de 2018 y es conocida por sus mejoras en la seguridad y la estabilidad. Java SE 11 también introdujo la noción de lanzamientos a largo plazo (LTS), que son lanzamientos que reciben soporte a largo plazo.

1.6.3. Java SE 16

Java SE 16 es la versión más reciente de Java en el momento de escribir esto. Fue lanzada en marzo de 2021 y es conocida por sus mejoras en la programación funcional y en la seguridad.

Es importante tener en cuenta que las diferentes versiones de Java pueden tener diferentes funcionalidades y características. Al seleccionar una versión de Java, es importante considerar las necesidades del proyecto y las características específicas de cada versión.

Además, es importante tener en cuenta que la elección del JDK también es importante para el desarrollo de programas en Java. Es recomendable utilizar la última versión del JDK para aprovechar al máximo las características y mejoras de Java.

1.6.4. Archivos .jar

Un archivo JAR (Java ARchive) es un archivo comprimido que contiene código y recursos que puede ser utilizado por una máquina virtual de Java. Estos archivos suelen utilizarse para distribuir bibliotecas de clases reutilizables o aplicaciones completas escritas en Java.

Para crear un archivo .jar desde la línea de comandos, se utiliza el comando `jar` proporcionado por el JDK de Java. La sintaxis básica del comando es la siguiente:

```
jar cf jar-file input-file(s)
```

Donde `jar-file` es el nombre del archivo .jar que se desea crear, y `input-file(s)` es la lista de archivos que se desean incluir en el archivo .jar. Por ejemplo, para crear un archivo .jar que contenga todos los archivos .class en el directorio actual, se puede utilizar el siguiente comando:

```
jar cf mylibrary.jar *.class
```

Para utilizar una biblioteca JAR en un proyecto de Java, primero se debe agregar la biblioteca al classpath del proyecto. Esto se puede hacer de varias formas, dependiendo del entorno de desarrollo utilizado. En Eclipse, por ejemplo, se puede agregar una biblioteca JAR al classpath de un proyecto haciendo clic derecho en el proyecto, seleccionando “Propiedades”, y luego

seleccionando “Java Build Path” en el menú de la izquierda. A continuación, se debe hacer clic en la pestaña “Librerías”, y luego en el botón “Agregar JARs...” para seleccionar el archivo .jar que se desea agregar.

Una vez que se ha agregado la biblioteca al classpath del proyecto, se pueden importar las clases y utilizarlas en el código de la misma manera que se importan y utilizan las clases de la biblioteca estándar de Java.

Es importante tener en cuenta que los archivos JAR pueden contener código malicioso o peligroso. Por esta razón, se debe tener precaución al descargar y utilizar archivos JAR de fuentes desconocidas.

2. Ciclos en Java

Los ciclos son estructuras de control que permiten repetir una o varias instrucciones mientras se cumpla una condición. En Java, existen varios tipos de ciclos, cada uno con su propia sintaxis y uso específico.

2.1. Ciclo condicional: `if`

El ciclo condicional `if` permite ejecutar una o varias instrucciones si se cumple una condición determinada. Su sintaxis es la siguiente:

```
if (condicion) {
// Instrucciones a ejecutar si se cumple la condicion
}
```

Si la condición es verdadera, se ejecutan las instrucciones dentro del bloque de código entre llaves. Si la condición es falsa, se omite la ejecución de las instrucciones y se continúa con la siguiente línea de código. También se puede utilizar la instrucción `else` para especificar un bloque de código a ejecutar en caso de que la condición sea falsa:

```
if (condicion) {
// Instrucciones a ejecutar si se cumple la condicion
} else {
// Instrucciones a ejecutar si NO se cumple la condicion
}
```

2.2. Ciclo repetitivo: `for`

El ciclo repetitivo `for` permite ejecutar una o varias instrucciones un número determinado de veces. Su sintaxis es la siguiente:

```
for (inicializacion; condicion; incremento) {
// Instrucciones a ejecutar en cada iteracion
}
```

La `inicializacion` define una variable de control y su valor inicial. La `condicion` define la condición que debe cumplirse para continuar ejecutando el ciclo. El `incremento` se ejecuta después de cada iteración y permite modificar el valor de la variable

de control. Si la condición es verdadera, se ejecutan las instrucciones dentro del bloque de código entre llaves. Después de cada iteración, se evalúa la condición nuevamente. Si la condición es falsa, se sale del ciclo.

2.3. Ciclo repetitivo: `while`

El ciclo repetitivo **`while`** permite ejecutar una o varias instrucciones mientras se cumpla una condición determinada. Su sintaxis es la siguiente:

```
while (condicion) {
// Instrucciones a ejecutar mientras se cumpla la condicion
}
```

Si la condición es verdadera, se ejecutan las instrucciones dentro del bloque de código entre llaves. Después de cada ejecución, se evalúa la condición nuevamente. Si la condición es falsa, se sale del ciclo.

2.4. Ciclo repetitivo: `do-while`

El ciclo repetitivo **`do-while`** es similar al ciclo **`while`**, pero la evaluación de la condición se realiza después de la primera ejecución del bloque de código. Su sintaxis es la siguiente:

```
do {
// bloque de codigo
} while (condicion);
```

En este caso, el bloque de código se ejecuta al menos una vez, independientemente de si la condición es verdadera o falsa en la primera evaluación. Después de la primera ejecución, la condición se evalúa nuevamente. Si es verdadera, el bloque de código se ejecuta de nuevo y así sucesivamente, hasta que la condición se vuelva falsa.

2.5. Ejemplo practico

Te dejo un ejemplo de método de análisis numérico de bisección implementado en Java:

```
public class BisectionMethod {

    public static double findRoot(double a, double b, double epsilon) {
        double mid = (a + b) / 2.0;
        while (Math.abs(a - b) > epsilon) {
            if (function(mid) == 0) {
                return mid;
            }
            if (function(a) * function(mid) < 0) {
                b = mid;
            } else {
                a = mid;
```

```
            }
            mid = (a + b) / 2.0;
        }
        return mid;
    }

    public static double function(double x) {
        // Define la funcion cuya raiz se quiere encontrar
        return Math.pow(x,2)-2;
    }

    public static void main(String[] args) {
        double a = 0;
        double b = 2;
        double epsilon = 0.001;
        double root = findRoot(a, b, epsilon);
        System.out.println("La raiz de la funcion es: " + root);
    }
}
```

Este programa calcula la raíz cuadrada de 2 usando el método de bisección. La función `f` define la función que se va a evaluar, en este caso $f(x) = x^2 - 2$. El método `bisection` toma como entrada los límites del intervalo, una tolerancia y el número máximo de iteraciones, y devuelve la aproximación de la raíz. El programa principal simplemente llama al método `bisection` con los parámetros deseados y muestra el resultado.

3. bibliografía

Referencias

[1] Martin, Robert C. *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice Hall, 2008.

[2] Platzi. *Curso de Java Básico*. Platzi, 2023. https://platzi.com/cursos/java-basico/

[3] Platzi. *Curso de Java Orientado a Objetos*. Platzi, 2023. https://platzi.com/cursos/java-poo/

[4] Oracle Corporation. *Java SE Documentation*. Oracle Corporation, 2023. https://docs.oracle.com/en/java/javase/index.html