

FUNDAMENTOS DE JAVA

Manual Tecnológico

Version 1.0

Presentación

Java es uno de los lenguajes de programación más importantes en la historia de la informática moderna. Desde su aparición, ha desempeñado un papel fundamental en el desarrollo de aplicaciones empresariales, sistemas de información, plataformas web, aplicaciones móviles y soluciones tecnológicas de diferentes dimensiones.

Su arquitectura, basada en la máquina virtual de Java (JVM), permite ejecutar programas en distintos sistemas operativos sin necesidad de modificar el código fuente para cada plataforma. Esta característica contribuyó a su popularidad y consolidación dentro de la industria del software.

El presente manual tecnológico, **Fundamentos de Java**, tiene como finalidad introducir al lector en los principios esenciales de este lenguaje de programación. A través de explicaciones progresivas, ejemplos prácticos y fundamentos conceptuales, se desarrollan los conocimientos necesarios para comprender la estructura de un programa Java y comenzar a construir aplicaciones.

El manual aborda desde los conceptos iniciales de programación hasta los principios fundamentales de la programación orientada a objetos, incluyendo variables, tipos de datos, operadores, estructuras de control, métodos, clases, objetos, excepciones y colecciones.

Objetivo general

Proporcionar los conocimientos fundamentales del lenguaje de programación Java, permitiendo comprender su funcionamiento, sintaxis, estructura y principales características para desarrollar programas básicos aplicando los principios de la programación estructurada y orientada a objetos.

Contenido

1. Introducción a Java
2. Entorno de desarrollo Java
3. Estructura de un programa Java
4. Variables y tipos de datos
5. Operadores y expresiones
6. Estructuras de control
7. Arreglos y matrices
8. Métodos y funciones
9. Programación orientada a objetos
10. Herencia y polimorfismo
11. Manejo de excepciones
12. Colecciones en Java
13. Entrada y salida de datos
14. Desarrollo de una aplicación básica
15. Buenas prácticas de programación
16. Glosario técnico

1. Introducción a Java

1.1. ¿Qué es Java?

Java es un lenguaje de programación de propósito general, orientado a objetos, diseñado para desarrollar aplicaciones que puedan ejecutarse en diferentes plataformas informáticas.

Fue desarrollado originalmente por James Gosling y su equipo en Sun Microsystems. Su primera versión pública apareció en 1995.

Java se caracteriza por ofrecer una sintaxis estructurada, mecanismos de administración de memoria, seguridad y compatibilidad entre diferentes sistemas operativos mediante la máquina virtual de Java.

Entre sus principales características se encuentran:

- **Orientado a objetos:** permite organizar los programas mediante clases y objetos.
- **Multiplataforma:** permite ejecutar aplicaciones en distintos sistemas operativos mediante una JVM compatible.
- **Robusto:** incorpora mecanismos para controlar errores y administrar recursos.
- **Seguro:** dispone de mecanismos de seguridad durante la ejecución.
- **Multihilo:** permite ejecutar diferentes tareas concurrentemente.
- **Portátil:** utiliza un formato de código intermedio conocido como bytecode.
- **Amplia biblioteca:** proporciona numerosas clases y herramientas para desarrollar aplicaciones.

1.2. Historia y evolución

Java surgió como resultado de un proyecto de investigación iniciado en 1991, conocido inicialmente como Green Project.

Su desarrollo estuvo relacionado con la necesidad de crear software capaz de funcionar en diferentes dispositivos y plataformas. Algunos acontecimientos importantes:

Año	Acontecimiento
1991	Inicio del proyecto Green en Sun Microsystems.
1995	Presentación pública de Java.
1996	Publicación de JDK 1.0.
2004	Introducción de Java 5 y mejoras importantes del lenguaje.
2006	Publicación de OpenJDK como proyecto de código abierto.
2010	Oracle adquiere Sun Microsystems.
2014	Java 8 incorpora expresiones lambda y Stream API.
2017	Java 9 introduce el sistema de módulos.
2021	Java 17 se establece como versión LTS.
2023	Java 21 incorpora importantes mejoras del lenguaje.

Java continúa evolucionando mediante nuevas versiones y actualizaciones.

1.3. Aplicaciones de Java

Java se utiliza en numerosos sectores tecnológicos:

- Desarrollo de aplicaciones empresariales.
- Sistemas de información y gestión administrativa.
- Aplicaciones web y servicios backend.
- Desarrollo de aplicaciones Android.
- Sistemas bancarios y financieros.
- Procesamiento de datos.
- Aplicaciones distribuidas.
- Servicios en la nube.

2. Entorno de desarrollo Java

2.1. ¿Qué es el JDK?

El Java Development Kit (JDK) es el conjunto de herramientas necesario para desarrollar aplicaciones Java.

Incluye el compilador, herramientas de desarrollo y componentes necesarios para ejecutar programas.

Entre sus herramientas principales se encuentran:

Herramienta	Función
javac	Compila el código fuente Java.
java	Ejecuta aplicaciones Java.
javadoc	Genera documentación técnica.
jar	Empaqueta clases y recursos.
jshell	Permite ejecutar instrucciones Java interactivamente.

2.2. JVM, JRE y JDK

Estos tres conceptos son fundamentales para comprender la arquitectura de Java.

JVM (Java Virtual Machine): máquina virtual encargada de ejecutar el bytecode generado durante la compilación.

JRE (Java Runtime Environment): concepto tradicional que representa el entorno necesario para ejecutar aplicaciones Java, incluyendo la JVM y las bibliotecas correspondientes.

JDK (Java Development Kit): entorno de desarrollo que incorpora las herramientas necesarias para crear, compilar y ejecutar programas.

2.3. Entornos de desarrollo integrado (IDE)

Un IDE es una aplicación que facilita la escritura, organización, depuración y ejecución de programas.

Algunos entornos utilizados para desarrollar aplicaciones Java son:

- IntelliJ IDEA.
- Eclipse IDE.
- Apache NetBeans.
- Visual Studio Code, mediante extensiones para Java.

3. Estructura de un programa Java

3.1. Primer programa

El primer paso para aprender Java consiste en crear un programa sencillo que muestre un mensaje en pantalla.

Ejemplo:

```
public class HolaMundo {

    public static void main(String[] args) {

        System.out.println("Hola, mundo Java!");

    }

}
```

Explicación del código

- `public`: establece que la clase o método es accesible desde otros componentes.
- `class`: palabra reservada para declarar una clase.
- `HolaMundo`: nombre de la clase.
- `main`: método de entrada convencional de una aplicación Java.
- `String[] args`: recibe argumentos enviados al programa.
- `System.out.println()`: muestra información en la consola.

3.2. Compilación y ejecución

Java utiliza un proceso de compilación y ejecución.

1. El programador escribe el código fuente en un archivo `.java`.
2. El compilador `javac` transforma el código fuente en bytecode.
3. El bytecode se almacena en un archivo `.class`.
4. La JVM interpreta o compila el bytecode para ejecutarlo en la plataforma correspondiente.

Ejemplo desde la terminal:

```
javac HolaMundo.java

java HolaMundo
```

Resultado:

```
Hola, mundo Java!
```

3.3. Reglas básicas de sintaxis

Java posee reglas sintácticas que deben respetarse:

- Las instrucciones normalmente terminan con punto y coma (`;`).
- Las clases y los métodos utilizan llaves `{ }`.
- El lenguaje distingue entre letras mayúsculas y minúsculas.
- Los nombres de las clases deben respetar las reglas de identificación.
- Los comentarios permiten documentar el código.

Tipos de comentarios:

```
// Comentario de una sola línea

/*
   Comentario
   de varias líneas
*/

/**
 * Comentario de documentación.
 */
```

4. Variables y tipos de datos

4.1. ¿Qué es una variable?

Una variable es un espacio identificado por un nombre que permite almacenar un valor durante la ejecución de un programa.

En Java, cada variable tiene un tipo de dato que determina qué clase de información puede almacenar.

Ejemplo:

```
int edad = 25;

double salario = 2500.50;

String nombre = "Franklin";

boolean activo = true;
```

4.2. Tipos de datos primitivos

Java dispone de ocho tipos de datos primitivos.

Tipo	Descripción	Ejemplo
byte	Entero de 8 bits	100
short	Entero de 16 bits	1500
int	Entero de 32 bits	25000
long	Entero de 64 bits	900000L
float	Decimal de 32 bits	12.5f
double	Decimal de 64 bits	125.75
char	Carácter Unicode de 16 bits	'A'
boolean	Valor lógico	true

4.3. Tipos de referencia

Además de los tipos primitivos, Java permite trabajar con tipos de referencia.

Entre ellos se encuentran:

- String.
- Arrays.
- Clases.
- Objetos.
- Interfaces.
- Enumeraciones.

Ejemplo:

```
String ciudad = "Londres";
```

4.4. Constantes

Una constante es una variable cuyo valor no puede modificarse después de su inicialización.

Se utiliza la palabra reservada `final`.

```
final double PI = 3.14159;
```

```
final int ANIO = 2026;
```

5. Operadores y expresiones

Los operadores permiten realizar operaciones matemáticas, comparaciones y evaluaciones lógicas.

5.1. Operadores aritméticos

Operador	Operación	Ejemplo
+	Suma	a + b
-	Resta	a - b
*	Multiplicación	a * b
/	División	a / b
%	Módulo o residuo	a % b

Ejemplo:

```
int a = 20;

int b = 5;

System.out.println(a + b);

System.out.println(a - b);

System.out.println(a * b);

System.out.println(a / b);
```

5.2. Operadores relacionales

Permiten comparar valores.

Operador	Significado
==	Igual a
!=	Diferente de
>	Mayor que
<	Menor que
>=	Mayor o igual que
<=	Menor o igual que

5.3. Operadores lógicos

Operador	Significado
&&	AND lógico
||	OR lógico
!	NOT lógico

Ejemplo:

```
int edad = 25;


if (edad >= 18 && edad <= 65) {
    System.out.println("Edad dentro del intervalo.");
}
```

6. Estructuras de control

Las estructuras de control permiten modificar el orden de ejecución de las instrucciones.

6.1. Estructura condicional if

Se utiliza para ejecutar instrucciones cuando se cumple una condición.

```
int edad = 20;


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

6.2. Estructura switch

Permite seleccionar una alternativa entre diferentes posibilidades.

```
int opcion = 2;


switch (opcion) {
    case 1:
        System.out.println("Nuevo registro");
        break;

    case 2:
        System.out.println("Consultar registro");
        break;
```

```
    case 3:
        System.out.println("Eliminar registro");
        break;

    default:
        System.out.println("Opción no válida");
}
```

6.3. Ciclo for

Permite repetir instrucciones un número determinado de veces.

```
for (int i = 1; i <= 5; i++) {
    System.out.println("Número: " + i);
}
```

6.4. Ciclo while

Repite instrucciones mientras una condición sea verdadera.

```
int contador = 1;

while (contador <= 5) {

    System.out.println(contador);

    contador++;
}
```

6.5. Ciclo do-while

Ejecuta las instrucciones al menos una vez y posteriormente evalúa la condición.

```
int numero = 1;

do {
    System.out.println(numero);
    numero++;
} while (numero <= 5);
```

7. Arreglos y matrices

7.1. Arreglos

Un arreglo (array) es una estructura de datos que permite almacenar varios elementos del mismo tipo bajo un único identificador.

Ejemplo:

```
int[] numeros = {10, 20, 30, 40, 50};


System.out.println(numeros[0]);
```

El resultado será:

```
10
```

Los índices de los arreglos comienzan en cero.

7.2. Recorrer un arreglo

```
int[] numeros = {10, 20, 30, 40, 50};


for (int i = 0; i < numeros.length; i++) {


    System.out.println(numeros[i]);


}
```

7.3. Matrices

Una matriz es una estructura bidimensional que permite organizar datos en filas y columnas.

Ejemplo:

```
int[][] matriz = {
    {1, 2, 3},
    {4, 5, 6},
    {7, 8, 9}
};
System.out.println(matriz[1][2]);
```

Resultado: 6

8. Métodos y funciones

8.1. Concepto de método

Un método es un bloque de código que realiza una tarea específica y puede ejecutarse cuando es invocado.

Los métodos permiten dividir un programa en unidades funcionales, mejorar la organización y evitar la repetición innecesaria de instrucciones.

Ejemplo:

```
public static void saludar() {

    System.out.println("Bienvenido a Java");

}
```

Para ejecutar el método:

```
saludar();
```

8.2. Métodos con parámetros

Los parámetros permiten enviar información a un método.

```
public static void mostrarNombre(String nombre) {

    System.out.println("Nombre: " + nombre);

}
```

Invocación:

```
mostrarNombre("Franklin");
```

8.3. Métodos con retorno

Un método puede devolver un resultado mediante la instrucción `return`.

```
public static int sumar(int a, int b) {

    return a + b;

}
```

Invocación:

```
int resultado = sumar(10, 20);

System.out.println(resultado);
```

9. Programación orientada a objetos

La programación orientada a objetos (POO) es un paradigma que organiza el software mediante entidades denominadas objetos.

Java utiliza este paradigma como uno de sus fundamentos principales.

9.1. Clases y objetos

Una clase es una estructura que define atributos y comportamientos.

Un objeto es una instancia concreta de una clase.

Ejemplo:

```
public class Persona {

    String nombre;

    int edad;

    public void presentar() {

        System.out.println("Mi nombre es " + nombre);

        System.out.println("Mi edad es " + edad);

    }

}
```

Creación de un objeto:

```
Persona persona1 = new Persona();

persona1.nombre = "Franklin";

persona1.edad = 30;

persona1.presentar();
```

9.2. Atributos y métodos

Los atributos representan las características o propiedades de un objeto.

Los métodos representan las acciones o comportamientos que puede realizar.

En una clase denominada Automovil, por ejemplo:

Atributos:

- Marca.
- Modelo.
- Año.
- Color.

Métodos:

- Arrancar.
- Acelerar.
- Frenar.
- Apagar.

9.3. Encapsulamiento

El encapsulamiento consiste en proteger los datos internos de un objeto y controlar el acceso a ellos.

Ejemplo:

```
public class CuentaBancaria {

    private double saldo;

    public void depositar(double cantidad) {

        if (cantidad > 0) {

            saldo += cantidad;

        }

    }

    public double consultarSaldo() {

        return saldo;

    }

}
```

El modificador `private` impide acceder directamente al atributo desde otras clases.

9.4. Constructores

Un constructor es un componente especial de una clase que se ejecuta cuando se crea un objeto.

```
public class Producto {


    String nombre;

    double precio;

    public Producto(String nombre, double precio) {

        this.nombre = nombre;

        this.precio = precio;

    }

}
```

Creación del objeto:

```
Producto producto1 = new Producto("Teclado", 25.50);
```

10. Herencia y polimorfismo

10.1. Herencia

La herencia permite crear una clase nueva a partir de otra existente.

La clase derivada puede utilizar los atributos y métodos accesibles de la clase base.

Ejemplo:

```
public class Animal {

    public void comer() {

        System.out.println("El animal está comiendo.");

    }

}
```

Clase derivada:

```
public class Perro extends Animal {

    public void ladrar() {

       System.out.println("Guau, guau!");

    }

}
```

Uso:

```
Perro miPerro = new Perro();

miPerro.comer();

miPerro.ladrar();
```

10.2. Polimorfismo

El polimorfismo permite que diferentes objetos respondan de manera distinta a una misma operación o método.

Ejemplo:

```
class Animal {

    public void sonido() {

        System.out.println("Sonido de animal");

    }

}


class Perro extends Animal {

    @Override

    public void sonido() {

        System.out.println("Guau");

    }

}


class Gato extends Animal {

    @Override

    public void sonido() {

        System.out.println("Miau");

   }

}
```

Cada clase proporciona una implementación particular del método `sonido()`.

11. Manejo de excepciones

11.1. ¿Qué es una excepción?

Una excepción es un evento que ocurre durante la ejecución de un programa y que altera el flujo normal de sus instrucciones.

Java dispone de mecanismos para detectar y gestionar estas situaciones.

Algunos ejemplos:

- División entre cero.
- Acceso a una posición inexistente de un arreglo.
- Conversión incorrecta de datos.
- Problemas durante operaciones de entrada y salida.

11.2. Bloques try y catch

El bloque `try` contiene instrucciones que pueden generar excepciones.

El bloque `catch` permite capturar y gestionar los errores correspondientes.

Ejemplo:

```
public class Division {
    public static void main(String[] args) {
        try {
            int resultado = 10 / 0;
            System.out.println(resultado);
        } catch (ArithmeticException e) {
            System.out.println("Error: división entre cero.");
        }
    }
}
```

11.3. Bloque finally

El bloque `finally` permite ejecutar instrucciones independientemente de que ocurra o no una excepción, salvo situaciones excepcionales en las que la ejecución de la JVM termina abruptamente.

12. Colecciones en Java

Las colecciones permiten almacenar y administrar grupos de objetos de manera flexible.

Java proporciona el framework Collections, que incluye diferentes estructuras de datos.

12.1. ArrayList

ArrayList es una colección que permite almacenar elementos en una lista dinámica.

Ejemplo:

```
import java.util.ArrayList;
public class ListaEjemplo {
    public static void main(String[] args) {
        ArrayList<String> nombres = new ArrayList<>();
        nombres.add("Ana");
        nombres.add("Pedro");
```

```
        nombres.add("Carlos");
        System.out.println(nombres);
    }
}
```

12.2. HashSet

HashSet permite almacenar elementos únicos, evitando duplicados.

12.3. HashMap

HashMap almacena información mediante pares de clave y valor.

Ejemplo:

```
import java.util.HashMap;
public class EjemploMapa {
    public static void main(String[] args) {
        HashMap<Integer, String> estudiantes = new HashMap<>();
        estudiantes.put(1, "Ana");
        estudiantes.put(2, "Pedro");
        estudiantes.put(3, "Carlos");
        System.out.println(estudiantes.get(2));
    }
}
```

13. Entrada y salida de datos

La entrada y salida de datos permite que los programas reciban información y presenten resultados.

13.1. Entrada de datos mediante Scanner

Java dispone de la clase Scanner para leer información introducida por el usuario.

Ejemplo:

```
import java.util.Scanner;
public class EntradaDatos {
    public static void main(String[] args) {
        Scanner entrada = new Scanner(System.in);
        System.out.print("Introduzca su nombre: ");
```

```
        String nombre = entrada.nextLine();
        System.out.println("Bienvenido, " + nombre);
        entrada.close();

    }
}
```

13.2. Salida de información

Las instrucciones más utilizadas son:

```
System.out.print("Mensaje");
System.out.println("Mensaje con salto de línea");
System.out.printf("Edad: %d%n", 25);
```

14. Desarrollo de una aplicación básica

En este capítulo se integran algunos de los conocimientos estudiados mediante el desarrollo de un pequeño programa de gestión de estudiantes.

14.1. Descripción del proyecto

Se desarrollará una aplicación de consola que permita registrar información básica de un estudiante y mostrar sus datos.

La aplicación utilizará:

- Variables.
- Tipos de datos.
- Entrada de información.
- Salida de resultados.
- Métodos.

14.2. Código fuente

```
import java.util.Scanner;
public class GestionEstudiante {
    public static void main(String[] args) {
        Scanner entrada = new Scanner(System.in);
        System.out.print("Nombre del estudiante: ");
        String nombre = entrada.nextLine();
```

```
        System.out.print("Edad: ");
        int edad = entrada.nextInt();

        System.out.print("Calificación: ");
        double nota = entrada.nextDouble();

        mostrarDatos(nombre, edad, nota);

        entrada.close();

    }

    public static void mostrarDatos(
            String nombre, int edad, double nota) {

        System.out.println("\n--- DATOS DEL ESTUDIANTE ---");

        System.out.println("Nombre: " + nombre);
        System.out.println("Edad: " + edad);
        System.out.println("Calificación: " + nota);

        if (nota >= 60) {

            System.out.println("Estado: Aprobado");

        } else {

            System.out.println("Estado: Reprobado");

        }
```

```
    }
}
```

Nota: El ejemplo utiliza una escala académica de 0 a 100 y establece 60 como calificación mínima aprobatoria.

14.3. Análisis del programa

El programa solicita tres datos al usuario: nombre, edad y calificación.

Posteriormente, utiliza el método `mostrarDatos()` para presentar la información y una estructura condicional para determinar el estado académico del estudiante.

Este ejercicio permite comprender cómo se integran diferentes elementos del lenguaje Java en una aplicación sencilla.

15. Buenas prácticas de programación

La calidad de un programa no depende únicamente de que funcione correctamente. También es importante que su código sea comprensible, organizado, mantenible y eficiente.

Algunas recomendaciones fundamentales son:

1. Utilizar nombres descriptivos para variables, métodos y clases.
2. Aplicar correctamente la indentación.
3. Dividir los programas en métodos y clases.
4. Evitar la repetición innecesaria de código.
5. Validar los datos introducidos por el usuario.
6. Gestionar adecuadamente las excepciones.
7. Documentar las partes importantes del programa.
8. Aplicar los principios de encapsulamiento.
9. Evitar utilizar variables globales innecesarias.
10. Probar los programas antes de distribuirlos.

15.1. Convenciones de nomenclatura

Java utiliza convenciones ampliamente aceptadas para nombrar los diferentes componentes de un programa.

Elemento	Convención	Ejemplo
Clases	PascalCase	GestionEstudiante
Métodos	camelCase	calcularPromedio()
Variables	camelCase	nombreEstudiante
Constantes	MAYÚSCULAS	VALOR_MAXIMO
Paquetes	Minúsculas	com.bitcomputacional

16. Glosario técnico

Algoritmo: conjunto ordenado y finito de instrucciones para resolver un problema.

Atributo: característica o propiedad de una clase u objeto.

Bytecode: código intermedio generado por el compilador Java.

Clase: estructura que define atributos y métodos.

Compilador: herramienta que transforma el código fuente en código ejecutable o intermedio.

Constructor: componente especial utilizado para inicializar objetos.

Encapsulamiento: mecanismo que protege los datos internos de una clase.

Excepción: evento que altera el flujo normal de ejecución.

Herencia: mecanismo que permite crear clases derivadas de otras clases.

IDE: entorno de desarrollo integrado.

JDK: conjunto de herramientas para desarrollar aplicaciones Java.

JVM: máquina virtual que ejecuta el bytecode.

Método: bloque de instrucciones que realiza una tarea específica.

Objeto: instancia de una clase.

Polimorfismo: capacidad de utilizar una misma interfaz o método con diferentes implementaciones.

Variable: espacio identificado que almacena un valor.

Conclusiones

Java constituye una herramienta tecnológica de gran importancia para el desarrollo de software. Su arquitectura, características de programación orientada a objetos y capacidad de ejecución multiplataforma lo convierten en un lenguaje apropiado para la formación de nuevos programadores y para la construcción de aplicaciones profesionales.

El aprendizaje de sus fundamentos permite comprender conceptos esenciales de las ciencias de la computación, tales como algoritmos, estructuras de datos, abstracción, encapsulamiento, herencia y polimorfismo.

Los conocimientos presentados en este manual representan una base para continuar estudiando temas más avanzados, como programación de interfaces gráficas, bases de datos, desarrollo web, aplicaciones empresariales y programación distribuida.

El dominio de Java no se alcanza únicamente mediante el estudio de su sintaxis. Requiere práctica constante, análisis de problemas y desarrollo progresivo de soluciones informáticas.

Referencias bibliográficas

- Bloch, J. (2018). *Effective Java* (3rd ed.). Addison-Wesley Professional.
- Deitel, P., & Deitel, H. (2017). *Java: How to Program, Early Objects* (11th ed.). Pearson.
- Oracle. (s. f.). *The Java Tutorials*. https://docs.oracle.com/javase/tutorial/
- Oracle. (s. f.). *Java Documentation*. https://docs.oracle.com/en/java/
- OpenJDK. (s. f.). *OpenJDK*. https://openjdk.org/

Autor: Franklin Medina Bucobo para Bitcomputacional

Área de conocimiento: Ciencias de la Computación y Tecnologías de Información
Especialidad: Desarrollo de Software y Programación