Unidad 7

Almacenamiento en memoria. Tipos básicos vs objetos

Contenidos

Parte 1: Almacenamiento en memoria. Tipos básicos vs objetos.

En Java, los datos y las estructuras se almacenan en diferentes áreas de memoria según su tipo. Entender cómo funciona este almacenamiento es esencial para escribir programas eficientes, evitar errores y optimizar el rendimiento.

1. Estructura de la Memoria en Java

Java organiza la memoria en dos áreas principales:

1. Stack (Pila):

- Memoria de acceso rápido.
- Almacena variables locales, referencias a objetos y llamadas a métodos.
- Se libera automáticamente al finalizar la ejecución de un método.

2. Heap (Montón):

- Memoria dinámica más grande.
- Contiene los objetos creados con `new` y las variables globales de clase.
- Gestionada por el **Garbage Collector**.

2. Tipos Básicos

Los tipos básicos (también llamados primitivos) son tipos de datos integrados en el lenguaje que almacenan valores simples.

Características:

1. Almacenamiento directo:

- Los valores se almacenan directamente en la stack.
- Son independientes y no necesitan instanciación.

2. Tamaño fijo:

- Cada tipo primitivo tiene un tamaño predefinido, optimizando el uso de memoria.

Tipos Básicos en Java:

Tipo	Tamaño	Valor predeterminado	Rango
`byte`	1 byte	`0`	-128 a 127
`short`	2 bytes	`0`	-32,768 a 32,767
`int`	4 bytes	`0`	-2,147,483,648 a 2,147,483,647
`long`	8 bytes	`0L`	-9,223,372,036,854,775,808 a 9,223,372,036,854,775,807
`float`	4 bytes	`0.0f`	±1.4E-45 a ±3.4028235E38
`double`	8 bytes	`0.0d`	±4.9E-324a±1.7976931348623157E308
`char`	2 bytes	`\u0000`	'\u0000' a '\uffff' (0 a 65,535)
`boolean`	Valor lógico	1 bit	`true` o `false`

3. Objetos

Un **objeto** es una instancia de una clase que puede contener múltiples atributos (estado) y métodos (comportamiento). Los objetos permiten modelar entidades del mundo real en un programa.

Características:

1. Almacenamiento en el heap:

- Los objetos se almacenan en memoria dinámica y se acceden mediante referencias almacenadas en la stack.

2. Tamaño variable:

- Depende de los atributos y métodos definidos en la clase.

Ejemplo Comparativo: Tipo Básico vs Objeto

```
public class Main {
    public static void main(String[] args) {
        // Tipo básico
        int numero = 42;
        // Objeto equivalente
        Integer objetoNumero = Integer.valueOf(numero);
        System.out.println("Tipo básico: " + numero);
        System.out.println("Objeto: " + objetoNumero);
    }
}
```

Salida:

```
Tipo básico: 42
Objeto: 42
```

4. Comparación: Tipos Básicos vs Objetos

Aspecto	Tipos Básicos	Objetos
Almacenamiento	Stack	Heap
Acceso	Directo, más rápido	Referenciado, más lento
Tamaño	Fijo	Variable
Complejidad	Simple	Complejo
Flexibilidad	Limitada	Alta
Recolección de Basura	No aplica	Gestionado por el Garbage Collector

5. Ciclo de Vida en Memoria

1. Tipos Básicos:

- Se crean en la stack al declararse.
- Se destruyen automáticamente al salir del bloque o método donde fueron declarados.

2. Objetos:

- Se crean en el heap con la palabra clave `new`.
- Permanecen en memoria hasta que no son accesibles, momento en que el **Garbage Collector** los elimina.

6. Ejemplo Completo: Tipos Básicos y Objetos

Crea un programa que compare el almacenamiento y uso de memoria de tipos básicos y objetos.

```
public class Main {
    public static void main(String[] args) {
        // Tipo básico
        int numero = 10;
        // Objeto
        Integer objetoNumero = Integer.valueOf(numero);
        // Comparación de almacenamiento
        System.out.println("Tipo básico: " + numero);
        System.out.println("Objeto: " + objetoNumero);
    }
}
```

Salida:

```
Tipo básico: 10
Objeto: 10
```

7. Ejercicios Prácticos

Ejercicio 1: Declaración y Almacenamiento

1. **Declara variables de todos los tipos básicos (`int`, `double`, `char`, `boolean`).**
2. **Declara objetos equivalentes (`Integer`, `Double`, `Character`, `Boolean`).**
3. **Analiza la diferencia entre su almacenamiento.**

Ejercicio 2: Crear y Comparar Objetos

1. **Crea una clase `Libro` con atributos `titulo` y `autor`.**
2. **Declara dos objetos `Libro` con los mismos valores y compara sus referencias.**

Ejercicio 3: Analizar Ciclo de Vida

1. **Declara una variable de tipo básico dentro de un método y analiza cómo desaparece al salir del bloque.**
2. **Crea un objeto dentro del mismo método y observa su comportamiento en memoria.**

8. Preguntas de Evaluación

1. ¿Dónde se almacenan los tipos básicos en Java?

a) Heap.
b) Stack.
c) En el disco duro.
d) Ninguna de las anteriores.

2. ¿Qué ocurre con un objeto sin referencias en Java?

a) Se elimina automáticamente.
b) Permanece en memoria.
c) Lanza un error en tiempo de ejecución.
d) Se destruye cuando se termina el método.

3. ¿Cuál de los siguientes es un tipo básico?

a) `Integer`.
b) `Double`.
c) `boolean`.
d) `String`.

4. ¿Qué ventaja tiene el almacenamiento de tipos básicos en la stack?

a) Mayor flexibilidad.

b) Más rápido y de acceso directo.

c) Tamaño variable.
d) Persistencia en memoria.

5. ¿Qué palabra clave se utiliza para crear un objeto en el heap?

a) `create`.
b) `allocate`.
c) `new`.
d) `this`.

Parte 2: Destrucción de objetos y liberación de memoria

En Java, la gestión de memoria y la destrucción de objetos es un proceso automático gestionado por el **Garbage Collector** (GC). Este sistema identifica y elimina objetos que ya no son accesibles, liberando así memoria en el heap para nuevos objetos.

1. ¿Qué es el Garbage Collector?

El **Garbage Collector** es una herramienta de la Java Virtual Machine (JVM) que:

1. **Detecta objetos no referenciados o inaccesibles.**
2. **Recupera la memoria ocupada por esos objetos.**
3. **Previene fugas de memoria al garantizar que no se acumulen objetos innecesarios en el heap.**

2. ¿Cómo Identifica el Garbage Collector los Objetos Inaccesibles?

Un objeto se considera inaccesible cuando:

- No tiene referencias activas apuntando a él.
- Las referencias han sido reasignadas o marcadas como `null`.

Ejemplo de Objeto Inaccesible

```
  public class Main {
      public static void main(String[] args) {
            Persona persona = new Persona(“Juan”, 30);
            persona = null; // Objeto ya no es accesible
            System.out.println(“El Garbage Collector eliminará el objeto en algún momento.”);
      }
  }
```

3. ¿Cuándo se Ejecuta el Garbage Collector?

La ejecución del Garbage Collector es controlada automáticamente por la JVM. Sin embargo, puedes:

1. **Solicitar su ejecución manualmente con `System.gc()`:**

```
  System.gc();
```

Esto es solo una solicitud; no garantiza que el Garbage Collector se ejecute inmediatamente.

2. Eventos comunes que disparan el Garbage Collector:

- Cuando el heap está lleno.
- Durante períodos de inactividad de la JVM.

4. Ciclo de Vida de un Objeto en Memoria

1. Creación:

- El objeto se crea en el heap usando la palabra clave `new`.

2. Referenciación:

- Se accede al objeto mediante una referencia almacenada en la stack.

3. Desreferenciación:

- La referencia se reasigna o marca como `null`.

4. Recolección de Basura:

- El Garbage Collector identifica el objeto como no accesible y lo elimina.

```
public class Persona {
    String nombre;
    public Persona(String nombre) {
        this.nombre = nombre;
    }
}
public class Main {
    public static void main(String[] args) {
        Persona persona1 = new Persona("Juan");
        Persona persona2 = persona1;
        persona1 = null; // Objeto aún es accesible desde persona2
        persona2 = null; // Ahora el objeto es inaccesible y será eliminado
        System.out.println("El Garbage Collector se encargará del objeto.");
    }
}
```

5. Importancia de la Gestión Automática

La gestión automática de memoria tiene varias ventajas:

1. Evita fugas de memoria:

- Los objetos no accesibles se eliminan automáticamente.

2. Simplifica la programación:

- El programador no necesita liberar memoria manualmente (a diferencia de lenguajes como C o C++).

3. Optimiza el uso del heap:

- Se libera memoria para nuevos objetos.

6. Limitaciones del Garbage Collector

1. No garantiza cuándo se ejecutará:

- Su activación depende de la JVM.

2. No elimina objetos con referencias cíclicas:

- Si los objetos se referencian mutuamente pero no hay referencias externas, el Garbage Collector puede no detectarlos.

Ejemplo de Referencias Cíclicas

```
  public class Nodo {
      Nodo referencia;
      public Nodo(Nodo referencia) {
            this.referencia = referencia;
      }
  }
  public class Main {
      public static void main(String[] args) {
            Nodo nodo1 = new Nodo(null);
            Nodo nodo2 = new Nodo(nodo1);
            nodo1.referencia = nodo2;
            nodo1 = null;
            nodo2 = null;
            System.out.println("El Garbage Collector debe manejar referencias
cíclicas.");
      }
  }
```

7. Métodos Especiales para la Liberación de Recursos

1. Finalización con `finalize`

El método `finalize` se invoca antes de que un objeto sea eliminado por el Garbage Collector.

```
public class Persona {
    @Override
    protected void finalize() {
          System.out.println("El objeto está siendo recolectado.");
    }
}
```

Nota: El uso de `finalize` está obsoleto y no se recomienda en versiones modernas de Java. Es preferible utilizar bloques `try-with-resources` para liberar recursos.

8. Buenas Prácticas para Gestión de Memoria

1. Evita referencias innecesarias:

- Libera referencias de objetos no utilizados asignándolas a `null`.

2. Usa estructuras adecuadas:

- Emplea colecciones como `WeakHashMap` para almacenar objetos que puedan ser recolectados.

3. Minimiza el uso de objetos pesados:

- Opta por tipos básicos cuando sea posible.

4. Usa bloques `try-with-resources`:

- Para liberar automáticamente recursos como conexiones, archivos, etc.

```
try (BufferedReader br = new BufferedReader(new FileReader("archivo.txt"))) {
    String linea = br.readLine();
    System.out.println(linea);
} catch (IOException e) {
    e.printStackTrace();
}
```

9. Ejercicios Prácticos

Ejercicio 1: Identificar Objetos Inaccesibles

1. Crea una clase `Persona` con atributos `nombre` y `edad`.
2. Declara varios objetos de la clase y reasigna sus referencias.
3. Identifica qué objetos son recolectables.

Ejercicio 2: Liberar Recursos con `try-with-resources`

1. Crea un programa que lea un archivo de texto.
2. Utiliza un bloque `try-with-resources` para garantizar que el archivo se cierre automáticamente.

1.10. Preguntas de Evaluación

1. ¿Qué hace el Garbage Collector en Java?

a) Libera memoria ocupada por objetos no accesibles.
b) Asigna espacio para nuevos objetos.
c) Evita errores en tiempo de compilación.
d) Lanza excepciones si hay referencias nulas.

2. ¿Qué ocurre si llamas manualmente a `System.gc()`?

a) Se garantiza la recolección de todos los objetos no referenciados.
b) La JVM ignora la solicitud.
c) Solicitas la ejecución del Garbage Collector, pero no hay garantía de cuándo ocurrirá.
d) Los objetos marcados como `null` se eliminan de inmediato.

3. ¿Qué método se invoca antes de que un objeto sea recolectado?

a) `delete`.
b) `destroy`.
c) `finalize`.
d) `dispose`.

4. ¿Qué ocurre con los objetos que tienen referencias cíclicas?

a) Se eliminan automáticamente.
b) Nunca se eliminan.
c) Dependen del algoritmo del Garbage Collector.
d) Lanzan un error en tiempo de ejecución.

5. ¿Qué bloque garantiza la liberación de recursos en Java?

a) `try-finally`.
b) `try-catch`.
c) `try-with-resources`.
d) `finally-with-resources`.

Conclusión del Bloque: Almacenamiento en memoria. Tipos básicos vs objetos

La comprensión del almacenamiento en memoria y las diferencias entre tipos básicos y objetos es fundamental para dominar Java y optimizar el rendimiento de las aplicaciones. Este bloque ha cubierto cómo se gestionan y almacenan los datos, la diferencia entre los tipos básicos y los objetos, y la importancia de la recolección de basura para la liberación de memoria.

1. Puntos Clave del Bloque

1. Almacenamiento en Memoria

- Java utiliza dos áreas principales de memoria:
  1. **Stack (Pila):**
     - Almacena variables locales y referencias a objetos.
     - Es rápida y se gestiona automáticamente.
  2. **Heap (Montón):**
     - Contiene los objetos creados dinámicamente con `new`.
     - Es más grande pero más lenta que la pila.
     - Gestionada por el **Garbage Collector**.

2. Tipos Básicos vs Objetos

- **Tipos básicos**:
  - Son simples, tienen un tamaño fijo y se almacenan en la stack.
  - Ejemplo: `int`, `double`, `boolean`.
- **Objetos**:
  - Representan estructuras más complejas y se almacenan en el heap.
  - Requieren instanciación mediante la palabra clave `new`.

3. Destrucción de Objetos y Liberación de Memoria

- Java automatiza la liberación de memoria a través del **Garbage Collector**.
- Un objeto se considera inaccesible cuando no hay referencias apuntando a él.
- Aunque el Garbage Collector simplifica la gestión de memoria, es importante:
  1. Liberar referencias cuando no sean necesarias.
  2. Usar bloques `try-with-resources` para liberar recursos no gestionados automáticamente, como archivos o conexiones.

4. Buenas Prácticas

1. **Minimizar el uso de memoria innecesaria:**
   - Utilizar tipos básicos siempre que sea posible para evitar sobrecarga.
2. **Gestionar referencias adecuadamente:**
   - Establecer referencias a `null` para indicar que ya no se necesitan.
3. **Evitar referencias cíclicas:**
   - Diseñar estructuras de datos que no dependan de referencias mutuas innecesarias.
4. **Comprender el Garbage Collector:**
   - Aunque es automático, entender cómo funciona ayuda a evitar problemas de rendimiento.

2. Ejercicio Final del Bloque

Objetivo

Crear un programa que modele un sistema de gestión de memoria para tipos básicos y objetos, e implementar prácticas que demuestren la liberación de memoria.

Problema

1. **Crear una clase `Archivo` que:**
   - Tenga atributos `nombre` y `tamaño` (en MB).
   - Simule abrir y cerrar archivos.
2. **Implementar el uso de `try-with-resources` para gestionar automáticamente los recursos.**
3. **Crear varios objetos de la clase `Archivo` y demostrar cómo liberar memoria configurando referencias a `null`.**

Solución

Clase `Archivo`

```
import java.io.Closeable;
public class Archivo implements Closeable {
    private String nombre;
    private int tamaño;
    public Archivo(String nombre, int tamaño) {
        this.nombre = nombre;
        this.tamaño = tamaño;
        System.out.println("Archivo abierto: " + nombre + ", Tamaño: " +
tamaño + " MB");
    }
    public void leer() {
        System.out.println("Leyendo contenido del archivo: " + nombre);
    }
    @Override
    public void close() {
        System.out.println("Cerrando el archivo: " + nombre);
    }
}
```

Clase Principal

```
public class Main {
    public static void main(String[] args) {
        // Usar try-with-resources para gestionar automáticamente los recur-
sos
        try (Archivo archivo1 = new Archivo("documento.txt", 5);
             Archivo archivo2 = new Archivo("imagen.jpg", 15)) {
            archivo1.leer();
            archivo2.leer();
        }
        // Crear objetos adicionales y liberar referencias
        Archivo archivo3 = new Archivo("video.mp4", 50);
        archivo3 = null; // Liberar referencia
        System.gc(); // Solicitar ejecución del Garbage Collector
        System.out.println("Fin del programa.");
    }
}
```

Salida Esperada

```
Archivo abierto: documento.txt, Tamaño: 5 MB
Archivo abierto: imagen.jpg, Tamaño: 15 MB
Leyendo contenido del archivo: documento.txt
Leyendo contenido del archivo: imagen.jpg
Cerrando el archivo: documento.txt
Cerrando el archivo: imagen.jpg
Archivo abierto: video.mp4, Tamaño: 50 MB
Fin del programa.
```

3. Preguntas de Evaluación

1. ¿Cuál es el propósito del Garbage Collector en Java?

a) Liberar memoria ocupada por objetos no accesibles.

b) Crear objetos en memoria.

c) Optimizar el rendimiento en tiempo de compilación.

d) Lanzar excepciones si hay fugas de memoria.

2. ¿Qué palabra clave se utiliza para solicitar manualmente la ejecución del Garbage Collector?

a) `gc.run()`.

b) `System.cleanup()`.

c) `System.gc()`.

d) `Garbage.run()`.

3. ¿Qué ocurre si una referencia apunta a `null`?

a) El objeto es eliminado automáticamente.

b) Se lanza un error en tiempo de ejecución.

c) El objeto queda inaccesible y es candidato para el Garbage Collector.

d) El programa deja de ejecutarse.

4. ¿Qué ventaja ofrece el uso de `try-with-resources`?

a) Automatiza la gestión de recursos como archivos o conexiones.

b) Garantiza la ejecución del Garbage Collector.

c) Permite crear referencias cíclicas.

d) Reemplaza la necesidad de constructores.

5. ¿Qué diferencia principal hay entre stack y heap?

a) El stack es más rápido pero más pequeño que el heap.

b) El heap almacena variables locales y el stack objetos.

c) Solo el stack es gestionado por el Garbage Collector.

d) No hay diferencias significativas.