Unidad 6

Utilización de objetos

Contenidos

Parte 1: Características de los objetos

Los **objetos** son la unidad básica de la programación orientada a objetos (POO). Representan entidades del mundo real, y cada objeto tiene un estado único, un comportamiento definido y una identidad propia.

1. Estado

El estado de un objeto se define por sus **atributos** o **propiedades**. Los atributos son variables que almacenan información sobre el objeto.

Ejemplo básico

```
public class Coche {
    String color;     // Estado
    int velocidad;   // Estado
}
```

Un objeto `Coche` puede tener diferentes estados, dependiendo de los valores de `color` y `velocidad`.

Ejemplo: Crear un objeto con un estado específico

```
public class Main {
    public static void main(String[] args) {
          Coche miCoche = new Coche();
          miCoche.color = "Rojo";
          miCoche.velocidad = 50;
          System.out.println("Color: " + miCoche.color);
          System.out.println("Velocidad: " + miCoche.velocidad + " km/h");
    }
}
```

Salida:

```
Color: Rojo
Velocidad: 50 km/h
```

2. Comportamiento

El comportamiento de un objeto se define mediante sus **métodos**. Los métodos son funciones asociadas al objeto que pueden manipular su estado o realizar acciones.

Ejemplo básico

```
public class Coche {
    String color;
    int velocidad;
    // Método para aumentar la velocidad
    void acelerar(int incremento) {
        velocidad += incremento;
    }
}
```

Usando el método para cambiar el comportamiento

```
public class Main {
    public static void main(String[] args) {
        Coche miCoche = new Coche();
        miCoche.color = "Azul";
        miCoche.velocidad = 0;
        // Aumentar la velocidad
        miCoche.acelerar(20);
        System.out.println("Velocidad actual: " + miCoche.velocidad + "
km/h");
    }
}
```

Salida:

```
Velocidad actual: 20 km/h
```

3. Identidad

Cada objeto tiene una **identidad única** que lo diferencia de otros objetos, incluso si tienen el mismo estado y comportamiento. Esto se debe a que los objetos se almacenan en ubicaciones distintas de memoria.

Ejemplo

```
  public class Main {
      public static void main(String[] args) {
            Coche coche1 = new Coche();
            Coche coche2 = new Coche();
            // Ambos objetos tienen el mismo estado
            coche1.color = “Verde”;
            coche1.velocidad = 100;
            coche2.color = “Verde”;
            coche2.velocidad = 100;
            System.out.println(coche1 == coche2); // false, diferentes identi-
dades
      }
  }
```

Salida:

```
  false
```

4. Ejemplo completo: Estado, Comportamiento e Identidad

Crea una clase `Persona` que represente las características y acciones de una persona.

Código

```
public class Persona {
    // Estado
    String nombre;
    int edad;
    // Comportamiento
    void saludar() {
          System.out.println("Hola, mi nombre es " + nombre);
    }
    void cumplirAños() {
          edad++;
          System.out.println("Ahora tengo " + edad + " años.");
    }
}
public class Main {
    public static void main(String[] args) {
          // Crear objetos
          Persona persona1 = new Persona();
          persona1.nombre = "Juan";
          persona1.edad = 25;
          Persona persona2 = new Persona();
          persona2.nombre = "Ana";
          persona2.edad = 30;
          // Usar métodos
          persona1.saludar();
          persona1.cumplirAños();
          persona2.saludar();
    }
}
```

Salida:

```
Hola, mi nombre es Juan
Ahora tengo 26 años.
Hola, mi nombre es Ana
```

5. Ejercicios Prácticos

Ejercicio 1: Crear una clase `Libro`

1. **Define los atributos `titulo`, `autor` y `numeroDePaginas`.**
2. **Agrega un método `mostrarInformacion` que imprima los detalles del libro.**
3. **Crea dos objetos `Libro` con diferentes valores y muestra su información.**

Ejercicio 2: Crear una clase `Banco`

1. **Define los atributos `nombre` del banco y `saldo` de la cuenta.**
2. **Agrega un método `depositar` que incremente el saldo.**
3. **Agrega un método `retirar` que reduzca el saldo (si hay suficiente dinero).**
4. **Crea un objeto `Banco` y realiza varias operaciones.**

6. Preguntas de Evaluación

1. ¿Qué representa el estado de un objeto?

a) Sus métodos.
b) Sus atributos y sus valores actuales.
c) Su nombre.
d) Su identidad.

2. ¿Qué define el comportamiento de un objeto?

a) Su ubicación en memoria.
b) Sus métodos.
c) Sus atributos.
d) Ninguna de las anteriores.

3. ¿Qué determina la identidad de un objeto?

a) El estado del objeto.
b) La clase del objeto.
c) Su referencia en memoria.
d) Sus atributos.

4. ¿Qué sucede si dos objetos tienen el mismo estado?

a) Son el mismo objeto.
b) Tienen identidades diferentes.
c) No se pueden comparar.
d) Ambos son referencias al mismo espacio de memoria.

Parte 2: Constructores

Un **constructor** es un método especial que se utiliza para inicializar objetos de una clase. Se ejecuta automáticamente al crear una instancia de la clase y permite establecer valores iniciales para sus atributos.

1. Características de los Constructores

1. Mismo nombre que la clase:

- Un constructor siempre tiene el mismo nombre que la clase.
- No tiene tipo de retorno, ni siquiera `void`.

2. Automático al instanciar un objeto:

- Se ejecuta automáticamente cuando se crea un objeto con la palabra clave `new`.

3. Sobrecarga:

- Puedes definir múltiples constructores con diferentes parámetros (sobrecarga de constructores).

4. Por defecto:

- Si no defines un constructor, Java proporciona uno por defecto sin parámetros.

2. Sintaxis de un Constructor

Constructor por defecto

```
public class Coche {
    String color;
    int velocidad;
    // Constructor
    public Coche() {
          color = “Blanco”;
          velocidad = 0;
    }
}
```

Ejemplo de uso

```
public class Main {
    public static void main(String[] args) {
          Coche miCoche = new Coche();
          System.out.println("Color: " + miCoche.color);
          System.out.println("Velocidad: " + miCoche.velocidad + " km/h");
    }
}
```

Salida:

```
Color: Blanco
Velocidad: 0 km/h
```

3. Constructores con Parámetros

Puedes definir constructores con parámetros para inicializar los atributos con valores específicos al crear un objeto.

Ejemplo

```
public class Coche {
    String color;
    int velocidad;
    // Constructor con parámetros
    public Coche(String color, int velocidad) {
          this.color = color;
          this.velocidad = velocidad;
    }
}
public class Main {
    public static void main(String[] args) {
          Coche miCoche = new Coche("Rojo", 50);
          System.out.println("Color: " + miCoche.color);
          System.out.println("Velocidad: " + miCoche.velocidad + " km/h");
    }
}
```

Salida:

```
Color: Rojo
Velocidad: 50 km/h
```

4. Sobrecarga de Constructores

Puedes definir múltiples constructores en la misma clase, siempre y cuando tengan diferentes listas de parámetros.

Ejemplo

```
public class Coche {
    String color;
    int velocidad;
    // Constructor sin parámetros
    public Coche() {
        this.color = “Blanco”;
        this.velocidad = 0;
    }
    // Constructor con un parámetro
    public Coche(String color) {
        this.color = color;
        this.velocidad = 0;
    }
    // Constructor con dos parámetros
    public Coche(String color, int velocidad) {
        this.color = color;
        this.velocidad = velocidad;
    }
}
public class Main {
    public static void main(String[] args) {
        Coche coche1 = new Coche();
        Coche coche2 = new Coche(“Negro”);
        Coche coche3 = new Coche(“Azul”, 60);
        System.out.println(“Coche 1: Color “ + coche1.color + “, Velocidad “
+ coche1.velocidad);
        System.out.println(“Coche 2: Color “ + coche2.color + “, Velocidad “
+ coche2.velocidad);
        System.out.println(“Coche 3: Color “ + coche3.color + “, Velocidad “
+ coche3.velocidad);
    }
}
```

Salida:

```
Coche 1: Color Blanco, Velocidad 0
Coche 2: Color Negro, Velocidad 0
Coche 3: Color Azul, Velocidad 60
```

5. Uso de `this` en Constructores

La palabra clave `this` se utiliza para referirse a los atributos de la clase actual, especialmente cuando los nombres de los parámetros son iguales a los nombres de los atributos.

Ejemplo

```
public class Coche {
    String color;
    int velocidad;
    // Constructor con this
    public Coche(String color, int velocidad) {
          this.color = color; // this diferencia el atributo del parámetro
          this.velocidad = velocidad;
    }
}
```

6. Ejemplo Completo: Clase `Persona` con Constructores

Crea una clase `Persona` que utilice constructores para inicializar atributos.

Código

```
public class Persona {
    String nombre;
    int edad;
    // Constructor por defecto
    public Persona() {
        this.nombre = “Sin Nombre”;
        this.edad = 0;
    }
    // Constructor con parámetros
    public Persona(String nombre, int edad) {
        this.nombre = nombre;
        this.edad = edad;
    }
    // Método para mostrar información
    public void mostrarInformacion() {
        System.out.println(“Nombre: “ + nombre);
        System.out.println(“Edad: “ + edad);
    }
}
public class Main {
    public static void main(String[] args) {
        Persona persona1 = new Persona();
        Persona persona2 = new Persona(“Ana”, 30);
        persona1.mostrarInformacion();
        persona2.mostrarInformacion();
    }
}
```

Salida:

```
Nombre: Sin Nombre
Edad: 0
Nombre: Ana
Edad: 30
```

7. Ejercicios Prácticos

Ejercicio 1: Crear un Constructor por Defecto

1. Crea una clase `Libro` con los atributos `titulo` y `autor`.
2. Define un constructor que inicialice ambos atributos con valores predeterminados.
3. Crea un objeto `Libro` y muestra su información.

Ejercicio 2: Crear un Constructor con Parámetros

1. Modifica la clase `Libro` para incluir un constructor que reciba `titulo` y `autor` como parámetros.
2. Crea dos objetos con valores diferentes y muestra su información.

Ejercicio 3: Sobrecarga de Constructores

1. Añade un constructor adicional a la clase `Libro` que solo reciba `titulo`.
2. Usa la sobrecarga para inicializar `autor` como `"Desconocido"` si no se proporciona.

8. Preguntas de Evaluación

1. ¿Qué ocurre si no defines un constructor en tu clase?

a) No puedes crear objetos de la clase.
b) Se genera automáticamente un constructor sin parámetros.
c) El programa lanza un error en tiempo de compilación.
d) Los atributos no se inicializan correctamente.

2. ¿Qué diferencia hay entre un constructor y un método normal?

a) Un constructor tiene tipo de retorno.
b) Un constructor puede llamarse varias veces manualmente.
c) Un constructor no tiene tipo de retorno y se ejecuta automáticamente.
d) No hay diferencias.

3. ¿Qué hace la palabra clave `this` en un constructor?

a) Se refiere a un objeto externo.

- b) Se usa para llamar a otro constructor.
- c) Se usa para acceder a los atributos de la clase actual.
- d) b y c son correctas.

4. ¿Qué permite la sobrecarga de constructores?

a) Usar el mismo nombre para múltiples constructores con diferentes parámetros.
b) Reutilizar un constructor en otra clase.
c) Inicializar atributos después de crear un objeto.
d) Ninguna de las anteriores.

Parte 3: Instanciación de objetos. Declaración y creación

La **instanciación de objetos** es el proceso de crear un objeto en Java a partir de una clase. Este proceso consta de dos pasos principales: **declaración** y **creación**.

1. Declaración de un Objeto

La **declaración** consiste en definir una variable que contendrá la referencia al objeto. La sintaxis es similar a la declaración de una variable normal, pero se especifica el tipo como el nombre de la clase.

Sintaxis

```
NombreClase nombreObjeto;
```

Ejemplo

```
public class Main {
    public static void main(String[] args) {
          Persona persona1; // Declaración del objeto
    }
}
```

En este punto, `persona1` es una referencia, pero aún no apunta a un objeto en memoria.

2. Creación de un Objeto

La **creación** consiste en asignar un nuevo objeto a la referencia declarada. Esto se realiza con la palabra clave `new`, que reserva espacio en memoria y ejecuta el constructor de la clase.

Sintaxis

```
nombreObjeto = new NombreClase();
```

Declaración y creación combinadas

Es habitual combinar la declaración y la creación en una sola línea.

```
NombreClase nombreObjeto = new NombreClase();
```

Ejemplo

```
public class Main {
    public static void main(String[] args) {
          Persona persona1 = new Persona(); // Declaración y creación
    }
}
```

3. Proceso Interno de Instanciación

Cuando se crea un objeto:

1. La JVM reserva memoria para almacenar el objeto.

2. Se ejecuta el constructor de la clase.

3. Se inicializan los atributos con valores predeterminados o definidos en el constructor.

4. La referencia al objeto se almacena en la variable declarada.

Ejemplo

```
public class Persona {
    String nombre;
    int edad;
    public Persona() {
          this.nombre = “Sin Nombre”;
          this.edad = 0;
    }
}
public class Main {
    public static void main(String[] args) {
          Persona persona1 = new Persona();
          System.out.println(persona1.nombre); // Sin Nombre
          System.out.println(persona1.edad);  // 0
    }
}
```

4. Asignación de Referencias

Una variable puede almacenar la referencia a un objeto. Si dos variables apuntan al mismo objeto, los cambios realizados a través de una variable afectarán al objeto referenciado por la otra.

Ejemplo

```
public class Main {
    public static void main(String[] args) {
          Persona persona1 = new Persona();
          persona1.nombre = “Juan”;
          Persona persona2 = persona1; // Ambas apuntan al mismo objeto
          persona2.nombre = “Ana”;
          System.out.println(persona1.nombre); // Ana
          System.out.println(persona2.nombre); // Ana
    }
}
```

Salida:

```
Ana
Ana
```

5. Instanciar Múltiples Objetos

Cada objeto creado con `new` es independiente, aunque sea de la misma clase.

Ejemplo

```
public class Main {
    public static void main(String[] args) {
          Persona persona1 = new Persona();
          Persona persona2 = new Persona();
          persona1.nombre = “Juan”;
          persona2.nombre = “Ana”;
          System.out.println(persona1.nombre); // Juan
          System.out.println(persona2.nombre); // Ana
    }
}
```

Salida:

```
Juan
Ana
```

6. Ejemplo Completo

Este ejemplo combina declaración, creación y uso de múltiples objetos.

```
public class Coche {
    String color;
    int velocidad;
    public Coche(String color, int velocidad) {
        this.color = color;
        this.velocidad = velocidad;
    }
    public void mostrarInformacion() {
        System.out.println("Color: " + color + ", Velocidad: " + velocidad +
" km/h");
    }
}
public class Main {
    public static void main(String[] args) {
        // Declaración y creación de objetos
        Coche coche1 = new Coche("Rojo", 120);
        Coche coche2 = new Coche("Azul", 90);
        // Mostrar información de los coches
        coche1.mostrarInformacion();
        coche2.mostrarInformacion();
    }
}
```

Salida:

```
Color: Rojo, Velocidad: 120 km/h
Color: Azul, Velocidad: 90 km/h
```

7. Ejercicios Prácticos

Ejercicio 1: Crear Objetos

1. Define una clase `Libro` con los atributos `titulo` y `autor`.
2. Declara y crea dos objetos `Libro` con valores diferentes.
3. Imprime los detalles de ambos libros.

Ejercicio 2: Referencias Compartidas

1. Declara y crea un objeto de la clase `Persona`.
2. Asigna este objeto a otra variable.
3. Modifica un atributo a través de una de las referencias y verifica los cambios.

Ejercicio 3: Múltiples Objetos

1. Define una clase `Banco` con los atributos `nombre` y `saldo`.
2. Declara y crea tres objetos `Banco` con diferentes valores.
3. Imprime los detalles de cada banco.

8. Preguntas de Evaluación

1. ¿Qué palabra clave se utiliza para crear un nuevo objeto en Java?

a) `new`
b) `create`
c) `object`
d) `instance`

2. ¿Qué sucede si dos variables apuntan al mismo objeto?

a) Ambas referencias son independientes.
b) Las modificaciones en una afectan a la otra.
c) Generan un error en tiempo de ejecución.
d) No pueden coexistir.

3. ¿Qué ocurre si no se ejecuta el constructor al crear un objeto?

a) El objeto no se inicializa.
b) Los atributos tienen valores aleatorios.
c) Los atributos toman valores predeterminados.
d) No es posible crear el objeto.

4. ¿Qué representa la referencia de un objeto?

- a) Su dirección en memoria.

b) Una copia del objeto.

c) Su comportamiento.

d) Sus atributos.

5. ¿Qué método garantiza que se inicialicen los atributos del objeto al crearlo?

a) Método `main`.

b) Método `toString`.

c) Constructor.

d) Ninguno.

Parte 4: Utilización de métodos. Parámetros y valores de retorno

Los **métodos** son bloques de código que realizan una tarea específica y pueden ser reutilizados en diferentes partes de un programa. Entender cómo funcionan los parámetros y los valores de retorno permite modularizar y estructurar mejor el código.

1. ¿Qué es un Método?

Un método es una función dentro de una clase que puede realizar acciones sobre los datos del objeto o devolver información.

Sintaxis Básica

```
[modificador] tipoDeRetorno nombreDelMetodo([parámetros]) {
    // Cuerpo del método
    return valor; // Opcional, dependiendo del tipo de retorno
}
```

- **Modificador**: Define el nivel de acceso (`public`, `private`, etc.).
- **Tipo de Retorno**: El tipo de dato que devuelve el método (o `void` si no devuelve nada).
- **Nombre del Método**: Identificador único del método.
- **Parámetros**: Lista opcional de variables que recibe el método como entrada.

2. Métodos sin Parámetros ni Retorno

Estos métodos ejecutan una acción sin requerir datos de entrada ni devolver un resultado.

```
public class Coche {
    String color;
    int velocidad;
    // Método que imprime información del coche
    public void mostrarInformacion() {
        System.out.println("Color: " + color + ", Velocidad: " + velocidad +
" km/h");
    }
}
public class Main {
    public static void main(String[] args) {
        Coche coche = new Coche();
        coche.color = "Rojo";
        coche.velocidad = 120;
        coche.mostrarInformacion();
    }
}
```

Salida:

```
Color: Rojo, Velocidad: 120 km/h
```

3. Métodos con Parámetros

Los métodos con parámetros reciben datos de entrada que afectan su comportamiento.

Ejemplo

```
public class Coche {
    String color;
    int velocidad;
    // Método que cambia la velocidad
    public void cambiarVelocidad(int nuevaVelocidad) {
        velocidad = nuevaVelocidad;
    }
}
```

Usando el método con parámetros

```
public class Main {
    public static void main(String[] args) {
          Coche coche = new Coche();
          coche.color = "Azul";
          coche.cambiarVelocidad(80);
          System.out.println("Velocidad actual: " + coche.velocidad + "
km/h");
    }
}
```

Salida:

```
Velocidad actual: 80 km/h
```

4. Métodos con Valores de Retorno

Un método puede devolver un resultado utilizando la palabra clave `return`.

Ejemplo

```
public class Coche {
    String color;
    int velocidad;
    // Método que devuelve la información del coche
    public String obtenerInformacion() {
          return "Color: " + color + ", Velocidad: " + velocidad + " km/h";
    }
}
```

Usando el método con retorno

```
public class Main {
    public static void main(String[] args) {
          Coche coche = new Coche();
          coche.color = "Verde";
          coche.velocidad = 100;
          String informacion = coche.obtenerInformacion();
          System.out.println(informacion);
    }
}
```

Salida:

```
Color: Verde, Velocidad: 100 km/h
```

5. Métodos con Parámetros y Retorno

Los métodos pueden recibir datos de entrada y devolver un resultado.

```
public class Calculadora {
    // Método que suma dos números
    public int sumar(int a, int b) {
        return a + b;
    }
}
```

Usando el método

```
public class Main {
    public static void main(String[] args) {
        Calculadora calc = new Calculadora();
        int resultado = calc.sumar(5, 10);
        System.out.println("La suma es: " + resultado);
    }
}
```

Salida:

```
La suma es: 15
```

6. Sobrecarga de Métodos

La sobrecarga permite definir varios métodos con el mismo nombre pero con diferentes parámetros.

```
public class Calculadora {
    // Suma dos enteros
    public int sumar(int a, int b) {
        return a + b;
    }
    // Suma tres enteros
    public int sumar(int a, int b, int c) {
        return a + b + c;
    }
}
```

Usando la sobrecarga

```
public class Main {
    public static void main(String[] args) {
        Calculadora calc = new Calculadora();
        System.out.println("Suma de 2 números: " + calc.sumar(5, 10));
        System.out.println("Suma de 3 números: " + calc.sumar(5, 10, 15));
    }
}
```

Salida:

```
Copiar código
Suma de 2 números: 15
Suma de 3 números: 30
```

7. Ejemplo Completo

Crea una clase `Rectangulo` que permita calcular el área y el perímetro de un rectángulo.

```
public class Rectangulo {
    int largo;
    int ancho;
    // Método que calcula el área
    public int calcularArea() {
        return largo * ancho;
    }
    // Método que calcula el perímetro
    public int calcularPerimetro() {
        return 2 * (largo + ancho);
    }
}
public class Main {
    public static void main(String[] args) {
        Rectangulo rectangulo = new Rectangulo();
        rectangulo.largo = 5;
        rectangulo.ancho = 3;
        System.out.println("Área: " + rectangulo.calcularArea());
        System.out.println("Perímetro: " + rectangulo.calcularPerimetro());
    }
}
```

Salida:

```
Área: 15
Perímetro: 16
```

8. Ejercicios Prácticos

Ejercicio 1: Métodos sin parámetros ni retorno

1. Crea una clase `Persona` con los atributos `nombre` y `edad`.
2. Define un método `presentarse` que imprima un saludo con el nombre y edad.

Ejercicio 2: Métodos con parámetros

1. Crea una clase `Calculadora` con un método `multiplicar` que reciba dos números y muestre el resultado.

Ejercicio 3: Métodos con retorno

1. Modifica la clase `Calculadora` para que el método `multiplicar` devuelva el resultado en lugar de imprimirlo.

Ejercicio 4: Sobrecarga de Métodos

1. Agrega un método `multiplicar` adicional en la clase `Calculadora` que reciba tres números y los multiplique.

9. Preguntas de Evaluación

1. ¿Qué palabra clave se utiliza para devolver un valor en un método?

a) `return`
b) `this`
c) `break`
d) `result`

2. ¿Qué ocurre si un método con tipo de retorno no incluye una instrucción `return`?

a) Lanza un error de compilación.
b) Devuelve un valor predeterminado.
c) El programa lanza un error en tiempo de ejecución.
d) El método se ejecuta pero no devuelve nada.

3. ¿Qué permite la sobrecarga de métodos?

a) Crear métodos con el mismo nombre pero diferente cantidad o tipo de parámetros.
b) Utilizar métodos en diferentes clases.
c) Reducir la cantidad de líneas en un programa.
d) Crear métodos más rápidos.

Parte 5: Utilización de propiedades

Las **propiedades** de un objeto, también conocidas como **atributos**, son variables asociadas a una clase que representan las características del objeto. Estas propiedades se utilizan para definir el estado del objeto y se pueden acceder y modificar a través de métodos, como **getters** y **setters**.

1. Declaración de Propiedades

En Java, las propiedades de una clase se declaran como variables dentro de la clase. Por convención, estas suelen ser privadas para proteger su acceso directo desde fuera de la clase.

Ejemplo de Declaración

```
public class Persona {
    private String nombre; // Propiedad privada
    private int edad;      // Propiedad privada
}
```

2. Acceso a las Propiedades

El acceso a las propiedades se realiza generalmente mediante métodos especiales llamados **getters** (para obtener el valor) y **setters** (para modificar el valor).

Getters y Setters

1. **Getter: Devuelve el valor de una propiedad.**
2. **Setter: Asigna un nuevo valor a la propiedad.**

Ejemplo Completo

```
public class Persona {
    private String nombre;
    private int edad;
    // Getter para nombre
    public String getNombre() {
        return nombre;
    }
    // Setter para nombre
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    // Getter para edad
    public int getEdad() {
        return edad;
    }
    // Setter para edad
    public void setEdad(int edad) {
        this.edad = edad;
    }
}
```

Uso de Getters y Setters

```
public class Main {
    public static void main(String[] args) {
        Persona persona = new Persona();
        // Usar setters para establecer propiedades
        persona.setNombre("Juan");
        persona.setEdad(25);
        // Usar getters para obtener valores
        System.out.println("Nombre: " + persona.getNombre());
        System.out.println("Edad: " + persona.getEdad());
    }
}
```

Salida:

```
Nombre: Juan
Edad: 25
```

3. Propiedades Calculadas

Además de las propiedades básicas, puedes crear **propiedades calculadas** que no almacenan un valor, sino que calculan su resultado dinámicamente.

Ejemplo

```
public class Rectangulo {
    private int largo;
    private int ancho;
    public Rectangulo(int largo, int ancho) {
          this.largo = largo;
          this.ancho = ancho;
    }
    // Propiedad calculada: área
    public int getArea() {
          return largo * ancho;
    }
}
```

Uso:

```
public class Main {
    public static void main(String[] args) {
          Rectangulo rect = new Rectangulo(5, 3);
          System.out.println("Área del rectángulo: " + rect.getArea());
    }
}
```

Salida:

```
Área del rectángulo: 15
```

4. Encapsulación de Propiedades

La **encapsulación** es una práctica fundamental en la programación orientada a objetos que protege las propiedades de acceso no autorizado o modificación indebida.

Beneficios de la Encapsulación

1. **Protección: Evita modificaciones directas a las propiedades.**
2. **Flexibilidad: Permite validar valores antes de asignarlos.**
3. **Control: Facilita el mantenimiento del código al centralizar el acceso.**

Ejemplo con Validación

```
public class Persona {
    private int edad;
    public int getEdad() {
          return edad;
    }
    public void setEdad(int edad) {
          if (edad >= 0) { // Validar edad positiva
                this.edad = edad;
          } else {
                System.out.println("La edad no puede ser negativa.");
          }
    }
}
```

Uso:

```
public class Main {
    public static void main(String[] args) {
          Persona persona = new Persona();
          persona.setEdad(-5); // Intento de asignar valor inválido
          persona.setEdad(30); // Asignación válida
          System.out.println("Edad: " + persona.getEdad());
    }
}
```

Salida:

```
La edad no puede ser negativa.
Edad: 30
```

5. Ejemplo Completo

Crea una clase `CuentaBancaria` que gestione las propiedades `saldo` y `titular`.

```
public class CuentaBancaria {
    private String titular;
    private double saldo;
    // Constructor
    public CuentaBancaria(String titular, double saldoInicial) {
        this.titular = titular;
        this.saldo = saldoInicial;
    }
    // Getter para titular
    public String getTitular() {
        return titular;
    }
    // Getter para saldo
    public double getSaldo() {
        return saldo;
    }
    // Método para depositar dinero
    public void depositar(double cantidad) {
        if (cantidad > 0) {
            saldo += cantidad;
            System.out.println("Se han depositado $" + cantidad);
        } else {
            System.out.println("Cantidad inválida.");
        }
    }
    // Método para retirar dinero
    public void retirar(double cantidad) {
        if (cantidad > 0 && cantidad <= saldo) {
            saldo -= cantidad;
            System.out.println("Se han retirado $" + cantidad);
        } else {
            System.out.println("Operación no válida.");
        }
    }
}
```

Uso

```
public class Main {
    public static void main(String[] args) {
        CuentaBancaria cuenta = new CuentaBancaria("Ana", 500);
        System.out.println("Titular: " + cuenta.getTitular());
        System.out.println("Saldo inicial: $" + cuenta.getSaldo());
        cuenta.depositar(200);
        System.out.println("Saldo actual: $" + cuenta.getSaldo());
        cuenta.retirar(150);
        System.out.println("Saldo actual: $" + cuenta.getSaldo());
    }
}
```

Salida:

```
Titular: Ana
Saldo inicial: $500.0
Se han depositado $200.0
Saldo actual: $700.0
Se han retirado $150.0
Saldo actual: $550.0
```

6. Ejercicios Prácticos

Ejercicio 1: Crear una Clase `Libro`

1. Define propiedades `titulo` y `autor` como privadas.
2. Crea getters y setters para ambas propiedades.
3. Usa los métodos para mostrar y modificar los valores.

Ejercicio 2: Validar Propiedades

1. Modifica la clase `Persona` para validar que el nombre no esté vacío al asignarlo.
2. Asegúrate de que la edad sea mayor o igual a 0.

Ejercicio 3: Propiedades Calculadas

1. Crea una clase `Circulo` con una propiedad privada `radio`.
2. Implementa un getter calculado para el área (`PI * radio^2`).

7. Preguntas de Evaluación

1. ¿Qué palabra clave se utiliza para declarar propiedades privadas?

a) `private`
b) `protected`
c) `final`
d) `static`

2. ¿Qué permite un método `getter`?

a) Modificar una propiedad.
b) Obtener el valor de una propiedad.
c) Validar valores.
d) Inicializar propiedades.

3. ¿Cuál es la ventaja de usar encapsulación en las propiedades?

a) Proteger las propiedades de acceso directo.
b) Permitir el acceso desde cualquier parte del código.
c) Reducir la complejidad del código.
d) Ninguna de las anteriores.

4. ¿Qué ocurre si se accede directamente a una propiedad privada desde otra clase?

a) Se lanza un error en tiempo de compilación.
b) Se lanza un error en tiempo de ejecución.
c) La propiedad se modifica sin restricciones.
d) El acceso es permitido automáticamente.

Parte 6: Utilización de métodos estáticos

Los **métodos estáticos** en Java son aquellos que pertenecen a la clase en lugar de a las instancias de la clase. Esto significa que se pueden llamar sin crear un objeto de la clase.

1. ¿Qué es un Método Estático?

Un método se declara como **estático** utilizando la palabra clave `static`. Estos métodos:

1. No dependen de una instancia de la clase.

2. Solo pueden acceder a otros miembros estáticos de la clase.

3. Son ideales para operaciones generales que no necesitan modificar el estado del objeto.

Sintaxis

```
public class ClaseEjemplo {
    public static void metodoEstatico() {
        // Código del método
    }
}
```

Para llamar a un método estático:

```
ClaseEjemplo.metodoEstatico();
```

2. Ejemplo Básico de Método Estático

```
public class Calculadora {
    public static int sumar(int a, int b) {
        return a + b;
    }
}
```

Uso:

```
public class Main {
    public static void main(String[] args) {
        int resultado = Calculadora.sumar(10, 20);
        System.out.println("La suma es: " + resultado);
    }
}
```

Salida:

```
La suma es: 30
```

3. Comparación: Métodos Estáticos vs No Estáticos

Característica	Métodos Estáticos	Métodos No Estáticos
Asociación	A la clase	A una instancia (objeto)
Acceso a miembros	Solo miembros estáticos	Miembros estáticos y no estáticos
Llamada	Clase.metodo()	objeto.metodo()
Uso típico	Operaciones generales o utilidades	Comportamientos específicos del objeto

4. Usos Comunes de Métodos Estáticos

1. Funciones de utilidad:

- Métodos que realizan cálculos o transformaciones genéricas.
- Ejemplo: `Math.sqrt()`, `Arrays.sort()`.

2. Métodos fábrica:

- Métodos que devuelven una instancia de la clase.
- Ejemplo:

```
public static MiClase crearInstancia() {
    return new MiClase();
}
```

3. Métodos auxiliares:

- Métodos que proporcionan servicios independientes del estado del objeto.

5. Limitaciones de Métodos Estáticos

1. No pueden acceder directamente a miembros no estáticos:

- Los métodos estáticos no tienen una referencia a `this`, por lo que no pueden interactuar con propiedades o métodos no estáticos.

2. No pueden ser sobrescritos:

- Aunque los métodos estáticos pueden ser redefinidos en una subclase, no son polimórficos y no pueden ser sobrescritos.

6. Ejemplo Completo

Clase `Utilidades`

```
public class Utilidades {
    // Método estático para calcular el máximo de dos números
    public static int maximo(int a, int b) {
        return (a > b) ? a : b;
    }
    // Método estático para calcular el cuadrado de un número
    public static int cuadrado(int numero) {
        return numero * numero;
    }
}
```

Uso en el Programa Principal

```
public class Main {
    public static void main(String[] args) {
        int mayor = Utilidades.maximo(10, 20);
        System.out.println("El mayor es: " + mayor);
        int cuadrado = Utilidades.cuadrado(5);
        System.out.println("El cuadrado de 5 es: " + cuadrado);
    }
}
```

Salida:

```
El mayor es: 20
El cuadrado de 5 es: 25
```

7. Métodos Estáticos en Clases Predefinidas

Java incluye muchas clases con métodos estáticos útiles. Por ejemplo:

1. Clase `Math`:

```
double raiz = Math.sqrt(25); // Raíz cuadrada
double potencia = Math.pow(2, 3); // Potencia
```

2. Clase `Arrays`:

```
int[] numeros = {3, 1, 4, 1, 5};
Arrays.sort(numeros); // Ordena el array
```

3. Clase `Collections`:`List<Integer> lista = Arrays.asList(1, 2, 3, 4);`

```
Collections.reverse(lista); // Invierte el orden de la lista
```

8. Ejercicios Prácticos

Ejercicio 1: Métodos de Utilidad

1. Crea una clase `Calculadora` con métodos estáticos:

- `sumar(int a, int b)`
- `restar(int a, int b)`
- `multiplicar(int a, int b)`
- `dividir(int a, int b)` (verifica que el divisor no sea 0).

Ejercicio 2: Generar Números Aleatorios

1. Crea una clase `RandomUtils` con un método estático:

- `generarNumeroAleatorio(int min, int max)` que devuelva un número aleatorio entre dos valores.

Ejercicio 3: Conversión de Unidades

1. Define una clase `Conversor` con métodos estáticos:

- `metrosAPies(double metros)`
- `piesAMetros(double pies)`
- Usa las constantes: `1 metro = 3.28084 pies`.

9. Preguntas de Evaluación

1. ¿Qué palabra clave se utiliza para declarar un método estático?

a) `static`
b) `final`
c) `class`
d) `void`

2. ¿Cuál es la principal diferencia entre un método estático y uno no estático?

a) Los métodos estáticos no pueden llamarse directamente.
b) Los métodos estáticos pertenecen a la clase y no a una instancia.
c) Los métodos no estáticos no pueden acceder a miembros estáticos.
d) No hay diferencias.

3. ¿Qué ocurre si un método estático intenta acceder a un miembro no estático?

a) El programa lanza un error en tiempo de ejecución.
b) El programa lanza un error en tiempo de compilación.
c) El método accede al miembro normalmente.
d) Se ignora el acceso.

4. ¿Qué permite la clase `Math` en Java?

a) Crear objetos matemáticos.
b) Realizar operaciones matemáticas mediante métodos estáticos.
c) Sobrescribir operadores aritméticos.
d) Nada relacionado con matemáticas.

5. ¿Cuál de las siguientes afirmaciones es verdadera?

a) Los métodos estáticos no pueden sobrecargarse.
b) Los métodos estáticos pertenecen a la instancia de una clase.
c) Los métodos estáticos pueden usarse sin crear un objeto.
d) Los métodos estáticos no pueden redefinirse.

Conclusión del Bloque: Utilización de objetos

La **utilización de objetos** es el núcleo de la programación orientada a objetos (POO). Este bloque ha abarcado los conceptos esenciales que permiten a los estudiantes comprender cómo modelar, instanciar y manipular objetos en Java, así como aprovechar las herramientas y características de este paradigma para escribir código estructurado y reutilizable.

1. Puntos Clave del Bloque

1. Características de los Objetos

- Cada objeto tiene un **estado** (atributos), un **comportamiento** (métodos) y una **identidad** única.
- Los objetos permiten modelar entidades del mundo real en el código, promoviendo un diseño más intuitivo y organizado.

2. Constructores

- Los **constructores** son métodos especiales que inicializan objetos al crearlos.
- Permiten asignar valores iniciales a los atributos y pueden ser sobrecargados para mayor flexibilidad.
- Usar constructores garantiza que los objetos siempre comiencen en un estado válido.

3. Instanciación de Objetos

- Crear un objeto en Java implica dos pasos: **declaración** y **creación**.
- La palabra clave `new` reserva memoria para el objeto e invoca su constructor.
- Los objetos pueden compartir referencias, lo que permite gestionar un mismo objeto desde diferentes variables.

4. Utilización de Métodos

- Los métodos encapsulan comportamientos del objeto y permiten interactuar con él.
- **Parámetros** y **valores de retorno** hacen que los métodos sean reutilizables y flexibles.
- La **sobrecarga de métodos** permite definir múltiples variantes con el mismo nombre, pero diferentes parámetros.

5. Utilización de Propiedades

- Las propiedades representan el estado del objeto y deben protegerse con la **encapsulación**.
- Los métodos **getters** y **setters** facilitan el acceso controlado a las propiedades, permitiendo validaciones y cálculos dinámicos.

6. Métodos Estáticos

- Los métodos estáticos pertenecen a la clase en lugar de a las instancias, por lo que son ideales para funciones generales o utilidades.
- Se pueden usar sin necesidad de crear un objeto, pero tienen limitaciones en cuanto a la interacción con miembros no estáticos.

2. Importancia de la Utilización de Objetos

Dominar el uso de objetos es esencial para trabajar con Java, ya que:

1. Estructura y modularidad:

- Los objetos dividen el programa en unidades independientes y manejables.

2. Reutilización:

- Clases bien diseñadas pueden ser reutilizadas en múltiples proyectos, ahorrando tiempo y esfuerzo.

3. Escalabilidad:

- El enfoque orientado a objetos facilita la construcción de sistemas grandes y complejos.

3.Ejercicio Final del Bloque

Objetivo

Diseñar un programa completo que utilice los conceptos de este bloque, incluyendo:

- Creación de una clase con propiedades y métodos.
- Uso de constructores.
- Métodos estáticos y no estáticos.
- Encapsulación con getters y setters.

Problema

Crea un programa para gestionar una **biblioteca** que permita:

1. Registrar libros con su título, autor y número de copias.
2. Consultar la información de un libro.
3. Actualizar el número de copias disponibles.
4. Contar el total de libros registrados utilizando un método estático.

Solución

Clase `Libro`

```
  public class Libro {
      private String titulo;
      private String autor;
      private int copiasDisponibles;
      // Atributo estático para contar libros
      private static int totalLibros = 0;
      // Constructor
      public Libro(String titulo, String autor, int copiasDisponibles) {
            this.titulo = titulo;
            this.autor = autor;
            this.copiasDisponibles = copiasDisponibles;
            totalLibros++; // Incrementar el contador de libros al crear uno
nuevo
      }
      // Getter para título
      public String getTitulo() {
            return titulo;
      }
      // Getter para autor
      public String getAutor() {
            return autor;
      }
      // Getter y setter para copias disponibles
      public int getCopiasDisponibles() {
            return copiasDisponibles;
      }
      public void setCopiasDisponibles(int copiasDisponibles) {
            if (copiasDisponibles >= 0) {
                  this.copiasDisponibles = copiasDisponibles;
            } else {
                  System.out.println("El número de copias no puede ser negati-
vo.");
            }
      }
      // Método para mostrar información del libro
      public void mostrarInformacion() {
            System.out.println("Título: " + titulo);
            System.out.println("Autor: " + autor);
            System.out.println("Copias disponibles: " + copiasDisponibles);
```

```
    }
    // Método estático para obtener el total de libros
    public static int getTotalLibros() {
        return totalLibros;
    }
}
```

Clase Principal

```
public class Main {
    public static void main(String[] args) {
        // Crear libros
        Libro libro1 = new Libro("Cien Años de Soledad", "Gabriel García Márquez", 5);
        Libro libro2 = new Libro("1984", "George Orwell", 3);
        // Mostrar información de los libros
        libro1.mostrarInformacion();
        libro2.mostrarInformacion();
        // Actualizar copias disponibles
        libro1.setCopiasDisponibles(10);
        System.out.println("Copias actualizadas de '" + libro1.getTitulo() + "': " + libro1.getCopiasDisponibles());
        // Mostrar el total de libros registrados
        System.out.println("Total de libros registrados: " + Libro.getTotal-Libros());
    }
}
```

Salida Esperada

```
Título: Cien Años de Soledad
Autor: Gabriel García Márquez
Copias disponibles: 5
Título: 1984
Autor: George Orwell
Copias disponibles: 3
Copias actualizadas de 'Cien Años de Soledad': 10
Total de libros registrados: 2
```