CLASE 6. Funciones y Arreglos

Introducción a Funciones en Java

Las funciones en Java son un componente esencial de la programación modular, permitiendo la reutilización de código y facilitando la organización de programas complejos. Una función es un bloque de código diseñado para realizar una tarea específica. En Java, las funciones deben estar definidas dentro de una clase, ya que no existen funciones independientes como en otros lenguajes de programación.

Declaración de Funciones

Para declarar una función en Java, es necesario definir su tipo de retorno, nombre y parámetros de entrada, si los hay. El formato general para la declaración de una función es:

```
public tipoRetorno nombreFuncion(tipoParametro
nombreParametro) {
    // Cuerpo de la función
}
```

Por ejemplo, una función que suma dos números enteros y devuelve el resultado sería:

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

Parámetros de Función

Los parámetros son variables que se utilizan para pasar información a una función. Java admite dos tipos de parámetros:

1. **Parámetros por valor**: En Java, los tipos primitivos (int, float, etc.) se pasan por valor, lo que significa que la función recibe una copia de los datos.
2. **Parámetros por referencia**: Los objetos y arreglos en Java se pasan por referencia, lo que permite modificar los datos originales dentro de la función.

Retorno de Valores

El tipo de retorno de una función indica el tipo de dato que la función devolverá al final de su ejecución. Si una función no devuelve ningún valor, se utiliza el tipo `void`.

```
public int multiplicar(int x, int y) {
    return x * y;
}
```

En este ejemplo, el tipo de retorno es `int`, por lo que la función debe devolver un valor entero.

Funciones sin Retorno

Cuando no se requiere que una función devuelva un valor, se utiliza el tipo de retorno `void`. Estas funciones simplemente ejecutan instrucciones sin devolver un resultado.

```
public void imprimirMensaje() {
    System.out.println("Hola, mundo");
}
```

Llamada de Funciones

Para invocar una función, se debe utilizar su nombre seguido de los paréntesis que contienen los parámetros, si los hay. La llamada puede realizarse desde cualquier parte del código, siempre que la función esté definida en el ámbito correspondiente.

Ámbito de las Variables

El ámbito de una variable define dónde puede ser utilizada dentro del programa. Las variables locales son aquellas definidas dentro de una función y solo son accesibles dentro de la misma. Las variables de clase o de instancia pueden ser accedidas desde diferentes partes de la clase.

Declaración y Uso de Arrays en Java

Los **arrays** en Java son estructuras de datos que permiten almacenar múltiples valores del mismo tipo. Son útiles cuando se necesita manejar grandes cantidades de datos de manera ordenada.

Arrays en Java

En Java, los arrays se declaran de la siguiente forma:

```
tipo[] nombreArray;
```

Por ejemplo:

```
int[] numeros;
```

Declaración y Creación de Arrays

La declaración solo define la referencia del array, mientras que la creación asigna memoria para almacenar los elementos. Para inicializar un array, se utiliza la palabra clave `new`:

```
numeros = new int[5];
```

En este caso, el array `numeros` puede almacenar cinco enteros.

Acceso a Elementos en un Array

Los elementos de un array se acceden mediante índices, comenzando desde cero:

```
int primerElemento = numeros[0];
```

Longitud de un Array

La propiedad `length` se utiliza para obtener el tamaño de un array:

```
int longitud = numeros.length;
```

Inicialización de Arrays

Los arrays pueden inicializarse al momento de su declaración:

```
int[] numeros = {1, 2, 3, 4, 5};
```

ArrayList en Java

La clase `ArrayList` en Java proporciona una estructura de datos dinámica, que permite almacenar y manipular colecciones de elementos sin tener que definir su tamaño inicial.

Declaración y Creación de ArrayList

La declaración de un `ArrayList` es similar a la de un array, pero con una sintaxis más flexible:

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

Agregar y Acceder a Elementos

Para agregar elementos a un `ArrayList`, se utiliza el método `add`:

```
lista.add("Elemento 1");
```

Para acceder a los elementos, se emplea el método `get`:

```
String elemento = lista.get(0);
```

Tamaño y Comprobaciones

El tamaño del `ArrayList` se obtiene mediante el método `size`:

```
int tamano = lista.size();
```

Para verificar si contiene un elemento específico, se puede usar el método `contains`:

```
boolean contiene = lista.contains("Elemento 1");
```

Eliminar y Buscar Elementos

Los elementos pueden eliminarse con el método `remove`:

```
lista.remove("Elemento 1");
```

También es posible buscar el índice de un elemento en el `ArrayList`:

```
int indice = lista.indexOf("Elemento 2");
```

Iteración sobre Arrays Utilizando Bucles

Java permite iterar sobre los arrays utilizando varios tipos de bucles:

- **Bucle `for`**: El bucle `for` es ideal cuando se conoce el número exacto de iteraciones:

```
for (int i = 0; i < numeros.length; i++) {
    System.out.println(numeros[i]);
}
```

- **Bucle `while`**: Utilizado cuando no se sabe cuántas iteraciones son necesarias de antemano:

```
int i = 0;
while (i < numeros.length) {
    System.out.println(numeros[i]);
    i++;
}
```

- **Bucle `for-each`**: Especialmente útil para recorrer arrays o colecciones:

```java
for (int num : numeros) {
    System.out.println(num);
}
```

API Swing

Swing es una biblioteca gráfica de Java que forma parte de la biblioteca estándar **Java Foundation Classes (JFC)**. Proporciona un conjunto extenso de componentes para crear interfaces gráficas de usuario (GUI), permitiendo a los desarrolladores construir aplicaciones visuales ricas en funcionalidad, tanto en entornos de escritorio como en otras plataformas.

Características de la API Swing

1. **Independencia de la plataforma**
   Swing se basa en el principio de independencia de plataforma, lo que significa que las aplicaciones desarrolladas con esta API funcionarán de la misma manera en diferentes sistemas operativos, manteniendo una apariencia consistente y uniforme.
2. **Componentes ligeros**
   A diferencia de la anterior API de Java llamada AWT (Abstract Window Toolkit), que dependía en gran medida de los componentes nativos del sistema operativo, Swing utiliza componentes ligeros, es decir, no requieren recursos nativos del sistema, lo que los hace más flexibles y portables.
3. **Modelo de Delegación de Eventos**
   Swing adopta un enfoque basado en eventos para la interacción

con la interfaz gráfica. Los eventos, como clics de botones o entrada de texto, se gestionan mediante el modelo de delegación de eventos, donde los componentes emiten eventos que son manejados por oyentes (listeners) específicos.

4. **Interfaz visual personalizable**
   Los desarrolladores pueden personalizar la apariencia de las aplicaciones Swing mediante la configuración de diferentes **Look and Feels**, lo que permite cambiar el diseño y estilo de los componentes.
5. **Compatibilidad con MVC (Modelo-Vista-Controlador)**
   Swing sigue el patrón **Modelo-Vista-Controlador (MVC)**, lo que facilita la separación de la lógica de la interfaz de usuario y la estructura de datos.

Componentes Fundamentales de Swing

1. JFrame

El `JFrame` es la ventana principal de una aplicación Swing. Representa un marco o contenedor que puede contener otros componentes de la interfaz, como botones, etiquetas, paneles, etc. Es el componente básico para crear aplicaciones de escritorio.

Creación de un JFrame

Para crear un `JFrame`, es necesario instanciar la clase `JFrame` y establecer sus propiedades básicas, como el tamaño y la visibilidad.

```
import javax.swing.JFrame;

public class MiVentana {
    public static void main(String[] args) {
```

```
        JFrame ventana = new JFrame("Mi Aplicación");
        ventana.setSize(400, 300);  // Establece el
tamaño de la ventana

ventana.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
// Cierra la aplicación al cerrar la ventana
        ventana.setVisible(true);  // Hace visible la
ventana
    }
}
```

Propiedades de JFrame

- **setSize(int ancho, int alto)**: Establece el tamaño de la ventana.
- **setTitle(String titulo)**: Define el título de la ventana.
- **setDefaultCloseOperation(int operación)**: Especifica lo que sucede cuando se cierra la ventana. El valor común es `JFrame.EXIT_ON_CLOSE` para cerrar la aplicación.
- **setVisible(boolean estado)**: Muestra u oculta la ventana.

2. JPanel

Un `JPanel` es un contenedor genérico que se utiliza para organizar componentes dentro de un `JFrame`. Puede contener otros componentes como botones, etiquetas, campos de texto, etc., y puede gestionarse mediante **layouts** (disposición).

Ejemplo de uso de JPanel

```
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JButton;

public class EjemploPanel {
    public static void main(String[] args) {
```

```
        JFrame ventana = new JFrame("Ejemplo de
JPanel");
        JPanel panel = new JPanel();  // Crear un nuevo
panel
        JButton boton = new JButton("Haz clic aquí");
// Crear un botón

        panel.add(boton);  // Agregar el botón al panel
        ventana.add(panel);  // Agregar el panel a la
ventana

        ventana.setSize(300, 200);

ventana.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        ventana.setVisible(true);
    }
}
```

En este ejemplo, el botón se añade a un `JPanel`, que a su vez se añade al `JFrame`. El `JPanel` es un contenedor flexible que facilita la disposición de los componentes.

3. JButton

El `JButton` es un componente básico en Swing que permite crear botones que pueden recibir eventos de clic. Cada botón puede contener texto o iconos.

Ejemplo básico de JButton

```
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;

public class EjemploBoton {
    public static void main(String[] args) {
```

```
        JFrame ventana = new JFrame("Ejemplo de
JButton");
        JPanel panel = new JPanel();
        JButton boton = new JButton("Presiona aquí");

        boton.addActionListener(e ->
System.out.println("Botón presionado"));  // Acción al
presionar el botón

        panel.add(boton);
        ventana.add(panel);

        ventana.setSize(200, 100);

ventana.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        ventana.setVisible(true);
    }
}
```

4. JLabel

El `JLabel` se utiliza para mostrar texto estático o imágenes en la interfaz gráfica. No es interactivo, pero es útil para proporcionar información visual al usuario.

Ejemplo básico de JLabel

```
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;

public class EjemploLabel {
    public static void main(String[] args) {
        JFrame ventana = new JFrame("Ejemplo de
JLabel");
        JPanel panel = new JPanel();
```

```
        JLabel etiqueta = new JLabel("Etiqueta de
ejemplo");

        panel.add(etiqueta);
        ventana.add(panel);

        ventana.setSize(300, 100);

ventana.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        ventana.setVisible(true);
    }
}
```

5. JTextField y JTextArea

El `JTextField` permite la entrada de una sola línea de texto, mientras que el `JTextArea` permite la entrada de varias líneas de texto.

Ejemplo de JTextField

```
import javax.swing.*;

public class EjemploTextField {
    public static void main(String[] args) {
        JFrame ventana = new JFrame("Ejemplo de
JTextField");
        JPanel panel = new JPanel();
        JTextField campoTexto = new JTextField(20);  //
Campo de texto con 20 columnas

        panel.add(campoTexto);
        ventana.add(panel);

        ventana.setSize(300, 100);
```

```
ventana.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        ventana.setVisible(true);
    }
}
```

Ejemplo de JTextArea

```
import javax.swing.*;

public class EjemploTextArea {
    public static void main(String[] args) {
        JFrame ventana = new JFrame("Ejemplo de
JTextArea");
        JPanel panel = new JPanel();
        JTextArea areaTexto = new JTextArea(5, 20);  //
Área de texto con 5 filas y 20 columnas

        panel.add(new JScrollPane(areaTexto));  //
Agrega un JScrollPane para permitir el desplazamiento
        ventana.add(panel);

        ventana.setSize(300, 150);

ventana.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        ventana.setVisible(true);
    }
}
```

Disposición de Componentes (Layout Managers)

Los **layout managers** en Swing permiten controlar cómo se distribuyen los componentes dentro de un contenedor. Existen varios

tipos de layouts, entre ellos:

1. **FlowLayout**: Coloca los componentes en una fila, ajustándolos al tamaño de la ventana.

```
panel.setLayout(new FlowLayout());
```

2. **BorderLayout**: Distribuye los componentes en cinco áreas: norte, sur, este, oeste y centro.

```
ventana.setLayout(new BorderLayout());
ventana.add(botonNorte, BorderLayout.NORTH);
```

3. **GridLayout**: Organiza los componentes en una cuadrícula de filas y columnas.

```
panel.setLayout(new GridLayout(2, 3));  // 2 filas
y 3 columnas
```

4. **BoxLayout**: Dispone los componentes en una sola fila o columna.

```
panel.setLayout(new BoxLayout(panel,
BoxLayout.Y_AXIS));  // Disposición vertical
```