Operadores Aritméticos

+-\*/%

Operadores relacionales preguntar por el swich

* Mayor que (>)
* Mayor o igual que (>=)
* Menor que (<)
* Menor o igual que (<=)
* Igual que (==)
* Distinto de (!=)

Operadores de bits

En Java, los operadores de bits se utilizan para manipular los bits individuales de un valor. Aquí tienes una lista de los operadores de bits más comunes y su funcionamiento:

1. **AND a nivel de bits (&)**: Compara cada bit de dos operandos. El resultado es 1 solo si ambos bits son 1.
2. int resultado = 5 & 3; // 0101 & 0011 = 0001 (1 en decimal)
3. **OR a nivel de bits (|)**: Compara cada bit de dos operandos. El resultado es 1 si al menos uno de los bits es 1.
4. int resultado = 5 | 3; // 0101 | 0011 = 0111 (7 en decimal)
5. **XOR a nivel de bits (^)**: Compara cada bit de dos operandos. El resultado es 1 si los bits son diferentes.
6. int resultado = 5 ^ 3; // 0101 ^ 0011 = 0110 (6 en decimal)
7. **Complemento a nivel de bits (~)**: Invierte todos los bits del operando.
8. int resultado = ~5; // ~0101 = 1010 (en complemento a dos, esto es -6 en decimal)
9. **Desplazamiento a la izquierda (<<)**: Desplaza los bits del operando a la izquierda, añadiendo ceros a la derecha.
10. int resultado = 5 << 1; // 0101 << 1 = 1010 (10 en decimal)
11. **Desplazamiento a la derecha (>>)**: Desplaza los bits del operando a la derecha, manteniendo el bit de signo.
12. int resultado = 5 >> 1; // 0101 >> 1 = 0010 (2 en decimal)
13. **Desplazamiento a la derecha sin signo (>>>)**: Desplaza los bits del operando a la derecha, llenando con ceros.
14. int resultado = -5 >>> 1; // Para números negativos, el resultado depende de la representación binaria.

[Estos operadores son útiles para tareas de bajo nivel, como la manipulación de datos binarios y la optimización de algoritmos](https://www.manualweb.net/java/operadores-bit-java/)[1](https://www.manualweb.net/java/operadores-bit-java/)[2](https://codegym.cc/es/groups/posts/es.10.operadores-bit-a-bit-de-java)[3](https://aprenderjava.net/base/guia-completa-operadores-utilizados-en-java-y-sus-funciones/).

¿Te gustaría ver algún ejemplo más detallado o tienes alguna otra pregunta sobre Java?

Operadores booleanos

[Los **operadores booleanos en Java** se utilizan para combinar y evaluar expresiones booleanas](https://www.bing.com/ck/a?!&&p=9de27b41cd7a309dJmltdHM9MTcyNTQwODAwMCZpZ3VpZD0xY2Q3ZDRhMy01NTZkLTY1MWUtMjc0ZC1jMDFmNTRjMjY0MWImaW5zaWQ9NTcxMA&ptn=3&ver=2&hsh=3&fclid=1cd7d4a3-556d-651e-274d-c01f54c2641b&psq=Operadores+booleanos++en+java&u=a1aHR0cDovL3d3dy5lZHU0amF2YS5jb20vZXMvcHJvZ2Jhc2ljYS9wcm9nYmFzaWNhNi5odG1s&ntb=1)[**1**](https://www.bing.com/ck/a?!&&p=f9d81dc3af2a3ef5JmltdHM9MTcyNTQwODAwMCZpZ3VpZD0xY2Q3ZDRhMy01NTZkLTY1MWUtMjc0ZC1jMDFmNTRjMjY0MWImaW5zaWQ9NTcxMQ&ptn=3&ver=2&hsh=3&fclid=1cd7d4a3-556d-651e-274d-c01f54c2641b&psq=Operadores+booleanos++en+java&u=a1aHR0cDovL3d3dy5lZHU0amF2YS5jb20vZXMvcHJvZ2Jhc2ljYS9wcm9nYmFzaWNhNi5odG1s&ntb=1)[**2**](https://www.bing.com/ck/a?!&&p=e5da941ea372b302JmltdHM9MTcyNTQwODAwMCZpZ3VpZD0xY2Q3ZDRhMy01NTZkLTY1MWUtMjc0ZC1jMDFmNTRjMjY0MWImaW5zaWQ9NTcxMg&ptn=3&ver=2&hsh=3&fclid=1cd7d4a3-556d-651e-274d-c01f54c2641b&psq=Operadores+booleanos++en+java&u=a1aHR0cHM6Ly9hcHJlbmRlcmphdmEubmV0L2Jhc2UvZ3VpYS1kZS11c28tZGUtYm9vbGVhbm9zLWVuLWphdmEtcGFyYS1vYmpldG9zLw&ntb=1). Algunos operadores comunes son:

* **&& (AND)**: Evalúa como verdadero si ambas expresiones son verdaderas.
* **|| (OR)**: Evalúa como verdadero si al menos una de las expresiones es verdadera.
* **! (NOT)**: Invierte el valor de la expresión booleana.

Operadores Incrementales y Aritmetico combinados

¡Claro! En Java, los operadores aritméticos e incrementales se pueden combinar para realizar operaciones de manera más eficiente. Aquí tienes una breve explicación y algunos ejemplos:

**Operadores Aritméticos**

Estos operadores permiten realizar operaciones matemáticas básicas:

* **Suma (+)**: Añade dos valores.
* **Resta (-)**: Resta un valor de otro.
* **Multiplicación (\*)**: Multiplica dos valores.
* **División (/)**: Divide un valor por otro.
* **Módulo (%)**: Devuelve el resto de una división.

**Operadores Incrementales**

Estos operadores incrementan o decrementan el valor de una variable en una unidad:

* **Incremento (++):** Aumenta el valor de una variable en 1.
* **Decremento (–):** Disminuye el valor de una variable en 1.

**Operadores Combinados**

Los operadores combinados permiten realizar una operación aritmética y asignar el resultado a una variable en una sola instrucción:

* **+=**: Suma y asigna.
* **-=**: Resta y asigna.
* \***=**: Multiplica y asigna.
* **/=**: Divide y asigna.
* **%=**: Calcula el módulo y asigna.

**Ejemplos en Java**

int a = 5;

int b = 3;

// Operadores aritméticos

int suma = a + b; // 8

int resta = a - b; // 2

int multiplicacion = a \* b; // 15

int division = a / b; // 1

int modulo = a % b; // 2

// Operadores incrementales

a++; // a es ahora 6

b--; // b es ahora 2

// Operadores combinados

a += b; // a es ahora 8 (6 + 2)

a -= b; // a es ahora 6 (8 - 2)

a \*= b; // a es ahora 12 (6 \* 2)

a /= b; // a es ahora 6 (12 / 2)

a %= b; // a es ahora 0 (6 % 2)

Estos operadores son muy útiles para simplificar y hacer más legible el código. Si necesitas más detalles, puedes revisar [este video](https://www.youtube.com/watch?v=7qFtLWpVJDM) [que explica los operadores aritméticos, incrementales y combinados en Java](https://www.youtube.com/watch?v=7qFtLWpVJDM)[1](https://www.youtube.com/watch?v=7qFtLWpVJDM).

¿Te gustaría ver algún otro ejemplo o tienes alguna pregunta específica sobre estos operadores?

Operador Condicional y de cadenas

¡Claro! En Java, los operadores condicionales y de cadenas son fundamentales para controlar el flujo del programa y manipular texto. Aquí tienes una breve explicación de ambos:

**Operadores Condicionales**

1. **if-else**: Permite ejecutar un bloque de código si una condición es verdadera y otro bloque si es falsa.
2. if (condicion) {
3. // Código si la condición es verdadera
4. } else {
5. // Código si la condición es falsa
6. }
7. **if-else if-else**: Permite evaluar múltiples condiciones en secuencia.
8. if (condicion1) {
9. // Código si la condición1 es verdadera
10. } else if (condicion2) {
11. // Código si la condición2 es verdadera
12. } else {
13. // Código si ninguna condición es verdadera
14. }
15. **Operador ternario**: Es una forma compacta de escribir una declaración if-else.
16. resultado = (condicion) ? valorSiVerdadero : valorSiFalso;
17. **switch**: Permite seleccionar entre múltiples opciones basadas en el valor de una variable.
18. switch (variable) {
19. case valor1:
20. // Código para valor1
21. break;
22. case valor2:
23. // Código para valor2
24. break;
25. default:
26. // Código si no coincide ningún caso
27. }

**Operadores de Cadenas**

1. **Concatenación**: Se utiliza el operador + para unir dos cadenas.
2. String saludo = "Hola, " + "mundo!";
3. **Métodos de la clase String**:
   * **length()**: Devuelve la longitud de la cadena.
   * int longitud = cadena.length();
   * **charAt()**: Devuelve el carácter en una posición específica.
   * char caracter = cadena.charAt(0);
   * **substring()**: Extrae una subcadena.
   * String subcadena = cadena.substring(1, 4);
   * **equals()**: Compara dos cadenas para ver si son iguales.
   * boolean sonIguales = cadena1.equals(cadena2);

Estos son solo algunos ejemplos básicos. ¿Hay algo específico que te gustaría saber más sobre operadores condicionales o de cadenas en Java?

Separadores

En Java, puedes utilizar varios métodos para dividir cadenas de texto utilizando separadores. Aquí te dejo algunas opciones comunes:

1. **Método split()**: Este método de la clase String te permite dividir una cadena en función de un separador específico, que puede ser un carácter, una expresión regular o una cadena de caracteres.
2. String texto = "uno,dos,tres";
3. String[] partes = texto.split(",");
4. for (String parte : partes) {
5. System.out.println(parte);
6. }
7. **Clase StringTokenizer**: Esta clase te permite dividir una cadena en función de un conjunto de delimitadores. Puedes utilizar caracteres simples o cadenas de caracteres como delimitadores.
8. import java.util.StringTokenizer;
9. String texto = "uno,dos,tres";
10. StringTokenizer tokenizer = new StringTokenizer(texto, ",");
11. while (tokenizer.hasMoreTokens()) {
12. System.out.println(tokenizer.nextToken());
13. }
14. **Método substring()**: Este método te permite extraer una parte específica de una cadena utilizando un índice de inicio y un índice de fin.
15. String texto = "uno,dos,tres";
16. String parte = texto.substring(0, 3); // "uno"
17. System.out.println(parte);

[Estos métodos te permiten manipular y analizar la información contenida en las cadenas de forma eficiente1](https://aprenderjava.net/base/dividir-cadena-en-java-usar-multiples-caracteres-como-separadores/)[2](https://www.programacionenjava.com/blog/2008/06/18/javabasico/separadores-en-java/)[3](https://parzibyte.me/blog/2019/08/08/java-separar-cadena-convertirla-arreglo/). ¿Hay algún método en particular que te gustaría explorar más a fondo?

Prioridad entreoperadores Aritmeticos

En Java, la prioridad de los operadores aritméticos sigue reglas específicas que determinan el orden en que se evalúan las expresiones. Aquí tienes un resumen de la prioridad de los operadores aritméticos, de mayor a menor:

1. **Paréntesis**: ()
2. **Incremento y decremento**: ++, --
3. **Multiplicación, división y módulo**: \*, /, %
4. **Suma y resta**: +, -

Por ejemplo, en la expresión int resultado = 10 + 5 \* 2;, la multiplicación se realiza antes que la suma, por lo que el resultado sería 20 y no 30.

Si necesitas cambiar el orden de evaluación, puedes usar paréntesis. [Por ejemplo, int resultado = (10 + 5) \* 2; dará como resultado 301](https://www.discoduroderoer.es/prioridad-operadores-java/).

¿Te gustaría ver un ejemplo más detallado o tienes alguna otra pregunta sobre Java?

Validaciones e ingreso de datos, Diseño de pruebas

¡Claro! Aquí tienes una guía básica sobre validaciones e ingreso de datos, así como el diseño de pruebas en Java.

**Validaciones e Ingreso de Datos**

**Validaciones**: En Java, puedes utilizar anotaciones para realizar validaciones de datos. Las anotaciones como @NotNull, @Size, y @Pattern son comunes para asegurar que los datos de entrada cumplen con ciertos criterios. Por ejemplo:

import javax.validation.constraints.NotNull;

import javax.validation.constraints.Size;

public class Usuario {

@NotNull(message = "El nombre no puede ser nulo")

@Size(min = 1, max = 50, message = "El nombre debe tener entre 1 y 50 caracteres")

private String nombre;

// Getters y setters

}

**Ingreso de Datos**: Para manejar el ingreso de datos, puedes usar formularios en aplicaciones web con frameworks como Spring Boot. Aquí, las validaciones se aplican automáticamente cuando los datos se envían al servidor.

**Diseño de Pruebas en Java**

**Pruebas Unitarias**: Utiliza JUnit para escribir pruebas unitarias. Estas pruebas verifican que métodos individuales funcionan correctamente. Un ejemplo básico sería:

import static org.junit.jupiter.api.Assertions.assertEquals;

import org.junit.jupiter.api.Test;

public class CalculadoraTest {

@Test

public void testSuma() {

Calculadora calc = new Calculadora();

assertEquals(5, calc.suma(2, 3));

}

}

**Pruebas de Integración**: Estas pruebas aseguran que diferentes partes de la aplicación funcionan juntas. Puedes usar herramientas como Spring Test para realizar pruebas de integración en aplicaciones Spring Boot.

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.test.context.SpringBootTest;

import org.springframework.test.context.junit4.SpringRunner;

import org.junit.runner.RunWith;

import org.junit.Test;

@RunWith(SpringRunner.class)

@SpringBootTest

public class UsuarioServiceTest {

@Autowired

private UsuarioService usuarioService;

@Test

public void testGuardarUsuario() {

Usuario usuario = new Usuario("Juan");

usuarioService.guardarUsuario(usuario);

// Verificar que el usuario se guardó correctamente

}

}

**Recursos Adicionales**

* [Guía completa para implementar validaciones en Java1](https://rodriguezizquierdo.com/guia-completa-para-implementar-validaciones-en-java-con-anotaciones-y-grupos-de-restricciones/)
* [Automatiza pruebas QA con Java2](https://www.pragma.co/es/blog/automatiza-pruebas-qa-con-java-parte-1-utilidades-avanzadas)

Espero que esto te sea útil. ¿Hay algo más específico que te gustaría saber?

Actualizar LECTURA

consultar los operadores.

dise˜nar una forma de emplear estos operadores o temas en una calculadora.

estructura de programaci´on

--- diagramas, métodos

cuidando la aplicaci´on del principio de Unica Responsabilidad y el Paradigma orientado a Objetos de manera adecuada (encapsulamiento, sobrecarga, calses, objetos, static, gesti´on de la excepciones, documentaci´on). Finalmente pruebe muy bien su c´odigo.

operaciones b´asicas con cada uno de los operadores

Necesito que crees una calculadora de operaciones en java que cumpla con lo siguiente:

Asignaciones

A continuación, se presentan los paradigmas a trabajar y el lenguaje de programación en que deben hacer la implementación de la calculadora de la vida, aplicando el paradigma dado.

1. Operadores Aritméticos

2. Operadores relacionales

3. Operadores de bits

4. Operadores booleanos

5. Operadores Incrementales y Aritmético combinados

6. Operador Condicional y de cadenas

7. Separadores

8. Prioridad entre operadores Aritméticos

9. Validaciones e ingreso de datos, Diseño de pruebas

El objetivo de la calculadora es utilizar los operadores del lenguaje de programación JAVA, mediante la creación de un programa que usando la clase “Lectura”, lea los datos desde teclado y permita realizar las operaciones básicas con cada uno de los operadores, que le fueron asignados. Por lo tanto, lo primero que deberás realizar, es consultar los operadores. A continuación, diseñar una forma de emplear estos operadores o temas en una calculadora. Luego armar la estructura de programación. Posteriormente implementar en el lenguaje Java la solución, cuidando la aplicación del principio de Única Responsabilidad y el Paradigma orientado a Objetos de manera adecuada (encapsulamiento, sobrecarga, clases, objetos, static, gestión de las excepcione). Finalmente pruebe muy bien su código.

Ingreso de datos: Todos los datos deben ser ingresados por teclado (Modificada con Buffer), deben ser validados adecuadamente. Para esto use la clase lectura.

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

BufferedReader bIn = new BufferedReader(new InputStreamReader(System.in));

Resultado: Deben imprimir en consola, todo el proceso de la acción del operador, de tal forma que el usuario que corre la aplicación entienda cómo funciona el operador, paso a paso.

Calidad: Aplique el principio de única responsabilidad. Mantenga la complejidad baja, es decir haga el proceso sencillo sin complicarse con conocimiento más avanzados. Enfóquese en solucionar el problema

Ten muy en cuenta los conceptos de interfaz (comportamiento y herencia múltiple), herencia (get and set), abstracción (códigos o métodos en común), sobrecargo, sobreescritura, asociación y composición