

**Lab Manual- Java Data Type**

**Prepared for**:

**Date:** 18th June 2024

**Prepared by:**

Document Name: Lab Manual **Document Number** SDLab333

**Contributor:**

Contents

[1. What are Java Data Type 3](#_Toc172221026)

[2. Create New Java Class - Arithmetic.java 3](#_Toc172221027)

[3. Create New Java Class - Arithmetic.java 5](#_Toc172221028)

[4. Create New Java Class – Increment.java 5](#_Toc172221029)

[5. Create New Java Class – Boolean.java 6](#_Toc172221030)

[6. Create New Java Class – Bits.java 7](#_Toc172221031)

[1. Key Concepts 8](#_Toc172221032)

[2. Step-by-Step Explanation 8](#_Toc172221033)

[7. Create New Java Class – Logical.java 9](#_Toc172221034)

[1. Key Concepts 10](#_Toc172221035)

[2. Step-by-Step Explanation 10](#_Toc172221036)

[3. Comparison with Bitwise Operations 11](#_Toc172221037)

[8. Create New Java Class – String.java 11](#_Toc172221038)

[9. Java Casting Example 12](#_Toc172221039)

[1. Explanation 14](#_Toc172221040)

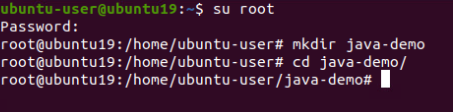
# What are Java Data Type

# Create New Java Class - Arithmetic.java

su root

mkdir java-demo

cd java-demo



nano Arithmetic.java



class Arithmetic {

public static void main(String[] args) {

int x = 12;

int y = 2 \* x;

System.out.println(y);

int z = (y - x) % 5;

System.out.println(z);

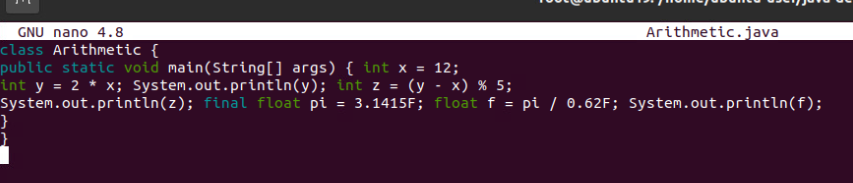
final float pi = 3.1415F;

float f = pi / 0.62F;

System.out.println(f);

}

}



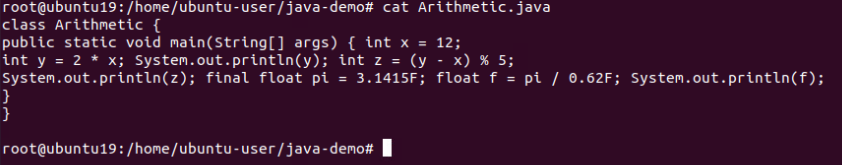
CTRL+X to save



Press Y and Enter

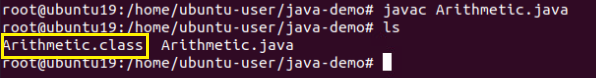


cat Arithmetic.java

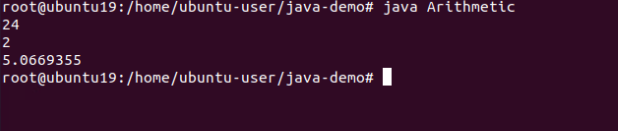


javac Arithmetic.java

ls



java Arithmetic



# Create New Java Class - Arithmetic.java

nano ShortHand.java



class ShortHand {

public static void main(String[] args) {

int x = 12;

x += 5; // x = x + 5

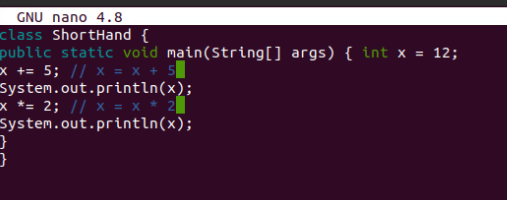
System.out.println(x);

x \*= 2; // x = x \* 2

System.out.println(x);

}

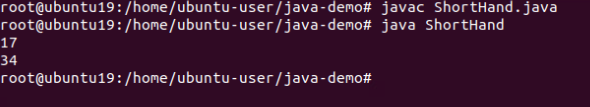
}



Save and Exit

javac ShortHand.java

java ShortHand



# Create New Java Class – Increment.java

nano Increment.java



class Increment {

public static void main(String[] args) {

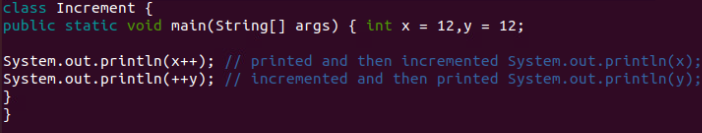
int x = 12,y = 12;

System.out.println(x++); // printed and then incremented System.out.println(x);

System.out.println(++y); // incremented and then printed System.out.println(y);

}

}

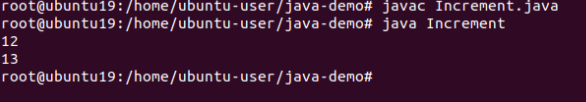


Save and Exit

Execute below command to run

Javac Increment.java

Java Increment



# Create New Java Class – Boolean.java

nano Boolean.java



class Boolean {

public static void main(String[] args) {

int x = 12,y = 33;

System.out.println(x < y);

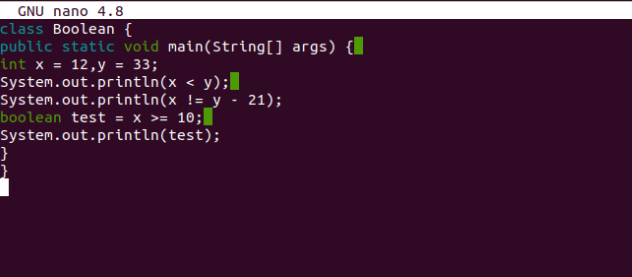
System.out.println(x != y - 21);

boolean test = x >= 10;

System.out.println(test);

}

}

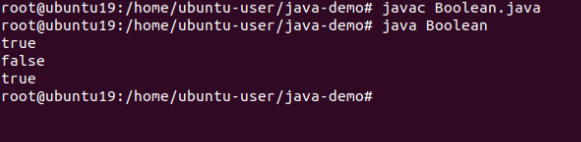


Save and Exit

Execute below command to run

Javac Boolean.java

Java Boolean



# Create New Java Class – Bits.java

This is a Java program that demonstrates bitwise operations using hexadecimal numbers.

nano Bits.java



class Bits {

public static void main(String[] args) {

int x = 0x16; // 00000000000000000000000000010110

int y = 0x33; // 00000000000000000000000000110011

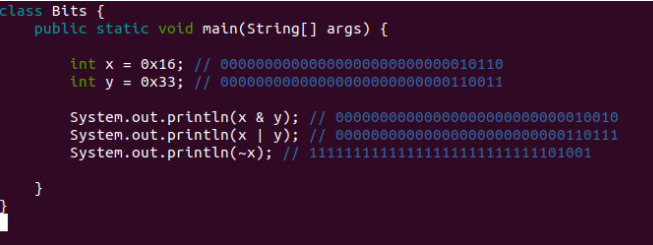
System.out.println(x & y);// 00000000000000000000000000010010

System.out.println(x | y);// 00000000000000000000000000110111

System.out.println(˜x); // 11111111111111111111111111101001

}

}

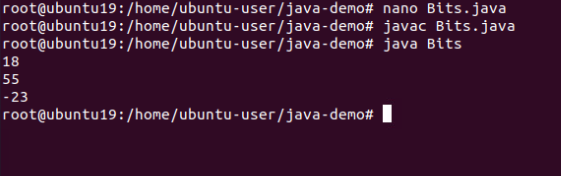


Save and Exit

Execute below command to run

Javac Bits.java

Java Bits

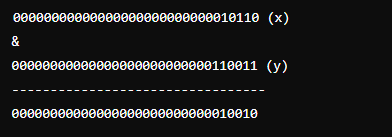


### Key Concepts

1. **Hexadecimal Numbers**:
   * 0x16 in hexadecimal is 22 in decimal.
   * 0x33 in hexadecimal is 51 in decimal.
2. **Bitwise Operations**:
   * **AND (&)**: Compares each bit of two numbers. If both bits are 1, the result is 1, otherwise, it's 0.
   * **OR (|)**: Compares each bit of two numbers. If at least one of the bits is 1, the result is 1, otherwise, it's 0.
   * **NOT (~)**: Inverts all the bits of a number. If a bit is 0, it becomes 1, and vice versa.

### Step-by-Step Explanation

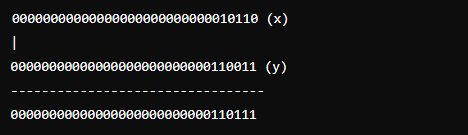
1. **Initialization**:
   * int x = 0x16;
     + x is assigned the value 22, which in binary is 00000000000000000000000000010110.
   * int y = 0x33;
     + y is assigned the value 51, which in binary is 00000000000000000000000000110011.
2. **Bitwise AND (&)**:
   * System.out.println(x & y);
   * Perform the AND operation



*  The result is 0x12 in hexadecimal or 18 in decimal.

 **Bitwise OR (|)**:

* System.out.println(x | y);
* Perform the OR operation



The result is 0x37 in hexadecimal or 55 in decimal.

# Create New Java Class – Logical.java

* **Logical operations** work with **boolean values** and are used to control the flow of a program based on conditions.
* **Bitwise operations** work with **bits of integer values** and are used for more **low-level data** manipulation.

nano Logical.java



class Logical {

public static void main(String[] args) {

int x = 12,y = 33;

double d = 2.45,e = 4.54;

System.out.println(x < y && d < e);

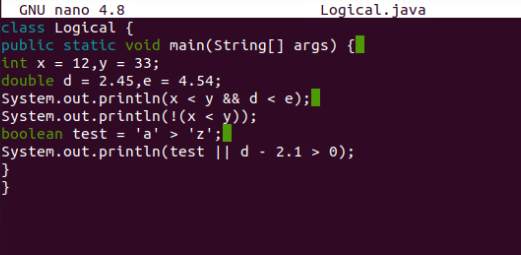
System.out.println(!(x < y));

boolean test = ‘a’ > ‘z’;

System.out.println(test || d - 2.1 > 0);

}

}

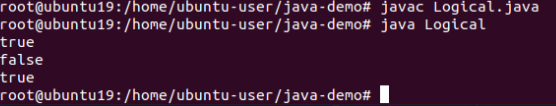


Save and Exit

Execute below command to run

Javac Logical.java

Java Logical



### Key Concepts

1. **Logical Operations**:
   * **AND (&&)**: Returns true if both conditions are true, otherwise returns false.
   * **OR (||)**: Returns true if at least one of the conditions is true, otherwise returns false.
   * **NOT (!)**: Inverts the truth value of a condition. If the condition is true, it returns false, and vice versa.

### Step-by-Step Explanation

1. **Initialization**:
   * int x = 12, y = 33;
     + x is assigned the value 12.
     + y is assigned the value 33.
   * double d = 2.45, e = 4.54;
     + d is assigned the value 2.45.
     + e is assigned the value 4.54.
2. **Logical AND (&&)**:
   * System.out.println(x < y && d < e);
   * Check if both conditions are true:
     + x < y (12 < 33) is true.
     + d < e (2.45 < 4.54) is true.
   * Since both conditions are true, the result is true.
3. **Logical NOT (!)**:
   * System.out.println(!(x < y));
   * Check the condition and invert its result:
     + x < y (12 < 33) is true.
     + !(x < y) inverts true to false.
   * The result is false.
4. **Logical OR (||)**:
   * boolean test = 'a' > 'z';
     + 'a' > 'z' compares the ASCII values of characters.
     + ASCII value of 'a' is 97 and 'z' is 122.
     + 97 > 122 is false, so test is assigned false.
   * System.out.println(test || d - 2.1 > 0);
   * Check if at least one condition is true:
     + test is false.
     + d - 2.1 > 0 (2.45 - 2.1 > 0) is true (0.35 > 0).
   * Since one of the conditions is true, the result is true.

### Comparison with Bitwise Operations

* **Logical Operations**:
  + Operate on boolean values (true or false).
  + Used for conditional statements and control flow in programs.
  + Example: x < y && d < e, test || d - 2.1 > 0.
* **Bitwise Operations**:
  + Operate on individual bits of integer values.
  + Used for low-level programming, manipulating data at the bit level.
  + Example: x & y, x | y, ~x.

# Create New Java Class – String.java

nano Strings.java



class Strings {

public static void main(String[] args) {

String s1 = "Hello" + " World!";

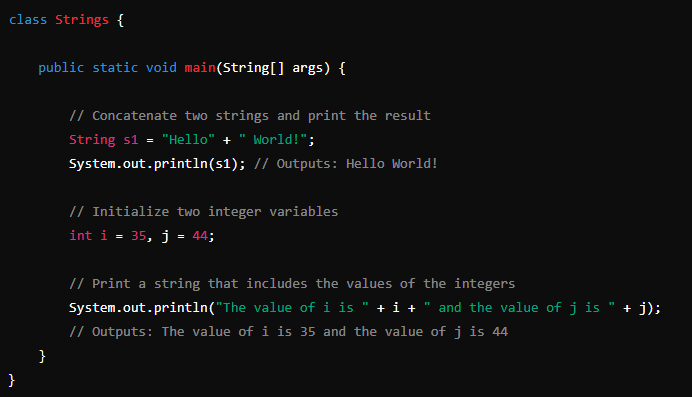
System.out.println(s1);

int i = 35,j = 44;

System.out.println("The value of i is " + i +" and the value of j is " + j);

}

}

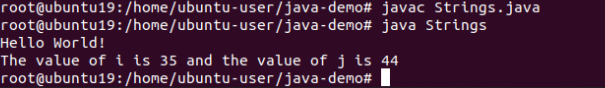


Save and Exit

Execute below command to run

Javac String.java

Java String



# Java Casting Example

nano TestCast.java



class TestCast {

public static void main(String[] args) {

int a = 'x'; // 'x' is a character

long b = 34; // 34 is an int

float c = 1002; // 1002 is an int

double d = 3.45F; // 3.45F is a float

// Print the initial values

System.out.println("Initial values:");

System.out.println("int a = 'x' -> a = " + a); // 'x' is 120 in ASCII

System.out.println("long b = 34 -> b = " + b);

System.out.println("float c = 1002 -> c = " + c);

System.out.println("double d = 3.45F -> d = " + d);

// Cast and print the values

char castToChar = (char) a;

int castToIntFromLong = (int) b;

int castToIntFromFloat = (int) c;

float castToFloatFromDouble = (float) d;

System.out.println("\nValues after casting:");

System.out.println("char castToChar = (char) a -> castToChar = " + castToChar);

System.out.println("int castToIntFromLong = (int) b -> castToIntFromLong = " + castToIntFromLong);

System.out.println("int castToIntFromFloat = (int) c -> castToIntFromFloat = " + castToIntFromFloat);

System.out.println("float castToFloatFromDouble = (float) d -> castToFloatFromDouble = " + castToFloatFromDouble);

}

}

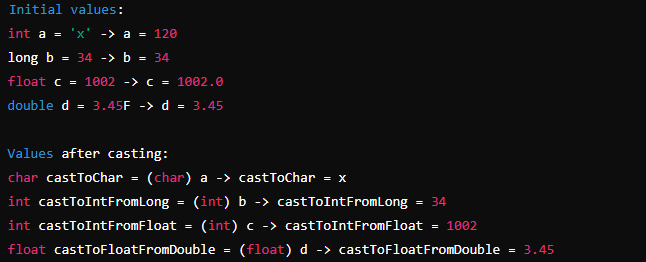


Save and Exit

Execute below command to run

Javac TestCast.java

Java TestCast



### Explanation

1. **Initial Values**:
   * int a = 'x';
     + 'x' is a character with an ASCII value of 120. This value is stored in a.
   * long b = 34;
     + 34 is an integer, and it can be directly assigned to a long variable without explicit casting.
   * float c = 1002;
     + 1002 is an integer, and it can be directly assigned to a float variable. It will be stored as 1002.0.
   * double d = 3.45F;
     + 3.45F is a float, and it can be directly assigned to a double variable. It will be stored as 3.45.
2. **Casting Values**:
   * char castToChar = (char) a;
     + This casts the integer value of a back to a character. Since a was initialized with 'x', castToChar will be 'x'.
   * int castToIntFromLong = (int) b;
     + This casts the long value b back to an integer. Since b was initialized with 34, castToIntFromLong will be 34.
   * int castToIntFromFloat = (int) c;
     + This casts the float value c back to an integer. Since c was initialized with 1002.0, castToIntFromFloat will be 1002.
   * float castToFloatFromDouble = (float) d;
     + This casts the double value d back to a float. Since d was initialized with 3.45, castToFloatFromDouble will be 3.45