Print binary format

print number in binary format System.out.println("~x"+ (~x));

Let's break down what this Java line does and how to **print a number in binary format** instead.

Your line:

```java
System.out.println("~x" + (~x));
```

Explanation:

- `~x` is the **bitwise NOT** operator — it inverts all bits of `x`.
- The result of `~x` is printed, but **in decimal form** by default.

For example:

```java
int x = 5;
System.out.println("~x" + (~x));
```

Output:

```
~x-6
```

Because the bitwise NOT of `5` (which is `0000 0101` in binary) is `1111 1010`, which equals `-6` in two's complement.

✅ To print in binary format, use `Integer.toBinaryString()`:

```java
int x = 5;
System.out.println("x in binary: " + Integer.toBinaryString(x));
System.out.println("~x in binary: " + Integer.toBinaryString(~x));
```

Output:

```php
x in binary: 101
```

```
~x in binary: 11111111111111111111111111111010
```

🧠 Optional: Limit to 8 bits (for readability)

You can mask the binary string to 8 bits:

```
java

System.out.println("~x (8 bits): " + String.format("%8s", Integer.toBinaryString(~x &
0xFF)).replace(' ', '0'));
```

Output:

```
scss

~x (8 bits): 11111010
```

Would you like me to show how to print the **bitwise NOT result step-by-step** (like showing both operands and result in binary)?

write following all to print in binary format

Sure! Here's a **complete Java example** that prints several bitwise operations — including the original number, its complement, AND, OR, XOR, and left/right shifts — all **in binary format**:

✅ Full Java Example

```
java

public class BinaryPrintExample {
    public static void main(String[] args) {
        int x = 5;  // example value
        int y = 3;  // another example value

        // Print in binary format (8 bits for clarity)
        System.out.println("x        = " + toBinary(x));
        System.out.println("y        = " + toBinary(y));
        System.out.println("~x       = " + toBinary(~x));
        System.out.println("x & y    = " + toBinary(x & y));
        System.out.println("x | y    = " + toBinary(x | y));
        System.out.println("x ^ y    = " + toBinary(x ^ y));
        System.out.println("x << 1   = " + toBinary(x << 1));
        System.out.println("x >> 1   = " + toBinary(x >> 1));
        System.out.println("x >>> 1  = " + toBinary(x >>> 1)); // unsigned right shift
    }

    // Helper method: convert to 8-bit binary string
    private static String toBinary(int n) {
        return String.format("%8s", Integer.toBinaryString(n & 0xFF)).replace(' ', '0');
    }
}
```

🧩

```
byte x, y;
x = 10;
y = 11;
System.out.println("~x=" + (~x));
System.out.println("x&y = " + (x & y));
System.out.println("x|y = " + (x | y));
System.out.println(("x^y=" + (x ^ y)));
System.out.println("x<<2=" + (x << 2));
System.out.println("x>>2=" + (x >> 2));
System.out.println("x>>>2=" + (x >>> 2));
```

Perfect 👍 — that's a good Java snippet showing bitwise operations on `byte` variables.

Right now, your code prints the **results in decimal form**.
If you want to print all of them in **binary format**, you can modify it like this 👇

✅ Full Java Code (prints everything in binary)

```java
public class BitwiseBinaryExample {
    public static void main(String[] args) {
        byte x, y;
        x = 10;  // 00001010
        y = 11;  // 00001011

        System.out.println("x     = " + toBinary(x));
        System.out.println("y     = " + toBinary(y));
        System.out.println("~x    = " + toBinary(~x));
        System.out.println("x & y = " + toBinary(x & y));
        System.out.println("x | y = " + toBinary(x | y));
        System.out.println("x ^ y = " + toBinary(x ^ y));
        System.out.println("x << 2 = " + toBinary(x << 2));
        System.out.println("x >> 2 = " + toBinary(x >> 2));
        System.out.println("x >>> 2 = " + toBinary(x >>> 2));
    }

    // Helper method: convert to 8-bit binary string
    private static String toBinary(int n) {
        // "& 0xFF" ensures we only keep the lower 8 bits
        return String.format("%8s", Integer.toBinaryString(n & 0xFF)).replace(' ', '0');
    }
}
```

🧩 Output (for `x=10`, `y=11`)

```bash
x     = 00001010
y     = 00001011
~x    = 11110101
x & y = 00001010
x | y = 00001011
```

```
x ^ y = 00000001
x << 2 = 00101000
x >> 2 = 00000010
x >>> 2 = 00000010
```

Would you like me to show the **decimal equivalents alongside the binary output** (e.g., **`x = 00001010 (10)`**)?

function in java to take input from user

Great question 👍

In Java, you can take input from the user using the **`Scanner`** class from the **`java.util`** package.
Here's how to create a **function** that reads input from the user.

✅ Example: Function to take input from the user

```java
import java.util.Scanner;

public class UserInputExample {

    // Function to take integer input
    static int getIntInput(String message) {
        Scanner sc = new Scanner(System.in);  // Create a Scanner object
        System.out.print(message);             // Display prompt
        int value = sc.nextInt();              // Read integer input
        return value;
    }

    public static void main(String[] args) {
        int x = getIntInput("Enter value for x: ");
        int y = getIntInput("Enter value for y: ");

        System.out.println("You entered:");
        System.out.println("x = " + x);
        System.out.println("y = " + y);
    }
}
```

🧠 Notes:

- **`Scanner`** can read **different data types**:

  ```java
  nextInt()      // for integers
  nextFloat()    // for floating-point numbers
  nextDouble()   // for double
  nextLine()     // for full line of text
  next()         // for a single word
  nextBoolean()  // for true/false
  ```

- Don't forget to **`import java.util.Scanner;`** at the top.

- Normally you should **reuse** one `Scanner` object instead of creating many inside functions, but for small programs like this, it's fine.

🔹 Example with Multiple Types

```java
static String getStringInput(String message) {
    Scanner sc = new Scanner(System.in);
    System.out.print(message);
    return sc.nextLine();
}
```

Would you like me to show how to take **multiple inputs (like x and y) on the same line** (e.g., `10 20`) using one Scanner call?

InputReaderStream kya hai fir

Excellent question 🙌 —
Let's clear this up: `InputStreamReader` (not *InputReaderStream*) is a real Java class, and it plays an important role in reading user input — but it works **a bit lower level** than `Scanner`.

Let's break it down 👇

🧩 1. What is `InputStreamReader`?

`InputStreamReader` is a **bridge between byte streams and character streams**.
It converts bytes (like those coming from the keyboard or a file) into characters that Java can read.

The most common usage:

```java
InputStreamReader reader = new InputStreamReader(System.in);
```

Here:

- `System.in` = standard input stream (keyboard)
- `InputStreamReader` = converts bytes → characters

🧩 2. Typically used with `BufferedReader`

Because `InputStreamReader` reads only one character at a time, developers usually wrap it in a `BufferedReader` to make reading easier and faster:

```java
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.IOException;

public class InputExample {
    public static void main(String[] args) throws IOException {
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

        System.out.print("Enter your name: ");
        String name = br.readLine();   // reads an entire line

        System.out.println("Hello, " + name + "!");
    }
}
```

⚖️ Comparison: `Scanner` vs `InputStreamReader`

Feature	`Scanner`	`BufferedReader + InputStreamReader`
Reads types (int, double, etc.)	✅ Yes	❌ No (reads only strings)
Reads a full line	✅ Yes (**`nextLine()`**)	✅ Yes (**`readLine()`**)
Easy for beginners	✅ Easier	⚠️ Slightly harder
Performance	⚠️ Slower	✅ Faster for large inputs
Typical use	Small console programs	Competitive programming / large data input

---

🧠 Summary

- ✅ **`Scanner`** → high-level, convenient for beginners and basic input
- ⚙️ **`InputStreamReader`** → low-level, converts bytes → characters
- 💨 **`BufferedReader` + `InputStreamReader`** → efficient combo for fast input

Would you like me to show a version of your **bitwise program** that takes input using **`BufferedReader`** instead of **`Scanner`**?