RandomClass in Java

1. `Random` Class Basics

What is it?

The `Random` class in Java is part of the `java.util` package. It is used to generate **random numbers**, such as integers, floating-point numbers, and booleans.

How to Use It?

Import the class

```
import java.util.Random;
```

Create a Random object

```
Random rand = new Random();
```

This object will now allow you to generate random numbers using its methods.

Useful Methods

Method	Description	Example Output
`nextInt()`	Returns any int (can be negative)	-5463728, 23423, etc.
`nextInt(bound)`	Returns int from 0 to bound-1	`rand.nextInt(5)` → 0–4
`nextDouble()`	Returns a double from 0.0 to 1.0	0.3723, 0.945, etc.
`nextBoolean()`	Returns true or false	`true` or `false`

Example

```
import java.util.Random;

public class Main {
    public static void main(String[] args) {
        Random rand = new Random();

        int anyInt = rand.nextInt();            // could be negative
        int zeroToNine = rand.nextInt(10);      // 0 to 9
        double randomDouble = rand.nextDouble(); // 0.0 to 1.0
        boolean randomBool = rand.nextBoolean(); // true or false

        System.out.println("Any int: " + anyInt);
        System.out.println("0-9: " + zeroToNine);
        System.out.println("Double: " + randomDouble);
        System.out.println("Boolean: " + randomBool);
```

```
    }
}
```

✅ 2. Generating Numbers in a Specific Range

Goal:

Generate random integers **within a fixed range**, like:

- 0 to 8
- 1 to 9
- 0 to 89
- 1 to 90
- 0 to 899
- 1 to 900

Using `rand.nextInt(bound)`

```
Random rand = new Random();
int num = rand.nextInt(9); // generates 0 to 8
```

This always gives numbers from `0` (inclusive) to `bound - 1` (exclusive).

Code	Range of values
`rand.nextInt(9)`	0 to 8
`rand.nextInt(90)`	0 to 89
`rand.nextInt(900)`	0 to 899

Adding `+1` to shift the range

Sometimes, you want the range to **start at 1** instead of 0.

```
int num = rand.nextInt(9) + 1; // generates 1 to 9
```

Code	Range of values
`rand.nextInt(9) + 1`	1 to 9
`rand.nextInt(90) + 1`	1 to 90
`rand.nextInt(900) + 1`	1 to 900

Example

```
Random rand = new Random();

int oneToNine = rand.nextInt(9) + 1;       // 1–9
int oneToNinety = rand.nextInt(90) + 1;    // 1–90
int oneToNineHundred = rand.nextInt(900) + 1; // 1–900
```

```
System.out.println("1 to 9: " + oneToNine);
System.out.println("1 to 90: " + oneToNinety);
System.out.println("1 to 900: " + oneToNineHundred);
```

Why use this method instead of `%`?

- It's **cleaner**
- It **doesn't need** `Math.abs()`
- It avoids possible **bugs** with negative numbers or edge cases

3. Using `%` Modulo Operator with Random

What is `%`?

The **modulo operator (`%`)** gives the **remainder** of a division.

Example:

```
10 % 3 = 1
7 % 5 = 2
```

Using `%` with Random

We can apply `%` to random numbers to limit the range. Example:

```
Random rand = new Random();
int num = Math.abs(rand.nextInt());
int result = num % 9; // gives 0–8
```

Why use `Math.abs()`? Because `rand.nextInt()` can return **negative** numbers, and `%` with negatives gives **unexpected** results.

Ranges with `%`

Code	Resulting Range
`Math.abs(rand.nextInt()) % 9`	0–8
`Math.abs(rand.nextInt()) % 90`	0–89
`Math.abs(rand.nextInt()) % 900`	0–899

Adding `+1` for 1-based ranges

Code	Resulting Range
`(Math.abs(rand.nextInt()) % 9) + 1`	1–9
`(Math.abs(rand.nextInt()) % 90) + 1`	1–90
`(Math.abs(rand.nextInt()) % 900) + 1`	1–900

Example

```
Random rand = new Random();
int num = Math.abs(rand.nextInt());

int oneToNine = (num % 9) + 1;        // 1–9
int oneToNinety = (num % 90) + 1;     // 1–90
int oneToNineHundred = (num % 900) + 1; // 1–900

System.out.println("1 to 9: " + oneToNine);
System.out.println("1 to 90: " + oneToNinety);
System.out.println("1 to 900: " + oneToNineHundred);
```

Why Not Use `%` All the Time?

- It's more **error-prone**
- Needs `Math.abs()` to avoid negative numbers
- Not recommended when `nextInt(bound)` gives the same result more safely

4. `Math.abs()` to Handle Negative Values

What is `Math.abs()`?

`Math.abs(x)` returns the **absolute value** of a number.

That means it turns **negative numbers into positive ones**.

```
Math.abs(-10) → 10
Math.abs(7) → 7
```

Why Use `Math.abs()` with Random?

When you use `rand.nextInt()` **without a bound**, it can return **negative numbers**.

```
Random rand = new Random();
int num = rand.nextInt(); // might be negative like -43278
```

If you want to do something like `num % 90`, a negative value can give **unexpected results** (negative modulo).

So you use:

```
int safeNum = Math.abs(rand.nextInt());
```

This ensures `safeNum` is **non-negative**.

Example

```
Random rand = new Random();
int num = Math.abs(rand.nextInt());
```

```
int oneTo90 = (num % 90) + 1; // always between 1 and 90
```

Important Edge Case

There's **one number** that `Math.abs()` can't fix:

```
int min = Integer.MIN_VALUE; // -2147483648
System.out.println(Math.abs(min)); // still -2147483648!
```

Because `Integer.MIN_VALUE` has **no positive equivalent** in 32-bit signed integers (this is just how binary works in Java).

This is rare, but good to **be aware** of.

When to Use `Math.abs()`

Situation	Use `Math.abs()`?
`rand.nextInt()` alone	Yes
`rand.nextInt(bound)`	No (already safe)
`rand.nextInt() % X`	Yes

5. Best Practices for Using `Random` in Java

These are the **recommended ways** to generate random values safely and clearly.

Use `rand.nextInt(bound)` for Ranges

Instead of using `%` and `Math.abs()`, always use `nextInt(bound)` when you can.

Example:

```
Random rand = new Random();

int zeroToEight = rand.nextInt(9);       // 0–8
int oneToNine = rand.nextInt(9) + 1;     // 1–9
int oneToNinety = rand.nextInt(90) + 1;  // 1–90
int oneTo900 = rand.nextInt(900) + 1;    // 1–900
```

Avoid Using `%` with `rand.nextInt()`

Even if it works, it's not the best way and may give unexpected results without `Math.abs()`.

Avoid this:

```
int num = Math.abs(rand.nextInt());
int oneToNine = (num % 9) + 1;
```

Use this instead:

```
int oneToNine = rand.nextInt(9) + 1;
```

Don’t Forget to Import

```
import java.util.Random;
```

Without this import, the code won’t compile.

Reuse the `Random` Object

Instead of creating a new `Random` object every time, reuse one:

Bad:

```
int a = new Random().nextInt(10);
int b = new Random().nextInt(10);
```

Good:

```
Random rand = new Random();
int a = rand.nextInt(10);
int b = rand.nextInt(10);
```

Optional: Use Seeding for Reproducible Results

If you want the same "random" sequence every time (for testing):

```
Random rand = new Random(123); // Fixed seed
```

Summary Table: What to Do

Goal	Do This
0–8	`rand.nextInt(9)`
1–9	`rand.nextInt(9) + 1`
1–90	`rand.nextInt(90) + 1`
Avoid negatives	Use `nextInt(bound)`, not `%`
Reproducible random	Use `new Random(seed)`