JAVA DEVELOPER INTERNSHIP BY ELEVATE LABS

Interview Questions:

1.What are Java loops?
2.What is enhanced for-loop?
3.How do you handle multiple user inputs?
4.How is a switch-case different from if-else?
5.What are collections in Java?
6.What is ArrayList?
7.How to iterate using iterators?
8.What is a Map?
9.How to sort a list?
10.How to shuffle elements?

1. What are Java Loops?

Definition:

A loop is a control structure in Java that **executes a block of code repeatedly** until a certain condition is met. Loops help reduce code duplication and handle repetitive tasks efficiently.

👉 **Types of Loops in Java:**

1. **for loop** – Used when we know the number of iterations beforehand.
2. **while loop** – Used when the condition is checked before execution, and we don't know iterations in advance.
3. **do-while loop** – Similar to while, but guarantees at least one execution because the condition is checked later.

👉 **Working Flow:**

Start → Check condition → If true → Execute code → Go back → Repeat

↓

If false → Exit loop

👉 **Code Example:**

```
for (int i = 1; i <= 5; i++) {
   System.out.println("Hello " + i);
}
```

Output:

Hello 1
Hello 2
Hello 3
Hello 4
Hello 5

👉 **Why useful?**

- Saves time and code.
- Handles repeated logic dynamically.

Real-Life Examples:

1. A washing machine drum rotates until the timer ends.
2. A cashier scanning items until the cart is empty.
3. Streetlights automatically turning on every evening in a loop daily.

2.What is enhanced for-loop?

Explanation (in depth)

Definition:

The **enhanced for-loop** (or for-each loop) is a simpler version of the traditional for loop used to iterate through **arrays and collections** without worrying about indexes.

👉 **Advantages:**

- Cleaner and more readable code.
- Avoids IndexOutOfBoundsException.
- Best when we need to **read** items, not modify them.

👉 **Syntax:**

```
for (datatype element : collection) {
   // use element
}
```

👉 **Code Example:**

```
String[] cars = {"BMW", "Tesla", "Audi"};
for (String car : cars) {
   System.out.println(car);
}
```

Output:

BMW
Tesla
Audi

Real-Life Examples:

1. Reading emails one by one from inbox.
2. Teacher taking attendance roll call.
3. Displaying each song in a playlist.

3) How do you handle multiple user inputs?

Definition:

Java provides the **Scanner class** (from java.util) to take user input. We can take multiple inputs by calling different scanner methods like:

- nextLine() → for Strings (whole line)
- nextInt() → for integers
- nextDouble() → for floating numbers

👉 **Code Example:**

```
Scanner sc = new Scanner(System.in);
System.out.print("Enter name: ");
String name = sc.nextLine();
System.out.print("Enter age: ");
int age = sc.nextInt();
System.out.print("Enter salary: ");
double salary = sc.nextDouble();
System.out.println(name + " | " + age + " | " + salary);
```

👉 **Why useful?**

- Makes interactive console applications.
- Handles different data types.

Real-Life Examples:

1. ATM asks for PIN, then withdrawal amount.
2. Railway booking form asking name, age, gender.
3. E-commerce checkout asking product, address, and payment.

4.How is a Switch-Case different from If-Else?

If-Else:

- Good for **range-based** or **complex conditions**.
- Slower for many conditions.

👉 **Switch-Case:**

- Good for **single variable matching with fixed values**.
- Cleaner than multiple if-else.

👉 **Code Example:**

```
int day = 2;
switch(day) {
   case 1: System.out.println("Monday"); break;
   case 2: System.out.println("Tuesday"); break;
   case 3: System.out.println("Wednesday"); break;
   default: System.out.println("Invalid");
}
```

👉 **Why use Switch?**

- More readable when handling fixed options.

Real-Life Examples:

1. Vending machine choosing drink based on button pressed.
2. Lift (Elevator) selecting floor number.
3. Mobile volume button → mute, low, medium, high

5. What are Collections in Java?

Definition:

The **Collections Framework** in Java is a set of classes & interfaces to store and manipulate groups of objects efficiently.

👉 **Main Interfaces:**

- **List** → ordered, allows duplicates (ArrayList, LinkedList).
- **Set** → unique elements (HashSet, TreeSet).
- **Map** → key-value pairs (HashMap, TreeMap).

👉 **Why use Collections instead of Arrays?**

- Arrays have fixed size.
- Collections can grow/shrink dynamically.
- Provides inbuilt methods for sorting, searching, etc.

👉 **Code Example:**

```
ArrayList<String> fruits = new ArrayList<>();
fruits.add("Apple");
fruits.add("Banana");
System.out.println(fruits);
```

Real-Life Examples:

1. Playlist of songs (List).
2. Unique Aadhaar numbers (Set).
3. Dictionary storing word → meaning (Map).

6.What is ArrayList?

Definition:

ArrayList is a **resizable array implementation of List interface**. Unlike arrays, its size increases or decreases dynamically.

👉 **Features:**

- Stores duplicate elements.
- Maintains insertion order.
- Allows random access.

👉 **Code Example:**

```
ArrayList<Integer> numbers = new ArrayList<>();
numbers.add(10);
numbers.add(20);
numbers.add(30);
System.out.println(numbers);
```

👉 **Why use it?**

- Flexible size.
- Useful when we don’t know the number of elements beforehand.

Real-Life Examples:

1. Shopping cart in Amazon (items keep changing).
2. Recent call logs on phone.
3. Task list in To-do apps.

7.How to iterate using Iterators?

👉 **Definition:**

Iterator is an object that lets us **traverse through collections** one element at a time.

👉 **Methods:**

- hasNext() → checks if more elements exist.
- next() → returns next element.
- remove() → removes last returned element.

👉 **Code Example:**

```
ArrayList<String> names = new ArrayList<>();
names.add("Amit");
names.add("Rita");

Iterator<String> it = names.iterator();
while (it.hasNext()) {
   System.out.println(it.next());
}
```

Real-Life Examples:

1. Reading books page by page.
2. Going through images in gallery.
3. Browsing YouTube videos in a playlist.

8.What is a Map?

👉 **Definition:**

Map stores **key-value pairs** where each key is unique but values can repeat.

👉 **Types:**

- **HashMap** → fast, unordered.
- **TreeMap** → sorted by key.
- **LinkedHashMap** → maintains insertion order.

👉 **Code Example:**

```
HashMap<String, Integer> marks = new HashMap<>();
marks.put("Raj", 85);
marks.put("Simran", 90);
System.out.println(marks);
```

👉 **Why use Map?**

- Best for lookup/search by unique key.

Real-Life Examples:

1. Student Roll No → Marks.
2. Employee ID → Salary.
3. Dictionary Word → Meaning.

9.How to Sort a List?

👉 **Definition:**

Sorting means arranging elements in order (ascending or descending).

We use Collections.sort() for simple data and Comparator for custom sorting.

👉 **Code Example:**

```
ArrayList<Integer> nums = new ArrayList<>();
nums.add(50); nums.add(20); nums.add(90);
Collections.sort(nums);
System.out.println(nums);
```

👉 **Custom Sort Example:**

```
Collections.sort(nums, Collections.reverseOrder());
```

Real-Life Examples:

1. Sorting students by marks.
2. Sorting Amazon products by price.
3. Sorting contacts alphabetically.

10.How to Shuffle Elements?

👉 **Definition:**

Shuffling means rearranging elements randomly. We use Collections.shuffle() to randomize list order.

👉 **Code Example:**

```
ArrayList<String> deck = new ArrayList<>();
deck.add("Ace"); deck.add("King"); deck.add("Queen");
Collections.shuffle(deck);
System.out.println(deck);
```

👉 **Why useful?**

- Randomization for fairness.
- Used in games and test systems.

Real-Life Examples:

1. Shuffling songs in music player.
2. Shuffling cards in card game.
3. Randomizing exam questions.