🔹 **What is Java?**

Java ek **high-level, object-oriented, platform-independent programming language** hai, jiska use **software, web applications, mobile apps (Android)** aur **automation testing (Selenium)** me hota hai.

👉 Java ka main focus hai:
✓ Reliability
✓ Security
✓ Write Once, Run Anywhere (WORA)

🔹 **History of Java**

- **1991** → Java ka development **Sun Microsystems** me start hua
- **James Gosling** ko Java ka **Father** mana jata hai
- Pehle Java ka naam **Oak** tha
- Baad me naam change hokar **Java** rakha gaya
- **1995** → Java officially launch hua
- **2010** → Oracle ne Sun Microsystems ko acquire kar liya

📌 *Automation point of view:*
Java stable aur secure hone ki wajah se **Selenium automation** me sabse zyada use hoti hai.

🔹 **Introduction to Java**

Java ek **object-oriented programming language** hai jo **bytecode** me convert hoti hai aur **JVM (Java Virtual Machine)** par run karti hai.

▶ **Java Working Flow:**

.java file → Compiler → .class (bytecode) → JVM → Output

🔹 **Features of Java**

1️⃣ **Platform Independent**
👉 Java code Windows, Linux, Mac sab par run hota hai

2️⃣ **Object Oriented**
👉 Class, Object, Inheritance, Polymorphism support karta hai

3️⃣ **Secure**
👉 No pointer, strong memory management

4️⃣ **Robust**
👉 Exception handling, garbage collection

5️⃣ **Multithreaded**
👉 Multiple tasks ek saath execute

6️⃣ **High Performance**
👉 JIT compiler use karta hai

🔹 **Why Java is Used in Automation Testing?**

✓ Selenium Java support karta hai
✓ Large community support
✓ Easy to learn
✓ Strong OOPS concept
✓ TestNG, Maven, Jenkins easily integrate

🔹 **Simple Java Example**

```
class Demo{
 public static void main(String[] args){
  System.out.println("Welcome to Java");
 }
}
```

📌 *Automation Example:*
Login automation, browser launch, test case execution sab Java se hota hai.

🔹 **Architecture of Java**

Java architecture ka matlab hai **Java program kaise compile, load aur execute hota hai**.

▶ **Java Execution Flow**

.java file
↓ (javac compiler)
.class file (Bytecode)
↓
JVM
↓
Output

📌 *Automation me*: Selenium scripts bhi isi process se run hote hain.

🔹 **JDK, JRE, JVM (MOST IMPORTANT)**

▶ JVM (Java Virtual Machine)

- JVM Java bytecode ko **machine code** me convert karta hai
- JVM har OS ke liye alag hoti hai
- Java ko **platform independent** banata hai

📌 *Example*:
Windows JVM → Windows par run
Linux JVM → Linux par run

▶ JRE (Java Runtime Environment)

JRE = **JVM + Libraries**

- JVM ko run karne ka environment deta hai
- Java program **execute** karne ke liye chahiye

📌 *Note*:
Sirf run karna hai → JRE enough
Automation me run ke liye bhi JRE use hota hai

▶ JDK (Java Development Kit)

JDK = **JRE + Development Tools**

Includes:

- javac (compiler)
- debugger
- JVM
- Libraries

📌 *Automation me*:
Selenium code likhne ke liye **JDK mandatory** hai

▶ Difference Table

JDK	JRE	JVM
Development + Run	Only Run	Executes code
Compiler included	No compiler	No compiler

JDK	JRE	JVM
Developers use	End users	System

🔹 **Memory Areas in Java (🔥 Interview Favorite)**

1️⃣ Heap Area

- Objects yahan store hote hain
- Runtime memory
- Garbage Collector yahin kaam karta hai

📌 *Automation Example*:
Page Object, WebDriver object heap me store hota hai

2️⃣ Method Area

- Class level data store hota hai
- Static variables
- Methods
- Class metadata

📌 *Example*:
static WebDriver driver;

3️⃣ Stack Area

- Method execution
- Local variables
- Each thread ka alag stack hota hai

📌 *Automation Example*:
login() method call hone par stack me entry banti hai

4️⃣ PC Register (Program Counter)

- Current executing instruction ka address rakhta hai
- Thread wise hota hai

📌 *Simple language*:
JVM ko batata hai next line kaun si execute karni hai

5️⃣ **Native Method Stack**

- Native (non-Java) methods ke liye
- C/C++ based code

🔹 **Java Memory Diagram (Conceptual)**

JVM

├── Method Area

├── Heap Area

├── Stack Area

├── PC Register

└── Native Method Stack

🔹 **Installation of Java (JDK)**

▶ **Steps:**

1️⃣ Oracle website se **JDK download**
2️⃣ Install JDK
3️⃣ Set **JAVA_HOME**
4️⃣ Set **Path** variable
5️⃣ Verify installation

▶ **Verify Command:**

java -version

javac -version

📌 *Automation Tip*:
Java version Selenium compatibility ke according rakho

🔹 **Installation of IDE – Eclipse**

▶ **Why Eclipse?**

✓ Free
✓ Selenium friendly
✓ Easy debugging
✓ Plugin support (TestNG, Maven)

▶ **Eclipse Installation Steps:**

1️⃣ Eclipse official website se download
2️⃣ Extract / Install
3️⃣ Workspace select karo
4️⃣ Open Eclipse

▶ **Eclipse me Java Setup**

- New → Java Project
- JDK select
- Package → Class create

📌 *Automation me*:
Selenium scripts Eclipse me likhe jate hain

🔹 **Java + Selenium Architecture (Short)**

Eclipse (IDE)

↓

Java Code

↓

JDK

↓

JVM

↓

Browser Automation (Selenium)

✅ **Interview Ready Line (2–3 Years Exp)**

🟢 "Java architecture includes JDK, JRE, and JVM where JVM manages memory areas like Heap, Stack, Method Area, and executes bytecode in a platform-independent way."

🔹 **OOPS Concepts (Automation Testing Focus)**

1️⃣ **Object and Class**

▶ **Class**

Class ek **blueprint / design** hoti hai jisme variables aur methods define hote hain.

```
class LoginPage{
 void login(){
   System.out.println("Login done");
 }
}
```

▶ **Object**

Object class ka **real instance** hota hai.

```
LoginPage lp = new LoginPage();
lp.login();
```

📌 **Real World Example**

- Class → Car
- Object → BMW, Audi

📌 **Automation Example**

- Class → LoginPage
- Object → new LoginPage()

2️⃣ **Inheritance**

Inheritance ka matlab hai **parent class ke properties child class me use karna**.

```
class BaseTest{
 WebDriver driver;
 void openBrowser(){}
}

class LoginTest extends BaseTest{
 void loginTest(){}
}
```

📌 **Real World Example**

- Parent → Vehicle
- Child → Car, Bike

📌 **Automation Use**

✓ Code reuse
✓ Framework structure

▶ **Types of Inheritance**

- Single
- Multilevel
- Hierarchical

⚠ Java me **multiple inheritance (class se)** allow nahi hai

3️⃣ **Polymorphism**

Polymorphism ka matlab hai **ek cheez, multiple forms**.

▶ **Compile Time (Method Overloading)**

void login(String user){}

void login(String user, String pass){}

▶ **Runtime (Method Overriding)**

WebDriver driver = new ChromeDriver();

📌 **Real World Example**

- Person → speak() (teacher, student)

📌 **Automation Use**

✓ Cross browser testing
✓ Dynamic behavior

4️⃣ **Abstraction**

Abstraction ka matlab hai **implementation hide karke sirf functionality dikhana**.

▶ **Using Interface**

```
interface Browser{
 void open();
}

class Chrome implements Browser{
 public void open(){}
```

}

📌 **Real World Example**

- ATM machine (process hidden)

📌 **Automation Use**

✓ Framework design
✓ Loose coupling

5️⃣ **Encapsulation**

Encapsulation ka matlab hai **data ko protect karna** using private variables + public methods.

```
class User{
 private String password;


 public String getPassword(){
   return password;
 }
}
```

📌 **Real World Example**

- ATM PIN

📌 **Automation Use**

✓ Security
✓ Maintainability

🔥 **OOPS Summary Table**

Concept	Meaning	Automation Use
Class	Blueprint	Page class
Object	Instance	Page object
Inheritance	Reuse	Base class
Polymorphism	Many forms	Browser
Abstraction	Hide logic	Framework
Encapsulation	Data security	Test data

✅ **Interview Ready Line (2–3 Years)**

🟢 “OOPS concepts like inheritance, polymorphism, abstraction, and encapsulation help in designing scalable and reusable Selenium automation frameworks.”

🔹 **Java – Basic Concepts**

1️⃣ **Identifiers**

Identifier ka matlab hota hai **name** jo hum:

- class
- method
- variable
  ko dene ke liye use karte hain.

▶ **Rules**

✓ Alphabet, digits, _ , $ allowed
✓ Digit se start nahi ho sakta
✓ Space allowed nahi
✓ Java keywords allowed nahi

▶ **Valid**

int age;

String userName;

double salary_amount;

▶ **Invalid**

int 1age; // ❌

int total amt; // ❌

int class; // ❌

📌 *Automation Example*:
WebDriver driver;

2️⃣ **Keywords**

Keywords wo **reserved words** hote hain jinka Java me special meaning hota hai.

▶ **Common Java Keywords**

class, public, static, void, int, if, else, for, while, return, try, catch, new

⚠ Inko identifiers ke roop me use nahi kar sakte.

📌 *Automation Example*:

```
public class LoginTest {
 public static void main(String[] args) {
 }
}
```

3️⃣ Variables

Variable ek **container** hota hai jo data store karta hai.

▶ Types of Variables

🔹 Local Variable

Method ke andar

```
void login(){
 int x = 10;
}
```

🔹 Instance Variable

Class ke andar, method ke bahar

```
class Test{
 String name;
}
```

🔹 Static Variable

static keyword ke saath

```
static WebDriver driver;
```

📌 *Automation Use*:
Browser driver ko common rakhna

4️⃣ Datatypes

Datatypes batate hain **variable kis type ka data store karega**.

▶ Primitive Datatypes

Type	Size	Example
int	4 byte	10
double	8 byte	10.5
char	2 byte	'A'
boolean	1 bit	true

▶ **Non-Primitive Datatypes**

- String
- Array
- Class
- Interface

String browser = "Chrome";

📌 *Automation Example*:
URL, username, password

5️⃣ **Operators**

Operators ka use **operations perform karne** ke liye hota hai.

🔹 **Arithmetic Operators**

Operator	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus

int a = 10, b = 5;

System.out.println(a + b);

🔹 **Relational Operators**

Operator	Use
==	Equal
!=	Not equal
>	Greater
<	Less
>=	Greater equal
<=	Less equal

```
if(a > b){
 System.out.println("a is greater");
}
```

📌 *Automation Use*:
Validation & assertions

🔹 **Bitwise Operators**

Operator	Name
&	AND
^	XOR
~	NOT
<<	Left shift
>>	Right shift

📌 *Interview Note*:
Automation me kam use hote hain, theory important hai

🔹 **Assignment Operators**

Operator	Example
=	a = 10
+=	a += 5
-=	a -= 2

Operator	Example
*=	a *= 3
/=	a /= 2

```
int x = 10;
x += 5;
```

🔥 **Automation Mapping (Quick)**

Concept	Automation Use
Identifiers	Element, class names
Variables	Store data
Datatypes	URL, text
Operators	Conditions
Keywords	Framework structure

✅ **Interview Ready Line**

🟢 “Java basic concepts like variables, datatypes, and operators are heavily used in Selenium automation for data handling and validations.”

🔹 **Access Specifiers (Access Modifiers)**

Access specifiers batate hain ki **class / method / variable ko kahaan se access** kiya ja sakta hai.

1️⃣ **Public**

- Kahin se bhi access ho sakta hai

```
public class Test {
 public int x = 10;
 public void show(){
  System.out.println("Public Method");
 }
```

}

📌 *Automation Use*:
Reusable methods (login, logout)

2️⃣ Private

- Sirf **same class** ke andar access

```
class Test{
 private int pin = 1234;
}
```

📌 *Automation Use*:
Sensitive data protection

3️⃣ Default (No keyword)

- Sirf **same package** me access

```
class Test{
 int age = 20;
}
```

📌 *Automation Use*:
Package level access

4️⃣ Protected

- Same package + **child class**

```
class BaseTest{
 protected WebDriver driver;
}
```

📌 *Automation Use*:
Framework inheritance

🔥 Access Specifiers Table

Specifier	Same Class	Same Package	Child Class	Anywhere
public	✓	✓	✓	✓

Specifier	Same Class	Same Package	Child Class	Anywhere
protected	✓	✓	✓	❌
default	✓	✓	❌	❌
private	✓	❌	❌	❌

🔹 **Constructors**

Constructor ek **special method** hota hai jo **object create hote hi call** hota hai.

▶ **Common Rules**

✓ Class name same
✓ Return type nahi hota
✓ Object creation ke time call hota hai

1️⃣ **Default Constructor**

- Java compiler automatically provide karta hai

```
class Demo{

}
```

2️⃣ **No Argument Constructor**

- Programmer khud banata hai (without parameters)

```
class Demo{
 Demo(){
  System.out.println("No Argument Constructor");
 }
}
```

3️⃣ **Parameterized Constructor**

- Parameters ke saath constructor

```
class LoginPage{
 WebDriver driver;
```

```
LoginPage(WebDriver driver){
 this.driver = driver;
}
}
```

📌 *Automation Use*:
Page Object Model (POM)

🔹 **Methods**

Method ek **block of code** hota hai jo koi specific task perform karta hai.

1️⃣ **Main Method**

- Program execution yahin se start hota hai

```
public static void main(String[] args){
 System.out.println("Program Start");
}
```

2️⃣ **Static Method**

- Class name se call hota hai
- Object ki zarurat nahi

```
static void openBrowser(){
 System.out.println("Browser Open");
}
```

Call:

```
Demo.openBrowser();
```

📌 *Automation Use*:
Utility methods

3️⃣ **Non-Static Method**

- Object ke through call hota hai

```
void login(){
 System.out.println("Login Done");
```

}

Call:

Demo d = new Demo();

d.login();

📌 *Automation Use*:
Page actions

4️⃣ Method Creation & Method Calling

▶ Method Creation

void add(int a, int b){

System.out.println(a+b);

}

▶ Method Calling

Demo d = new Demo();

d.add(10,20);

🔥 Automation Mapping (Quick)

Concept	Automation Use
public	Reusable methods
private	Data hiding
protected	Base class
Constructor	POM
Static method	Utilities
Non-static	Page actions

✅ Interview Ready Line (2–3 Years)

🟢 “Access specifiers control visibility, constructors initialize objects, and methods help in code reusability which is crucial for Selenium automation frameworks.”

🔹 Control Statements and Loops

1️⃣ If – Else

Condition ke basis par decision leta hai.

```
if(title.equals("Home")){
 System.out.println("Test Pass");
}else{
 System.out.println("Test Fail");
}
```

📌 **Automation Use**
✓ Validation
✓ Assertions logic

2️⃣ Switch Case

Multiple conditions ke liye better option.

```
switch(browser){
 case "chrome":
   System.out.println("Chrome launched");
   break;
 case "firefox":
   System.out.println("Firefox launched");
   break;
}
```

📌 **Automation Use**
✓ Cross browser testing

3️⃣ For Loop

Fixed number of times execution.

```
for(int i=0;i<5;i++){
 System.out.println(i);
}
```

📌 **Automation Use**
✓ Links

✓ Table rows
✓ Dropdown values

4️⃣ Do While Loop

Kam se kam **ek baar execute** hota hi hai.

```
int i=1;

do{

 System.out.println(i);

 i++;

}while(i<=3);
```

📌 **Use**
✓ Retry logic

5️⃣ For Each Loop

Collection/Array ke liye best.

```
for(WebElement e : links){

 System.out.println(e.getText());

}
```

📌 **Automation Use**
✓ Selenium elements list

🔹 String (Most Important 🔥)

1️⃣ Introduction to String

String ek **non-primitive immutable class** hai jo text store karti hai.

```
String name = "Selenium";
```

2️⃣ Creation of String

▶ Using Literal

```
String s1 = "Java";
```

▶ Using new Keyword

```
String s2 = new String("Java");
```

3️⃣ SCP & Heap Memory

- **SCP (String Constant Pool)** → literals store hote hain
- **Heap** → new keyword se banne wale objects

String s1 = "Test";

String s2 = "Test"; // SCP (same memory)

String s3 = new String("Test"); // Heap (new memory)

4️⃣ Why String is Immutable?

String change nahi hoti kyunki:
✓ Security
✓ Memory saving
✓ Thread safe

String s = "Java";

s.concat("Test");

System.out.println(s); // Java

5️⃣ String Methods (Important)

length()

toUpperCase()

toLowerCase()

charAt()

contains()

equals()

equalsIgnoreCase()

substring()

split()

trim()

replace()

📌 **Automation Use**

✓ Text validation

✓ Message comparison

6️⃣ **equals() vs ==**

```
String a = "Java";

String b = "Java";


System.out.println(a == b);      // true (same SCP)

System.out.println(a.equals(b)); // true (content)
```

📌 **Interview Rule**

✓ == → memory reference

✓ equals() → content compare

7️⃣ **Custom Immutable Class**

```
final class Employee{

 private final int id;


 Employee(int id){

  this.id = id;

 }

 public int getId(){

  return id;

 }

}
```

✓ class final

✓ variables private + final

✓ no setter

8️⃣ **Frequently Asked String Programs**

✓ Reverse String

✓ Palindrome

✓ Count characters

✓ Count vowels
✓ Remove spaces
✓ Duplicate characters
✓ Anagram

Example – Reverse String:

```
String s="Java";

String rev="";

for(int i=s.length()-1;i>=0;i--){

 rev=rev+s.charAt(i);

}

System.out.println(rev);
```

🔹 **Array**

1️⃣ Single Dimensional Array

```
int[] a = {10,20,30};
```

📌 **Use**
✓ Fixed data list

2️⃣ Multidimensional Array

```
int[][] a = {

 {1,2},

 {3,4}

};
```

📌 **Automation Use**
✓ Test data

3️⃣ Advantages of Array

✓ Fast access
✓ Less memory
✓ Easy to use

4️⃣ Limitations of Array

❌ Fixed size
❌ No built-in methods
❌ Insertion/deletion difficult

5️⃣ How Collection Framework Came?

👉 Array limitations ki wajah se:

- Dynamic size
- Ready-made methods
- Better data handling

Isliye **ArrayList, HashMap, Set** aaye.

6️⃣ Important Array Programs

✓ Find largest number
✓ Smallest number
✓ Duplicate elements
✓ Sorting
✓ Missing number

Example – Largest Number:

```
int[] a={10,50,20};

int max=a[0];


for(int i=1;i<a.length;i++){

 if(a[i]>max){

  max=a[i];

 }

}

System.out.println(max);
```

✅ Interview Ready Line (2–3 Years)

🟢 “Control statements manage program flow, Strings are immutable for security and performance, and Arrays store fixed data but Collections overcome their limitations.”

🔹 **Most Frequently Asked Logical Java Programs (Interview Ready)**

1️⃣ **Reverse a String**

```
String s = "Selenium";
String rev = "";
for(int i = s.length()-1; i>=0; i--){
 rev += s.charAt(i);
}
System.out.println(rev);
```

✅ *Output*: muineleS

2️⃣ **Check Palindrome**

```
String s = "madam";
String rev = "";
for(int i = s.length()-1; i>=0; i--){
 rev += s.charAt(i);
}
if(s.equals(rev)){
 System.out.println("Palindrome");
}else{
 System.out.println("Not Palindrome");
}
```

✅ *Use*: Validation logic in automation

3️⃣ **Find Largest and Smallest in Array**

```
int[] a = {10,50,20};
int max = a[0];
int min = a[0];

for(int i=1; i<a.length; i++){
 if(a[i]>max) max=a[i];
 if(a[i]<min) min=a[i];
```

```
}
System.out.println("Largest: " + max);
System.out.println("Smallest: " + min);
```

4️⃣ Count Vowels and Consonants

```
String s = "Automation";
int vowels=0, consonants=0;
for(int i=0; i<s.length(); i++){
 char ch = Character.toLowerCase(s.charAt(i));
 if(ch=='a'||ch=='e'||ch=='i'||ch=='o'||ch=='u') vowels++;
 else if(ch>='a' && ch<='z') consonants++;
}
System.out.println("Vowels: "+vowels);
System.out.println("Consonants: "+consonants);
```

5️⃣ Fibonacci Series

```
int n=10, a=0, b=1;
System.out.print(a+" "+b+" ");
for(int i=2;i<n;i++){
 int c = a+b;
 System.out.print(c+" ");
 a=b;
 b=c;
}
```

6️⃣ Prime Number Check

```
int num = 29;
boolean isPrime = true;
for(int i=2; i<=num/2; i++){
 if(num%i==0){
  isPrime=false;
```

```
 break;
 }
}
System.out.println(isPrime ? "Prime" : "Not Prime");
```

7️⃣ Factorial of a Number

```
int n=5, fact=1;
for(int i=1;i<=n;i++){
 fact*=i;
}
System.out.println("Factorial: "+fact);
```

8️⃣ Reverse Number

```
int num = 12345, rev=0;
while(num!=0){
 rev = rev*10 + num%10;
 num/=10;
}
System.out.println(rev);
```

9️⃣ Swap Two Numbers Without Temp Variable

```
int a=10, b=20;
a = a+b;
b = a-b;
a = a-b;
System.out.println(a + " " + b);
```

1️⃣0️⃣ Armstrong Number

```
int num = 153, sum=0, temp=num;
while(temp!=0){
 int digit = temp%10;
```

```
 sum += digit*digit*digit;
 temp/=10;
}
System.out.println(sum==num ? "Armstrong" : "Not Armstrong");
```

🔹 **Assignments (Practice Programs)**

1. Print multiplication table of given number
2. Count number of words in a string
3. Remove duplicate characters from string
4. Merge two arrays into a single array
5. Sort an array in ascending & descending order
6. Check Anagram of two strings
7. Find second largest element in array
8. Sum of digits of a number
9. Reverse each word in a sentence
10. Find common elements between two arrays

📌 *Automation Relevance*:

- String manipulations → Web text validations
- Arrays → Test data handling
- Loops & conditions → Dynamic testing scenarios
- Logic programs → Coding round for Automation interviews

✅ **Interview Tip**

🟢 "These logical Java programs demonstrate your problem-solving skills. For automation testing, string and array manipulations, loops, and conditionals are most commonly used in Selenium frameworks."

🔹 **Java – Exception Handling (Automation Focus)**

1️⃣ **What is Exception?**

Exception ek **unexpected event ya error** hai jo program execution ko **abnormally terminate** kar deta hai.

- Example: Division by zero, file not found, null pointer, array index out of bounds
- Automation context: Element not found, page load failure

int a = 10, b = 0;

int c = a/b; // ArithmeticException

2️⃣ Hierarchy of Exception

Throwable

├── Error (System errors)

└── Exception

  ├── Checked Exception (compile-time)

  └── Unchecked Exception (runtime)

▶ Checked Exception

- Compile-time error
- Example: FileNotFoundException, IOException

▶ Unchecked Exception

- Runtime error
- Example: ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException

---

3️⃣ Exception Handling using Try-Catch

▶ Syntax

```
try{
 // risky code
}catch(ExceptionType e){
 // handling code
}
```

▶ Example

```
try{
 int data = 10/0;
}catch(ArithmeticException e){
```

```
 System.out.println("Cannot divide by zero");
}
System.out.println("Rest of the code");
```

📌 *Automation Use*: Selenium scripts crash hone se bachana

4️⃣ Scenarios on Try, Catch, and Finally

▶ Try-Catch-Finally

```
try{
 int data = 10/0;
}catch(ArithmeticException e){
 System.out.println("Error handled");
}finally{
 System.out.println("Finally block always executes");
}
```

▶ Rules

- Finally block **always executes** (except System.exit())
- Multiple catch allowed for different exceptions
- Catch block order → **child first, parent later**

5️⃣ Throw and Throws

▶ throw

- Programmatically exception throw karte hain

```
int age = 15;
if(age<18){
 throw new ArithmeticException("Not eligible");
}
```

▶ throws

- Method declaration me exception declare karte hain

```
void checkAge(int age) throws ArithmeticException {
 if(age<18){
   throw new ArithmeticException("Not eligible");
```

```
}
}
```

📌 *Automation Use*:

- Page validations, preconditions check

6️⃣ Custom Exception

▶ How to create

```
class AgeException extends Exception{
 AgeException(String msg){
  super(msg);
 }
}
```

▶ How to handle

```
int age = 15;
try{
 if(age<18){
   throw new AgeException("Age must be >= 18");
 }
}catch(AgeException e){
 System.out.println(e.getMessage());
}
```

📌 *Automation Use*:

- Custom validation rules
- Framework preconditions

🔹 Exception Handling Table (Quick Reference)

Concept	Example	Automation Use
Try-Catch	Divide by zero	Prevent crash
Finally	Close browser	Always execute cleanup
Throw	throw new Exception()	Custom error

Concept	Example	Automation Use
Throws	Method declaration	Propagate exception
Custom Exception	AgeException	Business rules

✅ **Interview Ready Line (2–3 Years)**

🟢 “Exception handling in Java ensures robust Selenium scripts. Using try-catch-finally, throw, throws, and custom exceptions, we can gracefully handle unexpected scenarios and avoid test failures.”

🔹 **Java – Object Class**

1️⃣ **What is Object Class?**

- **Object class** Java me **root class** hai
- Java me jitni bhi classes hain, **directly ya indirectly Object class ko inherit karti hain**
- Object class ke methods **sabhi classes me available hote hain**

📌 *Automation Relevance*:

- Selenium me page object ya WebDriver objects Object class ke methods inherit karte hain
- Debugging aur logging me useful

2️⃣ **Object Class Methods (Commonly Used)**

Method	Description	Example / Automation Use
toString()	Object ka **string representation** return karta hai	System.out.println(driver.toString());
hashCode()	Object ka unique integer **hash code** return karta hai	Compare two WebElement objects
equals(Object obj)	Object **content equality** check karta hai	Compare two test data objects
getClass()	Object ka class name return karta hai	Logging / reflection
clone()	Object ka **copy** create karta hai	Test data cloning
finalize()	Garbage collector se pehle call hota hai	Clean up resources

Method	Description	Example / Automation Use
wait(), notify(), notifyAll()	Thread communication ke liye	Selenium multi-threaded execution me
equalsIgnoreCase()	(String object) case-insensitive comparison	Web text validation

---

3️⃣ Example: Using Object Class Methods

```
class Demo{
 int id;
 Demo(int id){ this.id = id; }

 public String toString(){
  return "Demo ID: "+id;
 }
}

public class TestObj{
 public static void main(String[] args){
  Demo d = new Demo(101);
  System.out.println(d.toString());   // Demo ID: 101
  System.out.println(d.hashCode());   //  Hash code value
 }
}
```

📌 *Automation Use*:

- toString() → Logging page objects
- hashCode() → Compare unique test objects

✅ Interview Ready Line

🟢 “In Java, the Object class is the superclass of all classes. Its methods like toString(), hashCode(), and equals() are inherited by every class and are commonly used for logging, object comparison, and debugging in Selenium automation frameworks.”

1️⃣ Introduction to Collection Framework

- **Collection Framework** Java me **predefined data structures** aur **interfaces** ka set hai
- Collections **dynamic data handling** provide karti hain, arrays ke limitations ko overcome karte hain
- Interfaces: **List, Set, Map, Queue**

📌 *Automation Relevance*:

- Test data handling (login credentials, multiple URLs)
- Web element lists, table rows

2️⃣ List Interface

- **Ordered collection**
- **Allows duplicates**
- Common implementations: **ArrayList, LinkedList, Vector, Stack**

▶ ArrayList

- Dynamic array, **fast random access**, slow insertion/deletion

```
ArrayList<String> browsers = new ArrayList<>();

browsers.add("Chrome");

browsers.add("Firefox");
```

▶ LinkedList

- Doubly linked list, **fast insertion/deletion**, slow random access

```
LinkedList<String> browsers = new LinkedList<>();
```

▶ Vector

- **Synchronized version of ArrayList**, thread-safe

```
Vector<String> v = new Vector<>();
```

▶ Stack

- LIFO (Last In First Out)

```
Stack<String> stack = new Stack<>();

stack.push("Login");

stack.pop();
```

3️⃣ Set Interface

- **No duplicates, unordered (except LinkedHashSet / TreeSet)**
- Common implementations: **HashSet, LinkedHashSet, SortedSet, TreeSet**

▶ **HashSet**

- No order

```
HashSet<String> set = new HashSet<>();
set.add("Chrome");
```

▶ **LinkedHashSet**

- Insertion order maintained

```
LinkedHashSet<String> lhs = new LinkedHashSet<>();
```

▶ **TreeSet / SortedSet**

- Sorted order

```
TreeSet<Integer> ts = new TreeSet<>();
ts.add(10);
ts.add(5);
```

📌 *Automation Use*:

- Unique test data, unique user IDs

4️⃣ **Map Interface**

- **Key-Value pair storage**
- Common implementations: **HashMap, LinkedHashMap, TreeMap, Hashtable**

▶ **HashMap**

- Unordered, allows null key/value

```
HashMap<String,String> map = new HashMap<>();
map.put("URL", "https://example.com");
```

▶ **LinkedHashMap**

- Maintains insertion order

▶ **TreeMap**

- Sorted by keys

▶ **Hashtable**

- Synchronized, thread-safe
- No null key/value

📌 *Automation Use*:

- Test data mapping, config properties

5️⃣ Iteration of Collections

▶ Using For Loop (List)

```
for(int i=0;i<browsers.size();i++){
 System.out.println(browsers.get(i));
}
```

▶ For-Each Loop

```
for(String b : browsers){
 System.out.println(b);
}
```

▶ Using Iterator

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

▶ Using Map (Key-Value)

```
for(Map.Entry<String,String> entry : map.entrySet()){
 System.out.println(entry.getKey() + " = " + entry.getValue());
}
```

6️⃣ Common Methods of Collections

Method	Description
add()	Add element
remove()	Remove element
size()	Get total elements
contains()	Check presence
isEmpty()	Check if empty

Method	Description
clear()	Remove all elements
get(index)	Access element (List)
put(key,value)	Add in Map
keySet()	Get all keys (Map)
values()	Get all values (Map)

🔹 **Automation Relevance**

Collection Use Case in Selenium

List	WebElement list, dropdown options
Set	Unique test data, unique users
Map	Test data key-value, config mapping
Stack	Undo / redo actions in automation

✅ **Interview Ready Line**

🟢 "Java Collection Framework provides dynamic data structures like List, Set, and Map for efficient data handling. In Selenium automation, these are used for managing test data, WebElements, and configurations effectively."

🔹 **Java 8 Features (Automation Focus)**

Java 8 ne programming me **functional programming aur concise syntax** introduce kiya. Selenium automation me modern frameworks me frequently use hota hai.

1️⃣ **Functional Interface**

- **Interface with exactly one abstract method**
- Lambda expressions ke saath use hota hai
- Annotation: @FunctionalInterface (optional but recommended)

▶ **Example**

```
@FunctionalInterface
interface Calculator{
```

```
 int add(int a, int b);
}

public class Test {
 public static void main(String[] args){
  Calculator c = (a,b) -> a+b;
  System.out.println(c.add(10,20));
 }
}
```

📌 *Automation Use*:

- Custom actions, reusable functional blocks

2️⃣ ForEach Loop (Java 8)

- Iterable interface me default method
- Collections and arrays ke liye concise iteration

▶ Example

```
List<String> browsers = Arrays.asList("Chrome","Firefox","Edge");
browsers.forEach(b -> System.out.println(b));
```

📌 *Automation Use*:

- Iterate WebElement lists
- Logging multiple elements

3️⃣ Static and Default Methods in Interface

▶ Static Methods

- Interface ke **static methods** directly call hote hain

```
interface Utils{
 static void openBrowser(){
  System.out.println("Browser Opened");
 }
}
```

Utils.openBrowser();

▶ **Default Methods**

- Interface me **default method** provide kiya ja sakta hai
- Implementation optional, inheritance ke saath override ho sakta hai

```
interface Logger{
 default void log(String msg){
  System.out.println("Log: "+msg);
 }
}

class Test implements Logger{}
Test t = new Test();
t.log("Automation started");
```

📌 *Automation Use*:

- Common reusable methods across framework interfaces

4️⃣ **Lambda Expression**

- **Shorter syntax** for anonymous classes
- Functional interfaces ke saath use hota hai

▶ **Syntax**

```
(parameters) -> expression
```

▶ **Example**

```
List<String> browsers = Arrays.asList("Chrome","Firefox");
browsers.forEach(b -> System.out.println(b));
```

▶ **Multi-line Lambda**

```
browsers.forEach(b -> {
 System.out.println("Browser: "+b);
 System.out.println("Test started");
});
```

📌 *Automation Use*:

- Actions on WebElement lists

- Event-driven frameworks

5️⃣ Streams (Optional for Selenium Advanced)

- Functional approach to process collections
- Filter, map, reduce

```
List<String> browsers = Arrays.asList("Chrome","Firefox","Edge");

browsers.stream().filter(b -> b.startsWith("C")).forEach(System.out::println);
```

📌 *Automation Use*:

- Filter WebElements
- Extract specific test data

🔹 Summary Table

Feature	Description	Automation Use
Functional Interface	One abstract method interface	Custom actions
Lambda Expression	Concise implementation	Iterate WebElements
Default Method	Interface method with body	Common framework utilities
Static Method	Interface method called via interface	Utility methods
ForEach Loop	Enhanced iteration	WebElement / test data lists
Streams	Functional processing	Filter / map WebElements

✅ Interview Ready Line

🟢 “Java 8 introduced functional programming features like functional interfaces, lambda expressions, default and static methods in interfaces, and enhanced forEach iteration, which are widely used in modern Selenium automation frameworks for cleaner, reusable, and concise code.”
