Java & Selenium Exception Handling – Detailed Notes

# 1. What is Exception Handling?

Exception handling is a mechanism that handles runtime errors so the normal flow of the program can continue. Java provides try, catch, finally, throw, and throws to manage exceptions.

# 2. Types of Exceptions

• Checked Exceptions: Checked at compile-time (e.g., IOException, SQLException)  
• Unchecked Exceptions: Checked at runtime (e.g., NullPointerException, ArithmeticException)

# 3. Java Exception Hierarchy

Throwable  
 ├── Error  
 └── Exception  
 ├── Checked (IOException, SQLException)  
 └── RuntimeException (NPE, AIOOBE, AE)

# 4. Syntax: try-catch-finally

try {  
 // Code that might throw an exception  
} catch (ExceptionType e) {  
 // Exception handling logic  
} finally {  
 // Cleanup code  
}

# 5. throw vs throws

• throw: Used to explicitly throw an exception  
• throws: Used to declare exceptions that a method may throw

throw new IllegalArgumentException("Invalid age");  
  
public void readFile() throws IOException {  
 // method code  
}

# 6. Common Java Exceptions

## NullPointerException

When: Accessing methods/variables on a null object

try {  
 String s = null;  
 System.out.println(s.length());  
} catch (NullPointerException e) {  
 System.out.println("Handled NPE");  
}

## ArrayIndexOutOfBoundsException

When: Accessing array with illegal index

try {  
 int[] arr = new int[3];  
 System.out.println(arr[5]);  
} catch (ArrayIndexOutOfBoundsException e) {  
 System.out.println("Index out of bounds");  
}

## ClassNotFoundException

When: `Class.forName()` fails to locate a class

try {  
 Class.forName("com.example.UnknownClass");  
} catch (ClassNotFoundException e) {  
 System.out.println("Class not found");  
}

## ArithmeticException

When: Division by zero

try {  
 int result = 10 / 0;  
} catch (ArithmeticException e) {  
 System.out.println("Cannot divide by zero");  
}

## IllegalArgumentException

When: Method receives an invalid argument

try {  
 Thread.sleep(-200);  
} catch (IllegalArgumentException e) {  
 System.out.println("Invalid argument passed");  
}

## FileNotFoundException

When: File does not exist or cannot be opened

try {  
 FileReader fr = new FileReader("nofile.txt");  
} catch (FileNotFoundException e) {  
 System.out.println("File not found");  
}

# 7. Common Selenium Exceptions

## NoSuchElementException

When: Element is not present in the DOM

try {  
 driver.findElement(By.id("invalidId"));  
} catch (NoSuchElementException e) {  
 System.out.println("Element not found");  
}

## TimeoutException

When: Waits exceed expected condition time

try {  
 WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));  
 wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("myBtn")));  
} catch (TimeoutException e) {  
 System.out.println("Timeout occurred");  
}

## ElementNotInteractableException

When: Element is present but not interactable (hidden, disabled, etc.)

try {  
 driver.findElement(By.id("hiddenButton")).click();  
} catch (ElementNotInteractableException e) {  
 System.out.println("Cannot interact with element");  
}

## StaleElementReferenceException

When: Element becomes stale after page update

try {  
 WebElement btn = driver.findElement(By.id("clickMe"));  
 driver.navigate().refresh();  
 btn.click();  
} catch (StaleElementReferenceException e) {  
 System.out.println("Stale element");  
}

## WebDriverException

When: Internal issue in the WebDriver

try {  
 driver.get("invalid://url");  
} catch (WebDriverException e) {  
 System.out.println("WebDriver crash or issue");  
}

## InvalidSelectorException

When: Invalid locator syntax used

try {  
 driver.findElement(By.xpath("//div[@"));  
} catch (InvalidSelectorException e) {  
 System.out.println("Invalid XPath selector");  
}

# 8. Best Practices

• Always catch specific exceptions, not just Exception  
• Use finally for clean-up (closing files, browser)  
• Use Explicit Waits over Thread.sleep()  
• Use try-catch in small blocks – avoid wrapping entire methods  
• Log all exceptions with meaningful messages

# 9. Custom Exception

class InvalidUserInputException extends Exception {  
 public InvalidUserInputException(String message) {  
 super(message);  
 }  
}

# 10. Real Selenium Use Case

public void loginTest() {  
 try {  
 driver.findElement(By.id("username")).sendKeys("admin");  
 driver.findElement(By.id("password")).sendKeys("admin123");  
 driver.findElement(By.id("login")).click();  
 } catch (NoSuchElementException e) {  
 System.out.println("Element missing");  
 } catch (Exception e) {  
 System.out.println("Unexpected error: " + e.getMessage());  
 }  
}