Java & Selenium Exception Handling – Advanced Notes for Automation Testers

# 1. Advanced Java Exception Concepts

## 1.1 Multi-catch Blocks

You can catch multiple exception types in a single catch block:

try {  
 // code that might throw IOException or SQLException  
} catch (IOException | SQLException e) {  
 e.printStackTrace();  
}

## 1.2 Suppressed Exceptions

Used in try-with-resources where exceptions in finally can be suppressed:

try (BufferedReader br = new BufferedReader(new FileReader("file.txt"))) {  
 // read file  
} catch (IOException e) {  
 e.printStackTrace();  
}

## 1.3 Checked vs Unchecked Exceptions

• Checked: Must be handled (IOException, SQLException)  
• Unchecked: Occur at runtime (NullPointerException, ArithmeticException)

# 2. Exception Handling in Testing Frameworks

## 2.1 TestNG – Expected Exceptions

@Test(expectedExceptions = NoSuchElementException.class)  
public void testElement() {  
 driver.findElement(By.id("invalid"));  
}

## 2.2 Assertions vs Exceptions

Use assertions to validate expected outcomes. Use try-catch to handle unexpected errors gracefully.

# 3. Logging and Reporting Integration

## 3.1 Logging with Log4j

private static Logger logger = Logger.getLogger(MyTest.class);  
  
try {  
 // test code  
} catch (Exception e) {  
 logger.error("Exception occurred", e);  
}

## 3.2 Reporting Failures

Use ExtentReports or Allure to capture exception details in HTML reports.

# 4. Retry Mechanisms and Recovery

## 4.1 Retry on Failure (TestNG IRetryAnalyzer)

public class RetryAnalyzer implements IRetryAnalyzer {  
 private int count = 0;  
 public boolean retry(ITestResult result) {  
 if (count < 2) {  
 count++;  
 return true;  
 }  
 return false;  
 }  
}

## 4.2 Handling Flaky Tests

Wrap unstable elements in retry blocks or re-fetch elements when stale.

# 5. Centralized Exception Strategy

Create a utility class for wrapping common WebDriver actions and handle exceptions uniformly.

public class ElementUtil {  
 public void click(By locator) {  
 try {  
 driver.findElement(locator).click();  
 } catch (ElementNotInteractableException e) {  
 System.out.println("Custom: Element not clickable");  
 }  
 }  
}

# 6. Custom Exception Categorization

Organize custom exceptions like:  
• UIException  
• DataLoadException  
• NetworkTimeoutException

# 7. Exception Propagation and Chaining

try {  
 methodA();  
} catch (IOException e) {  
 throw new CustomFrameworkException("Error in methodA", e);  
}

# 8. Global Exception Handling Design

Create a top-level handler for catching unhandled exceptions during test run and logging them appropriately.

# 9. Thread-Level Exception Handling

Use `Thread.setDefaultUncaughtExceptionHandler()` for managing uncaught exceptions in parallel test threads.

Thread.setDefaultUncaughtExceptionHandler((t, e) -> {  
 System.out.println("Uncaught exception: " + e);  
});