3.1 Exception Handling & Multithreaded

---

1. Error and Exception:

1.1. Types of Error [W-14]

Errors are broadly classified into two categories:-

1. Compile time errors
2. Runtime errors

1.1.1. Compile time error-

All syntax errors will be detected and displayed by Java compiler and therefore these errors are known as compile time errors.

The most common problems are:

- Missing semicolon
- Missing (or mismatch of) bracket in classes & methods
- Misspelling of identifiers & keywords
- Missing double quotes in string
- Use of undeclared variables.
- Bad references to objects.

1.1.2. Runtime error

Sometimes a program may compile successfully creating the .class file but may not run properly. Such programs may produce wrong results due to wrong logic or may terminate due to errors such as stack overflow. When such errors are encountered Java typically generates an error message and aborts the program.

The most common run-time errors are:

- Dividing an integer by zero
- Accessing an element that is out of bounds of an array
- Trying to store value into an array of an incompatible class or type
- Passing parameter that is not in a valid range or valid for method
- Trying to illegally change status of thread
- Attempting to use a negative size for an array
- Converting invalid string to a number
- Accessing character that is out of bound of a string

1.2. Exception

An exception is an event, which occurs during the execution of a program, that stop the flow of the program's instructions and takes appropriate actions if handled.

In Java, exceptions are categorized into **Checked Exceptions**, **Unchecked Exceptions**, and **Errors**. Below is a breakdown of each type:

1. Checked Exceptions (Compile-Time Exceptions)

These exceptions are checked at compile time, meaning the compiler requires the developer to handle them using **try-catch** or **throws**.

Examples:

- **IOException** – Raised when there is an issue with input/output operations.
- **SQLException** – Occurs when handling database operations.
- **FileNotFoundException** – Raised when trying to access a file that doesn't exist.
- **InterruptedException** – Occurs when a thread is interrupted while waiting.

2. Unchecked Exceptions (Runtime Exceptions)

These exceptions are not checked at compile time, meaning the program compiles successfully but may throw an error at runtime.

Examples:

- **NullPointerException** – Occurs when trying to use an object reference that is null.
- **ArrayIndexOutOfBoundsException** – Raised when accessing an array element outside its bounds.
- **ArithmeticException** – Occurs when there is an arithmetic error, such as division by zero.
- **ClassCastException** – Raised when trying to cast an object to an incompatible type.

3. Errors (Serious system-level issues)

Errors are serious problems that are typically beyond the control of the application and should not be caught.

Examples:

- **OutOfMemoryError** – Raised when the JVM runs out of memory.
- **StackOverflowError** – Occurs due to excessive recursive method calls.
- **VirtualMachineError** – Indicates problems with the Java Virtual Machine.
- **AssertionError** – Occurs when an assertion statement fails.

1.2.1. Exception Handling

Java handles exceptions with 5 keywords:

1. try
2. catch
3. finally
4. throw
5. throws

---

1) try:

This block applies a monitor on the statements written inside it. If there exists any exception, the control is transferred to catch or finally block.

Syntax

```java
try
{
    // block of code to monitor for errors
}
```

---

2) catch:

This block includes the actions to be taken if a particular exception occurs.

Syntax:

```java
catch(ExceptionType1 exOb)
{
    // exception handler for ExceptionType1
}
```

---

3) finally:

finally block includes the statements which are to be executed in any case, whether an exception is raised or not.

Syntax:

```java
finally
{
    // block of code to be executed before try block ends
}
```

---

4) throw:

This keyword is generally used in case of user-defined exceptions to forcefully raise the exception and take the required action.

The general form of throw is:

```java
throw new ThrowableInstance;
```

or

```java
throw throwableInstance;
```

throw statement explicitly throws a built-in or user-defined exception. When throw statement is executed, the flow of execution stops immediately after the throw statement, and any subsequent statements are not executed.

---

5) throws:

throws keyword can be used along with the method definition to name the list of exceptions which are likely to happen during the execution of that method. In that case, try ... catch block is not necessary in the code.

General form of method declaration that includes "Throws" clause:

```java
Type method-name (parameter list) throws exception list
{
    // body of method
}
```

1.3 Built-in Exceptions

Inside the standard package java.lang, Java defines several exception classes. A few have been used by the preceding examples. The most general of these exceptions are subclasses of the standard type RuntimeException.

These exceptions need not be included in any method's throws list, these are called **unchecked exceptions** because the compiler does not check to see if a method handles or throws these exceptions. The unchecked exceptions defined in java.lang are listed in **Table 1**.

Table 2 lists those exceptions defined by java.lang that must be included in a method's throws list if that method can generate one of these exceptions and does not handle it itself. These are called **checked exceptions**.

*

Unchecked Exceptions (Runtime Exceptions)

Exception	Meaning
ArithmeticException	Arithmetic error, such as divide-by-zero.
ArrayIndexOutOfBoundsException	Array index is out-of-bounds.
ArrayStoreException	Assignment to an array element of an incompatible type.
ClassCastException	Invalid cast.
EnumConstantNotPresentException	An attempt is made to use an undefined enumeration value.
IllegalArgumentException	Illegal argument used to invoke a method.
IllegalMonitorStateException	Illegal monitor operation, such as waiting on an unlocked thread.
IllegalStateException	Environment or application is in an incorrect state.
IllegalThreadStateException	Requested operation not compatible with current thread state.
IndexOutOfBoundsException	Some type of index is out-of-bounds.
NegativeArraySizeException	Array created with a negative size.
NullPointerException	Invalid use of a null reference.
NumberFormatException	Invalid conversion of a string to a numeric format.
SecurityException	Attempt to violate security.
StringIndexOutOfBoundsException	Attempt to index outside the bounds of a string.
TypeNotPresentException	Type not found.
UnsupportedOperationException	An unsupported operation was encountered.

Table 2: Checked Exceptions

Exception	Meaning
ClassNotFoundException	Class not found.
CloneNotSupportedException	Attempt to clone an object that does not implement the Cloneable interface.

Exception	Meaning
IllegalAccessException	Access to a class is denied.
InstantiationException	Attempt to create an object of an abstract class or interface.
InterruptedException	One thread has been interrupted by another thread.
NoSuchFieldException	A requested field does not exist.
NoSuchMethodException	A requested method does not exist.

1.4 Throwing our own exception

In Java, you can throw your own exception using the throw keyword. This allows you to create custom error handling for specific conditions in your application.

Steps to Throw Your Own Exception in Java:

1. **Use the throw keyword** to explicitly throw an exception.
2. **Create a custom exception class** by extending Exception (checked) or RuntimeException (unchecked).
3. **Use a constructor** in the custom exception class to pass error messages.
4. **Handle the exception** using try-catch or declare it with throws.

Example 1: Throwing a Built-in Exception

```
public class ThrowExample {
   public static void checkAge(int age) {
      if (age < 18) {
         throw new IllegalArgumentException("Age must be 18 or above.");
      }
      System.out.println("Access granted.");
   }

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

```
        checkAge(15); // This will throw an exception
    }
}
```

Output:

```
Exception in thread "main" java.lang.IllegalArgumentException: Age must be 18 or above.
```