Exception handling
Assignment

Q1. Explain different types of Errors in java?

Ans. Error is an illegal operation performed by the user which results in the abnormal working of the program. Programming errors often remain undetected until the program is compiled or executed.
Types of error:

1. Run Time Error:

Run Time errors occur or we can say, are detected during the execution of the program. Runtime errors occur when a program does not contain any syntax errors but asks the computer to do something that the computer is unable to reliably do. During compilation, the compiler has no technique to detect these kinds of errors. It is the JVM (Java Virtual Machine) that detects it while the program is running. To handle the error during the run time we can put our error code inside the try block and catch the error inside the catch block.

2. Compile Time Error:

Compile Time Errors are those errors which prevent the code from running because of an incorrect syntax such as a missing semicolon at the end of a statement or a missing bracket, class not found, etc. These errors are detected by the java compiler and an error message is displayed on the screen while compiling. Compile Time Errors are sometimes also referred to as Syntax errors.

Logical Error: A logic error is when your program compiles and executes, but does the wrong thing or returns an incorrect result or no output when it should be returning an output. These errors are detected neither by the compiler nor by JVM.

Q2. What is an Exception in java?

Ans. In Java, Exception is an unwanted or unexpected event, which occurs during the execution of a program, i.e. at run time, that disrupts the normal flow of the program's instructions. Exceptions can be caught and handled by the program.

Major reasons why an exception Occurs

- Invalid user input
- Device failure
- Loss of network connection
- Physical limitations (out-of-disk memory)
- Code errors
- Out of bound
- Null reference
- Type mismatch
- Opening an unavailable file
- Database errors
- Arithmetic errors

Q3. How can you handle exceptions in java? Explain with an example.
Ans. The Exception Handling in Java is one of the powerful *mechanisms to handle the runtime errors* so that the normal flow of the application can be maintained.

Java try-catch block

Java try block

Java try block is used to enclose the code that might throw an exception. It must be used within the method.

Java catch block

Java catch block is used to handle the Exception by declaring the type of exception within the parameter. The declared exception must be the parent class exception (i.e., Exception) or the generated exception type. However, the good approach is to declare the generated type of exception.

Syntax of Java try-catch

1. try{
2. //code that may throw an exception
3. }catch(Exception_class_Name ref){}

Example:-

```
public class TryCatchExample2 {


  public static void main(String[] args) {
    try
    {
    int data=50/0;
    }
          catch(ArithmeticException e)
    {
      System.out.println(e);
    }
    System.out.println("rest of the code");
  }


}
```

Java throw keyword

The Java throw keyword is used to throw an exception explicitly.

We specify the **exception** object which is to be thrown. The Exception has some message with it that provides the error description. These exceptions may be related to user inputs, server, etc.

Example-:

```
1.  public class TestThrow1 {
2.        public static void validate(int age) {
3.        if(age<18) {
4.                  throw new ArithmeticException("Person is not eligible to vote");
5.        }
6.        else {
7.           System.out.println("Person is eligible to vote!!");
8.        }
9.     }
10.     public static void main(String args[]){
11.           validate(13);
12.       System.out.println("rest of the code...");
13.  }
14.}
```

Q4. Why do we need exception handling in java?
Ans. Exception handling is the process of responding to unwanted or unexpected events when a computer program runs. Exception handling deals with these events to avoid the program or system crashing, and without this process, exceptions would disrupt the normal operation of a program.

Q5. What is the difference between exception and error in java?
Ans.

Errors	Exceptions

Recovering from Error is not possible.	We can recover from exceptions by either using try-catch block or throwing exceptions back to the caller.
All errors in java are unchecked type.	Exceptions include both checked as well as unchecked type.
Errors are mostly caused by the environment in which program is running.	Program itself is responsible for causing exceptions.
Errors can occur at compile time.	Unchecked exceptions occur at runtime whereas checked exceptions occur at compile time
They are defined in java.lang.Error package.	They are defined in java.lang.Exception package

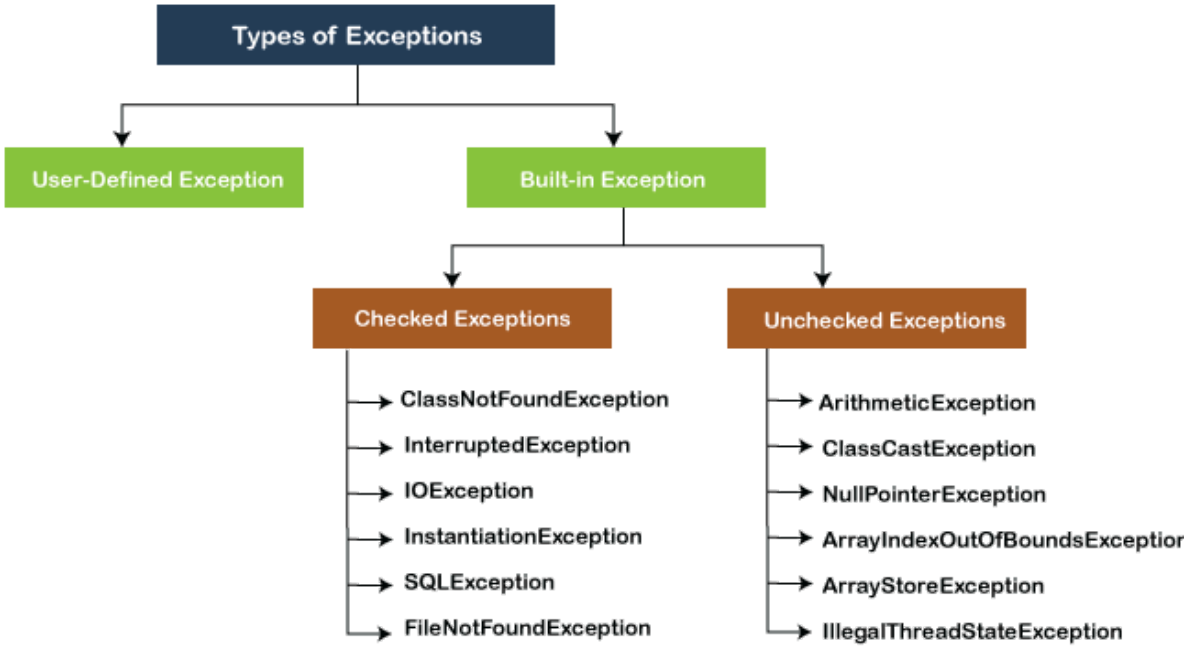
Examples : java.lang.StackOverflowError, java.lang.OutOfMemoryError	Examples : Checked Exceptions : SQLException, IOException Unchecked Exceptions : ArrayIndexOutOfBoundException, NullPointerException, ArithmeticException.

Q6. Name the different types of exceptions in java?

Ans. In Java, exception is an event that occurs during the execution of a program and disrupts the normal flow of the program's instructions. Bugs or errors that we don't want and restrict our program's normal execution of code are referred to as exceptions. In this section, we will focus on the types of exceptions in Java and the differences between the two.

Exceptions can be categorized into two ways:

1. Built-in Exceptions
    - Checked Exception
    - Unchecked Exception
2. User-Defined Exceptions



Built-in Exception

Exceptions that are already available in Java libraries are referred to as built-in exception. These exceptions are able to define the error situation so that we can understand the reason of getting this error. It can be categorized into two broad categories, i.e., checked exceptions and unchecked exceptions.

Checked Exception

Checked exceptions are called compile-time exceptions because these exceptions are checked at compile-time by the compiler. The compiler ensures whether the programmer handles the exception or not. The programmer should have to handle the exception; otherwise, the system has shown a compilation error

Example-:

```
1. import java.io.*;
2. class CheckedExceptionExample {
3.     public static void main(String args[]) {
4.         FileInputStream file_data = null;
5.         file_data = new
   FileInputStream("C:/Users/ajeet/OneDrive/Desktop/Hello.txt");
6.         int m;
```

```
7.          while(( m = file_data.read() ) != -1) {
8.              System.out.print((char)m);
9.          }
10.         file_data.close();
11.     }
12.}
```

Unchecked Exceptions

The **unchecked** exceptions are just opposite to the **checked** exceptions. The compiler will not check these exceptions at compile time. In simple words, if a program throws an unchecked exception, and even if we didn't handle or declare it, the program would not give a compilation error. Usually, it occurs when the user provides bad data during the interaction with the program

```
1.  class UncheckedExceptionExample1 {
2.     public static void main(String args[])
3.     {
4.      int positive = 35;
5.      int zero = 0;
6.      int result = positive/zero;
7.    System.out.println(result);
8.     }
9.  }
```

User-defined Exception

In Java, we already have some built-in exception classes like **ArrayIndexOutOfBoundsException, NullPointerException**, and **ArithmeticException**. These exceptions are restricted to trigger on some predefined conditions. In Java, we can write our own exception class by extends the Exception class. We can throw our own exception on a particular condition using the throw keyword. For creating a

user-defined exception, we should have basic knowledge of **the try-catch** block and **throw** keyword.

Q7. Can we just use try instead of finally and catch blocks?

Ans The try block is always followed by a catch block, which handles the exception that occurs in the associated try block. A try block must be used within the method and it must be followed by a catch block(s) or finally block or both.

The answer is "No, it is not mandatory that each try block must be followed by a catch block in Java."

- After try block, we can use either "catch" block or "finally" block.
- Generally, thrown exceptions should be declared in the thrown clause of the method.
- To understand the try-catch block, we will discuss three cases:
    1. Behaviour when each try block must be followed by a catch block?
    2. Behaviour when each try block must be followed by a finally block?
    3. Behaviour when each try block must be followed by both catch and finally block?