# **Assignment 5**

Q1. What is Exception in Java?

The term *exception* is shorthand for the phrase "exceptional event."

**Definition:** An *exception* is an event, which occurs during the execution of a program, that disrupts the normal flow of the program's instructions.

When an error occurs within a method, the method creates an object and hands it off to the runtime system. The object, called an *exception object*, contains information about the error, including its type and the state of the program when the error occurred. Creating an exception object and handing it to the runtime system is called *throwing an exception*.

After a method throws an exception, the runtime system attempts to find something to handle it. The set of possible "somethings" to handle the exception is the ordered list of methods that had been called to get to the method where the error occurred.

Q2. What is Exception Handling?

Exception Handling is a mechanism to handle runtime errors such as ClassNotFoundException, IOException, SQLException, RemoteException, etc.

### Advantage of Exception Handling

The core advantage of exception handling is **to maintain the normal flow of the application**. An exception normally disrupts the normal flow of the application; that is why we need to handle exceptions.

Let's consider a scenario:

1. statement 1;
2. statement 2;
3. statement 3;
4. statement 4;
5. statement 5;//exception occurs
6. statement 6;
7. statement 7;
8. statement 8;
9. statement 9;
10. statement 10;

Suppose there are 10 statements in a Java program and an exception occurs at statement 5; the rest of the code will not be executed, i.e., statements 6 to 10 will not be executed. However, when we perform exception handling, the rest of the statements will be executed. That is why we use exception handling in Java.

Q3. What is the difference between Checked and Unchecked Exceptions and Error?

Error

Errors are usually caused by serious problems that are outside the control of the program, such as running out of memory or a system crash. Errors are represented by the Error class and its subclasses. Some common examples of errors in Java include:

* **OutOfMemoryError:** Thrown when the Java Virtual Machine (JVM) runs out of memory.
* **StackOverflowError:** Thrown when the call stack overflows due to too many method invocations.
* **NoClassDefFoundError:** Thrown when a required class cannot be found.

Since errors are generally caused by problems that cannot be recovered from, it’s usually not appropriate for a program to catch errors. Instead, the best course of action is usually to log the error and exit the program.

**Exception**

**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.

In Java, there are two types of exceptions:

Checked exceptions

Unchecked exceptions

## Checked Exceptions in Java

Theseare the exceptions that are checked at compile time. If some code within a method throws a checked exception, then the method must either handle the exception or it must specify the exception using the *throws* keyword. In checked exceptions, there are two types: fully checked and partially checked exceptions.

// Java Program to Illustrate Checked Exceptions

// Where FileNotFoundException occurred

class Test {

// Main driver method

public static void main(String[] args)

{

// Reading file from path in local directory

FileReader file = new FileReader("C:\\test\\a.txt");

// Creating object as one of ways of taking input

BufferedReader fileInput = new BufferedReader(file);

// Printing first 3 lines of file "C:\test\a.txt"

for (int counter = 0; counter < 3; counter++)

System.out.println(fileInput.readLine());

// Closing file connections

// using close() method

fileInput.close();

}

}

## Unchecked Exceptions in Java

These are the exceptions that are not checked at compile time. In C++, all exceptions are unchecked, so it is not forced by the compiler’s to either handle or specify the exception. It is up to the programmers to be civilized and specify or catch the exceptions. In Java, exceptions under *Error* and *RuntimeException* classes are unchecked exceptions, everything else under throwable is checked.

**Here are some examples of unchecked exceptions in Java:**

***1. ArrayIndexOutOfBoundsException:*** *This exception is thrown when you attempt to access an array index that is out of bounds.*  
**2. NullPointerException:** *This exception is thrown when you attempt to access a null object reference.*  
**3. ArithmeticException:** *This exception is thrown when you attempt to divide by zero or perform an invalid arithmetic operation.*

Q4. What is the difference between throw and throws in Java?

The throw and throws is the concept of exception handling where the throw keyword throw the exception explicitly from a method or a block of code whereas the throws keyword is used in signature of the method.

There are many differences between throw and throws keywords. A list of differences between throw and throws are given below:

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr. no.** | **Basis of Differences** | **throw** | **throws** |
| 1. | Definition | Java throw keyword is used throw an exception explicitly in the code, inside the function or the block of code. | Java throws keyword is used in the method signature to declare an exception which might be thrown by the function while the execution of the code. |
| 2. | Type of exception Using throw keyword, we can only propagate unchecked exception i.e., the checked exception cannot be propagated using throw only. | Using throws keyword, we can declare both checked and unchecked exceptions. However, the throws keyword can be used to propagate checked exceptions only. |  |
| 3. | Syntax | The throw keyword is followed by an instance of Exception to be thrown. | The throws keyword is followed by class names of Exceptions to be thrown. |
| 4. | Declaration | throw is used within the method. | throws is used with the method signature. |
| 5. | Internal implementation | We are allowed to throw only one exception at a time i.e. we cannot throw multiple exceptions. | We can declare multiple exceptions using throws keyword that can be thrown by the method. For example, main() throws IOException, SQLException. |

Q5. What is multithreading in Java? mention its advantages

**Multithreading in Java** is a process of executing multiple threads simultaneously.

A thread is a lightweight sub-process, the smallest unit of processing. Multiprocessing and multithreading, both are used to achieve multitasking.

However, we use multithreading than multiprocessing because threads use a shared memory area. They don't allocate separate memory areas, so they save memory, and context-switching between the threads takes less time than process.

Advantages of Java Multithreading

1) It **doesn't block the user** because threads are independent, and you can perform multiple operations at the same time.

2) You **can perform many operations together, so it saves time**.

3) Threads are **independent**, so it doesn't affect other threads if an exception occurs in a single thread.

Q6. Write a program to create and call a custom exception

we can create our own exceptions that are derived classes of the Exception class. Creating our own Exception is known as custom exception or user-defined exception. Basically, Java custom exceptions are used to customize the exception according to user need.

Consider example 1 in which InvalidAgeException class extends the Exception class.

Using the custom exception, we can have your own exception and message. Here, we have passed a string to the constructor of superclass i.e. Exception class that can be obtained using getMessage() method on the object we have created.

**Example 1**

// class representing custom exception

**class** InvalidAgeException **extends** Exception

{

**public** InvalidAgeException (String str)

{

// calling the constructor of parent Exception

**super**(str);

}

}

// class that uses custom exception InvalidAgeException

**public** **class** TestCustomException1

{

// method to check the age

**static** **void** validate (**int** age) **throws** InvalidAgeException{

**if**(age < 18){

// throw an object of user defined exception

**throw** **new** InvalidAgeException("age is not valid to vote");

}

**else** {

System.out.println("welcome to vote");

}

}

// main method

**public** **static** **void** main(String args[])

{

**try**

{

// calling the method

validate(13);

}

**catch** (InvalidAgeException ex)

{

System.out.println("Caught the exception");

// printing the message from InvalidAgeException object

System.out.println("Exception occured: " + ex);

}

System.out.println("rest of the code...");

} }

Q7. How can you handle exceptions in Java?

Java provides five keywords that are used to handle the exception. The following table describes each.

|  |  |
| --- | --- |
| **Keyword** | **Description** |
| try | The "try" keyword is used to specify a block where we should place an exception code. It means we can't use try block alone. The try block must be followed by either catch or finally. |
| catch | The "catch" block is used to handle the exception. It must be preceded by try block which means we can't use catch block alone. It can be followed by finally block later. |
| finally | The "finally" block is used to execute the necessary code of the program. It is executed whether an exception is handled or not. |
| throw | The "throw" keyword is used to throw an exception. |
| throws | The "throws" keyword is used to declare exceptions. It specifies that there may occur an exception in the method. It doesn't throw an exception. It is always used with method signature. |

Let's see an example of Java Exception Handling in which we are using a try-catch statement to handle the exception.

**public** **class** JavaExceptionExample{

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

**try**{

//code that may raise exception

**int** data=100/0;

}

**catch**(ArithmeticException e)

{System.out.println(e);}

//rest code of the program

System.out.println("rest of the code...");

}

}

In the above example, 100/0 raises an ArithmeticException which is handled by a try-catch block.

Q8. What is Thread in Java?

A **Thread** is a very light-weighted process, or we can say the smallest part of the process that allows a program to operate more efficiently by running multiple tasks simultaneously.

Thread is a:

* Feature through which we can perform multiple activities within a single process.
* Lightweight process.
* Series of executed statements.
* Nested sequence of method calls.

## Thread Model

Just like a process, a thread exists in several states. These states are as follows:

**1) New (Ready to run)**

A thread is in **New** when it gets CPU time.

**2) Running**

A thread is in **a Running** state when it is under execution.

**3) Suspended**

A thread is in the **Suspended** state when it is temporarily inactive or under execution.

**4) Blocked**

A thread is in the **Blocked** state when it is waiting for resources.

**5) Terminated**

A thread comes in this state when at any given time, it halts its execution immediately.

Q9. What are the two ways of implementing thread in Java?

**Threads** can be used to perform complicated tasks in the background without interrupting the main program.

### **Different ways of creating Threads**

There are 2 different ways to create Threads in Java.

1 By extending Thread class

2 By implementing Runnable interface

#### **By extending Thread class**

Let’s look at the complete implementation

#### public class Main extends Thread {

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

#### Main thread = new Main();

#### thread.start();

#### System.out.println("This code is running outside of the thread");

#### }

#### public void run () {

#### System.out.println("This code is running in a thread");

#### }

#### }

#### **By implementing Runnable interface**

Let’s look at the complete implementation

public class Main implements Runnable {

public static void main(String[] args) {

Main obj = new Main();

Thread thread = new Thread(obj);

thread.start();

System.out.println("This code is running outside of the thread");

}

public void run () {

System.out.println("This code is running in a thread");

} }

Q10. What do you mean by garbage collection?

Garbage Collection is the process of reclaiming the runtime unused memory automatically. In other words, it is a way to destroy unused objects.

### Advantage of Garbage Collection

* It makes java **memory efficient** because garbage collector removes the unreferenced objects from heap memory.
* It is **automatically done** by the garbage collector(a part of JVM) so we don't need to make extra efforts.

## How can an object be unreferenced?

There are many ways:

* By nulling the reference
* By assigning a reference to another
* By anonymous object etc.

### 

### Simple Example of garbage collection

**public** **class** TestGarbage1{

**public** **void** finalize(){System.out.println("object is garbage collected");}

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

TestGarbage1 s1=**new** TestGarbage1();

TestGarbage1 s2=**new** TestGarbage1();

s1=**null**;

s2=**null**;

System.gc();

}

}