1. **What is exception?**

Ans:

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

1. **Who raises Exceptions in java?**

Ans:

In Java, exceptions can be raised by either an application ( by developer ) or by JVM.

1. **What do we write in try and what do we write in catch blocks?**

Ans:

In try block we write possible statements which might cause exception/s. in catch block we write reference of a particular exception class in order to handle it.

1. **What is the difference between throw and throws?**

Ans:

throw keyword is used to raise an exception whereas throws keyword is used to declare the exception/s.

1. **What is the exact syntax of raising an exception from your application code?**

Ans:

throw new <particular exception class name>();

1. **What is finally block in exception handling?**

Ans:

finally block gets executed irrespective of whether exception is raised or not.

it can be used to release resources such as file,socket,database connection etc. since you can not rely upon "finalized" method for the same task.

finally block can follow after catch (try..catch...finally ) or even after try (try...finally).

1. **What are the scenarios in which finally block does not get called?**

Ans:

finally block will not get executed if

a) System.exit(0) is called inside try ,catch or finally block.

b) exception gets raised in finally block itself.

**8) what exactly happens when an exception gets raised?**

Ans:

1. A particular exception class is instantiated
2. That instance is thrown to the application ( to the method ) in which exception has been raised.
3. **What is handle or declare rule?**

Ans:

a) whenever any method raises **checked** exception/s , method has to either **handle [try….catch]** or **declare [throws]** that checked exception/s.

b) whenever u invoke a method which has **declared [using throws] checked exception/s** , caller method has to either **handle [try….catch] or declare [throws]** that checked exception/s.

**10) does catch(Object) work in java?**

Ans:

No.

because if "catch(Object)" is allowed , then people can write

throw new String("hello");

throw new Integer(20);

throw new Sample();

etc.

and the rule is along with "throw" you can have class of type "Throwable" only.

1. **How do you create user defined checked and unchecked exceptions?**

Ans:

If you want to create user defined checked exception you’ve to derive the class from Exception class whereas in case of unchecked exception you’ve to derive the class from RuntimeException class.

1. On what basis you decide whether to create checked exception or unchecked exception?

Ans:

if you want that client code should take some corrective actions based on the exception raised , you will create checked exception. It is because when you create checked exception, client code will have to handle it compulsorily and while doing so, client code can take corrective action/s.

if you don't want client code to take corrective actions based on the exception , you will create unchecked exception.

1. **What are checked and unchecked exceptions?**

Ans:

* **Checked exceptions:** Exceptions that inherit from the Exception class are checked exceptions. Checked exceptions are those exceptions which can be raised in a correct program. Client code has to handle the checked exceptions thrown by the API, either in a catch clause or by forwarding it outward with the throws clause.
* **Unchecked exceptions:**Unchecked exceptions are those exceptions which can be raised due to programming mistakes. RuntimeException also extends from Exception. However, all of the exceptions that inherit from RuntimeException get special treatment. There is no requirement for the client code to deal with them, and hence they are called unchecked exceptions.

1. **What is ARM?**

Ans:

**ARM (Automatic Resource Management) in Exception Handling**

**ARM (Automatic Resource Management)** refers to Java's **try-with-resources** feature, introduced in **Java 7**, which automatically **closes resources** (like files, sockets, database connections) when they are no longer needed.

This helps prevent **resource leaks** (e.g., forgetting to close a file or database connection) and **reduces boilerplate code**.

**Why is ARM Important?**

Before ARM, resources like file streams had to be manually closed in a finally block, which was error-prone.

**Example (Before ARM - Without try-with-resources)**

import java.io.\*;

public class WithoutARM {

public static void main(String[] args) {

BufferedReader br = null;

try {

br = new BufferedReader(new FileReader("test.txt"));

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

} catch (IOException e) {

e.printStackTrace();

} finally {

try {

if (br != null) br.close(); // Manual closing

} catch (IOException ex) {

ex.printStackTrace();

}

}

}

}

**Problems:**

**Verbose Code**: Requires explicit closing in a finally block.  
**Error-Prone**: Forgetting to close resources leads to memory leaks.  
**Nested try-catch Blocks**: More complexity when handling exceptions.

**How ARM Works (try-with-resources)**

With ARM, you can use try-with-resources to automatically close resources.

**Example (With ARM - Using try-with-resources)**

import java.io.\*;

public class WithARM {

public static void main(String[] args) {

try (BufferedReader br = new BufferedReader(new FileReader("test.txt"))) {

System.out.println(br.readLine()); // No need for explicit close()

} catch (IOException e) {

e.printStackTrace();

}

}

}

**Advantages:**

**No finally Block Needed**: Resource is closed automatically.  
**Cleaner and Safer Code**: Reduces boilerplate code.  
**Exception Safety**: Resources are properly closed even if an exception occurs.

**How Does ARM Work Internally?**

* We write try with resource block i.e.

try(FileInputStream fis=new FileInputStream(“abc.txt”))

{

// some code

}

catch(IOException ie)

{

Ie.printStackTrace();

}

Though we haven’t written “finally” block compiler does the following trick:

Compiler converts above code into the following code:

FileInputStream fis;

try

{

fis=new FileInputStream(“abc.txt”);

// some code

}

catch(IOException ie)

{

Ie.printStackTrace();

}

finally

{

fis.close();

}

So you can see that compiler internally provides finally block inside which it invokes fis.close(). This way resource management is automatically or internally taken care.

1. **Can We Use Any Class Inside try-with-resources Block?**

Ans:

No, **only classes that implement AutoCloseable or Closeable can be used inside a try-with-resources block**.

Closeable is **specific to I/O operations**, while AutoCloseable is **more general**.

1. **What is the difference between Error and Exception?**

Ans:

In Java, both Error and Exception are subclasses of Throwable, but they represent different types of problems.

Error represents serious system-level issues (e.g., OutOfMemoryError, StackOverflowError) that cannot be recovered from. Even if caught, the JVM might be in an unstable state, so continuing execution is unsafe. The best solution is to fix the underlying issue or restart the application.

Exception represents application-level issues (e.g., NullPointerException, IOException) that can be handled using try-catch. The program can recover and continue execution normally.