**What is the base class for all exceptions in Java?**

Answer: The base class for all exceptions in Java is java.lang.Throwable. It has two main subclasses: Exception and Error.

**What is the difference between Exception and Error in Java?**

Answer: Exception represents conditions that a reasonable application might want to catch, such as IOException or NullPointerException. Error represents serious problems that a reasonable application should not try to catch, such as OutOfMemoryError or StackOverflowError.

**What is the difference between checked and unchecked exceptions?**

Answer: Checked exceptions: These are checked at compile time. The Java compiler ensures that your code handles these exceptions. Examples include IOException, SQLException.

Unchecked exceptions: These are not checked at compile time, but rather at runtime. They are subclasses of RuntimeException. Examples include NullPointerException, ArrayIndexOutOfBoundsException.

**How do you create a custom exception in Java?**

Answer: To create a custom exception, you need to extend the Exception class (for checked exceptions) or RuntimeException class (for unchecked exceptions). For example:

public class MyCustomException extends Exception {

public MyCustomException(String message) {

super(message);

}

}

**What is the purpose of the finally block in exception handling?**

Answer: The finally block is used to execute important code such as closing resources, regardless of whether an exception is thrown or not. The code inside the finally block always executes after the try block, and any associated catch blocks, even if an exception is not caught.

**Can a finally block be skipped?**

Answer: The finally block is almost always executed. However, it can be skipped in certain situations, such as when the JVM exits during the try or catch block due to a call to System.exit(), or if the thread executing the try or catch block is interrupted or killed.

**What happens if an exception is thrown in a catch or finally block?**

Answer: If an exception is thrown in a catch block, it can be caught by another catch block if it's nested within another try block. If an exception is thrown in a finally block, it will propagate up and can override any exception thrown in the try or catch block.

**How can you rethrow an exception in Java?**

Answer: You can rethrow an exception in a catch block by simply using the throw keyword followed by the exception object. For example:

try {

// code that may throw an exception

} catch (IOException e) {

// log the exception

throw e; // rethrow the exception

}

**What is the multi-catch block in Java?**

Answer: The multi-catch block allows you to catch multiple exceptions in a single catch block using the pipe (|) operator. This feature was introduced in Java 7. For example:

try {

// code that may throw multiple exceptions

} catch (IOException | SQLException ex) {

// handle either IOException or SQLException

ex.printStackTrace();

}

**What is the try-with-resources statement in Java?**

Answer: The try-with-resources statement is a try statement that declares one or more resources. A resource is an object that must be closed after the program is finished with it. The try-with-resources statement ensures that each resource is closed at the end of the statement. This feature was introduced in Java 7. For example:

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

// use the resource

} catch (IOException e) {

e.printStackTrace();

}

**Can you catch multiple exceptions in one catch block? If yes, how?**

Answer: Yes, you can catch multiple exceptions in one catch block using the multi-catch feature introduced in Java 7. This is done using the pipe (|) character to separate the exceptions. For example:

try {

// code that may throw multiple exceptions

} catch (IOException | SQLException e) {

// handle either IOException or SQLException

e.printStackTrace();

}

**What is the purpose of the throws keyword in Java?**

Answer: The throws keyword is used in a method signature to declare that the method can throw one or more exceptions. This informs the callers of the method that they need to handle or declare these exceptions. For example:

public void readFile(String fileName) throws IOException {

// method code

}

**What happens if an exception is not caught?**

Answer: If an exception is not caught, it will propagate up the call stack until it is either caught by a method higher in the stack or reaches the JVM, which will terminate the program and print a stack trace.

**Can you have a try block without a catch block?**

Answer: Yes, you can have a try block without a catch block if you use a finally block or if you are using a try-with-resources statement. For example:

try {

// code that may throw an exception

} finally {

// cleanup code

}

**What is the difference between final, finally, and finalize in Java?**

Answer: final: A keyword used to declare constants, prevent inheritance, and prevent method overriding.

finally: A block used in exception handling to execute important code such as closing resources, regardless of whether an exception is thrown or not.

finalize(): A method used to perform cleanup operations before an object is garbage collected. It is called by the garbage collector before the object is destroyed.

**What is a chained exception in Java?**

Answer: Chained exceptions allow you to relate one exception to another, forming a chain of exceptions. This is useful when you want to wrap a lower-level exception with a higher-level exception. You can use the initCause() method or the constructor of the exception class to set the cause. For example:

try {

// code that may throw an exception

} catch (IOException e) {

throw new RuntimeException("Custom message", e);

}

**How can you create a custom unchecked exception?**

Answer: To create a custom unchecked exception, extend the RuntimeException class. For example:

public class MyUncheckedException extends RuntimeException {

public MyUncheckedException(String message) {

super(message);

}

}

**What is the Throwable class in Java?**

Answer: Throwable is the superclass of all errors and exceptions in Java. Only objects that are instances of this class (or one of its subclasses) can be thrown by the Java Virtual Machine or the throw statement. It has two main subclasses: Error and Exception.

**Can a catch block catch multiple types of exceptions?**

Answer: Prior to Java 7, a single catch block could only catch one type of exception. With the introduction of multi-catch in Java 7, a single catch block can catch multiple types of exceptions using the pipe (|) operator.

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

Answer: throw: Used to explicitly throw an exception from a method or a block of code.

throws: Used in a method signature to declare the exceptions that a method can throw, informing the callers of the method to handle or declare these exceptions.