Exception handling is a crucial aspect of Java programming, and it's a common topic in interviews. Here are some frequently asked interview questions on exception handling in Java, along with their answers:

1. What is an exception in Java?
Answer: An exception in Java is an event that disrupts the normal flow of a program's execution. It is an object that represents an error or an unexpected event that occurs during the execution of a program. Exceptions can be caught and handled to prevent the program from crashing.

2. What is the difference between checked and unchecked exceptions?
Answer:

Checked Exceptions: These are exceptions that are checked at compile-time. The Java compiler requires that methods that can throw checked exceptions must either handle them with a try-catch block or declare them using the throws keyword. Examples include IOException, SQLException, etc.
Unchecked Exceptions: These are exceptions that are not checked at compile-time. They are subclasses of RuntimeException and include errors that are typically due to programming bugs, such as NullPointerException, ArrayIndexOutOfBoundsException, etc.
3. What is the difference between throw and throws?
Answer:

throw: The throw keyword is used to explicitly throw an exception in the code. It is followed by an instance of Throwable (or its subclass).
Java

```
1throw new IllegalArgumentException("Invalid argument");
```

throws: The throws keyword is used in a method signature to declare that a method can throw one or more exceptions. It informs the caller of the method about the exceptions that might be thrown.
Java

```
1public void readFile() throws IOException {
2    // Method implementation
3}
```

4. What is the purpose of the finally block?
Answer: The finally block is used to execute important code such as resource cleanup, regardless of whether an exception is thrown or not. It is executed after the try and catch blocks. The finally block is optional and can be used with or without a catch block.

5. Can a try block exist without a catch block?
Answer: Yes, a try block can exist without a catch block if it is followed by a finally block. The finally block will execute regardless of whether an exception is thrown or not.

Java

```
1try {
2    // Code that may throw an exception
3} finally {
4    // Cleanup code
5}
```

6. What happens if an exception is not caught?
Answer: If an exception is not caught, it propagates up the call stack to the previous method. If it reaches the main method without being caught, the Java runtime system terminates the program and prints the exception's stack trace to the console.

7. How can you create a custom exception in Java?
Answer: You can create a custom exception by extending the Exception class (for checked exceptions) or the RuntimeException class (for unchecked exceptions). Here's an example:

Java

```
1public class CustomException extends Exception {
2    public CustomException(String message) {
3        super(message);
4    }
5}
```

8. What is the try-with-resources statement?

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. It is used to manage resources like files, sockets, etc., and was introduced in Java 7.

Java

```
try (BufferedReader br = new BufferedReader(new FileReader("file.txt"))) {
    // Use the resource
} catch (IOException e) {
    // Handle exception
}
```

9. Can you catch multiple exceptions in a single catch block?
Answer: Yes, starting from Java 7, you can catch multiple exceptions in a single catch block using the pipe (|) operator. This is useful for handling multiple exceptions with the same handling code.

Java

```
try {
    // Code that may throw multiple exceptions
} catch (IOException | SQLException e) {
    // Handle both exceptions
}
```

10. What is the Error class in Java?
Answer: The Error class in Java represents serious problems that a reasonable application should not try to catch. Errors are typically external to the application and indicate issues such as system failures, memory exhaustion, or other critical problems. Examples include OutOfMemoryError, StackOverflowError, etc.

These questions cover a range of topics related to exception handling in Java, providing a solid foundation for understanding how to manage errors and exceptions in Java applications.