12. Exception Handling

1) Types of Exceptions: Checked and Unchecked

Answer:

Checked exceptions are exceptions that are **checked at compile-time**. The Java compiler requires the programmer to handle these exceptions explicitly using **try-catch blocks** or by declaring them with the throws keyword in the method signature.

Unchecked exceptions are exceptions that are **not checked at compile-time**. These exceptions occur at runtime, and the Java compiler does not enforce handling them.

Characteristics:

- Known only at runtime.
- Handling is optional, but it's good practice to handle them where necessary.
- Typically represent programming errors or unexpected conditions.

2) try, catch, finally, throw, throws

Answer:

1. try Block

The try block is used to enclose code that might throw an exception. If an exception occurs within the try block, it is caught and handled by a corresponding catch block.

Syntax:

```
try
{
   // Code that might throw an exception
}
```

2. catch Block

The `catch` block handles the exception thrown by the `try` block. You can specify the type of exception you want to catch.

Syntax:

```
catch (ExceptionType e)
{
   // Code to handle the exception
}
```

3. finally Block

The `finally` block is always executed, regardless of whether an exception is thrown or not. It is generally used for cleanup tasks, such as closing files, releasing resources, or resetting variables.

Syntax:

```
try
{
   // Risky code
} catch (ExceptionType e) {
   // Exception handling
} finally {
   // Cleanup code
}
```

4. throw Keyword

The throw keyword is used to explicitly throw an exception. It is typically used inside a method or block to signal that an exceptional condition has occurred.

Syntax:

```
throw new ExceptionType("Error Message");
```

5. throws Keyword

The throws keyword is used in a method declaration to indicate that the method might throw one or more exceptions. The caller of the method is responsible for handling the exceptions.

Syntax:

```
return_type method_name(parameters) throws ExceptionType1, ExceptionType2
{
   // Method body
}
```

3) Custom Exception Classes

Answer:

In Java, you can create your own custom exceptions to handle application-specific errors. Custom exceptions are particularly useful when the standard exceptions do not fit the needs of your application.

Creating a Custom Exception Class

1. Extend the Exception class for **checked exceptions** or the Runtime Exception class for **unchecked exceptions**.
2. Provide a constructor to initialize the exception with a custom message or additional data.

==========
==========
==========
==========