1.What is the difference between checked and unchecked exceptions?

->

In Java, exceptions are classified into two categories: checked exceptions and unchecked exceptions.

The main difference between them lies in how they are handled by the compiler and the programmer.

1. Checked Exceptions:

- Checked exceptions are the exceptions that are checked at compile-time by the Java compiler.

- These exceptions are derived from the `Exception` class but not from `RuntimeException` or its subclasses.

- Examples of checked exceptions include `IOException`, `SQLException`, and `ClassNotFoundException`.

- Checked exceptions are meant to handle exceptional conditions that are recoverable, such as file I/O errors, database errors, etc.

2. Unchecked Exceptions:

- Unchecked exceptions, also known as runtime exceptions, are not checked at compile-time by the Java compiler.

- These exceptions are derived from the `RuntimeException` class or its subclasses.

- Examples of unchecked exceptions include `NullPointerException`, `ArrayIndexOutOfBoundsException`, and `ArithmeticException`.

- Unchecked exceptions usually indicate programming errors or exceptional conditions that are unlikely to be recovered from.

- Although not enforced by the compiler, it is still possible to handle unchecked exceptions using `try-catch` blocks if desired.

The key distinction between checked and unchecked exceptions is that checked exceptions are checked by the compiler, while unchecked exceptions are not.

2.Explain the difference between a class and an object in Java.

->

In Java, a class and an object are two fundamental concepts of object-oriented programming.

Here's an explanation of the difference between a class and an object:

Class:

- A class is a blueprint or a template for creating objects.

- It defines the properties and behavior that objects of that class will have.

- For example, a class called `Car` can represent the concept of a car and define its properties (e.g., color, model) and behaviors (e.g., start engine, accelerate).

- It is not a Real World Entity. For Eg:car

Object:

- An object is an instance of a class.

- It is a specific, tangible entity that is created based on the blueprint defined by a class.

- When an object is created, it occupies memory and has its own set of values for the attributes defined in the class.

- Objects can have states (attribute values) and behaviors (method invocations).

- Objects can interact with each other by invoking methods and exchanging data.

3.What is the "final" keyword used for in Java?

->

In Java, the `final` keyword is used to create entities that cannot be modified.

It can be applied to variables, methods, and classes, each serving a slightly different purpose:

1. Final Variables:

When a variable is declared as `final`, its value cannot be changed once it is assigned.

It acts as a constant and can only be assigned once.

This is useful when you want to create a variable that should not be modified, such as mathematical constants or configuration values.

eg:

final int x = 10;

x = 20; // Error: Cannot assign a value to a final variable

2. Final Methods:

When a method is declared as `final`, it cannot be overridden by subclasses.

This means that the implementation of the method in the parent class is the final and definitive implementation that cannot be changed by any subclass.

3. Final Classes:

When a class is declared as `final`, it cannot be subclassed.

This is useful when you want to prevent further extension of a class, ensuring that its functionality remains intact and cannot be modified.

The `final` keyword provides immutability, restricts method overriding, and prevents class inheritance, depending on how it is used.

It helps ensure integrity, security, and maintainability in Java code.

4.How does exception handling work in Java? Explain the try-catch-finally block.

->

Exception handling in Java allows you to handle and recover from runtime errors or exceptional situations that may occur during the execution of a program.

Here's how the try-catch-finally block works:

1. `try` block:

The code that may potentially throw an exception is enclosed within the `try` block.

If an exception occurs within the `try` block, it will be caught and handled by the corresponding `catch` block.

2. `catch` block(s):

After the `try` block, you can have one or more `catch` blocks.

Each `catch` block is associated with a specific type of exception it can handle.

If an exception occurs within the `try` block and matches the type of exception specified in a `catch` block, the corresponding `catch` block is executed.

You can have multiple `catch` blocks to handle different types of exceptions.

To print exception we have following methods;

e.g:

catch (Exception e)

{

e.printStackTrace();

e.toString();

e.getmessage();

}

3. `finally` block:

The `finally` block is optional, but if present, it is executed regardless of whether an exception occurred or not.

The code within the `finally` block is executed after the execution of the `try` and `catch` blocks.

It is commonly used for cleanup operations or releasing resources, such as closing file handles or database connections.

Here's a basic example that demonstrates the usage of the try-catch-finally block:

try

{

// Code that may generate an exception

// ...

}

catch (ExceptionType1 e1)

{

// Exception handler for ExceptionType1

// ...

}

finally

{

// Code that will be executed regardless of exception occurrence

// ...

}

If an exception occurs within the `try` block, the program flow immediately jumps to the corresponding `catch` block that matches the exception type.

Once the exception is handled, the program continues execution from the point immediately after the `catch` block.

If no exception occurs, the `catch` block(s) are skipped, and the program proceeds to execute the `finally` block (if present) before moving on.