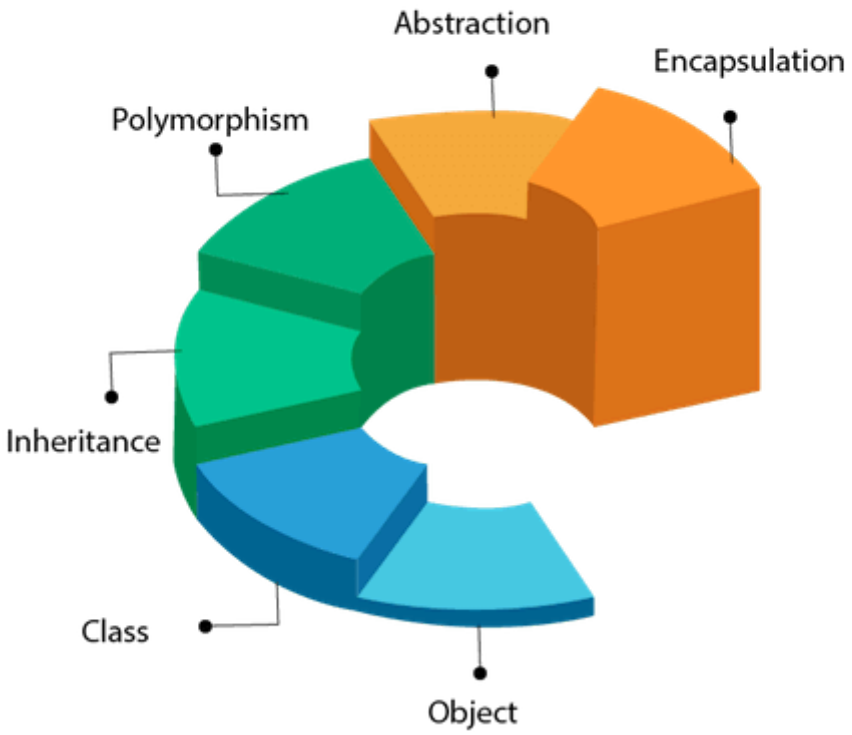
🌟 *Welcome to CodeCoders!* 🚀

OOPs (Object-Oriented Programming System)



OOPs

Object-Oriented Programming is a methodology or paradigm to design a program using classes and objects.

- ***Object***
- ***Class***
- ***Inheritance***
- ***Polymorphism***
- ***Abstraction***
- ***Encapsulation***

1.Object

- *Any entity that has a state and behavior is known as an object. For example, a chair, pen, table, keyboard, bike, etc. It can be physical or logical.*

- *An Object can be defined as an instance of a class.*

- *An object contains an address and takes up some space in memory.*

Example: *A dog is an **object** because it has **states** like color, name, breed, etc.*

*as well as **behaviors** like wagging the tail, barking, eating, etc.*

2.Class

- *Collection of objects is called class. It is a logical entity.*
- *A class is a user-defined blueprint or prototype from which objects are created.*
- *It represents the set of properties or methods that are common to all objects of one type.*
- *Using classes, you can create multiple objects with the same behavior instead of writing their code multiple times.*
- *Class doesn't consume any space.*

🌟 Welcome to CodeCoders! 🚀

Pillars of OOPs 4

1. Inheritance

When one object acquires all the properties and behaviors of a parent object, it is known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.

Inheritance enables runtime polymorphism through method overriding.

We are achieving inheritance by using extends keyword. Inheritance is also known as "is-a" relationship.

A subclass "is-a" Superclass: When a class extends another class, it means that the subclass is a type of superclass. For example, if we have classes like Animal and Dog, and Dog extends Animal, we can say that a Dog is an Animal. This is because a Dog inherits all the characteristics (properties and behaviors) of an Animal and adds its own specialized characteristics.

- ***Superclass:*** *The class whose features are inherited is known as superclass (also known as base or parent class).*

- ***Subclass:*** ***The class that inherits the other class is known as a subclass (also known as derived or extended or child class). The subclass can add its own fields and methods in addition to the superclass fields and methods.***

- ***Reusability:*** ***Inheritance supports the concept of "reusability", i.e. when we want to create a new class and there is already a class that includes some of the code that we want, we can derive our new class from the existing class. By doing this, we are reusing the fields and methods of the existing class.***

2. Polymorphism

Polymorphism is a fundamental concept in object-oriented programming (OOP) that allows objects of different types to be treated as objects of a common superclass. It enables flexibility in designing software systems by allowing tasks to be performed in multiple ways.

In Java, we use method overloading and method overriding to achieve polymorphism.

Method Overloading and Method Overriding:

Method Overloading:(Compile-time Polymorphism) This occurs when multiple methods in a class have the same name but different parameters. Java allows us to define multiple methods with the same name as long as their parameter lists are

different. This enables the same method name to be used for different behaviors based on the parameters passed to it.

Method Overriding:(Runtime Polymorphism) *Inheritance allows a subclass to provide a specific implementation of a method that is already defined in its superclass. This allows us to achieve different behaviors for the same method name in different classes.*

for example, a cat speaks meow, a dog barks woof, etc.

3. Abstraction

Abstraction in Java is the process in which we only show essential details/functionality to the user. The non-essential implementation details are not displayed to the user.

Simple Example to understand Abstraction:

Television remote control is an excellent example of abstraction. It simplifies the interaction with a TV by hiding the complexity behind simple buttons and symbols, making it easy without needing to understand the technical details of how the TV functions.

What is Abstraction in Java?

In Java, abstraction is achieved by interfaces and abstract classes. We can achieve 100% abstraction using interfaces.

Java Abstract classes and Java Abstract methods

Abstract Class: ***An abstract class in Java is declared using the abstract keyword. It's a class that cannot be instantiated on its own, meaning you cannot create objects directly from it. Instead, you must extend an abstract class and provide implementations for its abstract methods.***

Abstract Method: *An abstract method is declared within an abstract class but does not contain any implementation details. It's like a blueprint for a method that must be implemented by any subclass that extends the abstract class. Abstract methods are declared with the abstract keyword and end with a semicolon ';' instead of a method body.*

Partial Implementation: *An abstract class may contain both abstract methods (without implementation) and concrete methods (with implementation). Concrete methods provide default behavior that can be inherited by subclasses, while abstract methods define the structure that subclasses must implement.*

Requirement for Subclasses: *If a subclass extends an abstract class that contains abstract methods, it must provide implementations for all those abstract methods. This is necessary to make sure that the subclass is fully functional.*

Abstract Keyword: *Any class that contains at least one abstract method must itself be declared as abstract using the abstract keyword.*

No Direct Instantiation: *Since abstract classes cannot be instantiated directly, you cannot create objects of an abstract class using the new keyword. You can only create instances of concrete subclasses that extend the abstract class.*

Constructors in Abstract Classes: *Abstract classes can have constructors, including parameterized constructors. However, even if you don't explicitly define a constructor, a default constructor is always present in an abstract class.*

Demonstration of Abstract class

```
//abstract class
abstract class GFG {
   //abstract methods declaration
   abstract void add();
   abstract void mul();
   abstract void div();
}
```

Capsule

4. Encapsulation

Binding (or wrapping) code and data together into a single unit is known as encapsulation. For example, a capsule is wrapped with different medicines.

A Java class is an example of encapsulation. Java bean is the fully encapsulated class because all the data members are private here.

Important Points:

a.) Data Hiding: *Encapsulation hides the internal state of an object from the outside world. This means that the internal state (data members) of a class is not directly accessible from outside the class. Instead, access to the data is controlled through methods (getters and setters).*

b.) Access Specifiers: *In Java, access specifiers such as public, private, protected, and default are used to specify the accessibility of members (variables and methods) of a class. By default, members are private, which means they are accessible only within the same class. This is key to achieving encapsulation.*

c.) Getter and Setter Methods: *Encapsulation typically involves providing public methods (getters and setters) to access and modify the private data members of a class. Getter methods are used to retrieve the values of private variables, while setter methods are used to modify or update the values of private variables.*

d.) Data Validation: *Encapsulation allows for data validation, where you can control the values that are allowed to be set for the data members. This helps in maintaining the integrity and consistency of the data.*

e.) Flexibility and Maintainability: *Encapsulation promotes modularity and helps in building more flexible and maintainable code. By hiding the implementation details, it allows for easier modifications and enhancements without affecting other parts of the codebase.*

f.) Example: Here's a simple example to illustrate encapsulation in Java:

```
public class Person {
    private String name;
    private int age;

    // Getter method for name
    public String getName() {
        return name;
    }

    // Setter method for name
    public void setName(String name) {
        this.name = name;
    }
```

```
    // Getter method for age
    public int getAge() {
        return age;
    }

    // Setter method for age
    public void setAge(int age) {
        if(age >= 0) { // Data validation
            this.age = age;
        } else {
            System.out.println("Age cannot be negative.");
        }
    }
}
```

Constructors in Java

In Java, a constructor is a block of codes similar to the method. It is called when an instance of the class is created. At the time of calling constructor, memory for the object is allocated in the memory.

It is a special type of method which is used to initialize the object.

Every time an object is created using the new() keyword, at least one constructor is called.

It calls a default constructor if there is no constructor available in the class. In such case, Java compiler provides a default constructor by default.

There are two types of constructors in Java: no-arg constructor, and parameterized constructor.

Note: ***It is called constructor because it constructs the values at the time of object creation. It is not necessary to write a constructor for a class. It is because java compiler creates a default constructor if your class doesn't have any.***

Rules for creating Java constructor

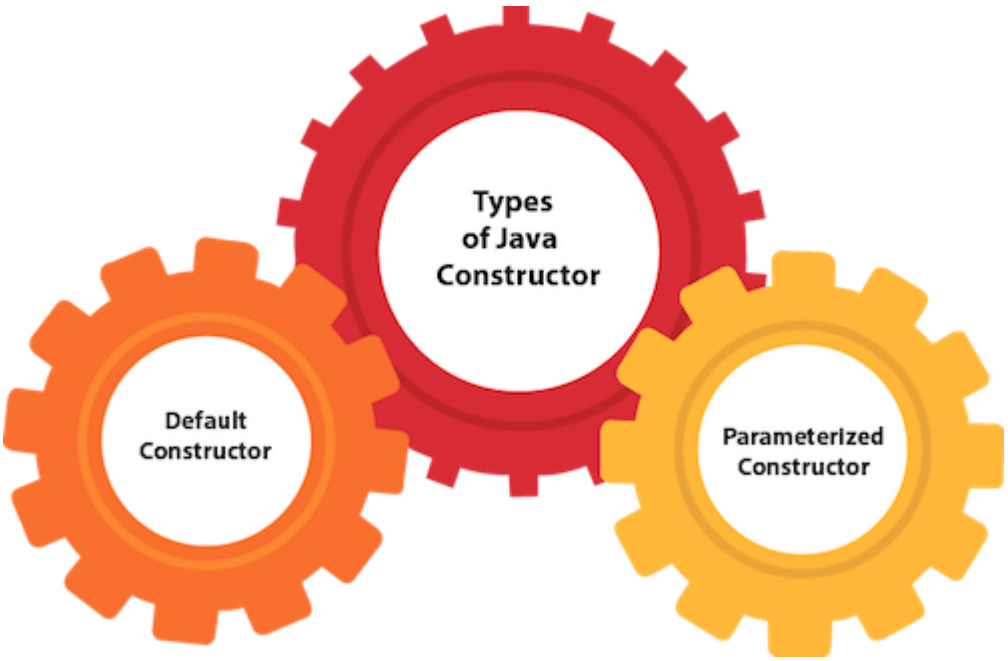
1. ***Constructor name must be the same as its class name***
2. ***A Constructor must have no explicit return type***
3. ***A Java constructor cannot be abstract, static, final, and synchronized***

> *Note: We can use access modifiers while declaring a constructor. It controls the object creation. In other words, we can have private, protected, public or default constructor in Java.*

Types of Java constructors

There are two types of constructors in Java:

1. ***Default constructor (no-arg constructor)***
2. ***Parameterized constructor***



Java Default Constructor

A constructor is called "Default Constructor" when it doesn't have any parameter.

Syntax of default constructor:

1. ***<class_name>(){}***

Example of default constructor

In this example, we are creating the no-arg constructor in the Bike class. It will be invoked at the time of object creation.

```
//Java Program to create and call a default constructor

class Bike1{

//creating a default constructor

Bike1(){

System.out.println("Bike is created");

}

//main method

public static void main(String args[]){

//calling a default constructor

Bike1 b=new Bike1();

}

}
```

Output: *Bike is created*

Rule: If there is no constructor in a class, compiler automatically creates a default constructor.

Q) What is the purpose of a default constructor?

The default constructor is used to provide the default values to the object like 0, null, etc., depending on the type.

Example of default constructor that displays the default values

```
//Let us see another example of default constructor

//which displays the default values

class Student3{

int id;

String name;

//method to display the value of id and name

void display(){System.out.println(id+" "+name);}



public static void main(String args[]){

//creating objects

Student3 s1=new Student3();

Student3 s2=new Student3();

//displaying values of the object

s1.display();

s2.display();

}

}
```

Output:

0 null

0 null

Explanation: ***In the above class,you are not creating any constructor so compiler provides you a default constructor. Here 0 and null values are provided by default constructor.***

Java Parameterized Constructor

A constructor which has a specific number of parameters is called a parameterized constructor.

Why use the parameterized constructor?

The parameterized constructor is used to provide different values to distinct objects. However, you can provide the same values also.

Example of parameterized constructor

In this example, we have created the constructor of Student class that have two parameters. We can have any number of parameters in the constructor.

```
//Java Program to demonstrate the use of the parameterized
constructor.
class Student4{
    int id;
    String name;
    //creating a parameterized constructor
    Student4(int i,String n){
    id = i;
    name = n;
    }
    //method to display the values
    void display(){System.out.println(id+" "+name);}

    public static void main(String args[]){
    //creating objects and passing values
    Student4 s1 = new Student4(111,"Karan");
    Student4 s2 = new Student4(222,"Aryan");
    //calling method to display the values of object
    s1.display();
    s2.display();
   }
}
```

Output:

111 Karan

222 Aryan

Constructor Overloading in Java

In Java, a constructor is just like a method but without return type. It can also be overloaded like Java methods.

Constructor overloading in Java is a technique of having more than one constructor with different parameter lists. They are arranged in a way that each constructor performs a different task. They are differentiated by the compiler by the number of parameters in the list and their types.

Example of Constructor Overloading

```
//Java program to overload constructors
class Student5{
    int id;
    String name;
    int age;
    //creating two arg constructor
    Student5(int i,String n){
```

```
    id = i;
    name = n;
    }
    //creating three arg constructor
    Student5(int i,String n,int a){
    id = i;
    name = n;
    age=a;
    }
    void display(){System.out.println(id+" "+name+" "+age);}

    public static void main(String args[]){
    Student5 s1 = new Student5(111,"Karan");
    Student5 s2 = new Student5(222,"Aryan",25);
    s1.display();
    s2.display();
   }
}
```

Output:

```
111 Karan 0
222 Aryan 25
```

Difference between constructor and method in Java

There are many differences between constructors and methods. They are given below.

Java Constructor	***Java Method***
A constructor is used to initialize the state of an object.	***A method is used to expose the behavior of an object.***
A constructor must not have a return type.	***A method must have a return type.***
The constructor is invoked implicitly.	***The method is invoked explicitly.***
The Java compiler provides a default constructor if you don't have any constructor in a class.	***The method is not provided by the compiler in any case.***
The constructor name must be same as the class name.	***The method name may or may not be same as the class name.***

Java Copy Constructor

There is no copy constructor in Java. However, we can copy the values from one object to another like copy constructor in C++.

There are many ways to copy the values of one object into another in Java. They are:

- ***By constructor***
- ***By assigning the values of one object into another***
- ***By clone() method of Object class***

In this example, we are going to copy the values of one object into another using Java constructor.

```
//Java program to initialize the values from one object to another object.
class Student6{
    int id;
    String name;
    //constructor to initialize integer and string
    Student6(int i,String n){
    id = i;
    name = n;
```

```
    }
    //constructor to initialize another object
    Student6(Student6 s){
    id = s.id;
    name =s.name;
    }
    void display(){System.out.println(id+" "+name);}

    public static void main(String args[]){
    Student6 s1 = new Student6(111,"Karan");
    Student6 s2 = new Student6(s1);
    s1.display();
    s2.display();
   }
}
```

Output:

```
111 Karan

111 Karan
```

Copying values without constructor

We can copy the values of one object into another by assigning the objects values to another object. In this case, there is no need to create the constructor.

```
class Student7{
   int id;
   String name;
   Student7(int i,String n){
   id = i;
   name = n;
   }
   Student7(){}
   void display(){System.out.println(id+" "+name);}

   public static void main(String args[]){
   Student7 s1 = new Student7(111,"Karan");
   Student7 s2 = new Student7();
   s2.id=s1.id;
   s2.name=s1.name;
   s1.display();
   s2.display();
  }
```

}

Output:

111 Karan

111 Karan

Q) Does constructor return any value?

Yes, it is the current class instance (You cannot use return type yet it returns a value).

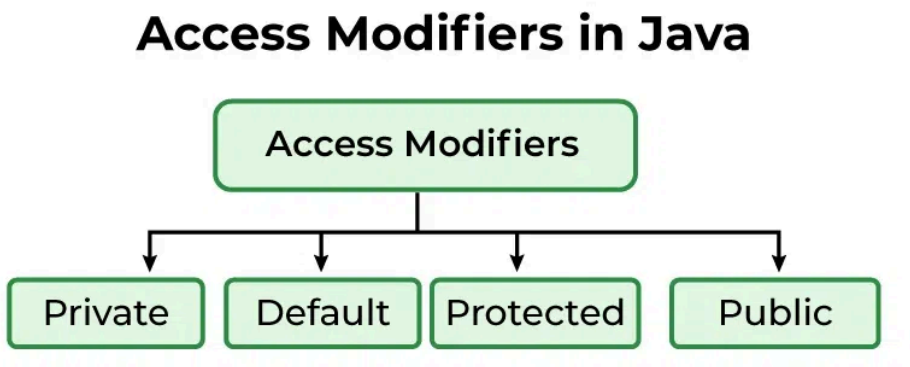
Can constructor perform other tasks instead of initialization?

Yes, like object creation, starting a thread, calling a method, etc. You can perform any operation in the constructor as you perform in the method.

Is there Constructor class in Java?

Yes.

What is the purpose of Constructor class? ***Java provides a Constructor class which can be used to get the internal information of a constructor in the class. It is found in the java.lang.reflect package.***



Access Modifiers in Java

There are two types of modifiers in Java: access modifiers and non-access modifiers.

The access modifiers in Java specifies the accessibility or scope of a field, method, constructor, or class. We can change the access level of fields, constructors, methods, and class by applying the access modifier on it.

There are many non-access modifiers, such as static, abstract, synchronized, native, volatile, transient, etc. Here, we are going to learn the access modifiers only.

Types of Access Modifiers in Java

There are four types of access modifiers available in Java:

1. ***Default – No keyword required***
2. ***Private***
3. ***Protected***
4. ***Public***

1. ***Private:*** ***The access level of a private modifier is only within the class. It cannot be accessed from outside the class.***
2. ***Default:*** ***The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.***
3. ***Protected:*** ***The access level of a protected modifier is within the package and outside the package through the child class. If you do not make the child class, it cannot be accessed from outside the package.***
4. ***Public:*** ***The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package, and outside the package.***

	Default	Private	Protected	Public
Same Class	Yes	Yes	Yes	Yes
Same Package Subclass	Yes	No	Yes	Yes
Same Package Non-Subclass	Yes	No	Yes	Yes
Different Package Subclass	No	No	Yes	Yes
Different Package Non-Subclass	No	No	No	Yes

Exception Handling in Java

The Exception Handling in Java is one of the powerful mechanism to handle the runtime errors so that the normal flow of the application can be maintained.

What is Exception in Java?

In Java, an exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime.

What is Exception Handling?

Exception Handling is a mechanism to handle runtime errors such as ClassNotFoundException, IOException, SQLException, RemoteException, etc.

Advantages of Exception Handling

The core advantage of exception handling is to maintain the normal flow of the application. An exception normally disrupts the normal flow of the application; that is why we need to handle exceptions. Let's consider a scenario:

1. ***statement 1;***
2. ***statement 2;***
3. ***statement 3;***
4. ***statement 4;***
5. ***statement 5;//exception occurs***
6. ***statement 6;***
7. ***statement 7;***
8. ***statement 8;***
9. ***statement 9;***
10. ***statement 10;***

Suppose there are 10 statements in a Java program and an exception occurs at statement 5; the rest of the code will not be executed, i.e., statements 6 to 10 will not be executed. However, when we perform exception handling, the rest of the statements will be executed. That is why we use exception handling in Java.

What is an Exception?

Definition:

- ***An exception is an abnormal condition that arises during the execution of a program.***
- ***It is an object that represents an error or an unexpected event that occurs during runtime.***

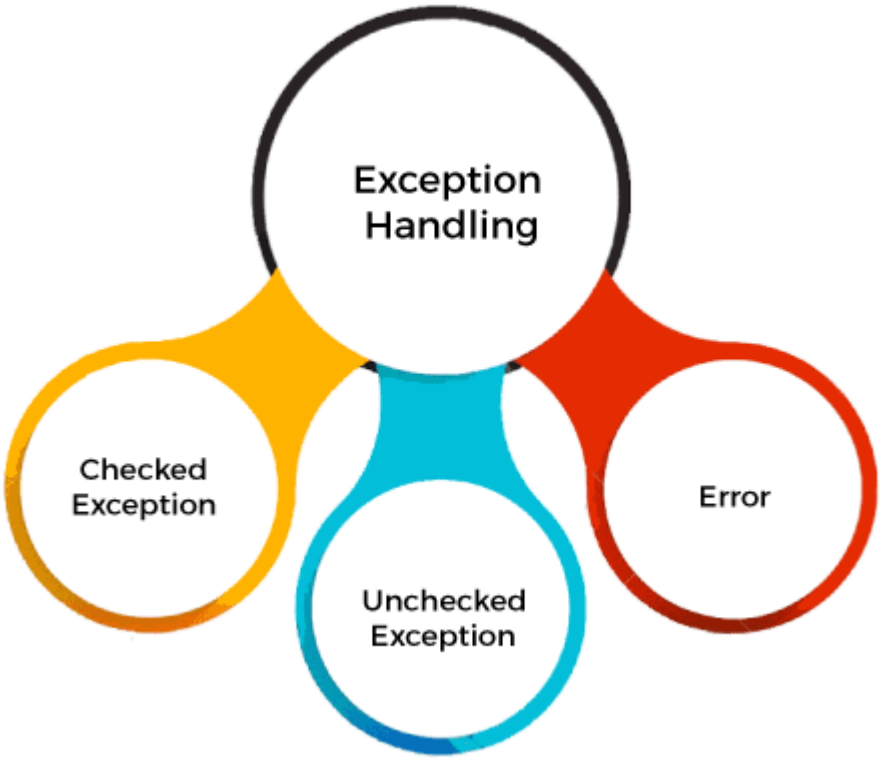
What is Exception Handling?

Exception handling in Java is a mechanism to handle runtime errors, so the normal flow of the application can be maintained. When an error occurs, Java creates an object representing the error. This object is called an exception object. Exception handling provides a way to catch this object and take appropriate actions instead of letting the program crash.

Types of Java Exceptions

There are mainly two types of exceptions: checked and unchecked. An error is considered as an unchecked exception. However, according to Oracle, there are three types of exceptions namely:

1. ***Checked Exception***
2. ***Unchecked Exception***
3. ***Error***



Difference between Checked and Unchecked Exceptions

1) Checked Exception

Checked exceptions are exceptions that are checked by the compiler at compile time. This means that if your code can potentially throw a checked exception, you must handle it using a try-catch block or declare it in the method signature with the `throws` keyword.

Why Are They Called "Checked"?

They are called "checked" because the compiler checks at compile time to ensure that these exceptions are either caught or declared. If they are not, the compiler will generate an error, and the code will not compile.

2) Unchecked Exception

Definition: Unchecked exceptions are exceptions that are not checked by the compiler at compile time. These exceptions are typically the result of programming errors and are not required to be caught or declared.

Why Are They Called "Unchecked"?

They are called "unchecked" because the compiler does not enforce handling these exceptions. It means that the programmer is not required to write explicit error handling code (try-catch blocks) or declare these exceptions in the method signature.

3) Error

Error is irrecoverable. Some example of errors are OutOfMemoryError, VirtualMachineError, AssertionError etc.

These are serious issues that are not typically handled by applications.

Examples: `OutOfMemoryError`, `StackOverflowError`.

1. Try-Catch Block

The try-catch block is used to handle exceptions.

try: Contains the code that might throw an exception.

catch: Contains the code to handle the exception.

Syntax:

```
try {
    // Code that might throw an exception
} catch (ExceptionType e) {
    // Code to handle the exception
}
```

Example:

```
public class TryCatchExample {
    public static void main(String[] args) {
        try {
            int result = 10 / 0; // This will throw ArithmeticException
            System.out.println("Result: " + result);
        } catch (ArithmeticException e) {
            System.out.println("Cannot divide by zero: " + e.getMessage());
        }
    }
}
```

2. Finally Block

The finally block is used to execute important code such as closing resources, regardless of whether an exception was thrown or caught.

Syntax:

```
try {
    // Code that might throw an exception
} catch (ExceptionType e) {
    // Code to handle the exception
} finally {
    // Code that will always run
}
```

Example:

```
public class FinallyExample {
    public static void main(String[] args) {
        try {
            int result = 10 / 0; // This will throw ArithmeticException
            System.out.println("Result: " + result);
        } catch (ArithmeticException e) {
            System.out.println("Cannot divide by zero: " + e.getMessage());
        } finally {
            System.out.println("This will always run.");
```

```
        }
    }
}
```

3. Throw Keyword

The throw keyword is used to explicitly throw an exception from a method or any block of code.

Syntax:

```
if (someCondition) {
    throw new ExceptionType("Error message");
}
```

Example:

```
public class ThrowExample {
    public static void main(String[] args) {
        try {
            checkAge(15);
        } catch (IllegalArgumentException e) {
            System.out.println("Exception: " + e.getMessage());
        }
    }
```

```
    static void checkAge(int age) {
        if (age < 18) {
            throw new IllegalArgumentException("Age must be 18 or older");
        }
        System.out.println("Age is valid");
    }
}
```

4. Throws Keyword

The throws keyword is used in the method signature to declare that the method might throw one or more exceptions.

Syntax:

```
public void someMethod() throws ExceptionType1, ExceptionType2 {
    // Method code
}
```

Java Exception Keywords

Java provides five keywords that are used to handle the exception. The following table describes each.

Keyword	*Description*
try	*The "try" keyword is used to specify a block where we should place an exception code. It means we can't use try block alone. The try block must be followed by either catch or finally.*
catch	*The "catch" block is used to handle the exception. It must be preceded by try block which means we can't use catch block alone. It can be followed by finally block later.*
finally	*The "finally" block is used to execute the necessary code of the program. It is executed whether an exception is handled or not.*
throw	*The "throw" keyword is used to throw an exception.*
throws	*The "throws" keyword is used to declare exceptions. It specifies that there may occur an exception in the method. It doesn't throw an exception. It is always used with method signature.*

Procedural Oriented Programming (POP)

Definition:

POP stands for Procedural Oriented Programming. It's a programming paradigm where the program is divided into small parts called functions or procedures.

Key Characteristics:

Division of Program:

The program is divided into functions. Each function performs a specific task.

Focus:

Importance is given to procedures or functions. The main focus is on how to perform tasks.

Modification:

Modifying the program can be difficult because changes in one function may affect other parts of the program.

Approach:

Uses a top-down approach, starting from a high-level design and breaking it down into smaller, manageable parts.

***Example:** Creating a simple calculator program that can perform addition, subtraction, multiplication, and division.*

Step-by-Step Breakdown:

1. ***Main Module:** Start by defining the main function that outlines the overall logic.*
2. ***Submodules:** Define submodules for each arithmetic operation.*
3. ***Detailed Functions:** Implement the specific functions that perform each operation.*

Access Specifiers:

Does not use access specifiers (like public, private, protected).

Data Hiding:

No concept of data hiding. All data is accessible throughout the program.

Real-world Modelling:

Poor at modelling real-world scenarios as it doesn't represent entities well.

Extensibility:

Less extensible; harder to add new features or functionalities.

Inheritance:

Inheritance is not supported.

Examples:

Languages like C, Pascal, FORTRAN, BASIC are examples of POP.