**Abstraction** **in Java is a fundamental concept in object-oriented programming (OOP) that involves**

**simplifying complex systems by modeling classes based on the essential features they share. It allows you to focus on the essential properties and behaviors of an object while hiding the unnecessary details. Abstraction is achieved through abstract classes and interfaces in Java.**

Here are the key components of abstraction in Java:

**Abstract Classes:**

An abstract class is a class that cannot be instantiated on its own and may contain abstract methods.

Abstract methods are declared without providing an implementation in the abstract class.

Example:

Java code

// Abstract class

abstract class Shape {

// Abstract method (no implementation)

public abstract void draw();

// Concrete method with implementation

public void displayArea() {

System.out.println("Displaying area of the shape.");

}

}

Abstract Methods:

**Abstract methods** are methods declared in an abstract class without providing a body.

Subclasses that extend the abstract class must provide implementations for these abstract methods.

Example:

Java code

// Abstract class with abstract method

abstract class Shape {

public abstract void draw();

}

// Concrete subclass providing implementation

class Circle extends Shape {

@Override

public void draw() {

System.out.println("Drawing a circle.");

}

}

**Interfaces:**

Interfaces are a way to achieve abstraction by defining a contract that implementing classes must adhere to.

All methods in an interface are implicitly **abstract and public.**

Example:

Java code

// Interface

interface Drawable {

void draw();

}

// Class implementing the interface

class Circle implements Drawable {

@Override

public void draw() {

System.out.println("Drawing a circle.");

}

}

**Encapsulation and Information Hiding:**

Abstraction is closely related to encapsulation, as it involves hiding the internal details and exposing only what is necessary.

Access modifiers (public, private, protected) are used to control access to fields and methods.

Example Usage:

Using abstraction to create objects based on their common features:

java code

public class Main {

public static void main(String[] args) {

// Using abstraction with abstract class

Shape circle = new Circle();

circle.draw();

circle.displayArea();

// Using abstraction with interface

Drawable drawable = new Circle();

drawable.draw();

}

}

**Abstraction** allows you to create generalized models that capture the essential characteristics of objects while hiding the implementation details**. It promotes code reuse, flexibility, and the ability to work at higher levels of abstraction.**