**Abstract class**

An **abstract class** in Java is a class that is declared using the abstract keyword.

It cannot be **instantiated** directly and serves as a blueprint for other classes.

Abstract classes can include both:

* **Abstract methods** (methods declared without an implementation).
* **Concrete methods** (methods with an implementation).

Abstract classes are used when you want to enforce certain behavior (method signatures) on all subclasses, but also want to provide some common functionality.

**Characteristics of Abstract Classes:**

1. **Cannot be instantiated**: You cannot create an object of an abstract class.
2. **May or may not contain abstract methods**: A class can be declared abstract even if it has no abstract methods.
3. **Can include concrete methods**: Unlike interfaces, abstract classes can have methods with full implementations.
4. **Can have constructors**: Abstract classes can define constructors, which are called when a subclass is instantiated.
5. **Can have fields (variables)**: Abstract classes can have instance variables, unlike interfaces which only allow public static final constants.

**Syntax:**

abstract class Animal {

abstract void sound(); // Abstract method (no implementation)

void eat() { // Concrete method (has implementation)

System.out.println("This animal eats food.");

}

}

**Example: Abstract Class in Action**

abstract class Animal {

abstract void sound(); // Abstract method

void eat() { // Concrete method

System.out.println("This animal eats food.");

}

}

class Dog extends Animal {

@Override

void sound() {

System.out.println("The dog barks.");

}

}

class Cat extends Animal {

@Override

void sound() {

System.out.println("The cat meows.");

}

}

public class AbstractClassExample {

public static void main(String[] args) {

Animal dog = new Dog();

dog.sound(); // Output: The dog barks.

dog.eat(); // Output: This animal eats food.

Animal cat = new Cat();

cat.sound(); // Output: The cat meows.

cat.eat(); // Output: This animal eats food.

}

}

**Key Points:**

1. **When to Use?**
   * Use an abstract class when you want to provide common functionality to subclasses but still need some methods to be implemented by the subclasses.
2. **Difference Between Abstract Class and Interface**:
   * Abstract classes can have both abstract and concrete methods, while interfaces (in older versions of Java) could only have abstract methods (Java 8 introduced default and static methods in interfaces).
   * Abstract classes can have constructors, instance variables, and access modifiers for methods, while interfaces primarily focused on public methods and constants.
3. **Why Use Abstract Classes?**
   * To enforce a common contract (via abstract methods) while allowing shared implementation for some functionality.

By combining abstract and concrete methods, abstract classes strike a balance between full inheritance and method enforcement.