**1. Introduction to Abstraction:**

* **Definition:** Abstraction is a process of hiding the implementation details and showing only the functionality to the user. It focuses on **what** an object can do, rather than **how** it does it.
* **Real-life Example:** Think about driving a car. You know how to accelerate, brake, or steer, but you don’t need to know how the engine works. This is an example of abstraction—you only deal with the interface (the pedals and steering wheel) and not the internal workings (engine, gearbox).
* **Why use abstraction?**
  + It helps in reducing complexity.
  + It allows us to focus on essential qualities rather than implementation details.

**2. Abstraction in Java:**

There are two ways to achieve abstraction in Java:

1. **Abstract Classes**
2. **Interfaces**

**a. Abstract Class:**

* An abstract class is a class that cannot be instantiated directly. It can have both **abstract methods** (methods without implementation) and **concrete methods** (methods with implementation).
* The abstract methods must be implemented by any subclass that extends the abstract class.

**Syntax Example:**

java

abstract class Animal {

// Abstract method (no implementation)

abstract void sound();

// Concrete method

void sleep() {

System.out.println("Animal is sleeping");

}

}

class Dog extends Animal {

// Implementing the abstract method

@Override

void sound() {

System.out.println("Dog barks");

}

}

public class Main {

public static void main(String[] args) {

Dog dog = new Dog();

dog.sound(); // Calls the implemented method

dog.sleep(); // Calls the inherited method from Animal

}

}

**Key Points:**

* **Abstract classes** can have both abstract and concrete methods.
* Any class that inherits from an abstract class must implement all the abstract methods.

**b. Interface:**

* An interface in Java is a blueprint of a class. It can have **only abstract methods** (before Java 8) or **default/static methods** (from Java 8 onwards).
* A class that implements an interface must provide implementations for all its abstract methods.

**Syntax Example:**

java

interface Vehicle {

void start();

}

class Car implements Vehicle {

@Override

public void start() {

System.out.println("Car starts");

}

}

class Bike implements Vehicle {

@Override

public void start() {

System.out.println("Bike starts");

}

}

public class Main {

public static void main(String[] args) {

Vehicle myCar = new Car();

Vehicle myBike = new Bike();

myCar.start(); // Calls Car's start method

myBike.start(); // Calls Bike's start method

}

}

**Key Points:**

* **Interfaces** cannot have concrete methods (until Java 8 introduced default/static methods).
* A class can implement multiple interfaces, achieving multiple inheritance in Java.

**3. Difference Between Abstract Class and Interface:**

| **Feature** | **Abstract Class** | **Interface** |
| --- | --- | --- |
| **Methods** | Can have both abstract and concrete methods | Can have only abstract methods (until Java 8) |
| **Multiple Inheritance** | Not supported | Supported (a class can implement multiple interfaces) |
| **Access Modifiers** | Can have different access modifiers | All methods are public and abstract by default |
| **Constructors** | Can have constructors | Cannot have constructors |

**4. Benefits of Abstraction:**

* **Reduces Complexity:** You only need to know what a method does, not how it works.
* **Code Reusability:** Abstract classes and interfaces allow developers to define a common structure that can be reused by multiple classes.
* **Maintainability:** Changes made in abstract classes/interfaces don't affect the classes that implement them, as long as the abstract method contracts are followed.

**5. Practical Exercise:**

* **Task 1:** Create an abstract class Shape with an abstract method draw(). Implement two subclasses Circle and Rectangle that provide specific implementations for draw().
* **Task 2:** Create an interface Appliance with a method turnOn(). Implement this interface in WashingMachine and AirConditioner classes.

**6. Summary:**

* **Abstraction** focuses on exposing only the necessary details while hiding the internal workings.
* **Abstract classes** allow partially defined classes that can be extended by other classes.
* **Interfaces** provide a way to define contracts that classes must follow without specifying the implementation.