

## 10.1 Abstract Keyword

### Introduction

In Java, the abstract keyword is a non-access modifier used with classes and methods (but not variables) to achieve **abstraction**, one of the four pillars of **Object-Oriented Programming (OOP)**. Abstraction allows us to define a template or blueprint without requiring a full implementation. It's mainly used to hide the complex implementation details and show only the essential features of an object.

---

### Abstract Methods in Java

Sometimes, you may only want to declare a method in a superclass without providing its implementation. This can be achieved using the abstract keyword. Such methods are often referred to as **subclass responsibilities**, as the subclass must provide their implementation. The superclass only specifies that the method exists but does not define its behaviour.

To declare an abstract method, use the following syntax:

```
abstract return_type methodName(parameters);
```

### Key Characteristics:

- Abstract methods have no body; they only include the method signature, followed by a semicolon.
  - A subclass that extends an abstract class **must** override and provide the implementation for all abstract methods; otherwise, the subclass will also be abstract.
- 

### Abstract Classes in Java

An **abstract class** is a class that may contain abstract methods (methods without implementation) as well as concrete methods (methods with implementation). You cannot instantiate an abstract class directly; it serves as a blueprint for other classes.

To declare an abstract class, you use the abstract keyword:

```
abstract class ClassName {  
    // Abstract methods  
    abstract void methodName();  
  
    // Concrete methods  
    void concreteMethod() {  
        // method body  
    }  
}
```

### Example:

```
abstract class Car {  
    // Abstract method  
    public abstract void drive();  
  
    // Concrete method  
    public void playMusic() {  
        System.out.println("Playing music...");  
    }  
}
```

---

### Characteristics of the Abstract Keyword

The abstract keyword in Java is mainly used to define abstract classes and methods. Here are some key characteristics:

- **Cannot Instantiate Abstract Classes:** You cannot create an object of an abstract class. The abstract class is meant to be extended by concrete (non-abstract) classes, which implement the abstract methods.
- **Abstract Methods Lack Implementation:** Abstract methods only provide a method signature and no implementation. The subclasses are responsible for defining these methods.
- **Abstract Classes Can Have Both Abstract and Concrete Methods:** Abstract classes may include concrete methods (methods with bodies) alongside abstract ones.
- **Abstract Classes Can Have Constructors:** Though you cannot instantiate abstract classes, they can still have constructors. These constructors are used by subclasses to initialize the inherited fields.

- **Can Contain Instance Variables:** Abstract classes can include instance variables, which can be accessed by both the abstract class and its subclasses.
- **Can Implement Interfaces:** Abstract classes can implement interfaces and must provide implementations for all interface methods unless the class itself is abstract.

### Real-World Example

Let's consider a scenario where we want to create a blueprint for different types of cars:

```
// Abstract class
abstract class Car {
    // Abstract method (no implementation)
    public abstract void drive();

    // Abstract method (no implementation)
    public abstract void fly();

    // Concrete method
    public void playMusic() {
        System.out.println("Playing music...");
    }
}

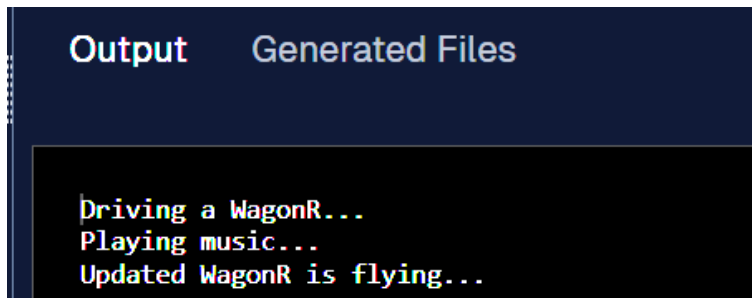
// Concrete subclass of Car
abstract class WagonR extends Car {
    // Implementing abstract method drive
    @Override
    public void drive() {
        System.out.println("Driving a WagonR...");
    }

    // Keep fly as abstract
    @Override
    public abstract void fly();
}

// Concrete subclass of WagonR
class UpdatedWagonR extends WagonR {
    // Implementing the fly method
    @Override
    public void fly() {
        System.out.println("Updated WagonR is flying...");
    }
}

public class MyClass {
    public static void main(String args[]) {
        // No need for typecasting, using dynamic dispatch
        Car car = new UpdatedWagonR();
        car.drive();    // Calls the drive() method from WagonR
        car.playMusic(); // Calls the concrete method from Car class
        car.fly();      // Calls the fly() method from UpdatedWagonR
    }
}
```

### Output:

A screenshot of a Java IDE's output window. The window has a dark blue header with the text "Output" and "Generated Files". Below the header, the output text is displayed on a black background with white and red text. The text shows the execution of a program: "Driving a WagonR...", "Playing music...", and "Updated WagonR is flying...".

```
Output    Generated Files

Driving a WagonR...
Playing music...
Updated WagonR is flying...
```

### Summary:

Here are the key takeaways regarding the abstract keyword in Java:

1. **Abstract Classes Cannot Be Instantiated:** You cannot create an object of an abstract class directly.
2. **Abstract Methods:** These methods are declared but not defined; they must be implemented in subclasses.
3. **Abstract Class Requirement:** If a class contains at least one abstract method, the class itself must be declared abstract.
4. **Concrete Methods in Abstract Classes:** An abstract class can contain both abstract and concrete methods.
5. **Abstract Class Constructors:** Abstract classes can have constructors, but they are only used by subclasses.