Abstract Classes & Interfaces in Python vs Java

Abstraction allows hiding implementation details and exposing only the essential features. This is achieved using Abstract Classes and Interfaces. Below is a comparison of their usage in Python and Java.

# 1. Abstract Classes

Abstract classes are classes that cannot be instantiated directly and may contain abstract methods that must be implemented by subclasses.

## Python Example:

from abc import ABC, abstractmethod  
  
class Vehicle(ABC): # Abstract Class  
 @abstractmethod  
 def start(self):  
 pass  
  
class Car(Vehicle):  
 def start(self):  
 return "Car engine started!"  
  
car = Car()  
print(car.start())

## Java Example:

abstract class Vehicle { // Abstract Class  
 abstract void start();  
}  
  
class Car extends Vehicle {  
 void start() {  
 System.out.println("Car engine started!");  
 }  
}  
  
public class Main {  
 public static void main(String[] args) {  
 Vehicle car = new Car();  
 car.start();  
 }  
}

# 2. Interfaces

Interfaces define a contract for classes. They contain only method signatures (and constants in Java). Implementing classes must provide implementations for all interface methods.

## Python Example (Using ABC):

from abc import ABC, abstractmethod  
  
class Vehicle(ABC):  
 @abstractmethod  
 def start(self):  
 pass  
 @abstractmethod  
 def stop(self):  
 pass  
  
class Car(Vehicle):  
 def start(self):  
 print("Car started")  
 def stop(self):  
 print("Car stopped")  
  
car = Car()  
car.start()  
car.stop()

## Java Example:

interface Vehicle {  
 void start();  
 void stop();  
}  
  
class Car implements Vehicle {  
 public void start() {  
 System.out.println("Car started");  
 }  
 public void stop() {  
 System.out.println("Car stopped");  
 }  
}  
  
public class Main {  
 public static void main(String[] args) {  
 Vehicle car = new Car();  
 car.start();  
 car.stop();  
 }  
}

# 3. Abstract Class vs Interface

|  |  |  |
| --- | --- | --- |
| Feature | Abstract Class | Interface |
| Purpose | Partial abstraction (template + implementation) | Full abstraction (contract only) |
| Methods | Can have abstract + concrete methods | Mostly abstract (Java 8+: default & static methods allowed) |
| Variables | Can have fields (state) | Only constants (in Java) |
| Multiple Inheritance | Not directly (in Java) | Yes (Java: multiple interfaces) |
| Instantiation | Cannot be instantiated | Cannot be instantiated |

# When to Use?

- Use Abstract Class when classes share common behavior + some abstract methods.  
- Use Interface when you want only the contract (no implementation) and multiple inheritance.