Assignment: Mastering Abstraction.
Resolving confusion between Abstract class and Interface in Java

Submitted by: IT22027 - Asraful Nahar Asa

Date: May 13, 2025

1. When to use Interface and When to Use Abstract class ? (with story and code)

Story: Imagine you're designing a vehicle system. All vehicles must have a way to start and stop, but not all vehicles share the same internal properties.

Define:

- An interface vehicle for common behaviors (start/stop)
- An abstract class car to represent cars with shared ~~files~~ fields like engine capacity and number of doors.

Interface Example (shared Behavior) :

```
interface Vehicle {
    void start();
    void stop();
}

class Bike implements Vehicle {
    public void start() {
        System.out.println("Bike started");
    }
    public void stop(){
        System.out.println("Bike stopped");
    }
}
```

Abstract class Example (Common State + Behavior):

```
abstract class Car implements Vehicle {
    int engineCapacity;

    Car(int capacity) {
        this.engineCapacity = capacity;
    }
    void displayInfo() {
        System.out.println("Engine capacity: " + engineCapa
                                                    city);
    }
    abstract void fuelType();

}
class Sedan extends Car {
    Sedan(int capacity) {
        super(capacity);
    }
    public void start() {
        System.out.println("Sedan started");
    }
```

```
    public void stop () {
        System.out.println ("Sedan stopped");
    }
    void fuelType () {
        System.out.println ("Runs on petrol");
    }
}
```

2. Are Interface Methods Slower Than Abstract class Methods ?

Interface methods can be slightly slower if accessed via an interface reference due to dynamic dispatch. However, the difference is negligible in most applications.

P.T.O

Code Example:

```
interface IExample {
    int add (int a, int b);
}
abstract class AExample {
    abstract int add (int a, int b);
}
class Implementation extends AExample implements IExample
    {
    public int add (int a, int b) {
        return a+b;
    }
}
```

Performance Test:

```
public class SpeedTest {
    public static void main (String [] args) {
        IExample iRef = new Implementation();
        AExample aRef = new Implementation();
```

```
long start = System.nanoTime();
for (int i=0; i<100000; i++) iRef.add(2,3);
long interfaceTime = System.nanoTime() - start;

start = System.nanoTime();
for (int i=0; i<1000000; i++) aRef.add(2,3);
long abstractTime = System.nanoTime() - start;

System.out.println("Interface time: " +interfaceTime);
System.out.println("Abstract class time: " +abstractTime);
}
}
```

Result:

The performance difference exists but is minor

Key Point:

It depends more on how the method is referenced, not whether it's in an interface or abstract class.

3. Table: Abstract Class Vs Interface in Java.

Feature	Abstract Class	Interface
Keyword	abstract	interface
Methods	can have both abstract and concrete methods	All methods are abstract by default (Java 7), from Java 8 can have default and static methods
Variables	can have instance variables	only public static final constants
Constructor	can have constructors	cannot have constructors
Inheritance	Single inheritance only	Multiple interfaces can be implemented
Access Modifiers	can be public, protected etc.	All methods are implicitly public.