IT 22026

Story - "The nissan skyline"

Nissan is building their next-gen skyline GT-R series.
. core skyline engineers → They design shared core features (like engine type, brand).
· External modification teams → like racing teams or customs shops, who only need to follow specific set of rules (like must implement turbo boost, support telemetry).

Abstract class → skyline Blueprint
The skyline base model has a common chassis, nissan brand and a predefined engine setup. These features are shared by all skyline cars but actual behaviour (like acceleration or top speed) is different depending on the model (GT-R, GT-R NISMO etc).
So, nissan defines an abstract class

Interface → skyline Racing standard
Now, a racing committee says:
"Any car entering the skyline Racing league must implement turbo boost and telemetry features."

IT22026

This is not about inheritance or shared code, but about a contract they must follow.

use of abstract class when we are defining a base template with common data and behavior, like Nissan's own Skyline models.

use an interface when we are defining a set of rules or capabilities, like what race-compliant vehicles must support - regardless of who makes them.

code:

```
public abstract class NissanSkyline {
    protected String model;
    protected String engineType = "V6 Twin Turbo";

    public NissanSkyline(String model) {
        this.model = model;
    }

public void showBrand() {
    System.out.println("Brand: Nissan");
    }

public void showEngine(){
    System.out.println("Engine: " + engineType);
}
```

```
    public abstract void accelerate();
    public abstract void topSpeed();
}

public class GTR extends NissanSkyline {    // extending abstract class
    public GTR(){
        super("GT-R");
    }

    @Override
    public void accelerate(){
        System.out.println(model + " accelerates from 0-100
            Km/h in 3.5 seconds.");
    }

    @Override
    public void topSpeed(){
        System.out.println(model + " top speed: 315 Km/h");
    }
}

public class GTRNismo extends NissanSkyline {
    public GTRNismo(){
        super("GT-R NISMO");
    }
```

```
@override
public void accelerate(){
    System.out.println(model+" accelerates from 0-100
        Km/h in 2.9 seconds.");
    }

@override
public void topSpeed(){
    System.out.println(model+"top speed : 330 km/h!");
    }
}
}

public interface RacingFeatures {   // define interface
    void enableTurboBoost();
    void enableTelemetry();
}

public class SkylineRaceMod implements RacingFeatures{
    @override
    public void enableTurboBoost(){
        System.out.println("Turbo Boost enabled : Additional 150
                                    hp speed added.");
        }
    @override
    public void enableTelemetry(){
        System.out.println("Telemetry System online: Monitoring
            speed, RPM, and G-Forces.");
        }
}
```

IT22026

```
public class Skyline{        // main class
    public static void main(String args[]){
        NissanSkyline baseModel = new GTR();
        baseModel.showBrand();
        baseModel.showEngine();
        baseModel.accelerate();
        baseModel.topspeed();

        System.out.println("-------");

        NissanSkyline proModel = new GTRNismo();
        proModel.showBrand();
        proModel.showEngine();
        proModel.accelerate();
        proModel.topspeed();

        System.out.println("-------");

        RacingFeatures modCar = new SkylineRaceMod();
        modCar.enableTurboBoost();
        modCar.enableTelemetry();

    }
}
```

IT22026

Ques: 02

Is it true that invoking methods in interface are slower than invoking it within the abstract classes? Explain and write new example.

Ans: yes, calling interface methods was slightly slower due to dynamic dispatch through the interface table. Abstract class method calls can be more direct, since there is more rigid inheritance structure:

method call via interface
method call via abstract class.

Code:

```
interface MyInterface{
    void doWork();
}
abstract class MyAbstractclass{
    abstract void doWork();
}
class InterfaceImpl implements MyInterface{
    public void doWork(){
        int x = 0;
        for (int i=0; i<100; i++){
            x+=i;
        }
    }
}
```

IT22026

```
class AbstractImpl extends MyAbstractClass {
    public void doWork () {
        int x = 0;
        for (int i = 0 ; i<100; i++ )  x+=i ;
    }
}

public class performancetest {
    public static void main (String args[]) {
        myInterface iobj = new InterfaceImpl ();
        MyAbstractClass aobj = new AbstractImpl ();

        long startTime , endTime ;

        startTime = System.nanoTime ();
        for (int i =0 ; i< 1_00_000 ; i++) iobj.doWork();

        endTime = System.nanoTime ();
        System.out.println ("Interface method time: " + (endTime
                - startTime) + " ns");

        startTime = system.nanoTime ();
        for (int i = 0; i< 1_00_000 ; i++) aobj.doWork ();

        endtime = System.nanaTime ();
        System.out.println ("Abstract class method time: " +
            (endtime - startTime) + "ns ");
    }
}
```

IT22026

Ques: 03

Make a table to summarize the differences between Abstract class and Interface.

Answer:

Feature	Abstract class	Interface
Inheritance	single	multiple
Method Implement	have both abstract & concrete	default & static
variables	Instances & static	public static final
construction	yes	no
Access modifier	private, protected	only public
performance	faster	slower