IT22037

Story - "The nissan Skyline"

Nissan is building their next-gen Skyline GT-R Series .

1. 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 s~~te~~tup. These features are shared by all Skyline cars but actual behaviour (like acceleration on 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 can entering the Skyline Racing league must implement turbo boost and telemetry features."

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, a Nissan's own Skyline models.

use an interface when we are defining a set of ru or capabilities, like what race-compliant vehicles m 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 0-10
              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.");
    }
}
```

```
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();

    }
}
```

Ques: 02

Is it true that invoking methods in interface are slower than invoking it within the abstract classes? Explain an 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's 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;
        }
    }
}
```

```
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: " + (end Tir
            - startTime) + "ns");

        startTime = System.nanoTime ();
        for (int i = 0; i<1_00_000 ; i++) aboj.dowork ();
        endTime = System.nanaTime ();
        System.out.println ("Abstract class method time:" +
            (endtime - startTime) + "ns ");
    }
}
```

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 fina
constructor	yes	no
Acess modifier	private, protected	only public
performance	faster	slower