using System;

namespace RefactoringGuru.DesignPatterns.TemplateMethod.Conceptual

{

abstract class AbstractClass

{

public void TemplateMethod()

{

this.BaseOperation1();

this.RequiredOperations1();

this.BaseOperation2();

this.Hook1();

this.RequiredOperation2();

this.BaseOperation3();

this.Hook2();

}

protected void BaseOperation1()

{

Console.WriteLine("AbstractClass says: I am doing the bulk of the work");

}

protected void BaseOperation2()

{

Console.WriteLine("AbstractClass says: But I let subclasses override some operations");

}

protected void BaseOperation3()

{

Console.WriteLine("AbstractClass says: But I am doing the bulk of the work anyway");

}

protected abstract void RequiredOperations1();

protected abstract void RequiredOperation2();

protected virtual void Hook1() { }

protected virtual void Hook2() { }

}

class ConcreteClass1 : AbstractClass

{

protected override void RequiredOperations1()

{

Console.WriteLine("ConcreteClass1 says: Implemented Operation1");

}

protected override void RequiredOperation2()

{

Console.WriteLine("ConcreteClass1 says: Implemented Operation2");

}

}

class ConcreteClass2 : AbstractClass

{

protected override void RequiredOperations1()

{

Console.WriteLine("ConcreteClass2 says: Implemented Operation1");

}

protected override void RequiredOperation2()

{

Console.WriteLine("ConcreteClass2 says: Implemented Operation2");

}

protected override void Hook1()

{

Console.WriteLine("ConcreteClass2 says: Overridden Hook1");

}

}

class Client

{

public static void ClientCode(AbstractClass abstractClass)

{

abstractClass.TemplateMethod();

}

}

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Same client code can work with different subclasses:");

Client.ClientCode(new ConcreteClass1());

Console.Write("\n");

Console.WriteLine("Same client code can work with different subclasses:");

Client.ClientCode(new ConcreteClass2());

}

}

}