Q # 01

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace lab7\_1

{

class MainApp

{

/// </summary>

static void Main()

{

LoadBalancer b1 = LoadBalancer.GetLoadBalancer();

LoadBalancer b2 = LoadBalancer.GetLoadBalancer();

LoadBalancer b3 = LoadBalancer.GetLoadBalancer();

LoadBalancer b4 = LoadBalancer.GetLoadBalancer();

// Same instance?

if (b1 == b2 && b2 == b3 && b3 == b4)

{

Console.WriteLine("Hey,\n");

}

// Load balance 15 persons

LoadBalancer balancer = LoadBalancer.GetLoadBalancer();

for (int i = 0; i < 15; i++)

{

string server = balancer.Server;

Console.WriteLine("Awesomist: " + server);

}

// Wait for user

Console.ReadKey();

}

}

/// <summary>

/// The 'Singleton' class

/// </summary>

class LoadBalancer

{

private static LoadBalancer \_instance;

private List<string> \_servers = new List<string>();

private Random \_random = new Random();

// Lock synchronization object

private static object syncLock = new object();

// Constructor (protected)

protected LoadBalancer()

{

// List of available servers

\_servers.Add("Noni");

\_servers.Add("Sara");

\_servers.Add("Ayesha");

\_servers.Add("Sadia");

\_servers.Add("JK");

\_servers.Add("Kookie");

}

public static LoadBalancer GetLoadBalancer()

{

// Support multithreaded applications through

// 'Double checked locking' pattern which (once

// the instance exists) avoids locking each

// time the method is invoked

if (\_instance == null)

{

lock (syncLock)

{

if (\_instance == null)

{

\_instance = new LoadBalancer();

}

}

}

return \_instance;

}

// Simple, but effective random load balancer

public string Server

{

get

{

int r = \_random.Next(\_servers.Count);

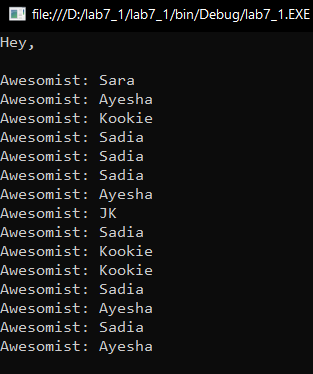
return \_servers[r].ToString();

}

}

}

}



Q # 02

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace lab7\_21

{

interface shape

{

string draw();

}

class rectangle : shape

{

public string draw()

{

return "Inside rectangle: draw() method";

}

}

class circle : shape

{

public string draw()

{

return "Inside circle: draw() method";

}

}

class square: shape

{

public string draw()

{

return "Inside square: draw() method";

}

}

static class shapefactory

{

public static shape Get(int id)

{

if (id==1)

{

return new circle();

}

else if (id==0)

{

return new rectangle();

}

else if (id==2)

{

return new square();

}

return null;

}

}

class Program

{

static void Main()

{

for (int i=0; i<=3; i++)

{

var type = shapefactory.Get(i);

if (type!= null)

{

Console.WriteLine(" Hey" + type.draw());

}

else

{

Console.ReadKey();

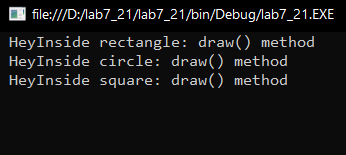
}

}

}

}

}



Q # 01

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace lab7nishaumaima

{

public interface Item

{

String name();

Packing packing();

float price();

}

public interface Packing

{

String pack();

}

public class Wrapper : Packing

{

public String pack()

{

return "Wrapper";

}

}

public class bottle : Packing

{

public String pack()

{

return "Bottle";

}

}

public abstract class burger : Item

{

public Packing packing()

{

return new Wrapper();

}

public abstract float price();

public abstract String name();

}

public class vegburger : burger

{

public override float price()

{

return 25.0f;

}

public override String name()

{

return "Veg Burger";

}

}

public class chickburger : burger

{

public override float price()

{

return 25.0f;

}

public override String name()

{

return "Chicken Burger";

}

}

public abstract class colddrink : Item

{

public Packing packing()

{

return new bottle();

}

public abstract float price();

public abstract String name();

}

public class coke : colddrink

{

public override float price()

{

return 35.0f;

}

public override String name()

{

return "Coke";

}

}

public class pepsi : colddrink

{

public override float price()

{

return 35.0f;

}

public override String name()

{

return "pepsi";

}

}

public class Meal

{

private List<Item> items = new List<Item>();

public void addItems(Item item)

{

items.Add(item);

}

public float getCost()

{

float cost = 0.0f;

foreach (Item item in items)

{

cost += item.price();

}

return cost;

}

public void showItems()

{

foreach (Item item in items)

{

System.Console.WriteLine("Item : " + item.name());

System.Console.WriteLine(", Packing : " + item.packing().pack());

System.Console.WriteLine(", Price : " + item.price());

System.Console.ReadKey();

}

}

}

public class MealBuilder

{

public Meal prepareVegMeal()

{

Meal meal = new Meal();

meal.addItems(new vegburger());

meal.addItems(new coke());

return meal;

}

public Meal prepareNonVegMeal()

{

Meal meal = new Meal();

meal.addItems(new chickburger());

meal.addItems(new pepsi());

return meal;

}

}

class Program

{

public static void Main(string[] args)

{

MealBuilder mealbuilder = new MealBuilder();

Meal vegMeal = mealbuilder.prepareVegMeal();

System.Console.WriteLine("Veg Meal");

Console.ReadKey();

vegMeal.showItems();

System.Console.WriteLine("Total Cost :" + vegMeal.getCost());

Console.ReadKey();

Meal nonVegMeal = mealbuilder.prepareNonVegMeal();

System.Console.WriteLine("\n\n NonVeg Meal");

Console.ReadKey();

nonVegMeal.showItems();

System.Console.WriteLine("Total Cost :" + nonVegMeal.getCost());

Console.ReadKey();

}

}

}

