DESIGN PATTERNS

COMMAND

using System;

using System.Collections.Generic;

// Receiver class

public class Employee

{

public string Name { get; set; }

public string Department { get; set; }

public void ChangeDepartment(string newDepartment)

{

Console.WriteLine($"Changing department of employee {Name} from {Department} to {newDepartment}");

Department = newDepartment;

}

}

// Command interface

public interface ICommand

{

void Execute();

void Undo();

}

// Concrete command classes

public class ChangeDepartmentCommand : ICommand

{

private Employee employee;

private string oldDepartment;

private string newDepartment;

public ChangeDepartmentCommand(Employee employee, string newDepartment)

{

this.employee = employee;

this.newDepartment = newDepartment;

oldDepartment = employee.Department;

}

public void Execute()

{

employee.ChangeDepartment(newDepartment);

}

public void Undo()

{

employee.ChangeDepartment(oldDepartment);

}

}

// Invoker class

public class EmployeeManager

{

private List<ICommand> commands;

public EmployeeManager()

{

commands = new List<ICommand>();

}

public void AddCommand(ICommand command)

{

commands.Add(command);

}

public void ProcessCommands()

{

foreach (var command in commands)

{

command.Execute();

}

}

public void UndoCommands()

{

for (int i = commands.Count - 1; i >= 0; i--)

{

commands[i].Undo();

}

}

}

class Program

{

static void Main(string[] args)

{

Employee employee1 = new Employee { Name = "John Doe", Department = "Sales" };

Employee employee2 = new Employee { Name = "Jane Smith", Department = "Marketing" };

EmployeeManager manager = new EmployeeManager();

// Change department commands

ICommand command1 = new ChangeDepartmentCommand(employee1, "IT");

ICommand command2 = new ChangeDepartmentCommand(employee2, "Finance");

manager.AddCommand(command1);

manager.AddCommand(command2);

// Process commands

manager.ProcessCommands();

// Undo commands

manager.UndoCommands();

}

}

STRATEGY

using System;

using System.Collections.Generic;

// Strategy interface

public interface IEmployeeDetailsStrategy

{

void DisplayDetails(Employee employee);

}

// Concrete strategy classes

public class BasicDetailsStrategy : IEmployeeDetailsStrategy

{

public void DisplayDetails(Employee employee)

{

Console.WriteLine($"Employee: {employee.Name}");

}

}

public class FullDetailsStrategy : IEmployeeDetailsStrategy

{

public void DisplayDetails(Employee employee)

{

Console.WriteLine($"Employee: {employee.Name}");

Console.WriteLine($"Department: {employee.Department}");

Console.WriteLine($"Position: {employee.Position}");

Console.WriteLine($"Salary: {employee.Salary}");

}

}

// Context class

public class Employee

{

public string Name { get; set; }

public string Department { get; set; }

public string Position { get; set; }

public decimal Salary { get; set; }

private IEmployeeDetailsStrategy detailsStrategy;

public void SetDetailsStrategy(IEmployeeDetailsStrategy strategy)

{

detailsStrategy = strategy;

}

public void DisplayDetails()

{

detailsStrategy.DisplayDetails(this);

}

}

class Program

{

static void Main(string[] args)

{

Employee employee = new Employee

{

Name = "John Doe",

Department = "Sales",

Position = "Manager",

Salary = 5000

};

employee.SetDetailsStrategy(new BasicDetailsStrategy());

employee.DisplayDetails();

Console.WriteLine();

employee.SetDetailsStrategy(new FullDetailsStrategy());

employee.DisplayDetails();

}

}

OBSERVER

using System;

using System.Collections.Generic;

// Subject interface

public interface IEmployeeSubject

{

void Attach(IEmployeeObserver observer);

void Detach(IEmployeeObserver observer);

void Notify();

}

// Observer interface

public interface IEmployeeObserver

{

void Update(Employee employee);

}

// Concrete subject class

public class EmployeeSubject : IEmployeeSubject

{

private List<IEmployeeObserver> observers;

private Employee employee;

public EmployeeSubject()

{

observers = new List<IEmployeeObserver>();

}

public void Attach(IEmployeeObserver observer)

{

observers.Add(observer);

}

public void Detach(IEmployeeObserver observer)

{

observers.Remove(observer);

}

public void Notify()

{

foreach (var observer in observers)

{

observer.Update(employee);

}

}

public void SetEmployeeDetails(Employee newEmployee)

{

employee = newEmployee;

Notify();

}

}

// Concrete observer class

public class EmployeeObserver : IEmployeeObserver

{

public void Update(Employee employee)

{

Console.WriteLine("Employee Details Updated");

Console.WriteLine($"Name: {employee.Name}");

Console.WriteLine($"Department: {employee.Department}");

Console.WriteLine($"Position: {employee.Position}");

Console.WriteLine($"Salary: {employee.Salary}");

Console.WriteLine();

}

}

// Employee class

public class Employee

{

public string Name { get; set; }

public string Department { get; set; }

public string Position { get; set; }

public decimal Salary { get; set; }

}

class Program

{

static void Main(string[] args)

{

EmployeeSubject subject = new EmployeeSubject();

EmployeeObserver observer = new EmployeeObserver();

subject.Attach(observer);

Employee employee1 = new Employee

{

Name = "John Doe",

Department = "Sales",

Position = "Manager",

Salary = 5000

};

subject.SetEmployeeDetails(employee1);

Employee employee2 = new Employee

{

Name = "Jane Smith",

Department = "Marketing",

Position = "Analyst",

Salary = 4000

};

subject.SetEmployeeDetails(employee2);

subject.Detach(observer);

}

}

ADAPTER

using System;

// Existing Employee class with different interface

public class Employee

{

public string Name { get; set; }

public string JobTitle { get; set; }

public decimal Salary { get; set; }

}

// Target interface

public interface IEmployeeDetails

{

string GetName();

string GetPosition();

decimal GetSalary();

}

// Adapter class

public class EmployeeAdapter : IEmployeeDetails

{

private Employee employee;

public EmployeeAdapter(Employee employee)

{

this.employee = employee;

}

public string GetName()

{

return employee.Name;

}

public string GetPosition()

{

return employee.JobTitle;

}

public decimal GetSalary()

{

return employee.Salary;

}

}

// Client class

public class EmployeeInfoPrinter

{

public void PrintEmployeeDetails(IEmployeeDetails employee)

{

Console.WriteLine("Employee Details:");

Console.WriteLine($"Name: {employee.GetName()}");

Console.WriteLine($"Position: {employee.GetPosition()}");

Console.WriteLine($"Salary: {employee.GetSalary()}");

}

}

class Program

{

static void Main(string[] args)

{

Employee employee = new Employee

{

Name = "John Doe",

JobTitle = "Manager",

Salary = 5000

};

EmployeeAdapter adapter = new EmployeeAdapter(employee);

EmployeeInfoPrinter infoPrinter = new EmployeeInfoPrinter();

infoPrinter.PrintEmployeeDetails(adapter);

}

}

DECORATOR

using System;

// Component interface

public interface IEmployeeDetails

{

string GetDetails();

}

// Concrete component class

public class BasicEmployeeDetails : IEmployeeDetails

{

private string name;

public BasicEmployeeDetails(string name)

{

this.name = name;

}

public string GetDetails()

{

return $"Name: {name}";

}

}

// Decorator base class

public abstract class EmployeeDetailsDecorator : IEmployeeDetails

{

protected IEmployeeDetails decoratedEmployee;

public EmployeeDetailsDecorator(IEmployeeDetails employee)

{

decoratedEmployee = employee;

}

public virtual string GetDetails()

{

return decoratedEmployee.GetDetails();

}

}

// Concrete decorator classes

public class PositionDecorator : EmployeeDetailsDecorator

{

private string position;

public PositionDecorator(IEmployeeDetails employee, string position) : base(employee)

{

this.position = position;

}

public override string GetDetails()

{

string baseDetails = base.GetDetails();

return $"{baseDetails}, Position: {position}";

}

}

public class SalaryDecorator : EmployeeDetailsDecorator

{

private decimal salary;

public SalaryDecorator(IEmployeeDetails employee, decimal salary) : base(employee)

{

this.salary = salary;

}

public override string GetDetails()

{

string baseDetails = base.GetDetails();

return $"{baseDetails}, Salary: {salary:C}";

}

}

class Program

{

static void Main(string[] args)

{

IEmployeeDetails employee = new BasicEmployeeDetails("John Doe");

employee = new PositionDecorator(employee, "Manager");

employee = new SalaryDecorator(employee, 5000);

Console.WriteLine(employee.GetDetails());

}

}

FACTORY

using System;

// Product interface

public interface IEmployee

{

string GetName();

string GetPosition();

decimal GetSalary();

}

// Concrete product classes

public class Manager : IEmployee

{

public string GetName()

{

return "John Doe";

}

public string GetPosition()

{

return "Manager";

}

public decimal GetSalary()

{

return 5000;

}

}

public class Developer : IEmployee

{

public string GetName()

{

return "Jane Smith";

}

public string GetPosition()

{

return "Developer";

}

public decimal GetSalary()

{

return 4000;

}

}

// Creator class

public abstract class EmployeeFactory

{

public abstract IEmployee CreateEmployee();

}

// Concrete creator classes

public class ManagerFactory : EmployeeFactory

{

public override IEmployee CreateEmployee()

{

return new Manager();

}

}

public class DeveloperFactory : EmployeeFactory

{

public override IEmployee CreateEmployee()

{

return new Developer();

}

}

class Program

{

static void Main(string[] args)

{

EmployeeFactory managerFactory = new ManagerFactory();

IEmployee manager = managerFactory.CreateEmployee();

EmployeeFactory developerFactory = new DeveloperFactory();

IEmployee developer = developerFactory.CreateEmployee();

Console.WriteLine("Manager Details:");

PrintEmployeeDetails(manager);

Console.WriteLine();

Console.WriteLine("Developer Details:");

PrintEmployeeDetails(developer);

}

static void PrintEmployeeDetails(IEmployee employee)

{

Console.WriteLine($"Name: {employee.GetName()}");

Console.WriteLine($"Position: {employee.GetPosition()}");

Console.WriteLine($"Salary: {employee.GetSalary()}");

}

}

ABSTRACT FACTORY

using System;

// Abstract product interface

public interface IEmployee

{

string GetName();

string GetPosition();

decimal GetSalary();

}

// Concrete product classes

public class Manager : IEmployee

{

public string GetName()

{

return "John Doe";

}

public string GetPosition()

{

return "Manager";

}

public decimal GetSalary()

{

return 5000;

}

}

public class Developer : IEmployee

{

public string GetName()

{

return "Jane Smith";

}

public string GetPosition()

{

return "Developer";

}

public decimal GetSalary()

{

return 4000;

}

}

// Abstract factory interface

public interface IEmployeeFactory

{

IEmployee CreateEmployee();

}

// Concrete factory classes

public class ManagerFactory : IEmployeeFactory

{

public IEmployee CreateEmployee()

{

return new Manager();

}

}

public class DeveloperFactory : IEmployeeFactory

{

public IEmployee CreateEmployee()

{

return new Developer();

}

}

class Program

{

static void Main(string[] args)

{

IEmployeeFactory factory = new ManagerFactory();

IEmployee employee = factory.CreateEmployee();

Console.WriteLine("Employee Details:");

PrintEmployeeDetails(employee);

}

static void PrintEmployeeDetails(IEmployee employee)

{

Console.WriteLine($"Name: {employee.GetName()}");

Console.WriteLine($"Position: {employee.GetPosition()}");

Console.WriteLine($"Salary: {employee.GetSalary()}");

}

}

SINGLETON

using System;

// Singleton class

public class EmployeeDetailsSingleton

{

private static EmployeeDetailsSingleton instance;

private EmployeeDetailsSingleton() { }

public static EmployeeDetailsSingleton Instance

{

get

{

if (instance == null)

{

instance = new EmployeeDetailsSingleton();

}

return instance;

}

}

public void PrintEmployeeDetails()

{

Console.WriteLine("Employee Details:");

Console.WriteLine("Name: John Doe");

Console.WriteLine("Position: Manager");

Console.WriteLine("Salary: 5000");

}

}

class Program

{

static void Main(string[] args)

{

EmployeeDetailsSingleton.Instance.PrintEmployeeDetails();

}

}

PRODUCER-CONSUMER

using System;

using System.Collections.Concurrent;

using System.Threading;

using System.Threading.Tasks;

// Producer class

public class EmployeeProducer

{

private readonly BlockingCollection<Employee> employeeQueue;

private readonly Random random;

public EmployeeProducer(BlockingCollection<Employee> queue)

{

employeeQueue = queue;

random = new Random();

}

public void StartProducing()

{

Task.Run(() =>

{

while (true)

{

Employee newEmployee = CreateRandomEmployee();

employeeQueue.Add(newEmployee);

Console.WriteLine($"Produced employee: {newEmployee.Name}");

Thread.Sleep(random.Next(1000, 2000));

}

});

}

private Employee CreateRandomEmployee()

{

string[] names = { "John Doe", "Jane Smith", "Alice Johnson", "Bob Thompson" };

string[] positions = { "Manager", "Developer", "Accountant", "HR Specialist" };

string name = names[random.Next(names.Length)];

string position = positions[random.Next(positions.Length)];

return new Employee(name, position);

}

}

// Consumer class

public class EmployeeConsumer

{

private readonly BlockingCollection<Employee> employeeQueue;

public EmployeeConsumer(BlockingCollection<Employee> queue)

{

employeeQueue = queue;

}

public void StartConsuming()

{

Task.Run(() =>

{

while (true)

{

Employee employee = employeeQueue.Take();

Console.WriteLine($"Consumed employee: {employee.Name} ({employee.Position})");

Thread.Sleep(2000);

}

});

}

}

// Employee class

public class Employee

{

public string Name { get; set; }

public string Position { get; set; }

public Employee(string name, string position)

{

Name = name;

Position = position;

}

}

class Program

{

static void Main(string[] args)

{

BlockingCollection<Employee> employeeQueue = new BlockingCollection<Employee>();

EmployeeProducer producer = new EmployeeProducer(employeeQueue);

EmployeeConsumer consumer = new EmployeeConsumer(employeeQueue);

producer.StartProducing();

consumer.StartConsuming();

Console.WriteLine("Press Enter to exit");

Console.ReadLine();

}

}