


ASSIGNMENT 2 FRONT SHEET

| | | | |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------|-------------------------------------|---------------------------------------------------------------------------------------|
| Qualification | BTEC Level 5 HND Diploma in Computing | | |
| Unit number and title | Unit 20: Advanced Programming | | |
| Submission date | December 26, 2022 | Date Received 1st submission | December 26, 2022 |
| Re-submission Date | | Date Received 2nd submission | |
| Student Name | Nguyen Tien Thinh | Student ID | GCH200796 |
| Class | GCH1002 | Assessor name | Le Viet Bach |
| Student declaration I certify that the assignment submission is entirely my own work and I fully understand the consequences of plagiarism. I understand that making a false declaration is a form of malpractice. | | | |
| | | Student's signature |  |

Grading grid

| | | | | | |
|----|----|----|----|----|----|
| P3 | P4 | M3 | M4 | D3 | D4 |
| | | | | | |

⚙ **Summative Feedback:**⚙ **Resubmission Feedback:****Grade:****Assessor Signature:****Date:****Lecturer Signature:**

Contents

| | | |
|------|---------------------------|----|
| I. | Introduction..... | 5 |
| II. | Scenario analysis..... | 5 |
| 1. | Scenario..... | 5 |
| 2. | Diagram..... | 6 |
| III. | Implementation | 8 |
| 1. | Code | 8 |
| a. | IWorkouts.cs | 9 |
| b. | Abs.cs | 9 |
| c. | Arm.cs | 10 |
| d. | Back.cs..... | 11 |
| e. | Chest.cs | 12 |
| f. | Leg.cs | 13 |
| g. | Shoulder.cs | 14 |
| h. | IWorkoutFactory.cs | 15 |
| i. | AbsWorkoutF.cs..... | 15 |
| j. | ArmWorkoutF.cs..... | 16 |
| k. | BackWorkoutF.cs | 16 |
| l. | ChestWorkoutF.cs | 16 |
| m. | LegWorkoutF.cs | 16 |
| n. | ShoulderWorkoutF.cs | 17 |
| o. | PlanTraining.cs | 17 |
| p. | Program.cs | 24 |
| 2. | Program screenshots | 26 |
| a. | Start Program..... | 26 |

| | | |
|-----|-------------------------------------------------------------------------|----|
| b. | Use Display schedule workouts function | 26 |
| c. | Use Training function..... | 27 |
| d. | Use Display tomorrow workouts function..... | 28 |
| e. | Use Change the number of reps in the workouts function | 29 |
| f. | Use Change the workouts tomorrow function. | 30 |
| g. | Use Exit function. | 32 |
| IV. | Discussion | 33 |
| 1. | Range of similar patterns | 33 |
| a. | Summary of some similar patterns..... | 33 |
| b. | Why Factory Method pattern is the most suitable for your scenario?..... | 33 |
| 2. | Usage of pattern | 34 |
| a. | Advantages | 34 |
| b. | Disadvantage | 34 |
| V. | Conclusion | 34 |
| VI. | Reference | 34 |

I. Introduction

The author will describe a basic scenario in this report on how to develop an application that gives workouts and utilize the factory methodology design pattern to put in the code for the aforesaid challenge. Code that is guaranteed to apply and build to the template's standards. The console interface contains several basic operations and verifies user input. Furthermore, the writer will particularly assess the chosen design pattern in terms of its grounds for selection as well as its benefits and drawbacks.

II. Scenario analysis

1. Scenario

FitnessAlpha is a gym owned by PT John. Three months ago, members did not practice here much, some new members did not know how to practice effectively, often guided by John.

However, recently, more and more members come to the fitness room, and he can't guide each member one by one. So John is looking to develop an app that provides workout guidance.

- Workouts will be suggested. In the immediate future, the system will have exercises for arms, chest, shoulders, back, abs, and legs.
- These muscle groups will be provided with exercises with a number of repetitions to perform.

| Name of workout | | Rep |
|-----------------|---------------------------|-----|
| Arm | Dumbbell Curl | 8 |
| | Overhead Tricep Extension | 8 |
| | Cable Triceps Pushdown | 10 |
| Chest | Dumbbell Press | 10 |
| | Barbell Press | 10 |
| | Cable Crossover | 10 |
| Shoulder | Lateral Raise | 8 |
| | Overhead Dumbbell Press | 10 |
| | Side Lateral Raises | 10 |

| | | |
|-------------|---------------------|----|
| Back | Deadlift | 10 |
| | Lat Pull Down | 10 |
| | Dumbbell Row | 12 |
| Abs | Crunch | 12 |
| | Vertical Leg Crunch | 12 |
| | Reverse Crunch | 12 |
| Leg | Squat | 12 |
| | Lying Leg Curl | 12 |
| | BarBell Squad | 8 |

- The exercises that have been created will be divided equally among the 3 days. Every day, because to save time and not be overpowered, leading to ineffective training, members will only exercise one or two muscle groups. Muscle groups are divided as follows:
 - Day 1: Arms and Abs.
 - Day 2: Chest and Back.
 - Day 3: Legs and Shoulders.
- Members can view the exercises directly on the system.
- After completing the practice, the system will record the member's feedback.
- If an exercise is too easy or too difficult for a member, they can ask the system to change the number of repetitions accordingly.
 - If the exercise is too easy, number of reps in each exercise will be increased by 2.
 - If the exercise is too difficult, number of reps in each exercise will be reduced by 2.
 - If the exercise is right, the number of reps will not change.
- The next day's exercises will be shown after the member's feedback. If the member does not want to train the planned muscle group, the member can leave or choose another exercise instead. However, members can only choose tomorrow's workouts instead.

2. Diagram

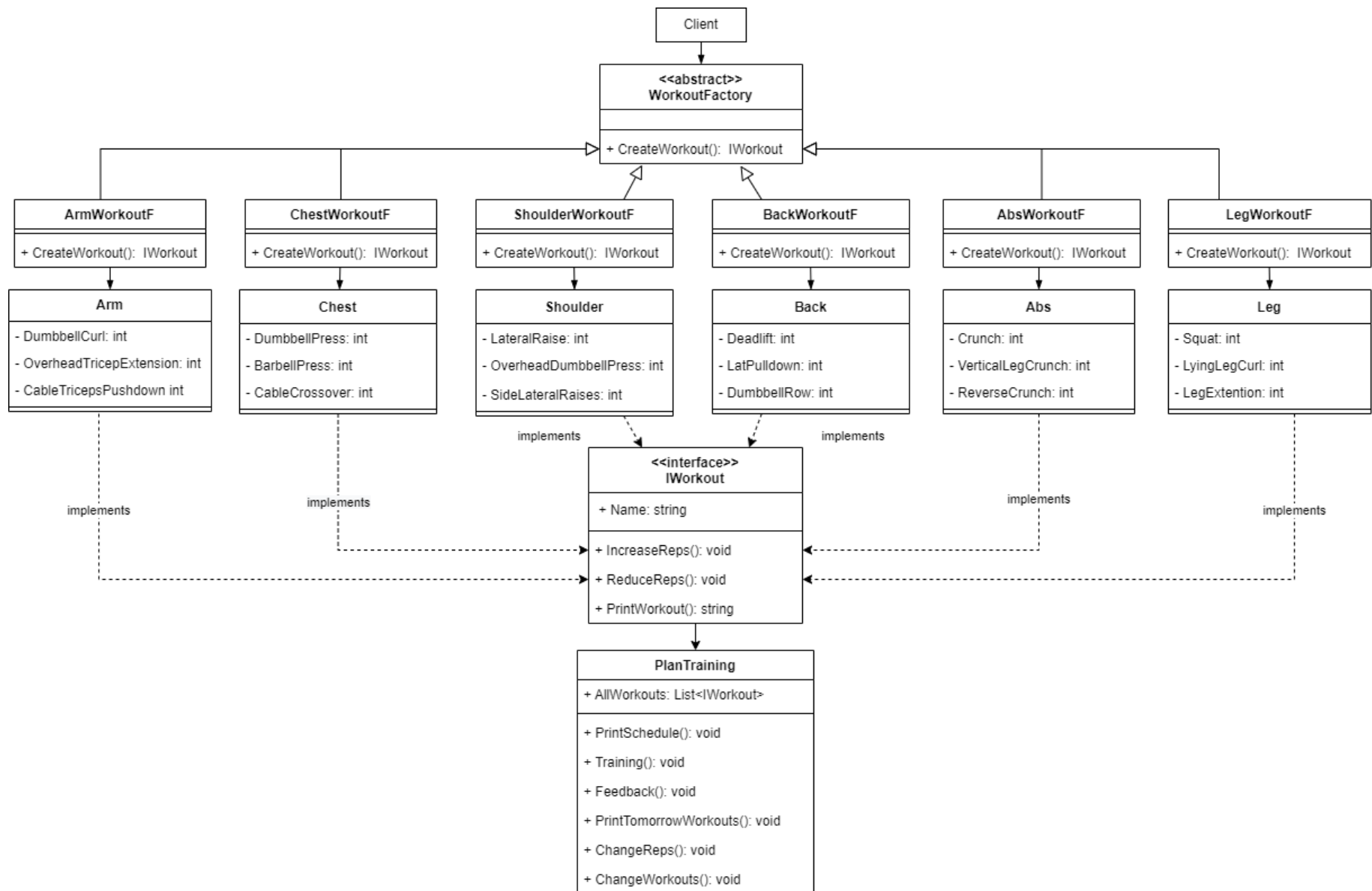


Figure 1. Class diagram.

Explain:

The first is the IWorkout interface, this interface class has common contracts for the exercises that are Name property, method IncreaseReps(), ReduceReps(), and PrintWorkouts(). This is the SuperClass of the workouts class. About the workouts class, there are 6 classes corresponding to 6 exercises for 6 muscle groups that are abs, arm, chest, shoulder, back and leg. These classes implement by Iworkout interface. Each class will have properties that are the number of reps of exercises exclusive to that muscle group and are stored with data type int.

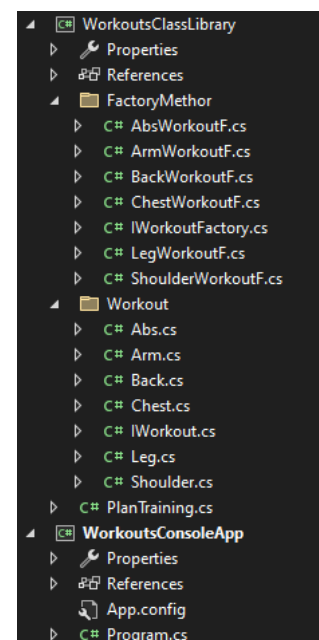
Next is the IWorkoutFactory class declares the factory method that returns new workouts objects. The return type of this method matches the IWorkout interface. ArmWorkoutF, AbsWorkoutF, ChestWorkoutF, LegWorkoutF, BackWorkoutF, and ShoulderWorkoutF, they override base factory method is IworkoutFactory, create and returns an object for the workouts.

Finally, the PlanTraining class perform saves the created workouts objects to a List. In addition, the PrintScheldule() methods are used to print out the workout schedule, Training() to print workouts to practice, Feedback() to collect user reviews after practice the workouts, PrintTomorrowWorkouts() to print workouts next day, ChangeReps() to change the number of reps of workouts simultaneously, and finally ChangeWorkouts() to change next day's workouts.

III. Implementation

1. Code

All folder and file to implement the project. The project is divided into 2 small projects, which are WorkoutsClassLibrary and WorkoutsConsoleApp. Project WorkoutsClassLibrary is built to perform the task of creating classes, interfaces containing objects, and methods to use for running the program. Project WorkoutsConsoleApp will use the objects and methods in the WorkoutsClassLibrary to run the program.



a. IWorkouts.cs

First, the author creates an IWorkout interface file with common contracts for the exercises that are Name property, method IncreaseReps(), ReduceReps() and PrintWorkouts().

```
namespace WorkoutsClassLibrary
{
    25 references
    public interface IWorkout
    {
        28 references
        string Name { get; set; }
        9 references
        void IncreaseReps();
        9 references
        void ReduceReps();
        10 references
        string PrintWorkout();
    }
}
```

b. Abs.cs

The author creates class Abs that implements the IWorkout interface. The property Name declared along with the property are specific exercises for the Abs muscle group, which are Crunch, Vertical Leg Crunch and Reverse Crunch.

```
namespace WorkoutsClassLibrary
{
    2 references
    public class Abs : IWorkout
    {
        23 references
        string IWorkout.Name {
            get { return Name; }
            set => Name = value;
        }
        4 references
        public string Name { get; private set; }
        4 references
        private int Crunch { get; set; }
        4 references
        private int VerticalLegCrunch { get; set; }
        4 references
        private int ReverseCrunch { get; set; }
    }
}
```

In the constructor function, the author will declare the initial values for each property corresponding to the number of reps. In methods IncreaseReps(), ReduceReps() when called, the number of reps of the workouts will be increased or decreased 2. Method PrintWorkouts() is used to print the name and number of reps of the workouts.

```
public Abs()
{
    this.Name = "Abs";
    this.Crunch = 12;
    this.VerticalLegCrunch = 12;
    this.ReverseCrunch = 12;
}

4 references
public void IncreaseReps()
{
    this.Crunch += 2;
    this.VerticalLegCrunch += 2;
    this.ReverseCrunch += 2;
}

4 references
public void ReduceReps()
{
    this.Crunch -= 2;
    this.VerticalLegCrunch -= 2;
    this.ReverseCrunch -= 2;
}

5 references
public string PrintWorkout()
{
    return this.Name + "\nCrunch: " + this.Crunch +
        "\nVertical Leg Crunch: " + this.VerticalLegCrunch +
        "\nReverse Crunch: " + this.ReverseCrunch;
}
```

c. Arm.cs

Similarly, the author creates an Arm class that implements the IWorkout interface. The property Name declared along with the property are specific exercises for the Arm muscle group, which are Dumbbell Curl, Overhead Tricep Extension and Cable Triceps Pushdown.

```
namespace WorkoutsClassLibrary
{
    2 references
    public class Arm : IWorkout
    {
        23 references
        string IWorkout.Name
        {
            get { return Name; }
            set => Name = value;
        }

        4 references
        public string Name { get; set; }
        4 references
        private int DumbbellCurl { get; set; }
        4 references
        private int OverheadTricepExtension { get; set; }
        4 references
        private int CableTricepsPushdown { get; set; }
    }
}
```

Similar to the methods above, the methods IncreaseReps(), ReduceReps() are used to increase and decrease the number of repetitions. Method PrintWorkouts() is used to print the name and number of reps of the workouts.

```

1 reference
public Arm()
{
    this.Name = "Arm";
    this.DumbbellCurl = 8;
    this.OverheadTricepExtension = 8;
    this.CableTricepsPushdown = 10;
}

4 references
public void IncreaseReps()
{
    this.DumbbellCurl += 2;
    this.OverheadTricepExtension += 2;
    this.CableTricepsPushdown += 2;
}

4 references
public void ReduceReps()
{
    this.DumbbellCurl -= 2;
    this.OverheadTricepExtension -= 2;
    this.CableTricepsPushdown -= 2;
}

5 references
public string PrintWorkout()
{
    return this.Name + "\nDumbbell Curl: " + this.DumbbellCurl +
        "\nOverhead Tricep Extension: " + this.OverheadTricepExtension +
        "\nCable Triceps Pushdown: " + this.CableTricepsPushdown ;
}

```

d. Back.cs

The author creates a Back class that implements the IWorkout interface with the name properties, the workouts are Dead lift, Lat Pulldown and Dumbbell Row.

```

namespace WorkoutsClassLibrary
{
    2 references
    public class Back : IWorkout
    {
        23 references
        string IWorkout.Name
        {
            get { return Name; }
            set => Name = value;
        }

        4 references
        public string Name { get; private set; }
        4 references
        private int Deadlift { get; set; }
        4 references
        private int LatPulldown { get; set; }
        4 references
        private int DumbbellRow { get; set; }
    }
}

```

IncreaseReps(), ReduceReps() methods are used to increase the number of repetitions. Method or PrintWorkouts() is used to print the name and number of repetitions of the exercise.

```

public Back()
{
    this.Name = "Back";
    this.Deadlift = 10;
    this.LatPulldown = 10;
    this.DumbbellRow = 12;
}

4 references
public void IncreaseReps()
{
    this.Deadlift += 2;
    this.LatPulldown += 2;
    this.DumbbellRow += 2;
}

4 references
public void ReduceReps()
{
    this.Deadlift -= 2;
    this.LatPulldown -= 2;
    this.DumbbellRow -= 2;
}

5 references
public string PrintWorkout()
{
    return this.Name + "\nDeadlift: " + this.Deadlift +
        "\nLat Pulldown: " + this.LatPulldown +
        "\nDumbbell Row: " + this.DumbbellRow;
}

```

e. Chest.cs

The author creates a Chest class that implements the IWorkout interface with the name properties, the workouts are Dumbbell Press, Barbell Press and Cable Crossover.

```

namespace WorkoutsClassLibrary
{
    2 references
    public class Chest : IWorkout
    {
        23 references
        string IWorkout.Name
        {
            get { return Name; }
            set => Name = value;
        }
        4 references
        public string Name { get; private set; }
        4 references
        private int DumbbellPress { get; set; }
        4 references
        private int BarbellPress { get; set; }
        4 references
        private int CableCrossover { get; set; }
    }
}

```

IncreaseReps(), ReduceReps() methods are used to increase the number of reductions. Method or PrintWorkouts() is used to print the name and number of repetitions of the exercise.

```

1 reference
public Chest()
{
    this.Name = "Chest";
    this.DumbbellPress = 10;
    this.BarbellPress = 10;
    this.CableCrossover = 10;
}

4 references
public void IncreaseReps()
{
    this.DumbbellPress += 2;
    this.BarbellPress += 2;
    this.CableCrossover += 2;
}

4 references
public void ReduceReps()
{
    this.DumbbellPress -= 2;
    this.BarbellPress -= 2;
    this.CableCrossover -= 2;
}

5 references
public string PrintWorkout()
{
    return this.Name + "\nDumbbell Press: " + this.DumbbellPress +
        "\nBarbell Press: " + this.BarbellPress +
        "\nCable Crossover: " + this.CableCrossover;
}

```

f. Leg.cs

The author creates a Chest class that implements the IWorkout interface with the name properties, the workouts are Squat, Lying Leg Curl and Leg Extention.

```

namespace WorkoutsClassLibrary
{
    2 references
    public class Leg : IWorkout
    {
        23 references
        string IWorkout.Name
        {
            get { return Name; }
            set => Name = value;
        }

        4 references
        public string Name { get; private set; }

        4 references
        private int Squat { get; set; }

        4 references
        private int LyingLegCurl { get; set; }

        4 references
        private int LegExtention { get; set; }
    }
}

```

IncreaseReps(), ReduceReps() methods are used to increase the number of reductions. Method or PrintWorkouts() is used to print the name and number of repetitions of the exercise

```

1 reference
public Leg()
{
    this.Name = "Leg";
    this.Squat = 12;
    this.LyingLegCurl = 12;
    this.LegExtension = 8;
}

4 references
public void IncreaseReps()
{
    this.Squat += 2;
    this.LyingLegCurl += 2;
    this.LegExtension += 2;
}

4 references
public void ReduceReps()
{
    this.Squat -= 2;
    this.LyingLegCurl -= 2;
    this.LegExtension -= 2;
}

5 references
public string PrintWorkout()
{
    return this.Name + "\nSquat: " + this.Squat +
        "\nLying Leg Curl: " + this.LyingLegCurl +
        "\nLeg Extension: " + this.LegExtension;
}

```

g. Shoulder.cs

The author creates a Shoulder class that implements the IWorkout interface with the name properties, the workouts are Lateral Raise, Overhead Dumbbell Press and Side Lateral Raises.

```

namespace WorkoutsClassLibrary
{
    2 references
    public class Shoulder : IWorkout
    {
        4 references
        public string Name { get; private set; }
        23 references
        string IWorkout.Name {
            get { return Name; }
            set { Name = value; }
        }
        4 references
        private int LateralRaise { get; set; }
        4 references
        private int OverheadDumbbellPress { get; set; }
        4 references
        private int SideLateralRaises { get; set; }
    }
}

```

IncreaseReps(), ReduceReps() methods are used to increase the number of repetitions. Method or PrintWorkouts() is used to print the name and number of repetitions of the exercise

```

1 reference
public Shoulder()
{
    this.Name = "Shoulder";
    this.LateralRaise = 8;
    this.OverheadDumbbellPress = 10;
    this.SideLateralRaises = 10;
}

4 references
public void IncreaseReps()
{
    this.LateralRaise += 2;
    this.OverheadDumbbellPress += 2;
    this.SideLateralRaises += 2;
}

4 references
public void ReduceReps()
{
    this.LateralRaise -= 2;
    this.OverheadDumbbellPress -= 2;
    this.SideLateralRaises -= 2;
}

5 references
public string PrintWorkout()
{
    return this.Name + "\nLateral Raise: " + this.LateralRaise +
        "\nOverhead Dumbbell Press: " + this.OverheadDumbbellPress +
        "\nSide Lateral Raises: " + this.SideLateralRaises;
}

```

h. IWorkoutFactory.cs

The IWorkoutFactory use to replace direct object construction calls (using the operator) with calls to a special factory method for each workout. The objects are created via the operator, it is called from within the factory method. Objects returned by a factory method are workouts.

```

12 references
public interface IWorkoutFactory
{
    12 references
    IWorkout CreateWorkout();
}

```

i. AbsWorkoutF.cs

AbsWorkoutsF override the base factory method IWorkoutFactory, so it creates and returns object is Abs workouts.

```

1 reference
public class AbsWorkoutF : IWorkoutFactory
{
    7 references
    public IWorkout CreateWorkout()
    {
        var absWorkout = new Abs();
        return absWorkout;
    }
}

```

j. ArmWorkoutF.cs

ArmWorkoutsF override the base factory method IWorkoutFactory, so it creates and returns object is Arm workouts.

```
1 reference
public class ArmWorkoutF : IWorkoutFactory
{
    7 references
    public IWorkout CreateWorkout()
    {
        var armWorkout = new Arm();
        return armWorkout;
    }
}
```

k. BackWorkoutF.cs

AbsWorkoutsF override the base factory method IWorkoutFactory, so it creates and returns object is Abs workouts.

```
1 reference
public class BackWorkoutF : IWorkoutFactory
{
    7 references
    public IWorkout CreateWorkout()
    {
        var backWorkout = new Back();
        return backWorkout;
    }
}
```

l. ChestWorkoutF.cs

ChestWorkoutsF override the base factory method IWorkoutFactory, so it creates and returns object is Chest workouts.

```
1 reference
public class ChestWorkoutF : IWorkoutFactory
{
    7 references
    public IWorkout CreateWorkout()
    {
        var chestWorkout = new Chest();
        return chestWorkout;
    }
}
```

m. LegWorkoutF.cs

LegWorkoutsF override the base factory method IWorkoutFactory, so it creates and returns object is Legs workouts.


```
1 reference
public class LegWorkoutF : IWorkoutFactory
{
    7 references
    public IWorkout CreateWorkout()
    {
        var legWorkout = new Leg();
        return legWorkout;
    }
}
```

n. ShoulderWorkoutF.cs

ShoulderWorkoutsF override the base factory method IWorkoutFactory, so it creates and returns object is shoulder workouts.

```
1 reference
public class ShoulderWorkoutF : IWorkoutFactory
{
    7 references
    public IWorkout CreateWorkout()
    {
        var shoulderWorkout = new Shoulder();
        return shoulderWorkout;
    }
}
```

o. PlanTraining.cs

After the workout objects are initialized at the class factory methods, they are added to an AllWorkouts list for user interaction and editing. In the constructor function, List will be initialized. The PrintSchedule() method will print the exercises in turn at index 0 and 1 for Day 1, similar to Day 2 and Day 3.

```
17 references
public class PlanTraining
{
    59 references
    public List<IWorkout> AllWorkouts { get; set; }

    1 reference
    public PlanTraining()
    {
        this.AllWorkouts = new List<IWorkout>();
    }

    1 reference
    public static void PrintSchedule(PlanTraining planTraining)
    {
        Console.WriteLine("Day 1: "
            + planTraining.AllWorkouts[0].Name + ", "
            + planTraining.AllWorkouts[1].Name);
        Console.WriteLine("Day 2: "
            + planTraining.AllWorkouts[2].Name + ", "
            + planTraining.AllWorkouts[3].Name);
        Console.WriteLine("Day 3: "
            + planTraining.AllWorkouts[4].Name + ", "
            + planTraining.AllWorkouts[5].Name);
    }
}
```

Method Training() will use the PrintWorkouts() function of each object to print out the workouts information at index position 0 and 1 (corresponding to Day 1). Similarly, the PrintTomorrowWorkouts() method, the program prints out the object at index position 2 and 3 (corresponding to Day 2).

```
1 reference
public static void Training(PlanTraining planTraining)
{
    Console.WriteLine(planTraining.AllWorkouts[0].PrintWorkout());
    Console.WriteLine();
    Console.WriteLine(planTraining.AllWorkouts[1].PrintWorkout());
}

1 reference
public static void PrintTomorrowWorkouts(PlanTraining planTraining)
{
    Console.WriteLine(planTraining.AllWorkouts[2]?.PrintWorkout());
    Console.WriteLine(planTraining.AllWorkouts[3]?.PrintWorkout());
}
```

The Feedback() method is called immediately after the user uses method Training(). The program will print out the question and let the user choose how they feel about the workouts. The feedback variable is created and uses the chooseOptionFeedback() method to check if the user data is valid. The switch case statement is used to catch the user input value.

```
public static void Feedback(PlanTraining planTraining)
{
    Console.WriteLine("\nDo you feel the level of exercise is appropriate?" +
        "\n1. It's a bit easy" +
        "\n2. Just enough" +
        "\n3. It's so hard");
    int feedback = chooseOptionFeedback();
}
```

If the feedback is 1, which means the exercise is quite easy for the user, the program will offer them a choice whether they want to increase the number of reps of each exercise or not. The variable increase is created and uses the YesOrNo() method to check the user input.

- If it is 1, the user wants to increase, the program will increase the reps of the exercise that day by 2 reps using the IncreaseReps() function of each object at the index position 0 and 1.
- If the selection is 2, the user does not want to change the number of reps, the program will print the message and break out.

```
switch (feedback)
{
    case 1:
        Console.WriteLine("Thanks for your feedback"
            + "\nDo you want to increase the number of reps of the workouts?(1 = Yes/ 2 = No)");
        int increase = YesOrNo();
        if (increase == 1)
        {
            planTraining.AllWorkouts[0].IncreaseReps();
            planTraining.AllWorkouts[1].IncreaseReps();
            Console.WriteLine("The number of reps has been increased by 2");
        }
        else if (increase == 2)
        {
            Console.WriteLine("Number of reps unchanged");
        }
        break;
}
```

If the feedback is 2, that is, the exercise fits the user, the program will print a message to cheer the user and break out.

If the feedback is 3, similar to the feedback equal to 1, the program prints the message, the decrease variable is created and uses the YesOrNo() method to check the user input.

- If it is 1, then the user wants to decrease, the program will increase the reps of that exercise by 2 reps using the ReduceReps() function of each object at index positions 0 and 1.
- If the choice is 2, that is, the user does not want to change the number of reps, the program will print a message and break out.

```
case 2:
    Console.WriteLine("Thanks for your feedback"
        + "\nKeep up the hard work!");
    break;
case 3:
    Console.WriteLine("Thanks for your feedback"
        + "\nDo you want to decrease the number of reps of the workouts?(1 = Yes/ 2 = No)");
    int decrease = YesOrNo();
    if (decrease == 1)
    {
        planTraining.AllWorkouts[0].ReduceReps();
        planTraining.AllWorkouts[1].ReduceReps();
        Console.WriteLine("The number of reps has been decreased by 2");
    }
    else if (decrease == 2)
    {
        Console.WriteLine("Number of reps unchanged");
    }
    break;
}
```

The ChangeReps() method when called print a notification question for the user to choose from. The change variable is created and uses the YesOrNo() method to check the user input.

If the choice is 1, that is, the user wants to increase the number of reps, the program will use the for loop and the IncreaseReps() function to increase the number of reps for each exercise in the List AllWorkouts.

If the option is 2, that is, the user wants to reduce the number of reps, the program will use the for loop and the ReduceReps() function to reduce the number of reps for each exercise in the List AllWorkouts.

```
public static void ChangeReps(PlanTraining planTraining)
{
    Console.WriteLine("You want to increase or decrease reps (enter 1 = Increase, 2 = Decrease)");
    int change = YesOrNo();
    if (change == 1)
    {
        for (int i = 0; i < planTraining.AllWorkouts.Count; i++)
        {
            planTraining.AllWorkouts[i].IncreaseReps();
        }
        Console.WriteLine("The number of reps has been increased by 2");
    }
    else if (change == 2)
    {
        for (int i = 0; i < planTraining.AllWorkouts.Count; i++)
        {
            planTraining.AllWorkouts[i].ReduceReps();
        }
        Console.WriteLine("The number of reps has been decreased by 2");
    }
}
```

The ChangeWorkouts method when called will print the names of Day 2's exercises (at index 2 and 3) corresponding to options 1 and 2, and then ask the user which exercise they want to change. The **changeWorkouts** variable is created and uses YesOrNo() method to check input.

```
public static void ChangeWorkouts(PlanTraining planTraining)
{
    Console.WriteLine("Next day workouts: \n1. "
        + planTraining.AllWorkouts[2]?.Name + "\n2. "
        + planTraining.AllWorkouts[3]?.Name);
    Console.WriteLine("What is the workouts do you want to change?");
    int changeWorkouts = YesOrNo();
}
```

The author **uses If condition for the first time** to check the exercise that the user chooses. If changeWorkouts is 1, ie the first exercise of Day 2 (index 2), the program will print a question informing whether the user wants to replace or delete this workout.

The author **uses If condition for second time** to check the user's choice to replace or delete these workouts. The workouts1 variable is created and uses YesOrNo() method to check input.

- If workouts1 is 2, the user wants to delete the workout. The program will assign the object at the index position with 2 null values to ensure that printing out workouts later on is not error-prone. Then print out the message to the user.

```
if(changeWorkouts == 1)
{
    Console.WriteLine("You choose to change "
        + planTraining.AllWorkouts[2].Name + " muscle group workouts " +
        "\nDo you want to remove or replace a new exercise?(1 = Replace / 2 = Remove)");
    int workouts1 = YesOrNo();
    if (workouts1 == 2)
    {
        planTraining.AllWorkouts[2] = null;
        Console.WriteLine("Workouts has been deleted");
    }
    else if (workouts1 == 1)
    {
        Console.WriteLine("Choose the muscle group you want to replace");
        Console.WriteLine("Next day workouts: \n1. "
            + planTraining.AllWorkouts[4].Name + "\n2. "
            + planTraining.AllWorkouts[5].Name);
        int changeworkouts1 = YesOrNo();
    }
}
```

- If workouts1 is 1, the user wants to change the exercise. The program will print two exercise options that the user has not practiced on the 3rd day (index 4 and 5) equivalent to options 1 and 2. The changeworkouts1 variable is created and uses YesOrNo() method to check input. The author **uses If condition for the third time** to test the workout that user chooses to replace.
 - If the selection is 1, i.e. the exercise at index position is 3, a variable temp will be initialized, and an assignment is performed to change the position of the element with index 2 and 4 in the List AllWorkouts.
 - If the selection is 2, i.e. workouts at index position is 5, the same steps are repeated and the position of element with index equal to 2 and 5 is swapped in the List AllWorkouts.

```
if (changeworkouts1 == 1)
{
    Console.WriteLine("The " + planTraining.AllWorkouts[2].Name
        + " muscle group has been changed to the " +
        planTraining.AllWorkouts[4].Name +
        " muscle group");
    Object temp = planTraining.AllWorkouts[2];
    planTraining.AllWorkouts[2] = planTraining.AllWorkouts[4];
    planTraining.AllWorkouts[4] = (IWorkout)temp;
}
else if (changeworkouts1 == 2)
{
    Console.WriteLine("The " + planTraining.AllWorkouts[2].Name
        + " muscle group has been changed to the " +
        planTraining.AllWorkouts[5].Name +
        " muscle group");
    Object temp = planTraining.AllWorkouts[2];
    planTraining.AllWorkouts[2] = planTraining.AllWorkouts[5];
    planTraining.AllWorkouts[5] = (IWorkout)temp;
}
}
```

Back to the first if condition, if changeWorkouts is 2, then choose workouts at index 3. The program will also repeat the same steps in case changeWorkouts is 1. The author will also use 2 loops, 1 is for check the option to delete or replace workouts. If it's replacing workouts, the next loop is used to select the workouts the user wants to replace.

```

else if(changeWorkouts == 2)
{
    Console.WriteLine("You choose to change "
        + planTraining.AllWorkouts[3].Name + " muscle group workout"
        + "\nDo you want to remove or replace a new exercise?(1 = Replace, 2 = Remove)");
    int workouts2 = YesOrNo();
    if (workouts2 == 2)
    {
        planTraining.AllWorkouts[3] = null;
        Console.WriteLine("Workouts has been deleted");
    }
    else if (workouts2 == 1)
    {
        Console.WriteLine("Choose the muscle group you want to replace");
        Console.WriteLine("Next day workouts: \n1. "
            + planTraining.AllWorkouts[4].Name + "\n2. "
            + planTraining.AllWorkouts[5].Name);
        int changeworkouts1 = YesOrNo();
        if (changeworkouts1 == 1)
        {
            Console.WriteLine("The " + planTraining.AllWorkouts[3].Name
                + " muscle group has been changed to the " +
                planTraining.AllWorkouts[4].Name +
                " muscle group");
            Object temp = planTraining.AllWorkouts[3];
            planTraining.AllWorkouts[3] = planTraining.AllWorkouts[4];
            planTraining.AllWorkouts[4] = (IWorkout)temp;
            Console.WriteLine();
        }
        else if (changeworkouts1 == 2)
        {
            Console.WriteLine("The " + planTraining.AllWorkouts[3].Name
                + " muscle group has been changed to the " +
                planTraining.AllWorkouts[5].Name +
                " muscle group");
            Object temp = planTraining.AllWorkouts[3];
            planTraining.AllWorkouts[3] = planTraining.AllWorkouts[5];
            planTraining.AllWorkouts[5] = (IWorkout)temp;
        }
    }
}
}

```


The chooseOption(), chooseOptionFeedback() and YesOrNo() methods are created with the task of checking user input. With similar structure but different in the number of choices that the user can enter. The author creates a choice variable with initial data of -1 (false). Then use a do while loop to check the user input repeatedly until it is correct (range 1 to 5 with chooseOption(), 1 to 3 with chooseOptionFeedback() and 1 or 2 with YesOrNo()). Inside the loop, the author uses try catch to catch user input errors. Until the user correctly enters the requested data, the choice variable will be assigned to that value and returned.

```
static public int chooseOption()
{
    int choice = -1;
    do
    {
        try
        {
            Console.WriteLine("Please enter your choice: ");
            choice = int.Parse(Console.ReadLine());
            if (choice < 0 || choice > 5)
            {
                Console.WriteLine("Please choose only 1 to 5");
            }
        }
        catch (System.Exception ex)
        {
            Console.WriteLine(ex.Message);
            Console.WriteLine("Please choose only 1 to 5");
        }
    } while (choice < 0 || choice > 5);
    return choice;
}
```

```
static public int chooseOptionFeedback()
{
    int choice = -1;
    do
    {
        try
        {
            Console.WriteLine("Please enter your choice: ");
            choice = int.Parse(Console.ReadLine());
            if (choice < 1 || choice > 3)
            {
                Console.WriteLine("Please choose only 1 to 3");
            }
        }
        catch (System.Exception ex)
        {
            Console.WriteLine(ex.Message);
            Console.WriteLine("Please choose only 1 to 3");
        }
    } while (choice < 1 || choice > 3);
    return choice;
}
```

```
static public int YesOrNo()
{
    int choice = -1;
    do
    {
        try
        {
            Console.WriteLine("Please enter your choice: ");
            choice = int.Parse(Console.ReadLine());
            if (choice < 1 || choice > 2)
            {
                Console.WriteLine("Please choose only 1 or 2");
            }
        }
        catch (System.Exception ex)
        {
            Console.WriteLine(ex.Message);
            Console.WriteLine("Please choose only 1 or 2");
        }
    } while (choice < 1 || choice > 2);
    return choice;
}
```

p. Program.cs

In the WorkoutsConsoleApp project, the author first imports the WorkoutsClassLibrary to reference to it and use objects and methods more conveniently.

```
using System;
using WorkoutsClassLibrary;
using WorkoutsClassLibrary.FactoryMethod;
```

Initially, the author creates a planTraining object with the properties of the PlanTraining class and IWorkoutFactory to create individual objects of the workouts. Next, he uses the CreateWorkout() function in the Factory Method to create exercises for the muscle groups along with creating a temporary variable that stores the muscle group information into the List AllWorkouts of the planTraining object.

```
namespace WorkoutsConsoleApp
{
    0 references
    class Program
    {
        0 references
        public static void Main(string[] args)
        {
            PlanTraining planTraining = new PlanTraining();
            IWorkoutFactory armF = new ArmWorkoutF();
            IWorkoutFactory absF = new AbsWorkoutF();
            IWorkoutFactory chestF = new ChestWorkoutF();
            IWorkoutFactory backF = new BackWorkoutF();
            IWorkoutFactory legF = new LegWorkoutF();
            IWorkoutFactory shoulderF = new ShoulderWorkoutF();

            var arm = armF.CreateWorkout();
            planTraining.AllWorkouts.Add(arm);
            var abs = absF.CreateWorkout();
            planTraining.AllWorkouts.Add(abs);
            var chest = chestF.CreateWorkout();
            planTraining.AllWorkouts.Add(chest);
            var back = backF.CreateWorkout();
            planTraining.AllWorkouts.Add(back);
            var leg = legF.CreateWorkout();
            planTraining.AllWorkouts.Add(leg);
            var shoulder = shoulderF.CreateWorkout();
            planTraining.AllWorkouts.Add(shoulder);
        }
    }
}
```

Then, the PrintMenu() method to print out the menu of functions and StopScreen() is created to help users better interact with the program.

```
3 references
static public void StopScreen()
{
    Console.WriteLine("\nPress any key to continue");
    Console.ReadKey();
}
```



```
2 references
static public void PrintMenu()
{
    Console.Clear();
    Console.WriteLine("*****MENU*****");
    Console.WriteLine(" *      Please enter your choice      *");
    Console.WriteLine(" * 1.Display schedule workouts         *");
    Console.WriteLine(" * 2.Training                         *");
    Console.WriteLine(" * 3.Display tomorrow workouts        *");
    Console.WriteLine(" * 4.Change the number of reps in the workouts *");
    Console.WriteLine(" * 5.Change the workouts tomorrow     *");
    Console.WriteLine(" * 0.Exit                             *");
    Console.WriteLine("*****\n");
}
```

Back to the Main function, the author prints the initial greeting for the user and then prints the menu for the user to choose with the PrintMenu() function. Selection variable is created and use the chooseOption() function to check the input. Next, the author combines using while loop and switch case with methods corresponding to the functions printed in PrintMenu() function.

```
Console.WriteLine("*****");
Console.WriteLine("***** WORKOUTS PROGRAM *****");
Console.WriteLine("*****");
Console.WriteLine("Welcome to the workouts program!");
StopScreen();
PrintMenu();
```

```
int selection = PlanTraining.chooseOption();
while (true)
{
    switch (selection)
    {
        case 1:
            Console.WriteLine("You choose to Display schedule workouts\n");
            PlanTraining.PrintSchedule(planTraining);
            break;
        case 2:
            Console.WriteLine("You choose to Training today workouts\n");
            PlanTraining.Training(planTraining);
            StopScreen();
            Console.Clear();
            PlanTraining.Feedback(planTraining);
            break;
        case 3:
            Console.WriteLine("You choose to Display tomorrow workouts\n");
            PlanTraining.PrintTomorrowWorkouts(planTraining);
            break;
        case 4:
            Console.WriteLine("You choose to Change the number of reps in the workouts\n");
            PlanTraining.ChangeReps(planTraining);
            break;
        case 5:
            Console.WriteLine("You choose to Change the workouts tomorrow\n");
            PlanTraining.ChangeWorkouts(planTraining);
            break;
        case 0:
            Console.WriteLine("You choose exit");
            Console.WriteLine("Press enter to exit...");
            Console.ReadKey();
            Environment.Exit(0);
            break;
    }
}
```

2. Program screenshots

a. Start Program

When starting, the program will print the welcome interface.

```

*****
***** WORKOUTS PROGRAM *****
*****
Welcome to the workouts program!

Press any key to continue
|

```

Then the program will print out the menu table.

```

*****MENU*****
*           Please enter your choice           *
* 1.Display schedule workouts                    *
* 2.Training                                    *
* 3.Display tomorrow workouts                   *
* 4.Change the number of reps in the workouts  *
* 5.Change the workouts tomorrow               *
* 0.Exit                                         *
*****
Please enter your choice: |

```

When the user enters the wrong data type, an error message will appear and ask to re-enter.

```

*****MENU*****
*           Please enter your choice           *
* 1.Display schedule workouts                    *
* 2.Training                                    *
* 3.Display tomorrow workouts                   *
* 4.Change the number of reps in the workouts  *
* 5.Change the workouts tomorrow               *
* 0.Exit                                         *
*****
Please enter your choice: a
Input string was not in a correct format.
Please choose only 1 to 5
Please enter your choice: |

```

b. Use Display schedule workouts function

When the user selects 1, the program will print out the names of workouts for each day that have been built according to the 3-day schedule.

```

*****MENU*****
*           Please enter your choice           *
* 1.Display schedule workouts                    *
* 2.Training                                    *
* 3.Display tomorrow workouts                   *
* 4.Change the number of reps in the workouts  *
* 5.Change the workouts tomorrow               *
* 0.Exit                                         *
*****

Please enter your choice: 1
You choose to Display schedule workouts

Day 1: Arm, Abs
Day 2: Chest, Back
Day 3: Leg, Shoulder

Press any key to continue
|

```

c. Use Training function

When the user selects 2, the program will print out the names of the workouts in day 1 along with the workouts that come with the number of reps to practice.

```

*****MENU*****
*           Please enter your choice           *
* 1.Display schedule workouts                    *
* 2.Training                                    *
* 3.Display tomorrow workouts                   *
* 4.Change the number of reps in the workouts  *
* 5.Change the workouts tomorrow               *
* 0.Exit                                         *
*****

Please enter your choice: 2
You choose to Training today workouts

Arm
Dumbbell Curl: 8
Overhead Tricep Extension: 8
Cable Triceps Pushdown: 10

Abs
Crunch: 12
Vertical Leg Crunch: 12
Reverse Crunch: 12

Press any key to continue
|

```

After training is complete, the user will receive a message asking for feedback on the workouts. Depending on the user's rating, the program will print the next selection but can increase reps if workouts are easy or can decrease reps if workouts are hard.

```
Do you feel the level of exercise is appropriate?  
1. It's a bit easy  
2. Just enough  
3. It's so hard  
Please enter your choice: |
```

If user evaluate the workouts just right.

```
Do you feel the level of exercise is appropriate?  
1. It's a bit easy  
2. Just enough  
3. It's so hard  
Please enter your choice: 2  
Thanks for your feedback  
Keep up the hard work!  
  
Press any key to continue  
|
```

If user evaluate the workouts are easy.

```
Do you feel the level of exercise is appropriate?  
1. It's a bit easy  
2. Just enough  
3. It's so hard  
Please enter your choice: 1  
Thanks for your feedback  
Do you want to increase the number of reps of the workouts?(1 = Yes/ 2 = No)  
Please enter your choice: |
```

```
Do you feel the level of exercise is appropriate?  
1. It's a bit easy  
2. Just enough  
3. It's so hard  
Please enter your choice: 1  
Thanks for your feedback  
Do you want to increase the number of reps of the workouts?(1 = Yes/ 2 = No)  
Please enter your choice: 1  
The number of reps has been increased by 2  
  
Press any key to continue  
|
```

d. Use Display tomorrow workouts function

When the user selects 3, the program will print out the names of the workouts in day 2 along with the workouts that come with the number of reps to practice.

```

*****MENU*****
*           Please enter your choice           *
* 1.Display schedule workouts                    *
* 2.Training                                    *
* 3.Display tomorrow workouts                  *
* 4.Change the number of reps in the workouts  *
* 5.Change the workouts tomorrow              *
* 0.Exit                                        *
*****

Please enter your choice: 3
You choose to Display tomorrow workouts

Chest
Dumbbell Press: 10
Barbell Press: 10
Cable Crossover: 10
Back
Deadlift: 10
Lat Pulldown: 10
Dumbbell Row: 12

Press any key to continue
|

```

e. Use Change the number of reps in the workouts function

When the user chooses 4, the program will ask the user to choose whether to increase or decrease the number of reps for each workouts.

```

*****MENU*****
*           Please enter your choice           *
* 1.Display schedule workouts                    *
* 2.Training                                    *
* 3.Display tomorrow workouts                  *
* 4.Change the number of reps in the workouts  *
* 5.Change the workouts tomorrow              *
* 0.Exit                                        *
*****

Please enter your choice: 4
You choose to Change the number of reps in the workouts

You want to increase or decrease reps (enter 1 = Increase, 2 = Decrease)
Please enter your choice: |

```

When user chooses to decrease the number of reps.

```

Please enter your choice: 4
You choose to Change the number of reps in the workouts

You want to increase or decrease reps (enter 1 = Increase, 2 = Decrease)
Please enter your choice: 2
The number of reps has been decreased by 2

Press any key to continue
|

```

Check back using Display tomorrow workouts function. Number of reps have been reduced

```
Please enter your choice: 3
You choose to Display tomorrow workouts

Chest
Dumbbell Press: 8
Barbell Press: 8
Cable Crossover: 8
Back
Deadlift: 8
Lat Pulldown: 8
Dumbbell Row: 10

Press any key to continue
|
```

f. Use Change the workouts tomorrow function.

When the user selects 3, the program will print out the names of the workouts in day 2 and ask the user which workout they want to change.

```
*****MENU*****
*           Please enter your choice           *
* 1.Display schedule workouts                   *
* 2.Training                                  *
* 3.Display tomorrow workouts                  *
* 4.Change the number of reps in the workouts *
* 5.Change the workouts tomorrow              *
* 0.Exit                                       *
*****

Please enter your choice: 5
You choose to Change the workouts tomorrow

Next day workouts:
1. Chest
2. Back
What is the workouts do you want to change?
Please enter your choice: |
```

After selecting the muscle group the user wants to change, they will be given the choice to change other workouts or delete them.

```
Please enter your choice: 5
You choose to Change the workouts tomorrow

Next day workouts:
1. Chest
2. Back
What is the workouts do you want to change?
Please enter your choice: 1
You choose to change Chest muscle group workouts
Do you want to remove or replace a new exercise?(1 = Replace / 2 = Remove)
Please enter your choice: |
```

In option 1 as replacement, users will be able to see the workouts on Day 3 and select the one they want to replace.

```
Next day workouts:
1. Chest
2. Back
What is the workouts do you want to change?
Please enter your choice: 1
You choose to change Chest muscle group workouts
Do you want to remove or replace a new exercise?(1 = Replace / 2 = Remove)
Please enter your choice: 1
Choose the muscle group you want to replace
Next day workouts:
1. Leg
2. Shoulder
Please enter your choice: |
```

After selecting muscle group the user wants to replace, the program will replace workout

```
Next day workouts:
1. Chest
2. Back
What is the workouts do you want to change?
Please enter your choice: 1
You choose to change Chest muscle group workouts
Do you want to remove or replace a new exercise?(1 = Replace / 2 = Remove)
Please enter your choice: 1
Choose the muscle group you want to replace
Next day workouts:
1. Leg
2. Shoulder
Please enter your choice: 2
The Chest muscle group has been changed to the Shoulder muscle group

Press any key to continue
|
```

Check back using Display tomorrow workouts function. Workouts have been replaced.

```
Please enter your choice: 3
You choose to Display tomorrow workouts

Shoulder
Lateral Raise: 6
Overhead Dumbbell Press: 8
Side Lateral Raises: 8
Back
Deadlift: 8
Lat Pulldown: 8
Dumbbell Row: 10

Press any key to continue
|
```

In option 2 as delete the workout, program will delete it.

```
Please enter your choice: 5
You choose to Change the workouts tomorrow

Next day workouts:
1. Shoulder
2. Back
What is the workouts do you want to change?
Please enter your choice: 1
You choose to change Shoulder muscle group workouts
Do you want to remove or replace a new exercise?(1 = Replace / 2 = Remove)
Please enter your choice: 2
Workouts has been deleted

Press any key to continue
|
```

Check back using Display tomorrow workouts function. Workouts have been deleted.

```
Please enter your choice: 3
You choose to Display tomorrow workouts

Back
Deadlift: 8
Lat Pulldown: 8
Dumbbell Row: 10

Press any key to continue
```

g. Use Exit function.

When the user choose 0, the program will print out a message and finish.

```
*****MENU*****
*           Please enter your choice           *
* 1.Display schedule workouts                    *
* 2.Training                                    *
* 3.Display tomorrow workouts                   *
* 4.Change the number of reps in the workouts  *
* 5.Change the workouts tomorrow               *
* 0.Exit                                         *
*****

Please enter your choice: 0
You choose exit
Press enter to exit...
|
```


IV. Discussion

1. Range of similar patterns

a. Summary of some similar patterns.

According to Richardson C. in the book “Microservices Patterns: With examples in Java” (2019), the definition of some similar patterns is provided as follows

Factory Method Pattern specifies an interface for producing an object while allowing subclasses to choose which class to instantiate. The Factory function allows a class to delegate instantiation to its subclasses. By constructing a sort of virtual constructor, this pattern delegated the job of initializing a class from the client to a specific factory class. To do this, we rely on a factory to provide us with the objects while concealing the real implementation details. A common interface is used to access the produced objects.

Builder Pattern creates several complicated creations from the same set of component elements. The Director class, for example, may pass a huge list of names and addresses into the Builder. Depending on the kind of Builder, the list might be used to generate an HTML page, an XML page, or a PostScript file.

Prototype pattern generates new objects by duplicating an existing prototype. You alter the object that is generated by changing the object that is copied. This is handy when you require several copies of a very complicated object with the same beginning state.

b. Why Factory Method pattern is the most suitable for your scenario?

First, the initialization process is relatively simple, and the constructor only requires a handful of parameters. So, using Factory Method will help save more time in the process of planning and writing code for the project.

Second, the exercises for the muscle groups are at a high level, when the current implementation cannot comfortably accommodate new change. When any changes take place. For example, adding workouts for a single biceps and forearms group instead of just one arm muscle group, we will find that modifying the code will become time consuming and complicated, and there will be many files that need to be edited. Along with that, if the editing is not thorough or there are errors, the program will crash and take more time to correct those errors.

Third, because the program caters to many customers who are club members. Everyone has a different fitness and training needs, so the implementation of an interface or an abstract class is expected to change frequently. Using the factory method makes it quicker and easier for the programmer to modify. Continuing the example above, when the user wants to have exercises for the biceps and forearms instead of just the biceps, the programmer can simply remove the Arm workouts classes and add new classes for Biceps. and Forests group.

2. Usage of pattern

a. Advantages

For example, when users want to add new workouts, programmers can easily extend and add new code to the program without breaking the original objects. In addition, this design pattern helps to group the code that generates workouts into one place in the program, making it easy to track and manipulate. Besides, reducing the possibility of compile errors, in case you need to create a workout object but forget to declare the class, the programmer can also handle the error in the Factory and declare the class for them later.

b. Disadvantage

Because there are many types of workouts for many muscle groups, the source code becomes more complicated than usual. Because it requires using many classes to be able to implement this pattern. Besides, refactoring an existing normal class into a class with Factory Method can lead to many errors in the system, breaking the existence of clients.

V. Conclusion

In conclusion, the author gave a straightforward illustration of how to write a program that includes exercises and applies the factory approach design pattern to include the necessary code to address the problem discussed before. The standards of the template will be followed and adhered to by the code. The console interface validates user input and is loaded with basic functions. The chosen design pattern has also been extensively evaluated by the author in terms of its reasons, advantages, and disadvantages.

VI. Reference

Richardson, C. (2019). *“Microservices patterns : with examples in Java”*. Shelter Island, New York: Manning Publications.