Chapter 0: Creational Pattern

Singleton

Name of function: Searching to find lesson by id or name

Why do you select this design pattern: Because the search action will often be performed by the user, it is not necessary to re-instantiate an object just to use the search functions, so we use the singleton pattern to ensure that the Search object is created only once and can reuse continuously

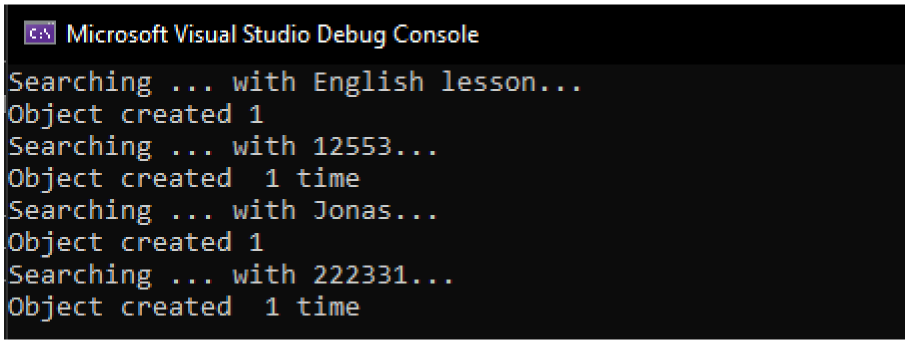
The characteristic of this design pattern:

- Singleton pattern ensures that a class has only one instance of existence at any given time
- If we try to instantiate an instance of the Singleton class for a second time, the new variable also points to the first instance that was created earlier

Code Demo: Translator demo code, users often search for lessons by name or Id

```
using System;
namespace Singleton
{
    class Program
    {
        static void Main(string[] args)
        {
            Search.GetInstance().searchByName("English
lesson");
            Search.GetInstance().searchById(12553);
            Search.GetInstance().searchByName("Jonas");
            Search.GetInstance().searchById(222331);
        }
    }

    class Search
    {
        //count the number of times from creating the Search
object
        int i = 0;
        private Search() { i++; }
        private static Search _instance;

        public static Search GetInstance()
        {
            if (_instance == null)
            {
                _instance = new Search();
            }
            return _instance;
        }

        public void searchByName(string name)
        {
            Console.WriteLine($"Searching ... with {name}...
\nObject created {i}");
        }
        public void searchById(int id)
        {
            Console.WriteLine($"Searching ... with {id}...
\nObject created  {i} time");
        }
    }
}
```

The result:


Microsoft Visual Studio Debug Console

```
Searching ... with English lesson...
Object created 1
Searching ... with 12553...
Object created  1 time
Searching ... with Jonas...
Object created 1
Searching ... with 222331...
Object created  1 time
```

Prototype

Name of function: Copy the mascot's outfit (representing each learner in the app)

Why do you select this design pattern: Because sometimes we want to improve the existing skins, we just need to copy that outfit without re-creating and adding value to it, so we use Prototype pattern is the right choice. We only care what the outfit looks like but don't know how to make it (just copy) .

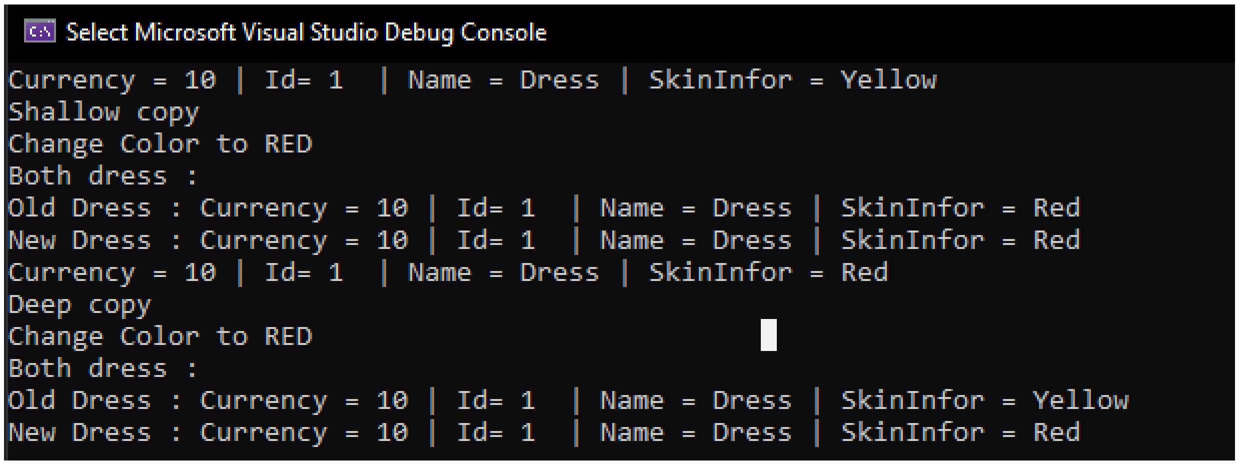
The characteristic of this design pattern:

- Prototype pattern design that lets you copy existing objects without making your code dependent on their classes.

Code Demo: Demo code : copy the mascot's outfit and change value of it.

```
using System;

namespace pattern
{
    class Program
    {
        static void Main(string[] args)
        {
            Stuff dress = new Stuff() { Currency = 10, Id =
1, Name = "Dress", skin = new Skin("Yellow") };
            Console.WriteLine(dress);
            Console.WriteLine("Shallow copy");

            Stuff newDress = dress.ShallowCopy();
            Console.WriteLine("Change Color to RED");
            newDress.skin.Color = "Red";
            Console.WriteLine("Both dress :");
            Console.WriteLine($"Old Dress : {dress} \nNew
Dress : {newDress}");



            Stuff dress2 = new Stuff() { Currency = 10, Id =
1, Name = "Dress", skin = new Skin("Yellow") };
            Console.WriteLine(dress);
            Console.WriteLine("Deep copy");
            Stuff modernDress = dress2.DeepCopy();
            Console.WriteLine("Change Color to RED");
            modernDress.skin.Color = "Red";
            Console.WriteLine("Both dress :");
            Console.WriteLine($"Old Dress : {dress2} \nNew
Dress : {modernDress}");

        }


    }

    public class Stuff
    {
        public int Currency { get; set; }
        public int Id { get; set; }
        public string Name { get; set; }
        public Skin skin { get; set; }
        public Stuff ShallowCopy()
        {
            return (Stuff)this.MemberwiseClone();
        }

        public Stuff DeepCopy()
        {
            Stuff clone = (Stuff)this.MemberwiseClone();
            clone.Currency = Currency;
            clone.Id = Id;
            clone.Name = Name;
            clone.skin = new Skin(skin.Color);
            return clone;
        }
        public override string ToString()
        {
            return $"Currency = {Currency} | Id= {Id}  |
Name = {Name} | SkinInfor = {skin}";
        }
    }
    public class Skin
    {
        public string Color { get; set; }
        public Skin(string color)
        {
            this.Color = color;
        }
        public override string ToString()
        {
            return Color;
        }

    }
}
```

The result:



```
Currency = 10 | Id= 1  | Name = Dress | SkinInfor = Yellow
Shallow copy
Change Color to RED
Both dress :
Old Dress : Currency = 10 | Id= 1  | Name = Dress | SkinInfor = Red
New Dress : Currency = 10 | Id= 1  | Name = Dress | SkinInfor = Red
Currency = 10 | Id= 1  | Name = Dress | SkinInfor = Red
Deep copy
Change Color to RED
Both dress :
Old Dress : Currency = 10 | Id= 1  | Name = Dress | SkinInfor = Yellow
New Dress : Currency = 10 | Id= 1  | Name = Dress | SkinInfor = Red
```

Chapter 1: Structural Pattern

ADAPTER PATTERN

Name of function: Lookup Dictionary

Why do you select this design pattern: because users want to interact or want to understand information in the process of learning English, they must use a dictionary, or use some other tool to be able to translate, such as translating from English to Vietnamese, so we use adapter pattern to solve this problem.

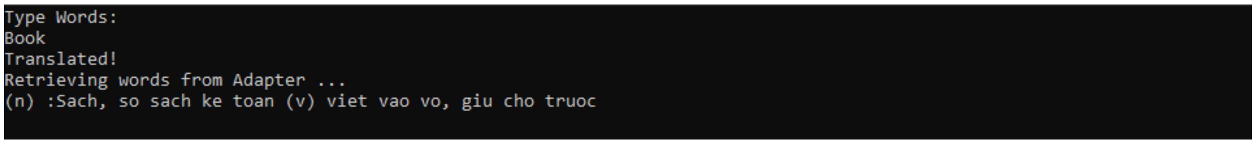
The characteristic of this design pattern:

- Adapter Pattern is a structural design pattern that convert the interface of a class into another interface clients expect. Adapter lets classes work together that couldn’t otherwise because of incompatible interfaces.
- The Adapter Pattern acts as an intermediary between two classes, converting the interface of one or more existing classes into another, appropriate for the class being written.

Code Demo: Translator demo code, users can translate any language through the dictionary interface

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace AdapterTest
{
    public interface Dictionary
    {

        void send(string words);

    }

    public class database
    {

        public virtual void receive(string words)
        {
            Console.WriteLine("Retrieving words from Adapter
...");
            Console.WriteLine(words);
        }
    }

    public class TranslatorAdapter : Dictionary
    {
        private database adaptee;

        public TranslatorAdapter(database adaptee)
        {
            this.adaptee = adaptee;
        }

        public void send(string words)
        {
            Console.WriteLine("Type Words:");
            Console.WriteLine(words);
            string vietnameseWords = this.translate(words);
            adaptee.receive(vietnameseWords);
        }

        private string translate(string vietnameseWords)
        {
            Console.WriteLine("Translated!");
            return "(n) :Sach, so sach ke toan (v) viet vao
vo, giu cho truoc";
        }
    }

    public class user
    {

        public static void Main(string[] args)
        {
            Dictionary user = new TranslatorAdapter(new
database());
            user.send("Book");
            Console.ReadLine();
        }
    }
}
```

The result:



```
Type Words:
Book
Translated!
Retrieving words from Adapter ...
(n) :Sach, so sach ke toan (v) viet vao vo, giu cho truoc
```

BRIDGE PATTERN

Name of function: Setup Object Goal

Why do you select this design pattern: Due to the large number of users, it is extremely difficult to create an additional class to manage, it is extremely important that we can minimize the creation of more classes, easy to handle 2 components What matters is abstraction and implementation. So we use the Bridge Pattern to solve this problem.

The characteristic of this design pattern:

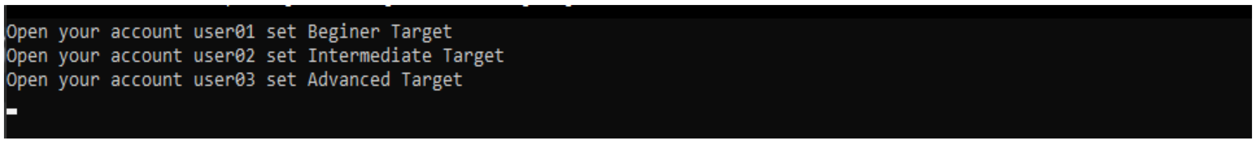
- Its idea is to separate its abstraction from its implementation. From there, it can be easily edited or replaced without affecting the places where the original layer is used.
- That is to say, we originally designed a class with a lot of handles, now we don't want to put those handlers in that class anymore. So we will create another class and move those handlers to the new class. Then, in the old class will keep an object belonging to the new class, and this object will be responsible for handling instead of the original class.

Code Demo: Demo code creating more goals for users.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace BridgePattern
{
    public interface Account
    {
        void openAccount();
    }

    public class BeginerLearner : Account
    {

        public void openAccount()
        {
            Console.WriteLine("Beginer Target");
        }
    }

    public class IntermediateLeaner : Account
    {

        public void openAccount()
        {
            Console.WriteLine("Intermediate Target");
        }
    }

    public class AdvancedLeaner : Account
    {

        public void openAccount()
        {
            Console.WriteLine("Advanced Target ");
        }
    }
```

```
public abstract class Learner
{
    protected internal Account account;

    public Learner(Account account)
    {
        this.account = account;
    }

    public abstract void openAccount();
}

public class user01 : Learner
{

    public user01(Account account) : base(account)
    {
    }

    public override void openAccount()
    {
        Console.Write("Open your account user01 set ");
        account.openAccount();
    }
}

public class user02 : Learner
{

    public user02(Account account) : base(account)
    {
    }

    public override void openAccount()
    {
        Console.Write("Open your account user02 set ");
        account.openAccount();
    }
    }

    public class user03 : Learner
    {

        public user03(Account account) : base(account)
        {
        }

        public override void openAccount()
        {
            Console.Write("Open your account user03 set ");
            account.openAccount();
        }
    }

    public class Admin
    {

        public static void Main(string[] args)
        {
            Learner user1 = new user01(new
BeginerLearner());
            user1.openAccount();

            Learner user2 = new user02(new
IntermediateLeaner());
            user2.openAccount();

            Learner user3 = new user03(new
AdvancedLeaner());
            user3.openAccount();
            Console.ReadLine();
        }
    }
}
```

The result:



```
Open your account user01 set Beginer Target
Open your account user02 set Intermediate Target
Open your account user03 set Advanced Target
```

Chapter 2: Behavior Pattern

Observer

Name of function: Working around with **Discussion forum** (View thread, answer thread and add new thread)

Why do you select this design pattern: Because when the user enter the forum, there must be something to keep track of user's activity in order to react back synchronously on time. The observer is best suit for this kind of scenario for its usability as every time the user interact with the forum there would be some kind of observable thing to notice every single act of the user and gave the immediate respond back.

The characteristic of this design pattern:

- Observer pattern is a pattern that belong to the group Behavior Pattern
- It defines a relationship of one-to-many between subject. When subject changes its own state, it notify the observer, so that the observer could also update any of its dependent automatically.

Code Demo:

For **Comment.cs**

```
class Comment
{
    private int commentdid;
    private string messsage;
    private DateTime posteddate;

    public int CommendID
    {
        get
        {
            return this.commentdid;
        }
        set
        {
            this.commentdid = value;
        }
    }
    public string Message
    {
        get
        {
            return this.messsage;
        }
        set
        {
            this.messsage = value;
        }
    }
    public DateTime PostedDate
    {
        get
        {
            return this.posteddate;
        }
        set
        {
            this.posteddate = value;
        }
    }
```

```
        public List<Comment> ListofComments()
        {
            List <Comment> comments = new List<Comment>()
            {
                new Comment(){commentdid = 0, messsage =
"test", posteddate = new DateTime(2021, 5, 28)},
                new Comment(){commentdid = 1, messsage =
"testXD", posteddate = new DateTime(2021, 5, 30)},
                new Comment(){commentdid = 2, messsage =
"XD", posteddate = new DateTime(2021, 5, 31)},
                new Comment(){commentdid = 3, messsage =
"JustTesting", posteddate = new DateTime(2021, 5, 31)},
                new Comment(){commentdid = 4, messsage =
"TestComment", posteddate = new DateTime(2021, 5, 31)}
            };

            return comments;
        }
    }
```

For **Program.cs**

```
    class Program
    {

        static void Main()
        {
            //Show the thread here
            Thread t = new Thread();

            //get the list of threads
            List<Thread> listthreads = t.ListofThread();

            for (int i=0; i< listthreads.Count; i++)
            {

Console.WriteLine("////////////////////////////////////");
                Console.WriteLine("Thread number:
"+listthreads[i].Threadid);
Console.WriteLine(listthreads[i].Threadtitle);

Console.WriteLine(listthreads[i].Thread_PostedDate);

Console.WriteLine(listthreads[i].Thread_LastUpdated);

Console.WriteLine(listthreads[i].Thread_Image);

Console.WriteLine(listthreads[i].Thread_Description);
                Console.WriteLine("\n");
                Console.WriteLine("List of comments:");

Console.WriteLine(listthreads[i].ListComments[i].CommendID);

Console.WriteLine(listthreads[i].ListComments[i].Message);

Console.WriteLine(listthreads[i].ListComments[i].PostedDate)
;

Console.WriteLine("///////////////////////////////////");
                Console.WriteLine("\n");
            }

            //Create new observer
            var concreteobserverWatchThread = new
ConcreteObserverWatchThread();
            var concreteobserverAnswerThread = new
ConcreteObserverAnswerThread();
            var concreteobserverAddThread = new
ConcreteObserverAddThread();

            t.Attach(concreteobserverWatchThread);
            t.WatchSubject(0);

            //For thread that does not exist
            t.WatchSubject(6);
            //Detach observer so observer is unable to
retrieve thread data to show
            t.Detach(concreteobserverWatchThread);
            t.WatchSubject(3);

            //Answering Thread
            t.Attach(concreteobserverAnswerThread);
            t.AnswerSubject();

            //Detach AnswerObserverConcrete
            t.Detach(concreteobserverAnswerThread);
            t.AnswerSubject();

            //For Adding new thread

            t.Attach(concreteobserverAddThread);
            t.AddSubject();
        }
    }


    //Observer
    public interface IObserver
    {
        // Receive update from subject
        void Update(ISubject subject);
    }

    //ConcereteObserver for WatchingThread
    class ConcreteObserverWatchThread : IObserver
    {
        public void Update(ISubject subject)
        {
            Console.WriteLine("Shown Thread");
            if ((subject as Thread).Index == 0)
            {
                Console.WriteLine("Thread id is " + (subject
as Thread).ListofThread()[0].Threadid);
```

```
                Console.WriteLine("Thread titile is " +
(subject as Thread).ListofThread()[0].Threadtitle);
                Console.WriteLine("Thread posted date is " +
(subject as Thread).ListofThread()[0].Thread_PostedDate);
                Console.WriteLine("Thread last time update
is " + (subject as
Thread).ListofThread()[0].Thread_LastUpdated);
                Console.WriteLine("Thread image is " +
(subject as Thread).ListofThread()[0].Thread_Image);
                Console.WriteLine("Thread description is " +
(subject as Thread).ListofThread()[0].Thread_Description);
            }
            else if ((subject as Thread).Index == 1)
            {
                Console.WriteLine("Thread id is " + (subject
as Thread).ListofThread()[1].Threadid);
                Console.WriteLine("Thread titile is " +
(subject as Thread).ListofThread()[1].Threadtitle);
                Console.WriteLine("Thread posted date is " +
(subject as Thread).ListofThread()[1].Thread_PostedDate);
                Console.WriteLine("Thread last time update
is " + (subject as
Thread).ListofThread()[1].Thread_LastUpdated);
                Console.WriteLine("Thread image is " +
(subject as Thread).ListofThread()[1].Thread_Image);
                Console.WriteLine("Thread description is " +
(subject as Thread).ListofThread()[2].Thread_Description);
            }
            else if ((subject as Thread).Index == 2)
            {
                Console.WriteLine("Thread id is " + (subject
as Thread).ListofThread()[2].Threadid);
                Console.WriteLine("Thread titile is " +
(subject as Thread).ListofThread()[2].Threadtitle);
                Console.WriteLine("Thread posted date is " +
(subject as Thread).ListofThread()[2].Thread_PostedDate);
                Console.WriteLine("Thread last time update
is " + (subject as
Thread).ListofThread()[2].Thread_LastUpdated);
```

```
                Console.WriteLine("Thread image is " +
(subject as Thread).ListofThread()[2].Thread_Image);
                Console.WriteLine("Thread description is " +
(subject as Thread).ListofThread()[2].Thread_Description);
            }
            else if ((subject as Thread).Index == 3)
            {
                Console.WriteLine("Thread id is " + (subject
as Thread).ListofThread()[3].Threadid);
                Console.WriteLine("Thread titile is " +
(subject as Thread).ListofThread()[3].Threadtitle);
                Console.WriteLine("Thread posted date is " +
(subject as Thread).ListofThread()[3].Thread_PostedDate);
                Console.WriteLine("Thread last time update
is " + (subject as
Thread).ListofThread()[3].Thread_LastUpdated);
                Console.WriteLine("Thread image is " +
(subject as Thread).ListofThread()[3].Thread_Image);
                Console.WriteLine("Thread description is " +
(subject as Thread).ListofThread()[3].Thread_Description);
            }
            else if ((subject as Thread).Index == 4)
            {
                Console.WriteLine("Thread id is " + (subject
as Thread).ListofThread()[4].Threadid);
                Console.WriteLine("Thread titile is " +
(subject as Thread).ListofThread()[4].Threadtitle);
                Console.WriteLine("Thread posted date is " +
(subject as Thread).ListofThread()[4].Thread_PostedDate);
                Console.WriteLine("Thread last time update
is " + (subject as
Thread).ListofThread()[4].Thread_LastUpdated);
                Console.WriteLine("Thread image is " +
(subject as Thread).ListofThread()[4].Thread_Image);
                Console.WriteLine("Thread description is " +
(subject as Thread).ListofThread()[4].Thread_Description);
            }

            //non exist thread
            else
            {
                Console.WriteLine("Thread not exist please
try again !!!");
            }

            Console.WriteLine("\n");
        }
    }

    //ConcreteObserver for AnsweringThread
    class ConcreteObserverAnswerThread : IObserver
    {
        public void Update(ISubject subject)
        {
            Console.WriteLine("Answer saved!\n");
        }
    }

    class ConcreteObserverAddThread : IObserver
    {
        public void Update(ISubject subject)
        {
            Thread t = new Thread();

            t.Threadid = 5;
            t.Threadtitle = "Just add XD";
            t.Thread_PostedDate = new DateTime(2021, 5, 30);
            t.Thread_LastUpdated = new DateTime(2021, 5,
31);
            t.Thread_Image = 5;
            t.Thread_Description = "Just Description XD";

            (subject as Thread).ListofThread().Add(t);

            Console.WriteLine("Information of new thread
added");

Console.WriteLine("////////////////////////////////////");
```

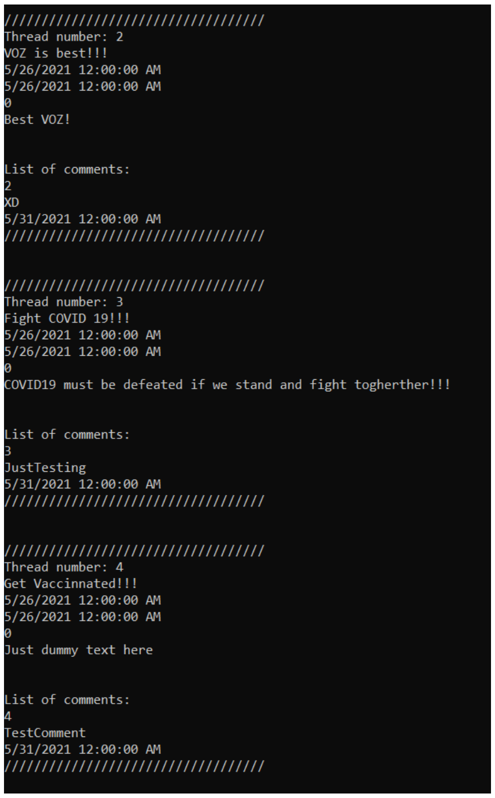
```
            Console.WriteLine("New thread id is " +
t.Threadid);
            Console.WriteLine("New thread title is " +
t.Threadtitle);
            Console.WriteLine("New thread posted date is " +
t.Thread_PostedDate);
            Console.WriteLine("New thread last updated is "
+ t.Thread_LastUpdated);
            Console.WriteLine("New thread image is " +
t.Thread_Image);
            Console.WriteLine("New thread description is " +
t.Thread_Description + "\n");
        }
    }

    //Subject
    public interface ISubject
    {
        // Attach an observer to the subject.
        void Attach(IObserver observer);

        // Detach an observer from the subject.
        void Detach(IObserver observer);

        // Notify all observers about an event.
        void Notify();
    }

    //ConcreteSubject class
    class Thread : ISubject
    {
        private int threadid;
        private string threadtitle;
        private DateTime thread_posteddate;
        private DateTime thread_lastupdated;
        private int thread_image;
        private string thread_description;
        List<Comment> listofcomments;
        private int index;
        private List<IObserver> observers = new
List<IObserver>();

        // The subscription management methods.
        public void Attach(IObserver observer)
        {
            Console.WriteLine("\nSubject: Attached an
observer.\n");
            this.observers.Add(observer);
        }

        public void Detach(IObserver observer)
        {
            this.observers.Remove(observer);
            Console.WriteLine("Subject: Detached an
observer.\n");
        }

        // Trigger an update in each observer
        public void Notify()
        {
            Console.WriteLine("Subject: Notifying
observers...");

            foreach (var observer in observers)
            {
                observer.Update(this);
            }
        }

        public int Threadid
        {
            get
            {
                return this.threadid;
            }
            set
            {
```

```
        this.threadid = value;
    }
}
public string Threadtitle
{
    get
    {
        return this.threadtitle;
    }
    set
    {
        this.threadtitle = value;
    }
}
public DateTime Thread_PostedDate
{
    get
    {
        return this.thread_posteddate;
    }
    set
    {
        this.thread_posteddate = value;
    }
}
public DateTime Thread_LastUpdated
{
    get
    {
        return this.thread_lastupdated;
    }
    set
    {
        this.thread_lastupdated = value;
    }
}
public int Thread_Image
{
    get
    {
        return this.thread_image;
    }
    set
    {
        this.thread_image = value;
    }
}
public string Thread_Description
{
    get
    {
        return this.thread_description;
    }
    set
    {
        this.thread_description = value;
    }
}
public List<Comment> ListComments
{
    get
    {
        return this.listofcomments;
    }
    set
    {
        this.listofcomments = value;
    }
}

public int Index
{
    get
    {
        return this.index;
    }
    set
    {
```

```
                this.index = value;
            }
        }

        public void WatchSubject(int index)
        {
            Console.WriteLine("Waching Thread number: " +
index);
            Console.WriteLine("Feching from database...");
            this.index = index;
            this.Notify();
        }

        public void AnswerSubject()
        {
            Random random = new Random();
            int indexrandom = random.Next(0, 4);

            //Change subject
            this.index = indexrandom;
            Console.WriteLine("Answering thread number: " +
this.index);
            Notify();
        }

        public void AddSubject()
        {
            Console.WriteLine("Adding new Thread!");
            Notify();
        }

        public List<Thread> ListofThread()
        {
            Comment comment = new Comment();
            //Create new list of thread here
            List<Thread> threads = new List<Thread>()
            {
                new Thread(){threadid = 0, threadtitle =
"Something cool here!!!", thread_posteddate = new
DateTime(2021, 5, 26), thread_lastupdated = new
DateTime(2021, 5, 26), thread_image = 0, thread_description
= "Nothing new here!", listofcomments =
comment.ListofComments()},
                new Thread(){threadid = 1, threadtitle =
"Welcome to our VOZ thread!!!", thread_posteddate = new
DateTime(2021, 5, 26), thread_lastupdated = new
DateTime(2021, 5, 26), thread_image = 0, thread_description
= "VOZ!", listofcomments = comment.ListofComments()},
                new Thread(){threadid = 2, threadtitle =
"VOZ is best!!!", thread_posteddate = new DateTime(2021, 5,
26), thread_lastupdated = new DateTime(2021, 5, 26),
thread_image = 0, thread_description = "Best VOZ!",
listofcomments = comment.ListofComments()},
                new Thread(){threadid = 3, threadtitle =
"Fight COVID 19!!!", thread_posteddate = new DateTime(2021,
5, 26), thread_lastupdated = new DateTime(2021, 5, 26),
thread_image = 0, thread_description = "COVID19 must be
defeated if we stand and fight togherther!!!",
listofcomments = comment.ListofComments()},
                new Thread(){threadid = 4, threadtitle =
"Get Vaccinnated!!!", thread_posteddate = new DateTime(2021,
5, 26), thread_lastupdated = new DateTime(2021, 5, 26),
thread_image = 0, thread_description = "Just dummy text
here", listofcomments = comment.ListofComments()}
            };
            return threads;
        }
    }
```

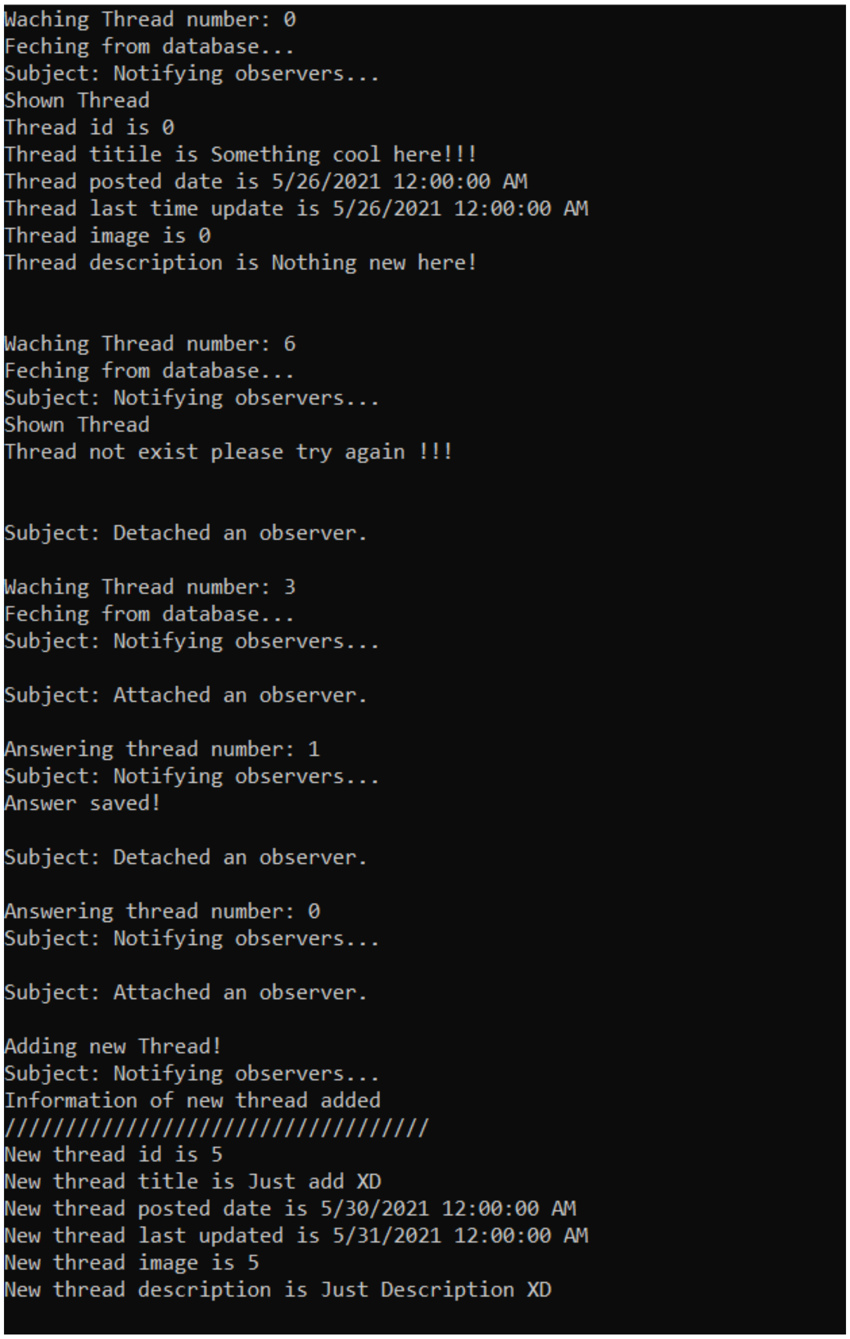
The result:

List of Thread:



```
//////////////////////////////////
Thread number: 2
VOZ is best!!!
5/26/2021 12:00:00 AM
5/26/2021 12:00:00 AM
0
Best VOZ!


List of comments:
2
XD
5/31/2021 12:00:00 AM
//////////////////////////////////


//////////////////////////////////
Thread number: 3
Fight COVID 19!!!
5/26/2021 12:00:00 AM
5/26/2021 12:00:00 AM
0
COVID19 must be defeated if we stand and fight togherther!!!


List of comments:
3
JustTesting
5/31/2021 12:00:00 AM
//////////////////////////////////


//////////////////////////////////
Thread number: 4
Get Vaccinnated!!!
5/26/2021 12:00:00 AM
5/26/2021 12:00:00 AM
0
Just dummy text here


List of comments:
4
TestComment
5/31/2021 12:00:00 AM
//////////////////////////////////
```

Watching thread, Answer thread, Adding new thread



```
Waching Thread number: 0
Feching from database...
Subject: Notifying observers...
Shown Thread
Thread id is 0
Thread titile is Something cool here!!!
Thread posted date is 5/26/2021 12:00:00 AM
Thread last time update is 5/26/2021 12:00:00 AM
Thread image is 0
Thread description is Nothing new here!


Waching Thread number: 6
Feching from database...
Subject: Notifying observers...
Shown Thread
Thread not exist please try again !!!


Subject: Detached an observer.

Waching Thread number: 3
Feching from database...
Subject: Notifying observers...

Subject: Attached an observer.

Answering thread number: 1
Subject: Notifying observers...
Answer saved!

Subject: Detached an observer.

Answering thread number: 0
Subject: Notifying observers...

Subject: Attached an observer.

Adding new Thread!
Subject: Notifying observers...
Information of new thread added
///////////////////////////////////
New thread id is 5
New thread title is Just add XD
New thread posted date is 5/30/2021 12:00:00 AM
New thread last updated is 5/31/2021 12:00:00 AM
New thread image is 5
New thread description is Just Description XD
```

Template Method

Name of function: User choose language to learn then proceed moving to create flash card session to create, edit, delete the flash card.

Why do you select this design pattern: Because the program will run in a template that consists a bunch of methods came from a class and its derived one, they all implement methods that are inside the template. Which one implements which method then it only affect that method and it doesn't affect the whole template structure which is beneficial to test out new way of coding.

The characteristic of this design pattern:

- Template pattern is a pattern that belong to the group Behavior Pattern
- It consists of 2 classes AbstractClass and ConcreteClass
  + AbstractClass define a template method that program will run and all of method that is inside the template.
  + Some methods in AbstractClass will be left behind empty, this is where its derived class come in and override those method and implement it.
  + After both of the method are implemented in AbstractClass and its derived one, then the program will run successfully without any problem.

Code Demo:

For Language.cs (To show list of languages for user to chooose)

```
class Languague
    {
        private int languageid;
        private string languagename;

        public int LanguageID
        {
            get
            {
                return this.languageid;
            }
            set
            {
                this.languageid = value;
            }
        }
        public string LanguageName
        {
            get
            {
                return this.languagename;
            }
            set
            {
                this.languagename = value;
            }
        }

        public List<Languague> getListLanguage()
        {
            List<Languague> languagues = new
List<Languague>()
            {
                new Languague(){ languageid = 1,
languagename = "English"},
                new Languague(){ languageid = 2,
languagename = "Vietnamese"},
                new Languague(){ languageid = 3,
languagename = "German"},
                new Languague(){ languageid = 4,
languagename = "Japanese"}
            };

            return languagues;
        }
    }
```

For **FlashCard.cs**

```
class FlashCard
    {
        private int flashcard_id;
        private string flashcard_title;
        private string flashcard_answer;
```

```
public int Flashcard_ID
{
    get
    {
        return this.flashcard_id;
    }
    set
    {
        this.flashcard_id = value;
    }
}

public string Flashcard_Title
{
    get
    {
        return this.flashcard_title;
    }
    set
    {
        this.flashcard_title = value;
    }
}
public string Flashcard_Answer
{
    get
    {
        return this.flashcard_answer;
    }
    set
    {
        this.flashcard_answer = value;
    }
}

public List<FlashCard> flashcardEng()
{
    var flashcard_eng = new List<FlashCard>()
    {
```

```
                new FlashCard(){flashcard_id = 0,
flashcard_title = "What does programming mean ?",
flashcard_answer = "the process or activity of writing
computer programs."},
                new FlashCard(){flashcard_id = 1,
flashcard_title = "What does light mean ?", flashcard_answer
= "It is kinda a thing that shed light to see things"}
            };

            return flashcard_eng;
        }

        public List<FlashCard> flashcardViet()
        {
            var flashcard_viet = new List<FlashCard>()
            {
                new FlashCard(){flashcard_id = 0,
flashcard_title = "Lap trinh la gi ?", flashcard_answer =
"La qua trinh viet ra phan mem de ung dung trong cuoc
song"},
                new FlashCard(){flashcard_id = 1,
flashcard_title = "Bong den la gi?", flashcard_answer = "Den
la vat dung de chieu anh sang ra de su dung cho nhieu viec
khac nhau"}
            };

            return flashcard_viet;
        }

        public List<FlashCard> flashcardGer()
        {
            var flashcard_ger = new List<FlashCard>()
            {
                new FlashCard(){flashcard_id = 0,
flashcard_title = "Was bedeutet Programmieren?",
flashcard_answer = "den Prozess oder die Aktivität des
Schreibens von Computerprogrammen."},
                new FlashCard(){flashcard_id = 1,
flashcard_title = "Was bedeutet Licht?", flashcard_answer =
"Es ist eine Sache, die Licht ins Dunkel bringt, um Dinge zu 
sehen"}
            };
            return flashcard_ger;
        }

        public List<FlashCard> flashcardJap()
        {
            var flashcard_jap = new List<FlashCard>()
            {
                new FlashCard(){flashcard_id = 0, 
flashcard_title = "プログラミングとはどういう意味ですか？", 
flashcard_answer = "コンピュータプログラムを書くプロセスまたは活
動"},
                new FlashCard(){flashcard_id = 1, 
flashcard_title = "光とはどういう意味ですか？", 
flashcard_answer = "物事を見るのに光を当てるのはちょっと物"}
            };
            return flashcard_jap;
        }

        public List<FlashCard> flashcardAnyCountry(int 
countryindex)
        {
            if (countryindex == 1)
            {
                return flashcardEng();
            }
            else if (countryindex == 2)
            {
                return flashcardViet();
            }
            else if (countryindex == 3)
            {
                return flashcardGer();
            }
            else
            {
                return flashcardJap();
            }
        }
    }
```

For **Program.cs**

```
    abstract class AbstractClass
    {

        public List<FlashCard> ListofFlashCard;
        public void TemplateMethod()
        {
            this.ShowListLanaguage();
            this.ChooseLanguageToShowFlashCard();
            this.WatchFlashCardList();
            this.AddNewFlashCard();
            Console.WriteLine("List after adding new
card\n");
            this.ShowAllFlashCard();
            this.EditFlashCard();
            Console.WriteLine("List after edditing\n");
            this.ShowAllFlashCard();
            this.DeleteFlashCard();
            Console.WriteLine("List after deleting\n");
            this.ShowAllFlashCard();
        }

        // These operations already have implementations.
        protected void ShowListLanaguage()
        {
            Languague lan = new Languague();


Console.WriteLine("\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\");
            for (int i = 0; i < lan.getListLanguage().Count;
i++)
            {
```

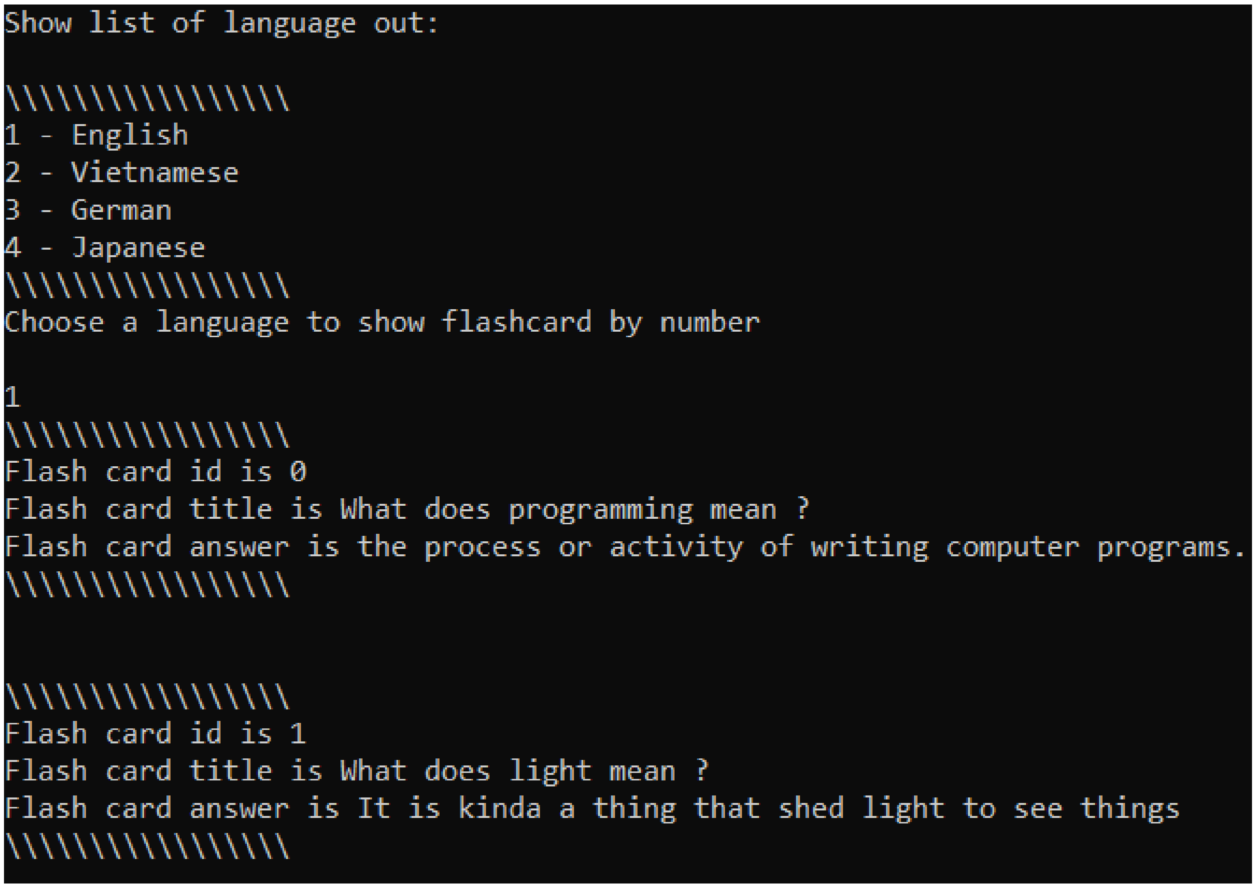
```
Console.WriteLine(lan.getListLanguage()[i].LanguageID + " -
" + lan.getListLanguage()[i].LanguageName);
            }

Console.WriteLine("\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\");
        }

        // These operations have to be implemented in
subclasses.
        protected abstract void
ChooseLanguageToShowFlashCard();

        protected abstract void WatchFlashCardList();

        protected abstract void AddNewFlashCard();

        protected abstract void ShowAllFlashCard();

        protected abstract void EditFlashCard();

        protected abstract void DeleteFlashCard();

    }


    class ConcreteClass1 : AbstractClass
    {
        protected override void AddNewFlashCard()
        {
            Console.WriteLine("Type in new flashcard id");
            int flashcardid =
Int32.Parse(Console.ReadLine());
            Console.WriteLine("Type in new flashcard
title");
            string flashcardtitle = Console.ReadLine();
            Console.WriteLine("Type in new flashcard
answer");
            string flashcardanswer = Console.ReadLine();
            this.ListofFlashCard.Add(new FlashCard {
Flashcard_ID = flashcardid, Flashcard_Title =
flashcardtitle, Flashcard_Answer = flashcardanswer });
            Console.WriteLine("New flash card added\n");
        }

        protected override void
ChooseLanguageToShowFlashCard()
        {
            int countryindex;
            Console.WriteLine("Choose a language to show
flashcard by number\n");
            do
            {
                countryindex =
Convert.ToInt32(Console.ReadLine());

                if (countryindex < 1 || countryindex > 4)
                {
                    Console.WriteLine("Please chooose the
correct nation");
                }
                //type correct
                else
                {
                    FlashCard fc = new FlashCard();
                    this.ListofFlashCard =
fc.flashcardAnyCountry(countryindex);
                }
            }
            while (countryindex < 1 || countryindex > 4);

        }

        protected override void DeleteFlashCard()
        {
            int flashcardindex;
            Console.WriteLine("Choose a flashcard to delete 
by flashcardid \n");

            do
            {
                flashcardindex = 
Int32.Parse(Console.ReadLine());

                if (flashcardindex < 0 || flashcardindex > 
2)
                {
                    Console.WriteLine("Please try again with 
correct index number");
                }
                else
                {
                    
this.ListofFlashCard.RemoveAt(flashcardindex);
                    Console.WriteLine("Done removing 
flashcard number " + flashcardindex);
                }
            }
            while (flashcardindex < 0 || flashcardindex > 
2);
        }

        protected override void EditFlashCard()
        {
            int flashcardindex;
            Console.WriteLine("Choose a flashcard to edit by 
flashcardid \n");

            do
            {
                flashcardindex = 
Int32.Parse(Console.ReadLine());

                if (flashcardindex < 0 || flashcardindex > 
2)
```

```
                {
                    Console.WriteLine("Please try again with
correct index number");
                }
                else
                {
                    Console.WriteLine("Type in new
title\n");
                    string newtitle = Console.ReadLine();

this.ListofFlashCard[flashcardindex].Flashcard_Title =
newtitle;
                    Console.WriteLine("Type in new
answer\n");
                    string newanswer = Console.ReadLine();

this.ListofFlashCard[flashcardindex].Flashcard_Answer =
newanswer;
                    Console.WriteLine("Done edditing
flashcard number " + flashcardindex);
                }
            }
            while (flashcardindex < 0 || flashcardindex >
2);
        }

        protected override void ShowAllFlashCard()
        {
            for (int i = 0; i < this.ListofFlashCard.Count;
i++)
            {

Console.WriteLine("\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\");
                Console.WriteLine("Flash card id is " +
this.ListofFlashCard[i].Flashcard_ID);
                Console.WriteLine("Flash card title is " +
this.ListofFlashCard[i].Flashcard_Title);
                Console.WriteLine("Flash card answer is " +
this.ListofFlashCard[i].Flashcard_Answer);
```

```
Console.WriteLine("\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\");
                Console.WriteLine("\n");
            }
        }

        protected override void WatchFlashCardList()
        {
            for (int i =0; i< this.ListofFlashCard.Count;
i++)
            {

Console.WriteLine("\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\");
                Console.WriteLine("Flash card id is " +
this.ListofFlashCard[i].Flashcard_ID);
                Console.WriteLine("Flash card title is " +
this.ListofFlashCard[i].Flashcard_Title);
                Console.WriteLine("Flash card answer is " +
this.ListofFlashCard[i].Flashcard_Answer);

Console.WriteLine("\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\");
                Console.WriteLine("\n");
            }
        }
    }



    class Client
    {

        public static void ClientCode(AbstractClass
abstractClass)
        {
            // ...
            abstractClass.TemplateMethod();
            // ...
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Show list of language 
out:\n");

            Client.ClientCode(new ConcreteClass1());

        }
    }
```

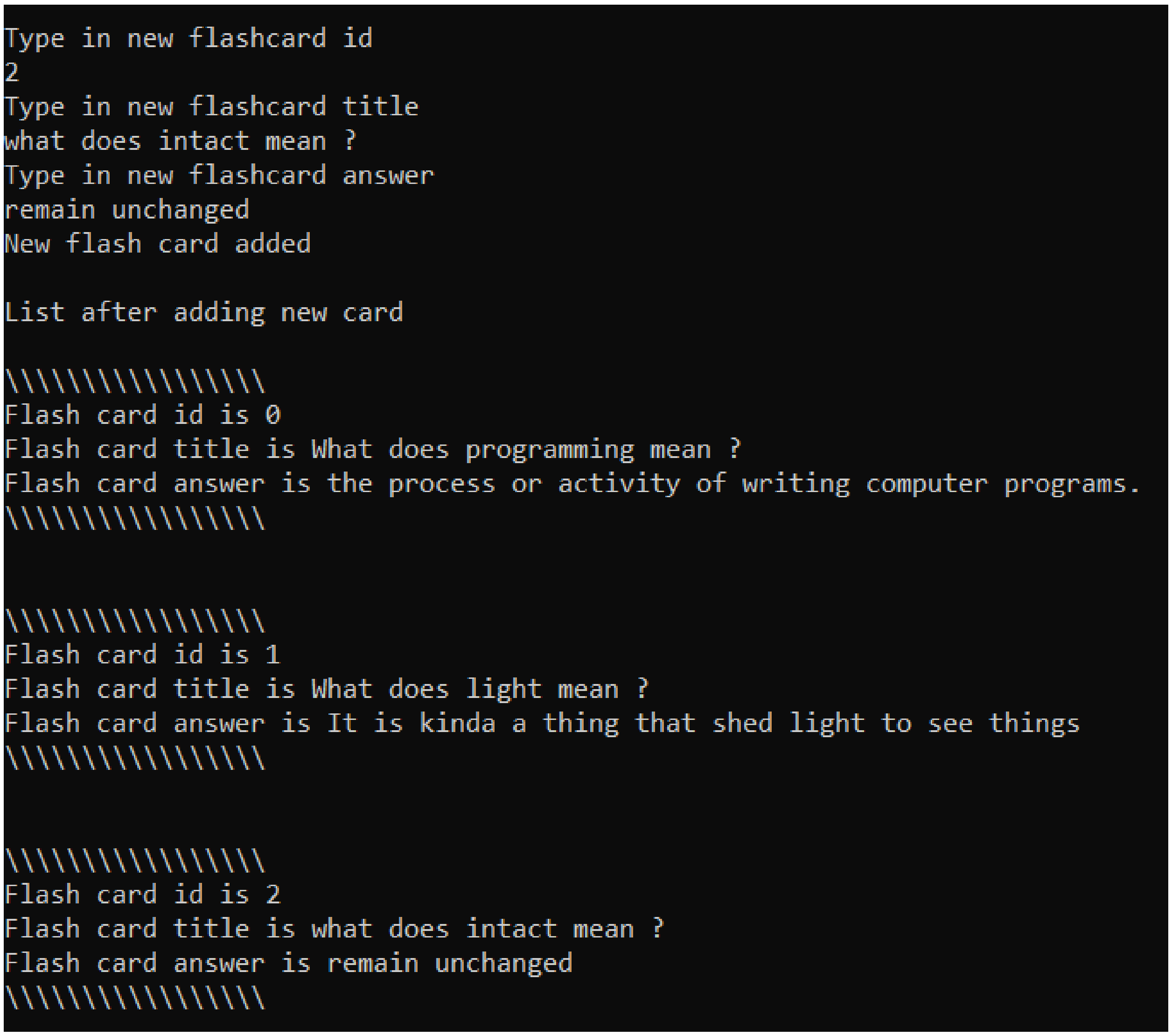
The result:

Choose language to show list of flashcards



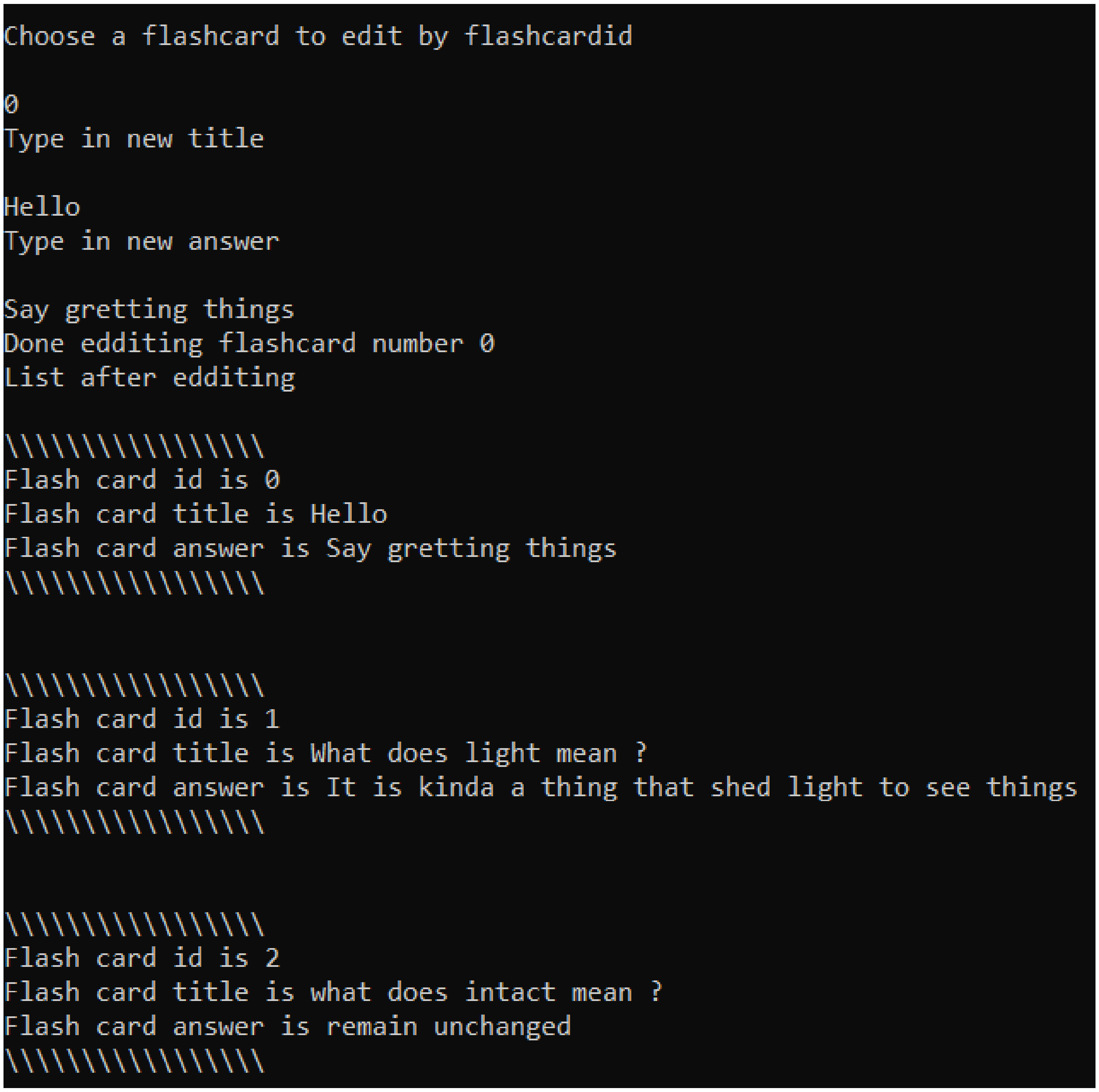
```
Show list of language out:

\\\\\\\\\\\\\\\\\
1 - English
2 - Vietnamese
3 - German
4 - Japanese
\\\\\\\\\\\\\\\\\
Choose a language to show flashcard by number

1
\\\\\\\\\\\\\\\\\
Flash card id is 0
Flash card title is What does programming mean ?
Flash card answer is the process or activity of writing computer programs.
\\\\\\\\\\\\\\\\\


\\\\\\\\\\\\\\\\\
Flash card id is 1
Flash card title is What does light mean ?
Flash card answer is It is kinda a thing that shed light to see things
\\\\\\\\\\\\\\\\\
```

Add new FlashCard



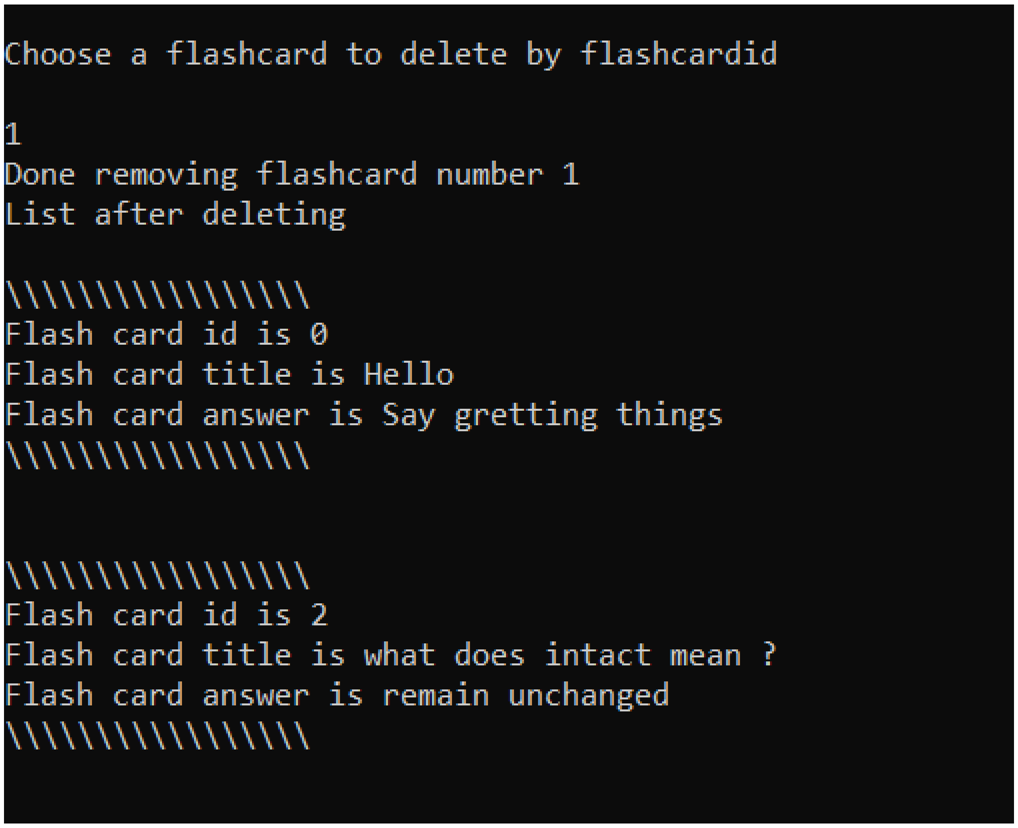
```
Type in new flashcard id
2
Type in new flashcard title
what does intact mean ?
Type in new flashcard answer
remain unchanged
New flash card added

List after adding new card

\\\\\\\\\\\\\\\\\
Flash card id is 0
Flash card title is What does programming mean ?
Flash card answer is the process or activity of writing computer programs.
\\\\\\\\\\\\\\\\\


\\\\\\\\\\\\\\\\\
Flash card id is 1
Flash card title is What does light mean ?
Flash card answer is It is kinda a thing that shed light to see things
\\\\\\\\\\\\\\\\\


\\\\\\\\\\\\\\\\\
Flash card id is 2
Flash card title is what does intact mean ?
Flash card answer is remain unchanged
\\\\\\\\\\\\\\\\\
```

Edit FlashCard



```
Choose a flashcard to edit by flashcardid

0
Type in new title

Hello
Type in new answer

Say gretting things
Done edditing flashcard number 0
List after edditing

\\\\\\\\\\\\\\\\\
Flash card id is 0
Flash card title is Hello
Flash card answer is Say gretting things
\\\\\\\\\\\\\\\\\


\\\\\\\\\\\\\\\\\
Flash card id is 1
Flash card title is What does light mean ?
Flash card answer is It is kinda a thing that shed light to see things
\\\\\\\\\\\\\\\\\


\\\\\\\\\\\\\\\\\
Flash card id is 2
Flash card title is what does intact mean ?
Flash card answer is remain unchanged
\\\\\\\\\\\\\\\\\
```

Delete FlashCard



```
Choose a flashcard to delete by flashcardid

1
Done removing flashcard number 1
List after deleting

\\\\\\\\\\\\\\\\\
Flash card id is 0
Flash card title is Hello
Flash card answer is Say gretting things
\\\\\\\\\\\\\\\\\


\\\\\\\\\\\\\\\\\
Flash card id is 2
Flash card title is what does intact mean ?
Flash card answer is remain unchanged
\\\\\\\\\\\\\\\\\
```

Chapter 3: Index