CECS 475
Lab Assignment 2
Assigned date: 1/23
Due date: 1/30
35 points

1. [20 points] Create class IntegerSet. Each IntegerSet object can hold integers in the range 0–100. The set is represented by an array of bools. Array element a[i] is true if integer i is in the set. Array element a[j] is false if integer j is not in the set. The parameterless constructor initializes the array to the "empty set" (i.e., a set whose array representation contains all false values).

Provide the following methods:

a) Method Union creates a third set that is the set-theoretic union of two existing sets (i.e., an element of the third set's array is set to true if that element is true in either or both of the existing sets—otherwise, the element of the third set is set to false).

b) Method Intersection creates a third set which is the set-theoretic intersection of two existing sets (i.e., an element of the third set's array is set to false if that element is false in either or both of the existing sets—otherwise, the element of the third set is set to true).

c) Method InsertElement inserts a new integer k into a set (by setting a[k] to true).

d) Method DeleteElement deletes integer m (by setting a[m] to false).

e) Method ToString returns a string containing a set as a list of numbers separated by spaces. Include only those elements that are present in the set. Use --- to represent an empty set.

f) Method IsEqualTo determines whether two sets are equal.

Test your class IntegerSet by using the main method below:

```
// initialize two sets
Console.WriteLine( "Input Set A" );
IntegerSet set1 = InputSet();
Console.WriteLine( "\nInput Set B" );
IntegerSet set2 = InputSet();
```

```
IntegerSet union = set1.Union( set2 );
IntegerSet intersection = set1.Intersection( set2 );

// prepare output
Console.WriteLine( "\nSet A contains elements:" );
Console.WriteLine( set1.ToString() );
Console.WriteLine( "\nSet B contains elements:" );
Console.WriteLine( set2.ToString() );
Console.WriteLine(
"\nUnion of Set A and Set B contains elements:" );
Console.WriteLine( union.ToString() );
Console.WriteLine(
"\nIntersection of Set A and Set B contains elements:" );
Console.WriteLine( intersection.ToString() );

// test whether two sets are equal
if ( set1.IsEqualTo( set2 ) )
Console.WriteLine( "\nSet A is equal to set B" );
else
Console.WriteLine( "\nSet A is not equal to set B" );

// test insert and delete
Console.WriteLine( "\nInserting 77 into set A..." );
set1.InsertElement( 77 );
Console.WriteLine( "\nSet A now contains elements:" );
Console.WriteLine( set1.ToString() );

Console.WriteLine( "\nDeleting 77 from set A..." );
set1.DeleteElement( 77 );
Console.WriteLine( "\nSet A now contains elements:" );
Console.WriteLine( set1.ToString() );

// test constructor
int[] intArray = { 25, 67, 2, 9, 99, 105, 45, -5, 100, 1 };
IntegerSet set3 = new IntegerSet( intArray );

Console.WriteLine( "\nNew Set contains elements:" );
Console.WriteLine( set3.ToString() );
} // end Main
```

Grading requirements

- A hard copy of your source code.
- Document your program
- Demonstrate the result to the instructor
- Submit the lab assigment to BeachBoard

2. [15 points] The factory concept is probably the most common design patterns and recurs throughout the object-oriented programming.

You will find countless references and uses of the factory pattern within the .net core foundational libraries and throughout the .net framework source code, most notable and probably one of the most commonly used factories can be found in the System.Data.Common namespace and the DbProviderFactories.

The factory method pattern is a clever but subtle extension to the concept that a class acts as a traffic cop and decides which subclass of a single hierarchy will be instantiated.

In the factory pattern, developers create an object without exposing the creation logic. An interface is used for creating an object, but lets subclass decide which class to instantiate. Rather than defining each object manually, developers can do it programmatically.

In short, a factory is an object that creates objects without the use of a constructor.

The pattern does not actually have a decision point where one subclass is directly selected over another class. Instead, programs are developed implementing this pattern usually define an abstract class that creates objects but lets each subclass decide which object to create.

The are a number of circumstances when developing an application when making use of the Factory Method is suitable. These situations include :

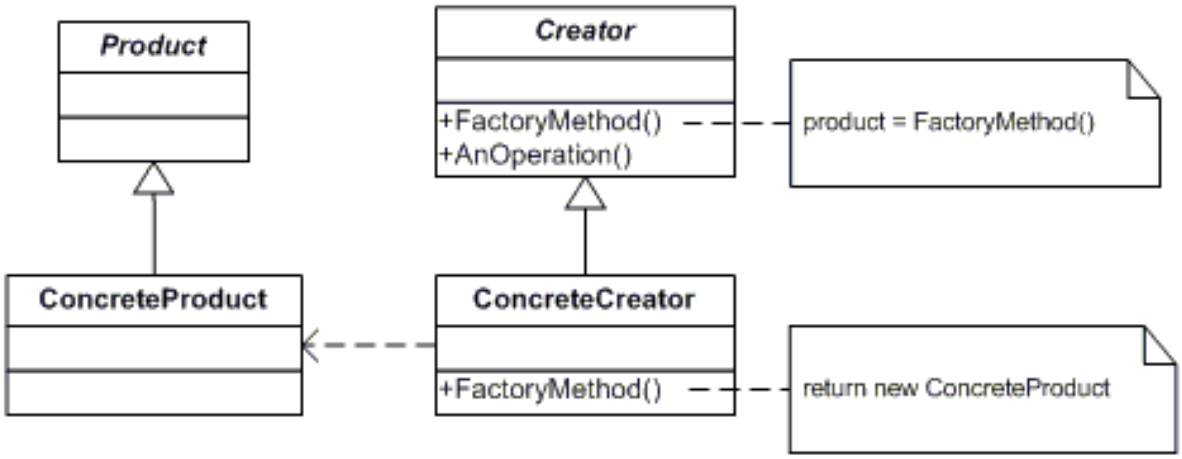
- A class can't anticipate which class objects it must create
- A class uses its subclasses to specify which objects it creates
- You need to localize the knowledge of which class gets created

It is generally considered a bad idea for base classes to know implementation details of their derived types. It is in situations like this is when you should use the Abstract Factory pattern.

A Typical situation maybe when a constructor needs to return an instance of type within which it resides, a factory method is able to return many different types of objects, all which belong to the same inheritance hierarchy.

It is tedious when the client needs to specify the class name while creating the objects. So, to resolve this problem, we can use Factory Method design pattern. It provides the client a simple way to create the object.

UML Class Diagram



The classes and objects participating in the above UML class diagram are as follows:

1. *Product*
   This defines the interface of objects the factory method creates

2. *ConcreteProduct*
   This is a class which implements the Product interface.

3. *Creator*
   This is an abstract class and declares the factory method, which returns an object of type Product.

   This may also define a default implementation of the factory method that returns a default ConcreteProduct object.

   This may call the factory method to create a Product object.

4. *ConcreteCreator*
   This is a class which implements the Creator class and overrides the factory method to return an instance of a ConcreteProduct.

Now let's understand this with a real world example

Assume you have three different healthcare plans which are considered here as classes HMO, PPO, ObamaCare, all of them implement abstract class HealthPlan. You need to instantiate one of these classes, but you don't know which of them, it depends on the user. This is perfect scenario for the Factory Method design pattern.

Each plan should have annual charge, deduction amount, plan type (HMO, PPO, ObamaCare).

Write a console application that asks the users the healthcare plan they want to apply and output the information about the plan (plan type, annual charge, and deduction amount).

Grading requirements

- A hard copy of your source code.
- Document your program
- Demonstrate the result to the instructor
- Submit the lab assigment to BeachBoard