**ABSTRACT FACTORY**

Name: Ngô Viết Hoàng Linh

Class: SE1612-NET

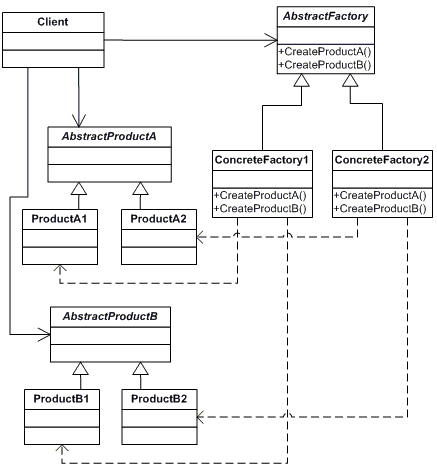
Student code: HE163171

**I – Research**

1. **Purpose of Abstract Factory**

* The Abstract Factory design pattern provides an interface for creating families of related or dependent objects without specifying their concrete classes.

1. **Abstract Factory Structural**
2. UML Class Diagram:

* 
* Explained class diagram:

+ Abstract Factory have the job to define the abstract methods to create Product

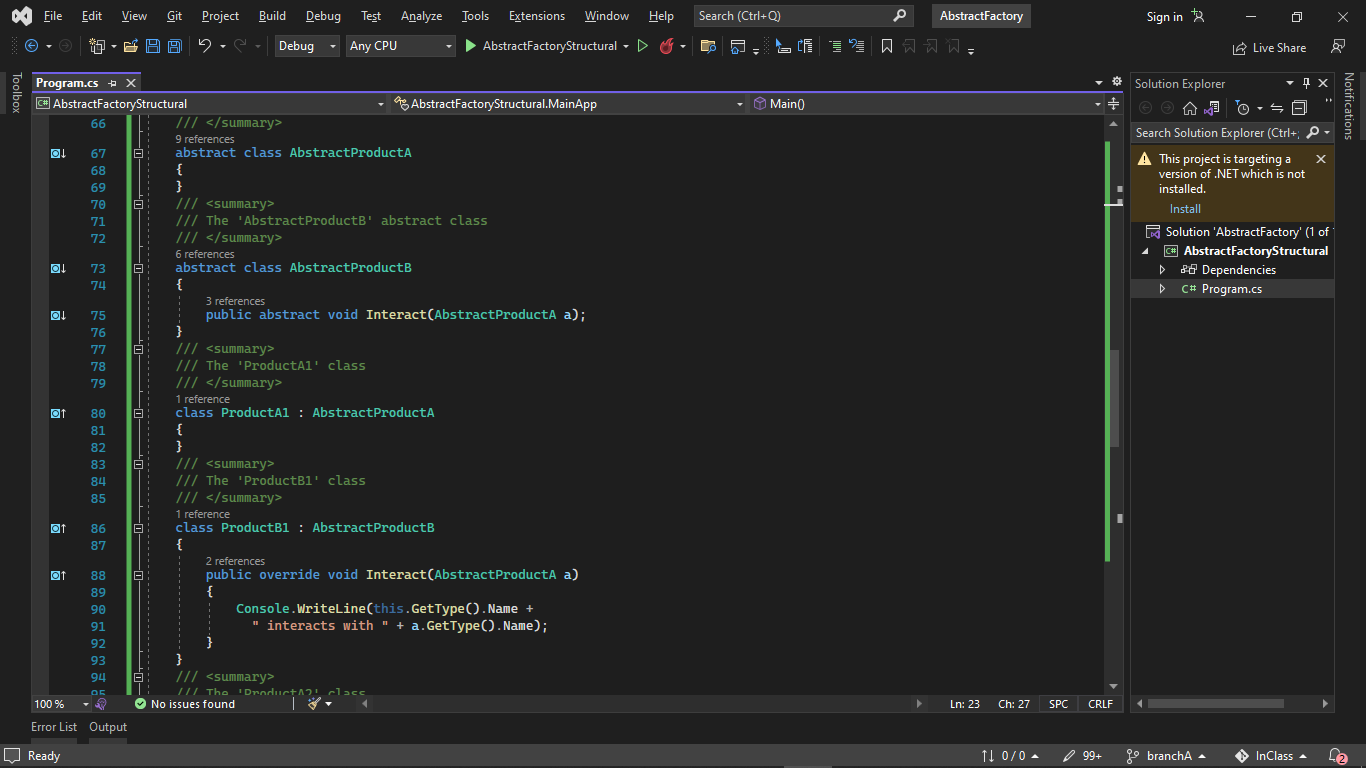
+ Concrete Factory will inherit abstract factory and override abstract method to create its Product

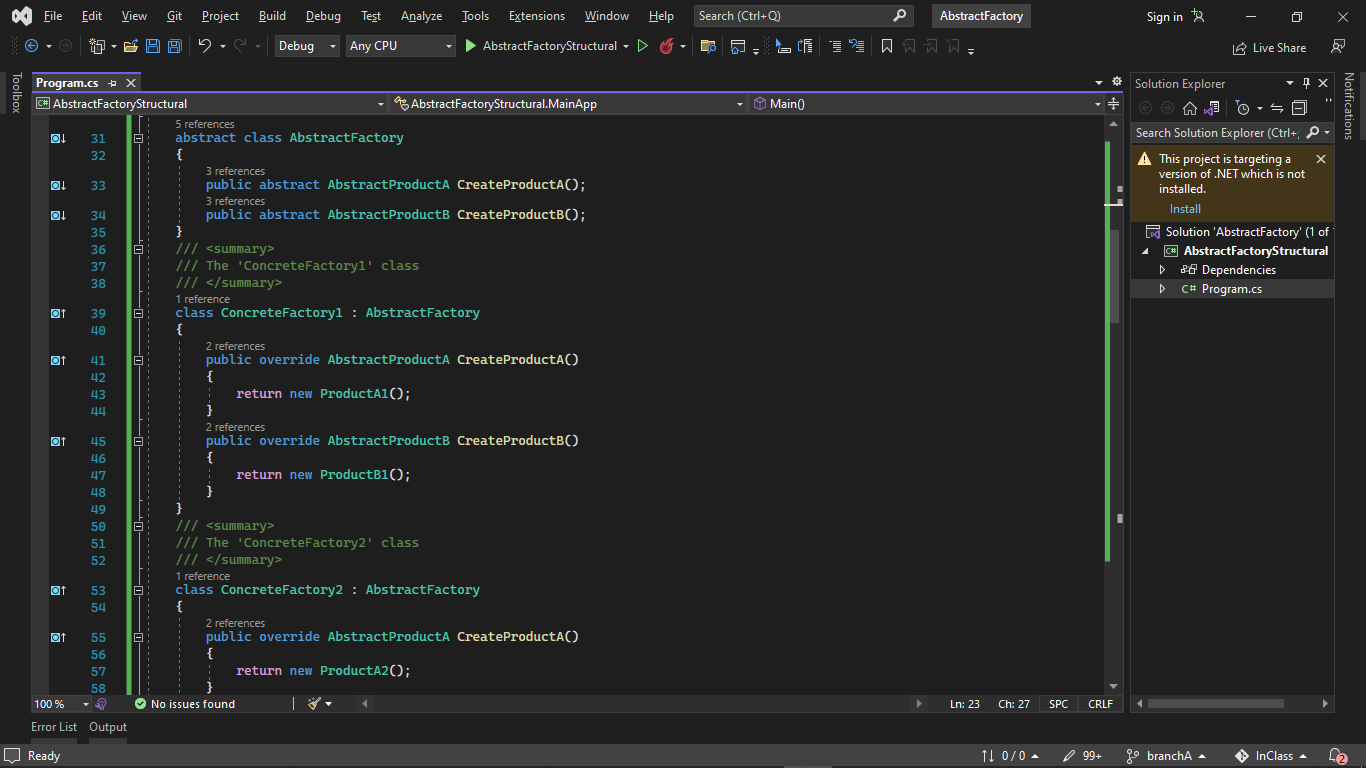
+ Abstract product have the job to define to abstract product to creeate.

+ Concrete product will inherit abstract product to implement their own product.

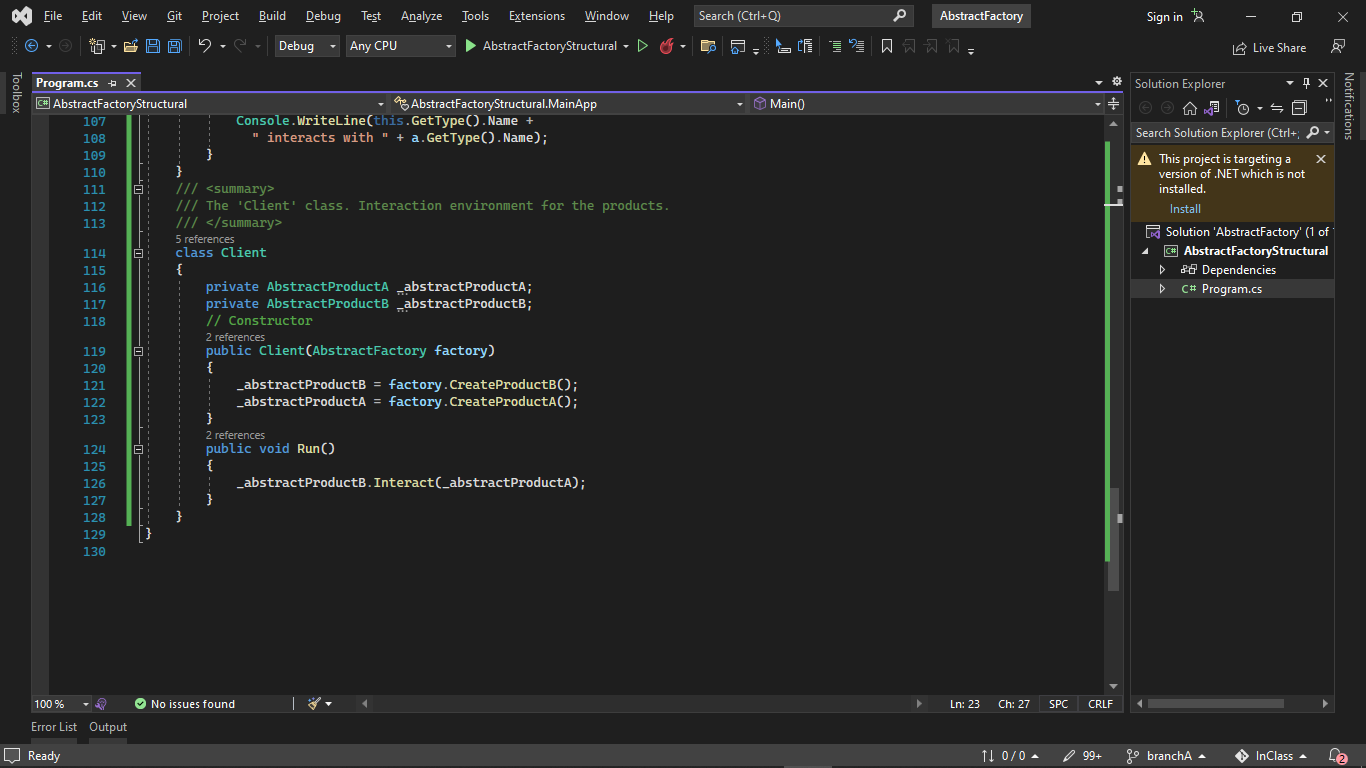
+ Client will only need to use AbstractProduct and AbstractFactory to define which product be created.

1. Understand structural code example:

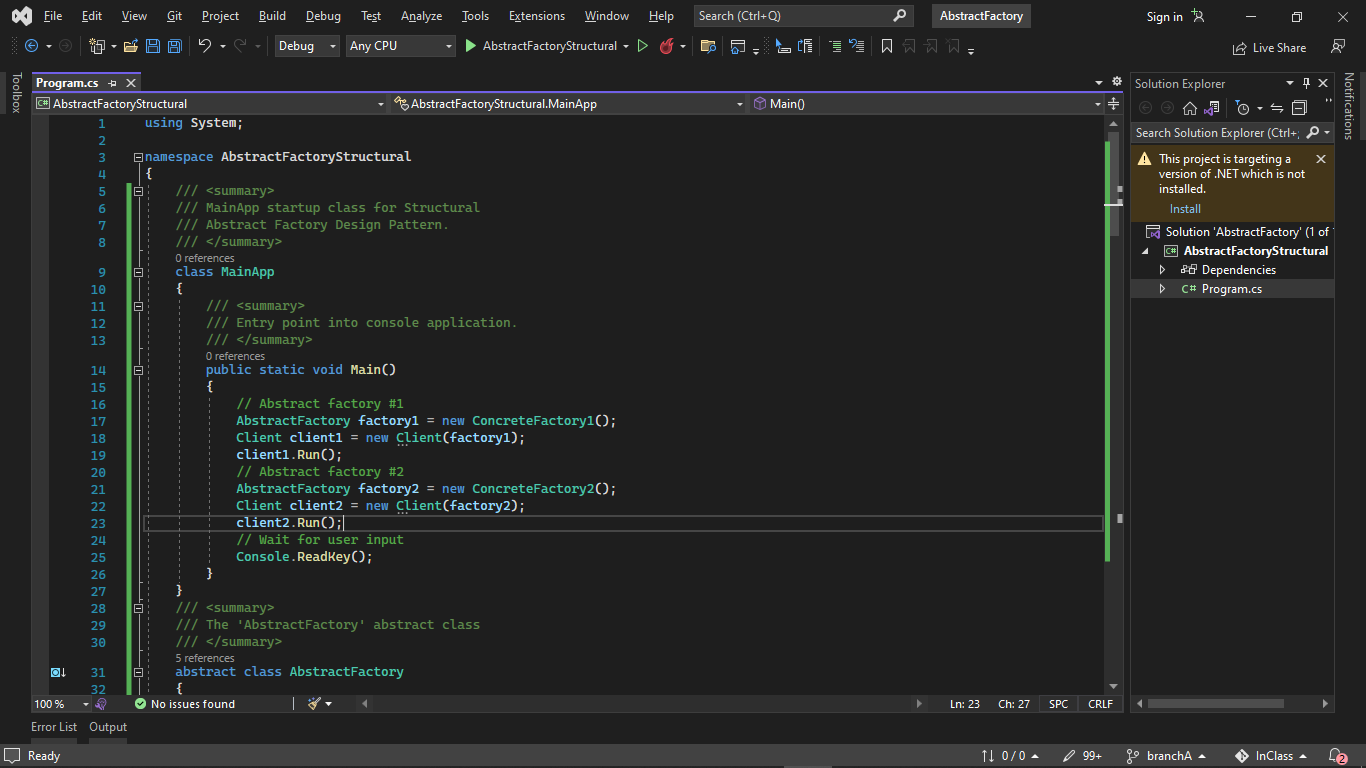
* 
* Abstract Product is used to declare an interface for type of product object ProdcutA and and Product B and define way concrete product must interact with each other.



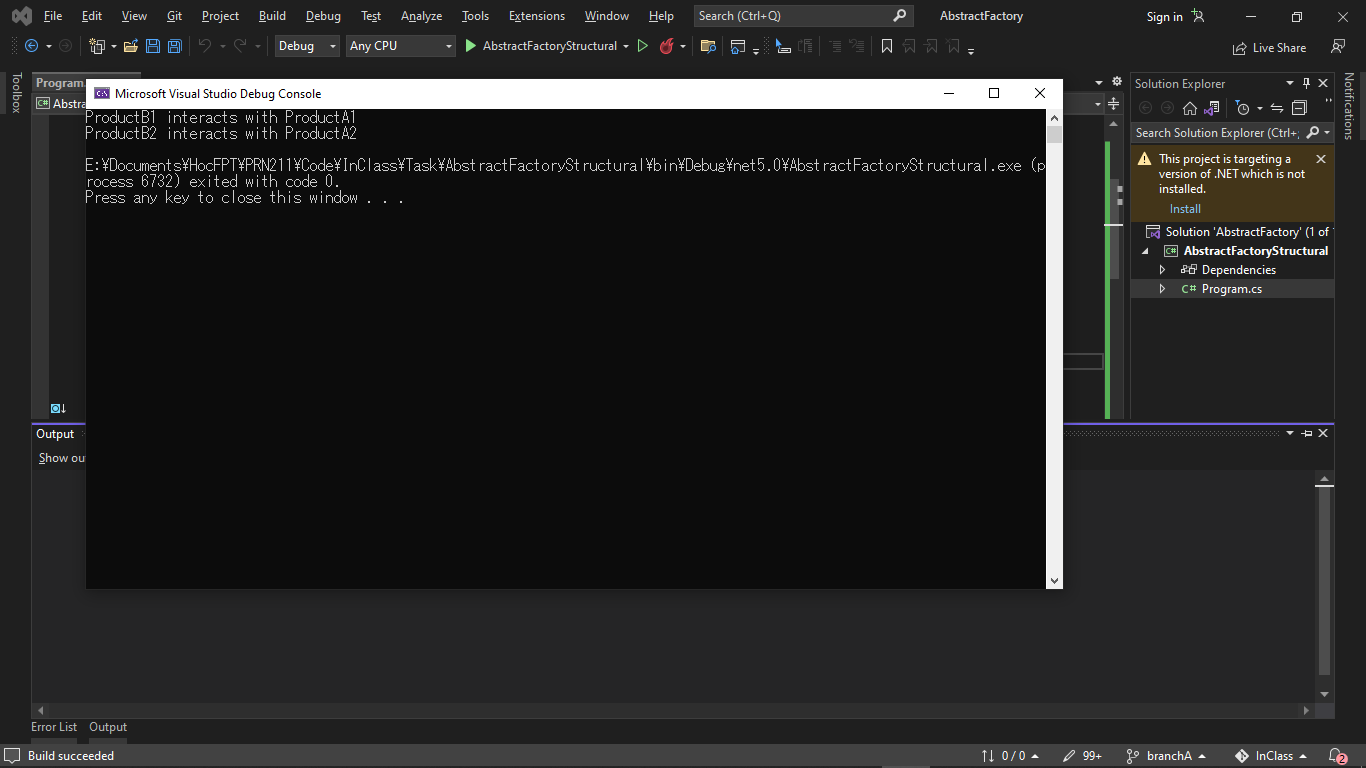
* Abstract Factory class is created to declare an interface for method to create abstract product
* Concrete Factory ConcreteFactory1 and ConcreteFactory2 implement method of abstract factory to create concrete product.



* Client class only need to use AbstractFactory and AbstructProduct to create product without needing the detail concrete factory and product.



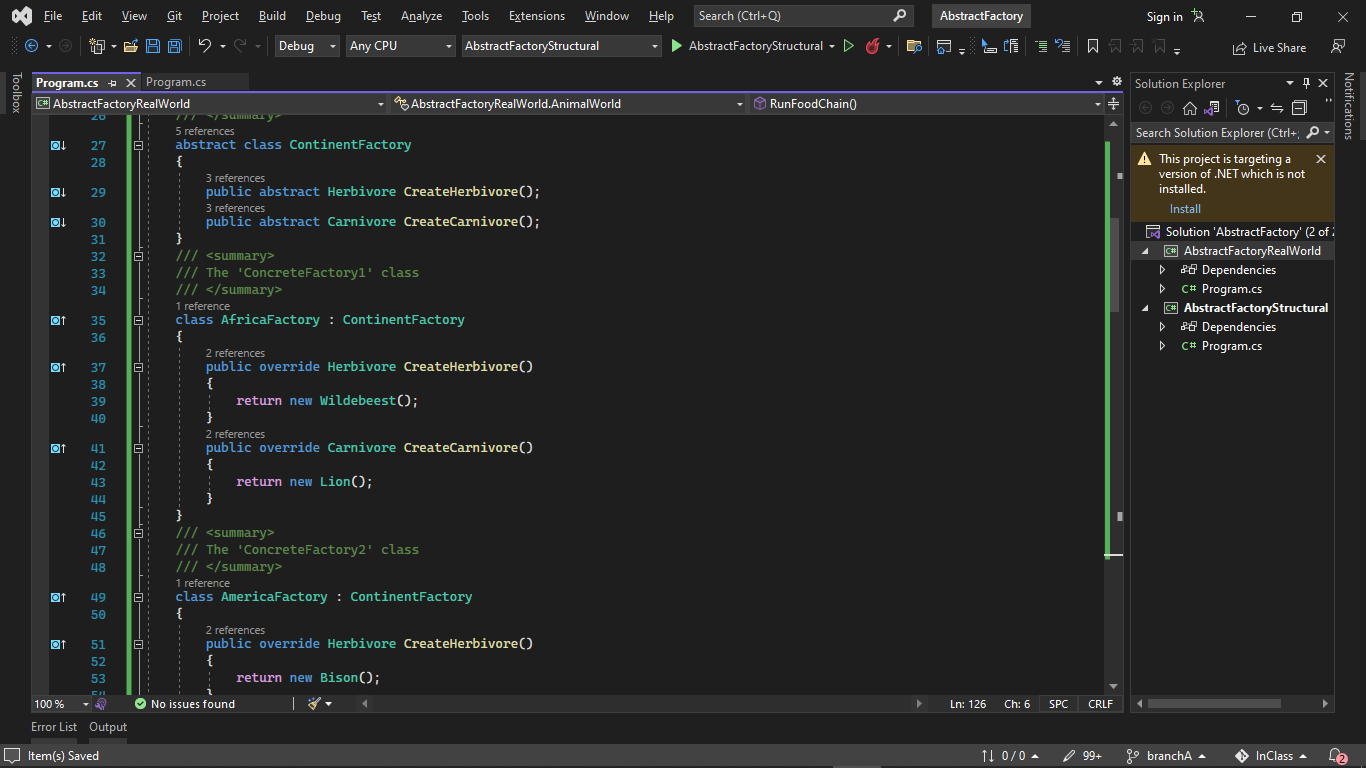
* Main program to test: factory1 and 2 in turn is concrete factory and was cast into Abstract factory object.
* Client only need to call these abstractFactory object in order to create product.

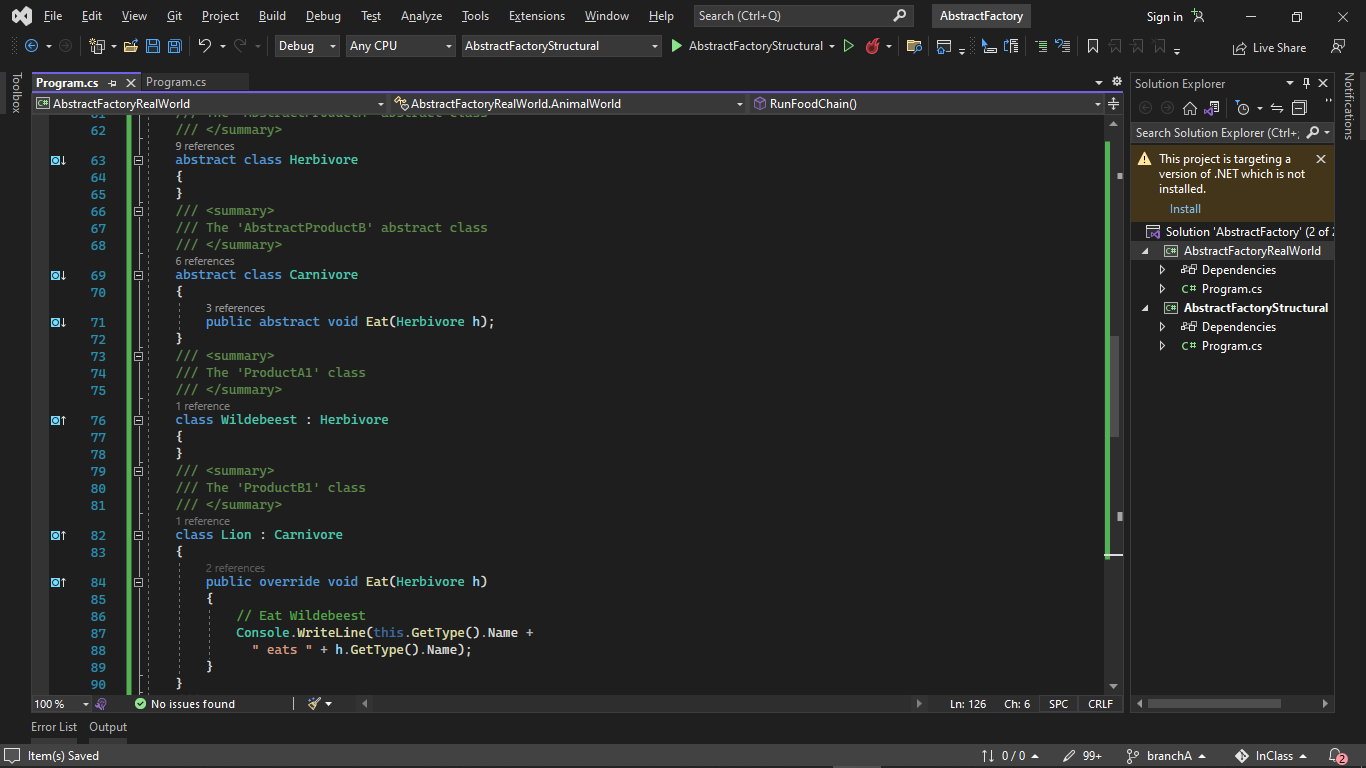


* Result show ProductB1 and ProductA1 was created by concreteFactory1
* Result show ProductB2 and ProductA2 was created by concreteFactory2
* It shows Abstract Factory pattern creating parallel hierarchies of objects. Object creation has been abstracted and there is no need for hard-coded class names in the client code. Client can create object and use them with Abstract Factory without needing to know or call all the detail in each concrete factory or concrete product.

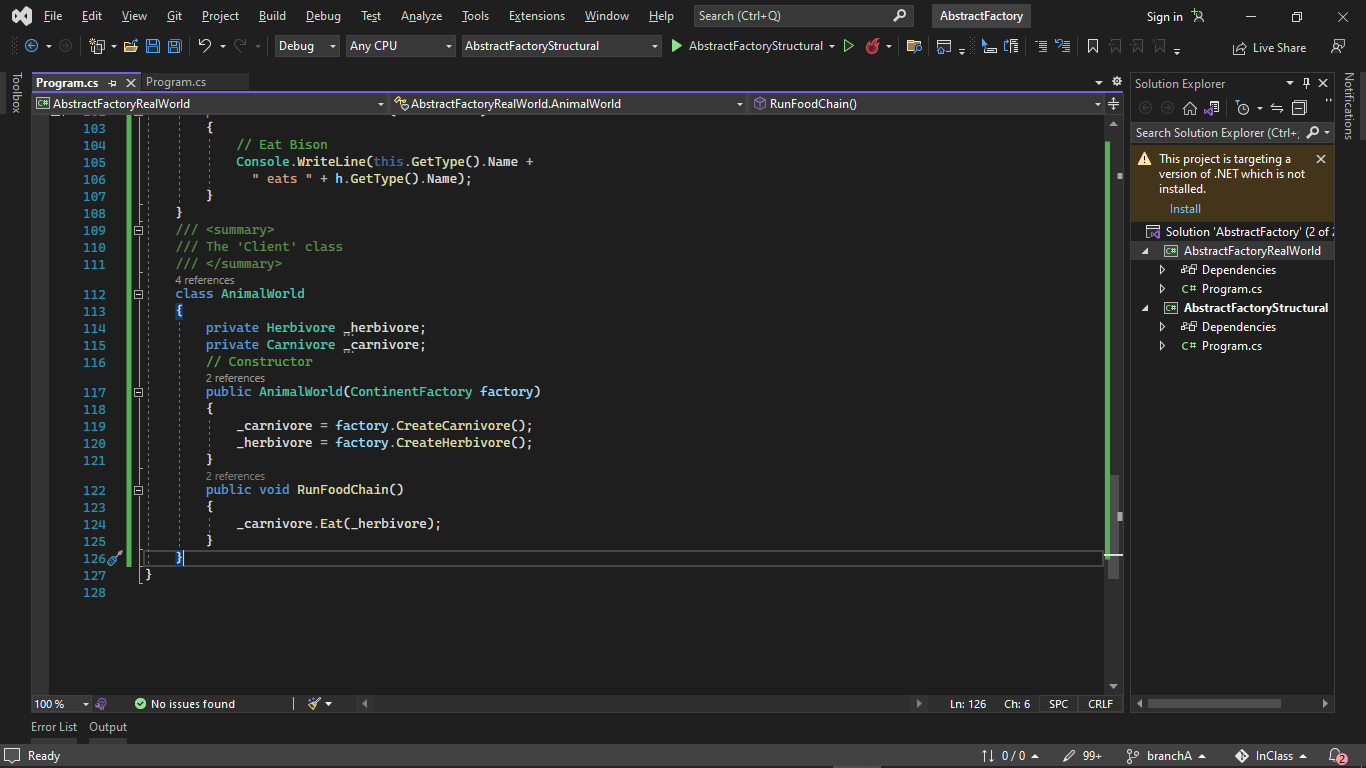
**II – Real application**

1. **Abstract Factory real-world code**

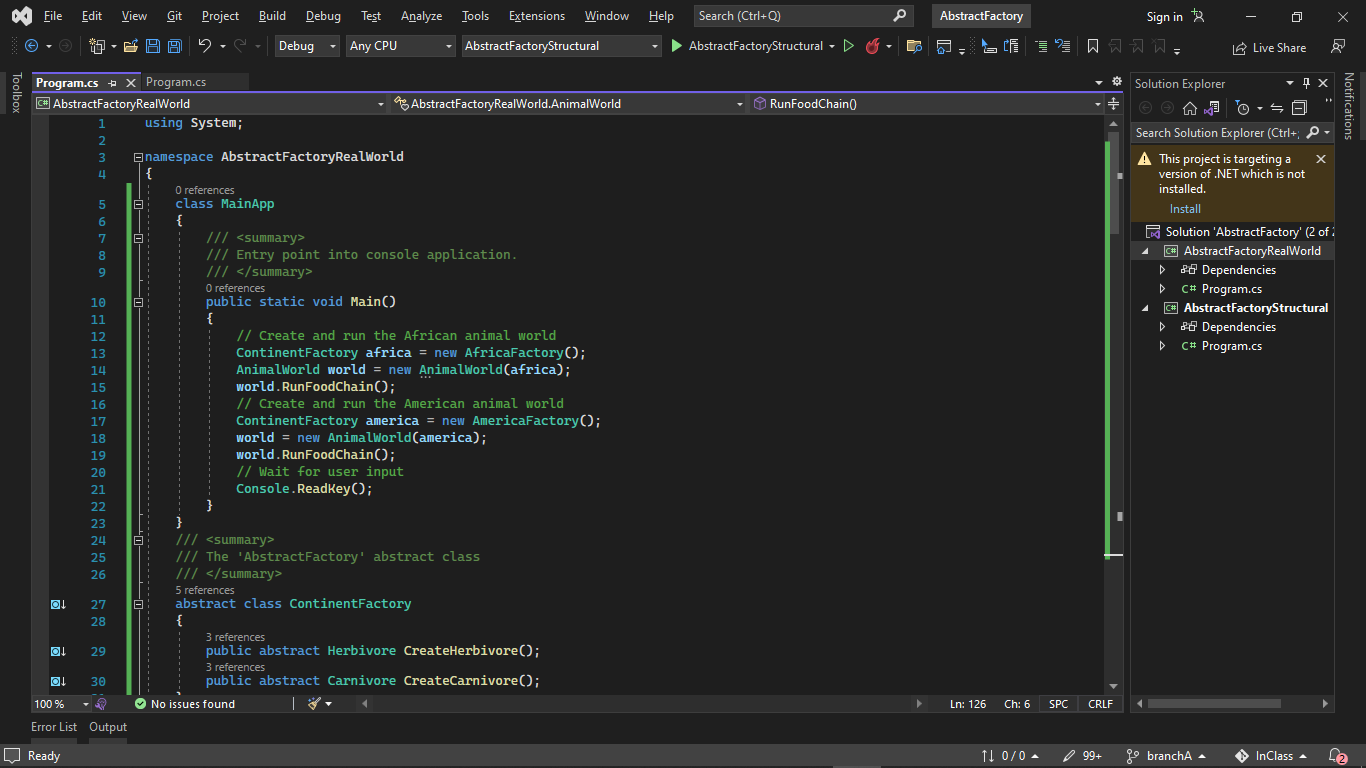
* 
* ContinentFactory is the AbstractFactory and define method to create Object Herbivore and Carnivore
* AfricaFactory and AmericaFactory is concrete factory to create specific Herbivore and Carnivore

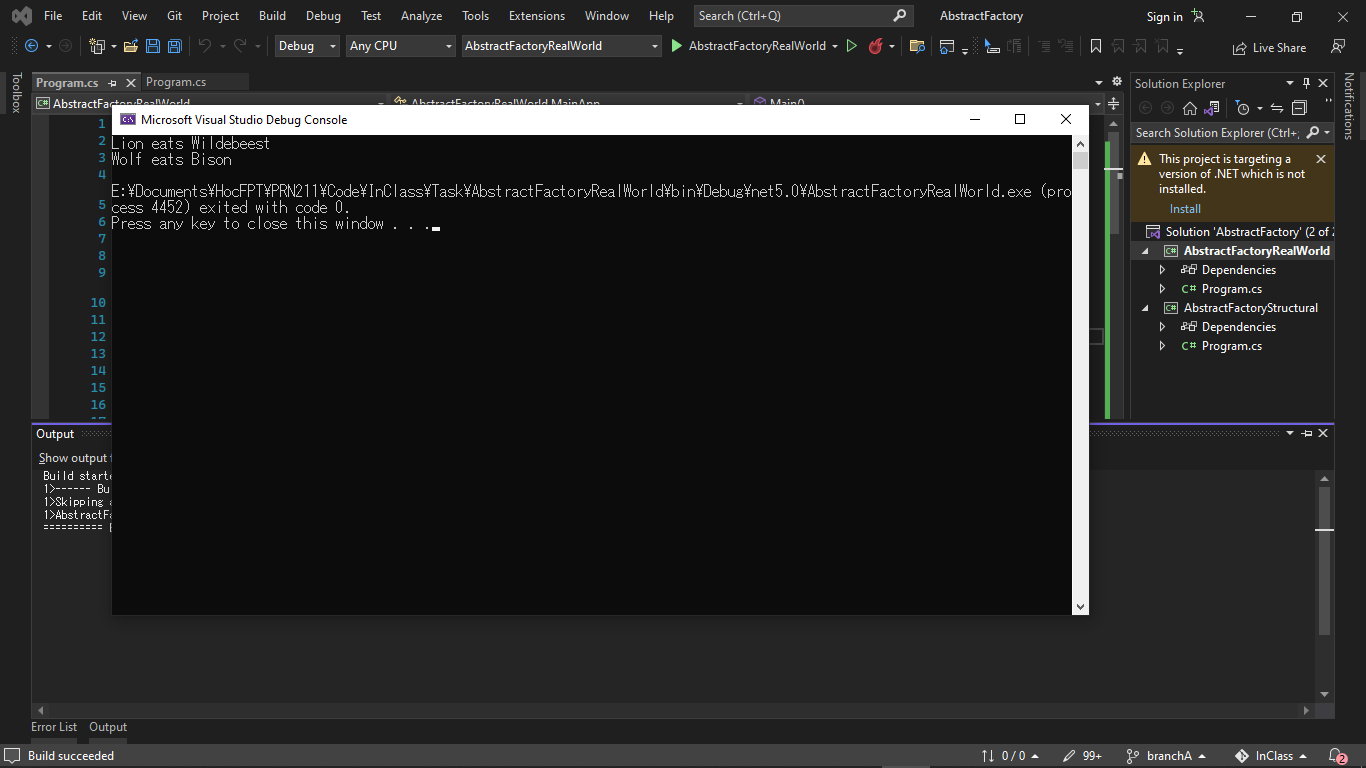
- Herbivore and Carnivore are Abstract Product to declare interface for the type of object and their interaction

- Wildbeast, Lion, Bison and Wolf inherit and override the abstract product and their implement all the method that define their interaction.



* AnimalWorld act as Client and use AbstractProduct Herbivore and Carnivore and AbstractFactory without needing the detail of concrete product and factory.



* Main program to test:
* Client calling 2 AbstractFactory which was created as Concrete Factory then Casted into Abstract Factory.
* 
* Result show that only by calling the abstract factory that contain the concrete factory, client are able to create different animal, but their interaction are still kept the same as expected.
* This  code demonstrates the creation of different animal using different factories. Although the animals created by the Continent factories are different, the interactions among the animals remain the same.

**III – Conclusion:**

* Abstract Factory is a creational design pattern, which solves the problem of **creating entire product families without specifying their concrete classes**. Abstract Factory defines an interface for creating all distinct products but leaves the actual product creation to concrete factory classes.
* Abstract Factory and Factory method difference:

**+ factory pattern provides a method of creating objects without specifying the exact class used to create it => Hide construction of a single object**

**+ the abstract factory pattern provides a method to combine a group of individual factories without specifying their concrete classes. => Hide construction of a family of related objects**