# Generics Lab

Perform these labs on your own computer using Visual Studio 2022 to ensure you understand the lessons presented in the corresponding videos and lectures.

## Lab 1: Organize Files in Project

Your OOPLab project is getting quite a few files in it. It is a good idea to start organizing these files into folders. Create the following folders on the OOPLab project.

- EntityClasses
- EventArgsClasses
- HelperClasses
- Interfaces

Move the appropriate classes into the appropriate folders.

**NOTE**: When prompted if you want to adjust the namespace for the class being moved, answer **No**.

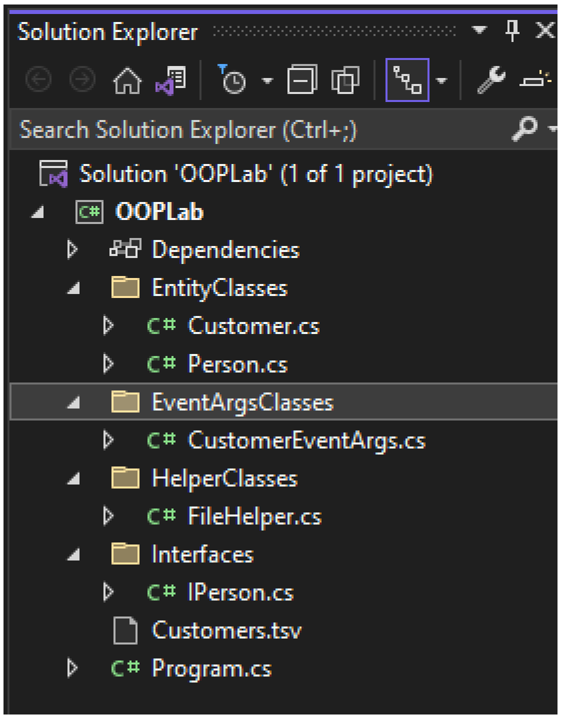
Customer.cs and Person.cs into **EntityClasses**.

CustomerEventArgs.cs into **EventArgsClasses**.

FileHelper.cs into **HelperClasses**.

IPerson.cs into **Interfaces**.

Your solution explorer should now look like the following:



```
Solution 'OOPLab' (1 of 1 project)
  OOPLab
    Dependencies
    EntityClasses
      Customer.cs
      Person.cs
    EventArgsClasses
      CustomerEventArgs.cs
    HelperClasses
      FileHelper.cs
    Interfaces
      IPerson.cs
    Customers.tsv
    Program.cs
```

## Lab 2: Create Repository Interface

Right mouse-click on the **Interfaces** folder and add a new file named **IRepository**.

```
namespace OOPLab;

public interface IRepository<TEntity>
{
  TEntity Insert(TEntity entity);
  TEntity Update(TEntity entity);
  bool Delete(int id);
}
```

## Lab 3: Create Customer Repository

You are now going to separate the properties in the Customer class from some of the methods. You are going to move some of the methods from the Customer class to a CustomerRepository class.

Right mouse-click on the OOPLab project and add a new folder named **RepositoryClasses**.

Right mouse-click on the RepositoryClasses folder and add a new class named **CustomerRepository**.

Make the code in the CustomerRepository class look like the following

```
namespace OOPLab;

public class CustomerRepository : IRepository<Customer>
{

}
```

You will have a red squiggly under the IRepository<Customer>, this is fine, we are going to fix this.

Open the **Customer.cs** file and cut the Save(), Insert() and Update() methods and paste them into the **CustomerRepository** class.

Remove the line of code from the Save() method that sets the **LastModified** property.

Add a new method named Delete() to the CustomerRepository class

```
public bool Delete(int id) {
  return true;
}
```

## Compile the Project

If you now compile the code, everything should still compile. This is because the **CustomerRepository** class implements the three interface methods defined in the **IRepository** interface.

## Move More Code to Customer Repository

Open the **Customer.cs** file and cut the GetCustomers() and the ProcessCustomerLines() methods and paste them into the **CustomerRepository** class.

Also cut the public event **CustomerProcessed** from the Customer class and paste it into the CustomerRepository class.

## Try it Out

Open the **Program.cs** file and modify the file to look like the following:

```
using OOPLab;

CustomerRepository repo = new();
Customer entity = new();

repo.CustomerProcessed += Repo_CustomerProcessed;

void Repo_CustomerProcessed(object sender,
CustomerEventArgs e) {
  Console.WriteLine(e.CustomerObject);
}

Customer[] customers =
  repo.GetCustomers(Directory.GetCurrentDirectory()
    + "\\Customers.tsv");
```

Run the application and view the output. It should still list all the customers read from the Customers.tsv file.

You now have a nice separation of data and processes. The Customer class contains just properties and those methods that operate on those properties. The CustomerRepository class contains methods that works on Customer class(es).

# Lab 4: Create Person Repository

You are now going to separate the properties in the Person class from some of the methods. You are going to move some of the methods from the Person class to a PersonRepository class.

Right mouse-click on the RepositoryClasses folder and add a new class named **PersonRepository**.

Make the code in the PersonRepository class look like the following

```
namespace OOPLab;

public class PersonRepository : IRepository<Person> {

}
```

Open the **Person.cs** file and cut the Save(), Insert() and Update() methods and paste them into the **PersonRepository** class.

Remove the line of code from the Save() method that sets the **LastModified** property.

Add a new method named Delete() to the PersonRepository class

```
public bool Delete(int id) {
  return true;
}
```

Open the **IPerson.cs** interface file and delete the **Insert()** and **Update()** methods because these methods are now defined in the IRepository.cs file.

## Compile the Project

If you now compile the code, everything should still compile. This is because the **PersonRepository** class implements the three interface methods defined in the **IRepository** interface.

You now have a nice separation of data and processes. The Person class contains just properties and those methods that operate on those properties. The PersonRepository class contains methods that works on Person class(es).