# Exercises: Implementing an OOP Hierarchy

Test your tasks in the Judge system: https://judge.softuni.org/Contests/4475/OOP-Hierarchy

## 1. Menu Item

We will be making a simple **OOP project** representing a **restaurant**. You'll be able to create **different menu items**, make new **customers** and manage their **orders**. Let's start by going into the **`MenuItem` class:**

```
namespace ExerciseOopHierarchy;

22 references
public abstract class MenuItem...
```

You can observe the class is **abstract**. We will use **abstraction** and **inheritance** to be able to make different items.

The class needs the following:

- **Property Name** – **`string`**.
- **Property `Description`** – **`string`**.
- **Property `Price`** – **`decimal`**.
- **Method override `ToString()`** - **`"{this.Name} - {this.Description} - ${this.Price}"`.**
- **Constructor accepting** the **three properties**.

## 2. AppetizerMenuItem

Create a new class called **`AppetizerMenuItem`** and **inherit** from **`MenuItem`**. Call the **base constructor** and **override** the **`ToString()` method**: **`"Appetizer: {base.ToString()}"`**

## 3. MainCourseMenuItem

Create a new class called **`MainCourseMenuItem`** and **inherit** from **`MenuItem`**. Call the **base constructor** and **override** the **`ToString()` method**: **`"Main Course: {base.ToString()}"`**

## 4. DessertMenuItem

Create a new class called **`DessertMenuItem`** and **inherit** from **`MenuItem`**. Call the **base constructor** and **override** the **`ToString()` method**: **`"Dessert: {base.ToString()}"`**

## 5. Order

Create a new class called **`Order`**. This class will be responsible for **keeping a list of menu items** and being able to **tell us the total all items cost**.

Start by making a **private list of menu items named `_items`**. We will use **encapsulation** to protect the collection from outside use. To still be able to use it create an **`AddItem(MenuItem item)` method** to be able to add an item to the collection from the outside. Using **polymorphism,** we will be able to add **any menu item**:

```
public void AddItem(MenuItem item)
{
    this._items.Add(item);
}
```

We also need the method for getting the total amount, so create a **method `decimal GetTotal()`** and return the **total price of each item in the collection**.

To wrap the class up let's allow **read only access to the collection** by adding a **`IReadOnlyCollection` property**:

```
public IReadOnlyCollection<MenuItem> Items => this._items.AsReadOnly();
```

# 6. Customer

Create a new class called **`Customer`**. The customer will hold information for each of his orders. The class needs:

- **Field `_orderHistory`** – **list** of **orders**.
- **Property `Name`** – **`string`**.
- **Property `Email`** – **`string`**.
- **Property `OrderHistory`** – **read only order collection** from **`_orderHistrory`**.
- **Constructor accepting** the **two properties**.
- **Method `AddOrder(Order order)` –** adds the given **order** to the **`_orderHistory` list.**

# 7. Restaurant

The final class **`Restaurant`** will hold the most logic, combining all classes. Here is what the class will have:

- **Field `_customers`** – **list** of **customers**.
- **Field `_menu`** – **list** of **menu items**.
- **Method `AddCustomer(Customer customer)` –** adds the given **customer** to the **`_customers` list.**
- **Method `GetMenuItem(int index)` – returns** the **menu item** at the **given index.**
  - Check if the index **is in bounds**! If not **throw** an **`IndexOutOfRangeException`**.
- **Method `AddMenuItem(MenuItem item)` –** adds the given **menu item** to the **`_menu` list.**
- **Method `PlaceOrder(Customer customer, Order order)` –** adds the given **order** to the **customers `_orderHistory` list through the method we wrote.**
- **Method `DisplayMenu()` –** First write to the console **`"Menu Items:"`** then **foreach menu item** in **`_menu`** **write the item to the console**.
- **Method `DisplayOrderHistory(Customer customer)` –** First:
  - Write to the console **`"{customer.Name}'s Order History:"`.**
  - Then **foreach order in the customers read only order collection write to the console "`Order Total: ${order.GetTotal()}`"**.
  - Finally **foreach item in the orders items write to the console on each line " `{item}`".**

All these methods allow us to communicate with the collection not only in this class but in others.

Now go ahead and run the code in **`Program.cs`** and look at the result!

```
Menu Items:
Main Course: Pasta - Delicious pasta dish - $12.99
Appetizer: Salad - Fresh garden salad - $7.99
Dessert: Cheesecake - Creamy cheesecake - $5.99
John Doe's Order History:
Order Total: $18.98
  Main Course: Pasta - Delicious pasta dish - $12.99
  Dessert: Cheesecake - Creamy cheesecake - $5.99
```