Programowanie obiektowe – C#

INTERFEJSY

Interfejs to struktura podobna do klasy z tą różnicą, że posiada jedynie deklaracje składowych metod a nie implementuje ich. Nie można utworzyć obiektu, który będzie instancją interfejsu. Klasa natomiast może implementować interfejs. Klasa implementująca interfejs musi posiadać implementacje wszystkich składników interfejsu. Oznacza to, że interfejs mówi klasie, co ma zawierać (jakie metody i właściwości). Co bardzo ważne klasa może dziedziczyć tylko po jednej klasie, ale może implementować wiele interfejsów naraz. Interfejs może być interfejsem rozszerzającym dla wielu innych interfejsów, jednak żaden interfejs rozszerzający nie może być bardziej dostępny od interfejsu bazowego. Klasa implementująca interfejs może być bardziej dostępna niż interfejs.

```
modyfikatorDostepu interface INazwaInterfejsu [: listaBazowa]
{
    składniki interfejsu;
}
```

Domyślnie wszystkie modyfikatory składników interfejsu są publiczne.

```
public interface IHasCreationTime
{
    DateTime CreationTime
    {
        get;
        set;
    }
}
public interface IDataRepository
{
    List<int> GetData();
}
```

Interfejs rozszerzający:

```
public interface ICreationAudited : IHasCreationTime
{
    int? CreatorUserId
    {
        get;
        set;
    }
}
```

Klasa może dziedziczyć po wielu interfejsach

```
public class Ticket : IHasCreationTime, IHasModificationTime, IHasDeletionTime
{
    public DateTime CreationTime { get; set; }
    public DateTime? DeletionTime { get; set; }
    public bool IsDeleted { get; set; }
    public DateTime? LastModificationTime { get; set; }
}
```

Wykorzystanie:

```
IHasCreationTime item = new IHasCreationTime(); //BŁĄD
IHasCreationTime item2 = new FilmShow();
```

Interfejs nie może zawierać:

- stałych,
- pól,
- operatorów,
- konstruktorów i destruktorów,
- zagnieżdżonych typów.

Najpopularniejsze wykorzystanie interfejsów.

Obudowanie klas oraz metody uniwersalne

```
public class Ticket : ICreationAudited
{
    public int? CreatorUserId
    {
        get;
        set;
    }
    public DateTime CreationTime
    {
        get;
        set;
    }
    public string Name { get; set; }
}
public class FilmShow : ICreationAudited
{
    public int? CreatorUserId
    {
        get;
        set;
    }
    public DateTime CreationTime
    {
        get;
        set;
    }
}
public static void SetCreationUser(ICreationAudited creationAudited)
{
    creationAudited.CreatorUserId = Session.GetUserId();
}
```

Wykorzystanie

```
var filmShow = new FilmShow();
SetCreationUser(filmShow);
var ticket = new Ticket();
SetCreationUser(ticket);
```

Repozytoria – pobieranie danych

```
public interface ITicketsRepository
{
    List<Ticket> GetAllTickets();
    void AddNewTicket(Ticket item);
}
public class TicketsRepository : ITicketsRepository
{
    public void AddNewTicket(Ticket item)
    {
```

```
        //instrukcje dodawania do bazy
    }
    public List<Ticket> GetAllTickets()
    {
        //instrukcja pobierania z bazy danych
        throw new NotImplementedException();
    }
}
```

Testowy interfejs:

```
public class MockTicketsRepository : ITicketsRepository
{
    private List<Ticket> list;
    public MockTicketsRepository()
    {
        list = new List<Ticket>()
{
new Ticket() { Name = "Item1" },
new Ticket() { Name = "Item2" },
new Ticket() { Name = "Item3" }
};
    }
    public void AddNewTicket(Ticket item)
    {
        list.Add(item);
    }
    public List<Ticket> GetAllTickets()
    {
        return list;
    }
}
```

Wykorzystanie:

```
ITicketsRepository ticketsRepository;
#if DEBUG
ticketsRepository = new MockTicketsRepository();
#else
ticketsRepository = new TicketsRepository();
#endif
var ticketsList = ticketsRepository.GetAllTickets();
var ticket = new Ticket();
SetCreationUser(ticket);
ticketsRepository.AddNewTicket(ticket);
```

Zadania do samodzielnego wykonania:

1) Zdefiniuj interfejs IOsoba. Powinien on nakazywać implementację właściwości Imię, Nazwisko oraz metodę ZwrocPelnaNazwe. Następnie stwórz klasę Osoba dziedziczącą po tym interfejsie i implementującą go. Stwórz kilka egzemplarzy tej klasy dodaj je do listy List<Osoba>.

2) Napisz metodę rozszerzającą List<IOsoba> void WypiszOsoby(), a następnie zaimplementuj ją. (wypisanie imię i nazwisko osób na konsole)

3) Napisz metodę rozszerzającą List<IOsoba> void PosortujOsobyPoNazwisku(), a następnie zaimplementuj ją.

4) Napisz interfejs IStudent, roszerzający interfejs IOsoba o właściwości Uczelnia, Kierunek, Rok, Semestr. Następnie stwórz klasę Student implementującą interfejs IStudent oraz zawierającą

dodatkową metodę: string WypiszPelnaNazweIUczelnie(), która wypisze pełną informację o studencie np.: Jan Kowalski – 2INF 2021 UR

5) Napisz i zaimplementuj klasę StudentUR, dziedziczącą po klasie Student. Stwórz kilka egzemplarzy tej klasy dodaj je do listy, a następnie przeciąż metodę z zadania 2, by pozwalała korzystać z metody WypiszPelnaNazweIUczelnie().