Технології .Net

Лекція 1.
Делегати. Події

Ігор Жовтяк,
Київ, 2021

Визначення делегата

- Тип **Делегат –** це клас, об’єкт якого вказує на методи.

Коли говорять просто делегат, мають на увазі саме об’єкт.!!!

- Методи, які адресує делегат повинні мати параметри і тип результату вказані при його визначенні.
- Делегат можна передавати як параметр, а потім викликати інкапсульований в нього метод.

[атрибути] [специфікатори] delegate тип им’я([параметри])

```
delegate int Operation(int x, int y);
delegate void GetMessage();
```

- Базовим класом делегата є клас System.Delegate

Приклад 1

```
class Program {
 delegate void GetMessage();
 static void Main(string[] args)  {
  GetMessage del;

  if (DateTime.Now.Hour < 12) del = GoodMorning;
  else                        del = GoodEvening;
  del.Invoke();
}
  private static void GoodMorning(){
      Console.WriteLine("Good Morning");
  }
  private static void GoodEvening(){
      Console.WriteLine("Good Evening");
  }}
```

Приклад 2.

```
class Program {
  delegate int Operation(int x, int y);

static void Main(string[] args)
  {
        Operation del = new Operation(Add);
        int result = del.Invoke(4, 5); //або result=del(4, 5);

        Console.WriteLine(result);

        del = Multiply;
        result = del.Invoke(3, 8); //або result=del(3, 8);
        Console.WriteLine(result);
   }
   private static int Add(int x, int y) {   return x + y;    }
   private static int Multiply(int x, int y) { return x * y; }
}
```

Приклад 3. Делегати, як параметри

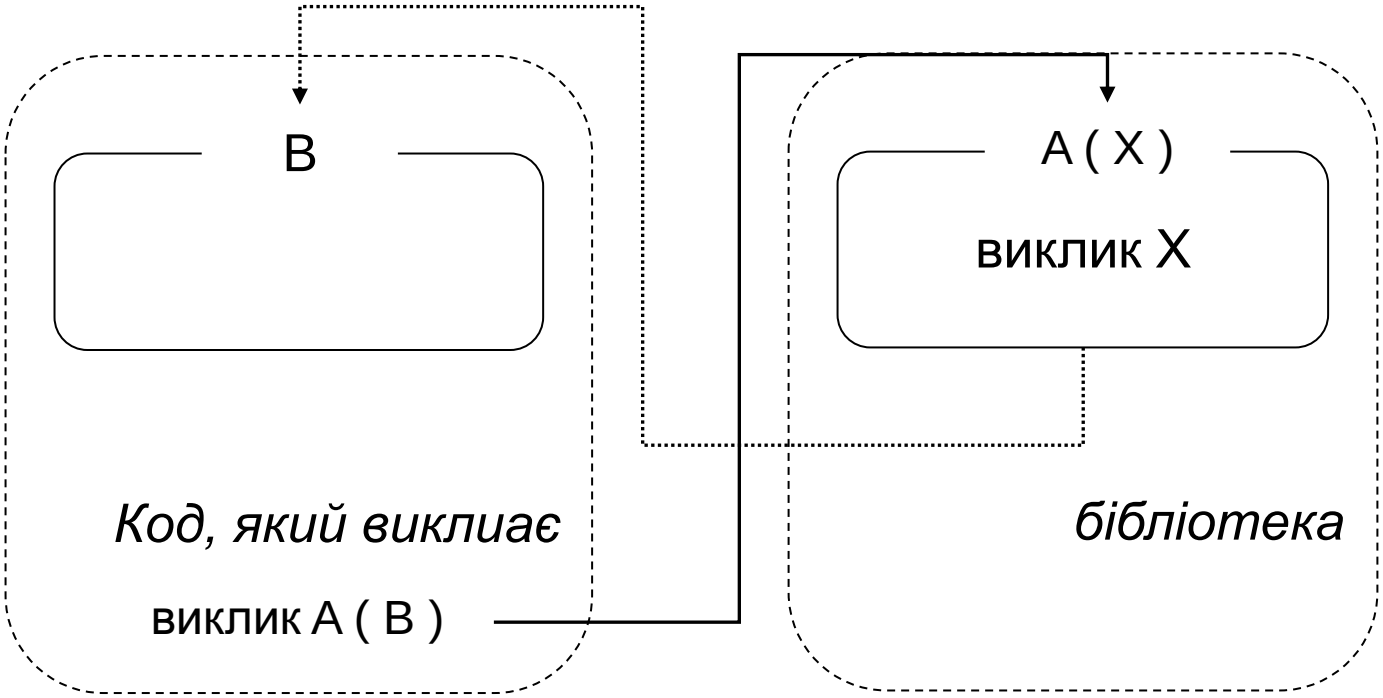
```
class Program
{
   delegate void GetMessage();
   static void Main(string[] args)
   {
    if (DateTime.Now.Hour < 12)    ShowMessage(GoodMorning);
    else                           ShowMessage(GoodEvening);
    }
  private static void ShowMessage(GetMessage fun) {
        fun.Invoke();
    }
   private static void GoodMorning() {
        Console.WriteLine("Good Morning"); }

   private static void GoodEvening() {
        Console.WriteLine("Good Evening");  }
}
```

Використання делегатів

Делегати застосовуються:

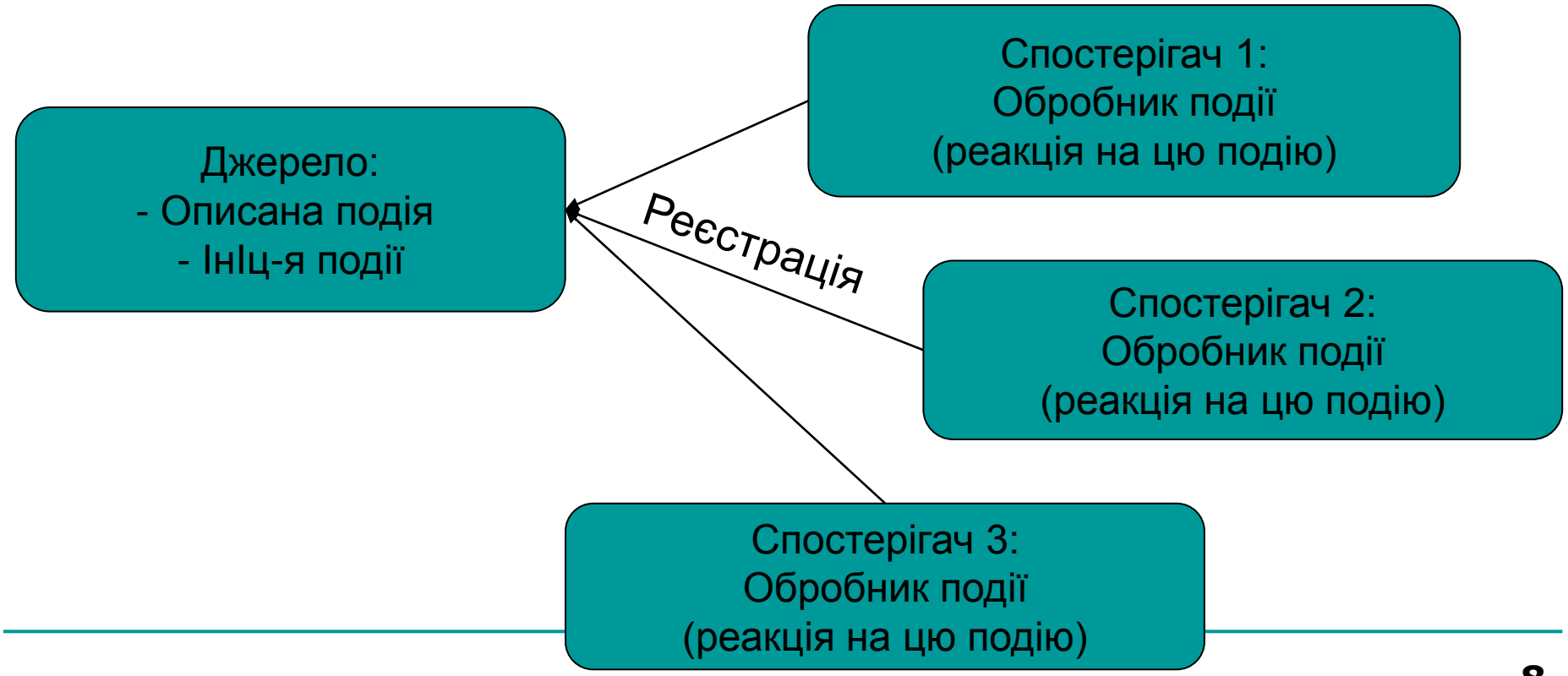
- щоб вказувати метод не під час компіляції, а динамічно, під час виконання програми;
- для забезпечення зв'язку між об'єктами по типу «джерело - спостерігач»;
- для створення універсальних методів, в які можна передавати інші методи (підтримки механізму зворотних викликів).

Обернений виклик (callback)



Події

- *Подія - елемент класу, що дозволяє його об’єкту (джерелу) надсилати іншим об'єктам (спостерігачам) повідомлення про зміну свого стану.*
- *Спостерігач повинен містити обробник події зареєстровний в об'єкті-джерелі*



Приклад.
Реалізувати одну з функцій роботи банкомату. Користувач знімає з рахунку певну суму коштів. Після їх зняття необхідно виводити повідомлення про операцію

```
class Account
 {
    int  sum;
    public Account(int _sum) {   sum = _sum; }
    public int  Sum {  get { return sum; } }
    public void Put(int _sum) {  sum += sum; }
    public void Withdraw(int _sum) {
       if (_sum <= sum)
       {
          sum -= _sum;
       }
    }
 }
```

Вважаємо, що після зняття грошей необхідно виводити повідомлення про операцію

```
class Account
 {
    public delegate void AccountStateHandler(string message);
    AccountStateHandler fun;

    public void RegisterHandler(AccountStateHandler _fun)
     {
         fun = _fun;
     }
```

```
public void Withdraw(int _sum)
    {
        if (_sum <= sum)
        {
            sum -= _sum;
            if (fun != null)
                    fun($"Суму {sum} знято з рахунку");
        }
        else
        {
            if (fun != null)
                    fun("Недостатньо коштів на рахунку");
        }
    }
```

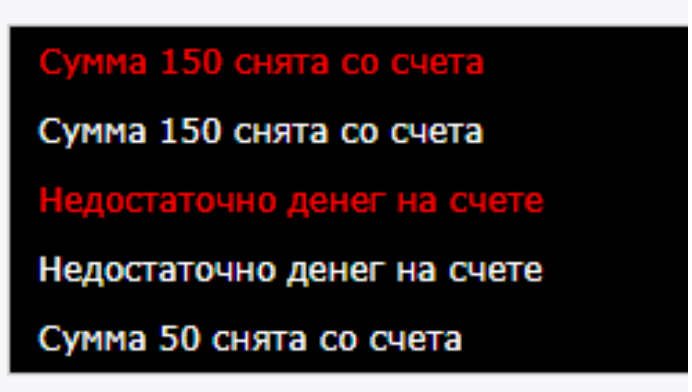
```
class Program {
    static void Main(string[] args) {
        Account account = new Account(200);

        var stateHandler  =
            new Account.AccountStateHandler(ShowMessage);

        account.RegisterHandler(stateHandler);


        account.Withdraw(100);
        account.Withdraw(150);
        Console.ReadLine();
    }
    private static void ShowMessage(string message) {
        Console.WriteLine(message);  }
 }
```

```
Сумма 100 снята со счета
Недостаточно денег на счете
```

Тепер змінимо та доповнимо клас

```
public void RegisterHandler(AccountStateHandler _fun)
  {
     Delegate mainDel = System.Delegate.Combine(fun,_fun);
      fun = mainDel as AccountStateHandler;
  }
  public void UnregisterHandler(AccountStateHandler del)
  {
     Delegate mainDel = System.Delegate.Remove(fun, del);
      fun = mainDel as AccountStateHandler;
  }
```

```
class Program{
    static void Main(string[] args)   {
        Account account = new Account(200);
        Account.AccountStateHandler colorDelegate =
                    new Account.AccountStateHandler(Color_Message);
        account.RegisterHandler(
                    new Account.AccountStateHandler(Show_Message));

          account.RegisterHandler(colorDelegate);
        account.Withdraw(100);
        account.Withdraw(150);
        account.UnregisterHandler(colorDelegate);
        account.Withdraw(50);
        Console.ReadLine();
    }
    private static void Show_Message(String message) {
        Console.WriteLine(message);
    }
    private static void Color_Message(string message) {
        Console.ForegroundColor = ConsoleColor.Red;
        Console.WriteLine(message);
        Console.ResetColor();       }}
```



Можна використовувати скорочену форму додавання та видалення методу з делегату:

```
public void RegisterHandler(AccountStateHandler del)
{
    _del += del;
}
public void UnregisterHandler(AccountStateHandler del)
{
    _del -= del;
}
```