

Модуль 6: Принципы объектно- ориентированного программирования

Обзор

- **Классы и объекты**
- **Использование инкапсуляции**
- **С# и объектно-ориентированное программирование**
- **Разработка объектно-ориентированных систем**

◆ Классы и объекты

- Что такое класс?
- Что такое объект?
- Сравнение классов со структурами

Что такое класс?

■ С точки зрения философа...

- Искусственное понятие, используемое людьми для классификации
- Классификация основывается на общности свойств или общности в поведении
- Согласованность в описании и именовании используемых классов
- Для создания словаря; в общении; когда думаем.

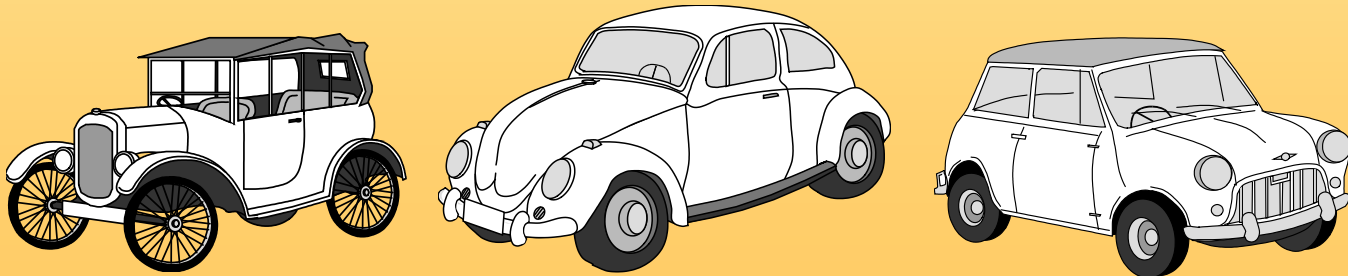
■ С точки зрения ООП-программиста...

- Именованная синтаксическая конструкция, описывающая общие свойства и общее поведение
- Структура данных, содержащая как данные, так и функции



Что такое объект?

- **Объект – это экземпляр класса**
- **Характеристики объекта:**
 - **Индивидуальность:** Объекты должны отличаться друг от друга
 - **Поведение:** объекты могут выполнять задачи
 - **Состояние:** Объекты хранят информацию



Синтаксис класса

■ Синтаксис описания класса:

```
[атрибуты] [спецификаторы] class имя[: список_родителей]
{тело_класса}
```

№	Спецификатор	Описание
1	new	Задаёт новое описание класса взамен унаследованного от предка. Используется для вложения классов (в иерархии объектов).
2	public	Доступ к классу не ограничен
3	protected	Доступ только из данного или производного класса. Используется для вложенных классов.
4	internal	Доступ только из данной программы (сборки).
5	protected internal	Доступ только из данного и производного класса и из данной программы (сборки).
6	private	Доступ только из элементов класса, внутри которых описан данный класс. Используется для вложенных классов.
7	static	Статический класс. Позволяет обращаться к методам класса без создания экземпляра класса
8	sealed	Бесплодный класс. Запрещает наследование данного класса. Применяется в иерархии объектов.
9	abstract	Абстрактный класс. Применяется в иерархии объектов.

Состав класса

- Класс можно описывать непосредственно внутри пространства имен или внутри другого класса.

В теле класса могут быть объявлены:

- константы;
- поля;
- конструкторы и деструкторы;
- методы;
- события;
- делегаты;
- классы (структуры, интерфейсы, перечисления).

Сравнение классов со структурами

■ Структуры определяют размерные типы данных

- Отсутствие уникальности, доступность состояния, отсутствие дополнительной функциональности

■ Классы определяют объекты

- Уникальность, скрытое состояние, дополнительная функциональность

```
struct Time
{
    public int hour;
    public int minute;
}
```

```
class BankAccount
{
    ...
    ...
}
```


◆ Использование инкапсуляции

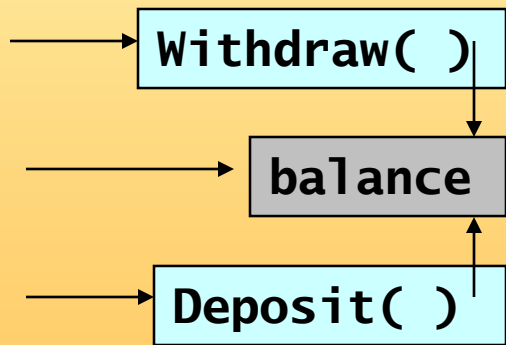
- Объединение данных и методов
- Контроль видимости доступа
- Зачем нужна инкапсуляция?
- Объектные данные
- Использование статических данных
- Использование статических методов

Объединение данных и методов

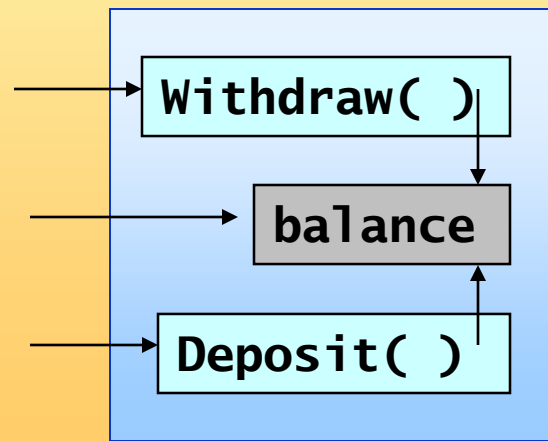
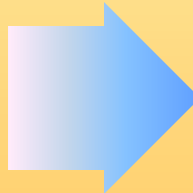
- Объединяйте данные и методы в одной *капсуле*
- Граница капсулы определяет внешнее и внутреннее

Синтаксис описания элемента данных:

[атрибуты] [спецификаторы] [const] тип имя [= начальное_значение]



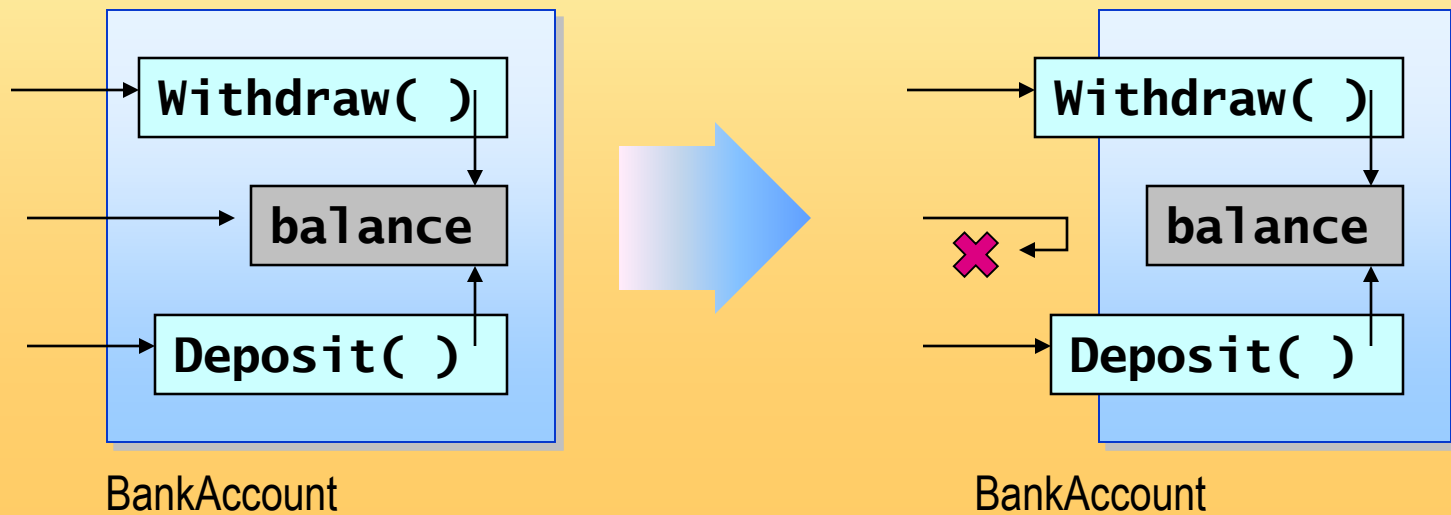
BankAccount



BankAccount

Контроль видимости доступа

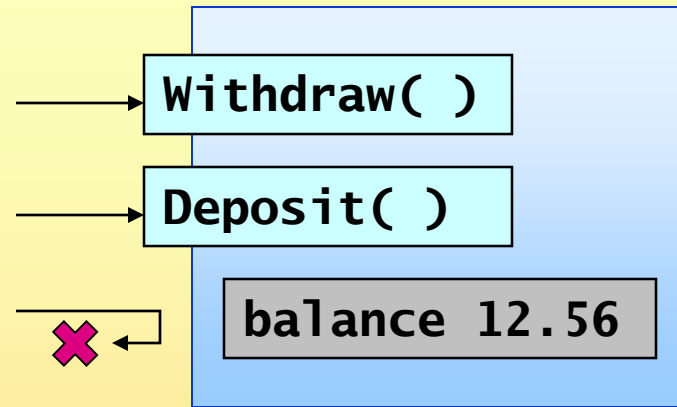
- Методы определяются как *public* и доступны извне
- Данные определяются как *private* и доступны только изнутри



Зачем нужна инкапсуляция?

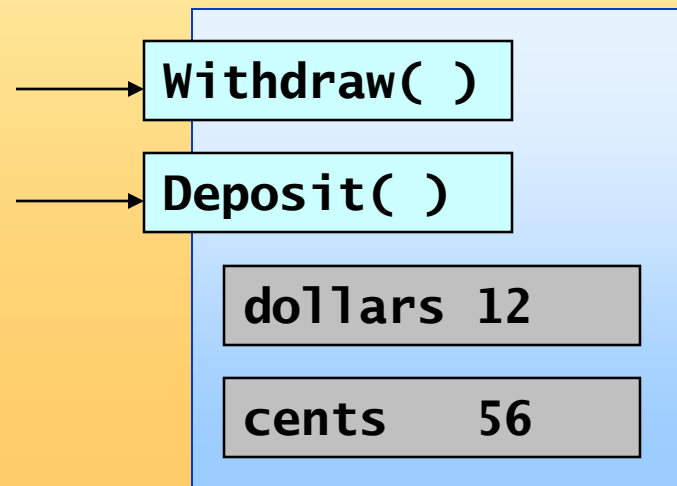
■ Можно контролировать

- Объект используется только через public-методы



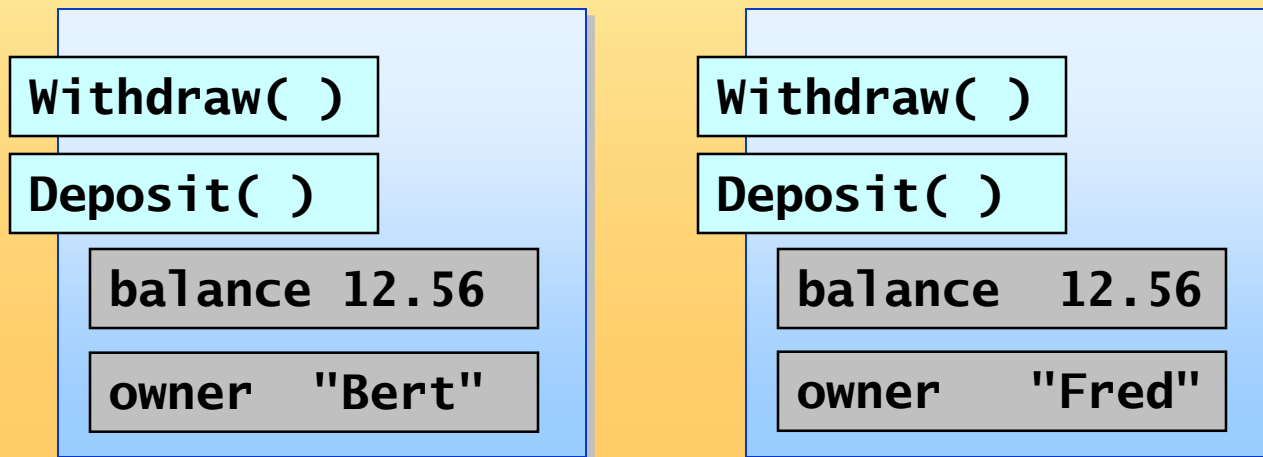
■ Можно изменять

- На использование объекта не влияют возможные изменения типов скрытых данных



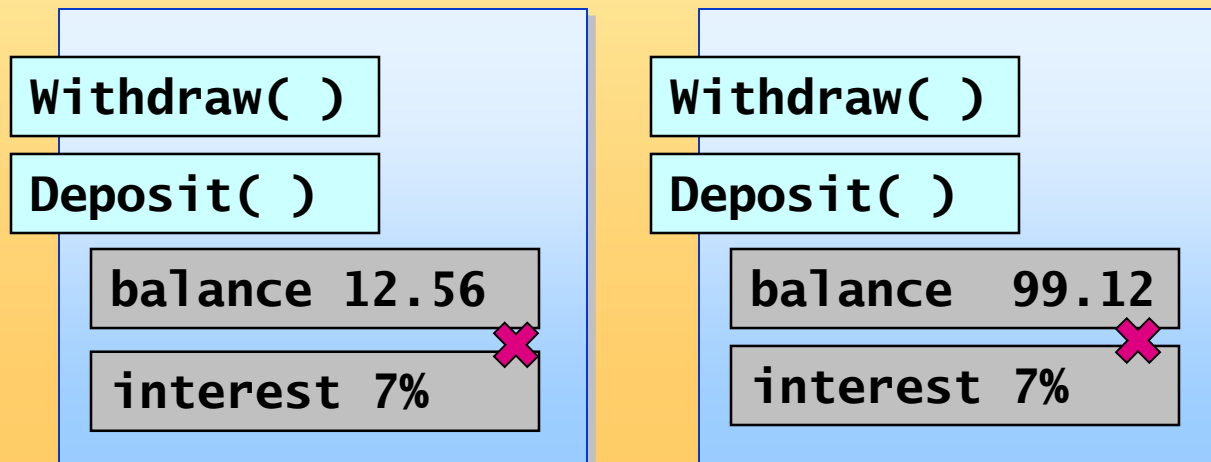
Объектные данные

- Объектные данные содержат индивидуальную информацию об объекте
 - Например, каждый банковский счет содержит свой баланс. Если два банковских счета имеют одинаковый баланс, то это всего лишь совпадение



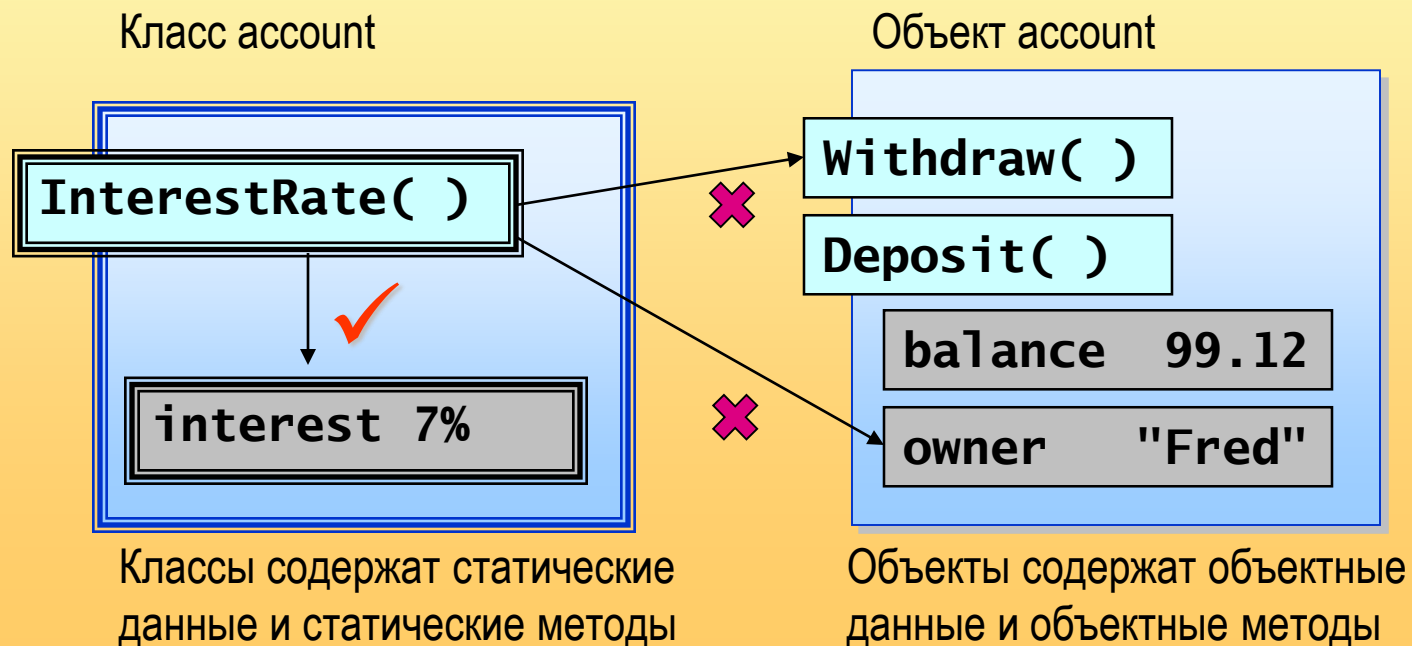
Использование статических данных

- Статические данные содержат информацию обо всех объектах класса
 - Например, предположим, что все счета имеет одинаковую процентную ставку. Хранение этого значения в каждом счете нецелесообразно. Почему?



Использование статических методов

- Статические методы имеют доступ только к статическим данным
 - Статический метод применяется на уровне класса, а не объекта



◆ С# и ООП

- Hello, World
- Определение простых классов
- Создание новых экземпляров объектов
- Использование ключевого слова `this`
- Создание вложенных классов
- Доступ к вложенным классам

Hello, World

```
using System;

class Hello
{
    public static int Main( )
    {
        Console.WriteLine("Hello, World");
        return 0;
    }
}
```

Определение простых классов

- Данные и методы вместе внутри класса
- Методы `public`, данные `private`

```
class BankAccount
{
    public void Withdraw(decimal amount)
    { ... }
    public void Deposit(decimal amount)
    { ... }
    private decimal balance;
    private string name;
}
```

Public методы
описывают
доступное
поведение

Private поля
описывают
недоступное
состояние

Создание новых экземпляров объектов

- При объявлении переменной класса объект не создается
 - Для создания объекта используйте оператор **new**

```
class Program
```

```
{
```

```
    static void Main( )
```

```
    {
```

```
        Time now;
```

```
        now.hour = 11;
```

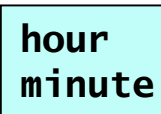
```
        BankAccount yours = new BankAccount( );
```

```
        yours.Deposit(999999M);
```

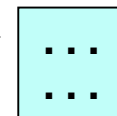
```
    }
```

```
}
```

now



yours



новый
BankAccount
объект

Использование ключевого слова **this**

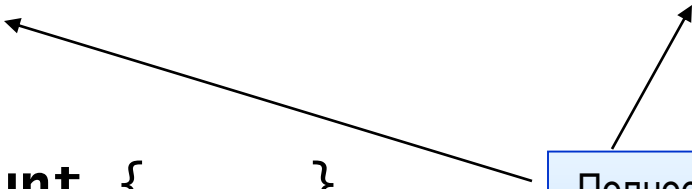
- Ключевое слово **this** ссылается на объект, для которого вызывается метод
- Используется для избежания конфликтов между именами

```
class BankAccount
{
    ...
    public void SetName(string name)
    {
        this.name = name;
    }
    private string name;
}
```

Создание вложенных классов

- Классы можно вкладывать в другие классы

```
class Program
{
    static void Main( )
    {
        Bank.Account yours = new Bank.Account( );
    }
}
class Bank
{
    ... class Account { ... }
}
```



Полное имя вложенного
класса включает в себя
имя внешнего класса

Доступ к вложенным классам

- Вложенные классы также могут объявляться как **public** или **private**

```
class Bank
{
    public class Account { ... }
    private class AccountNumberGenerator { ... }
}
class Program
{
    static void Main( )
    {
        Bank.Account                accessible; ✓
        Bank.AccountNumberGenerator inaccessible; ✗
    }
}
```