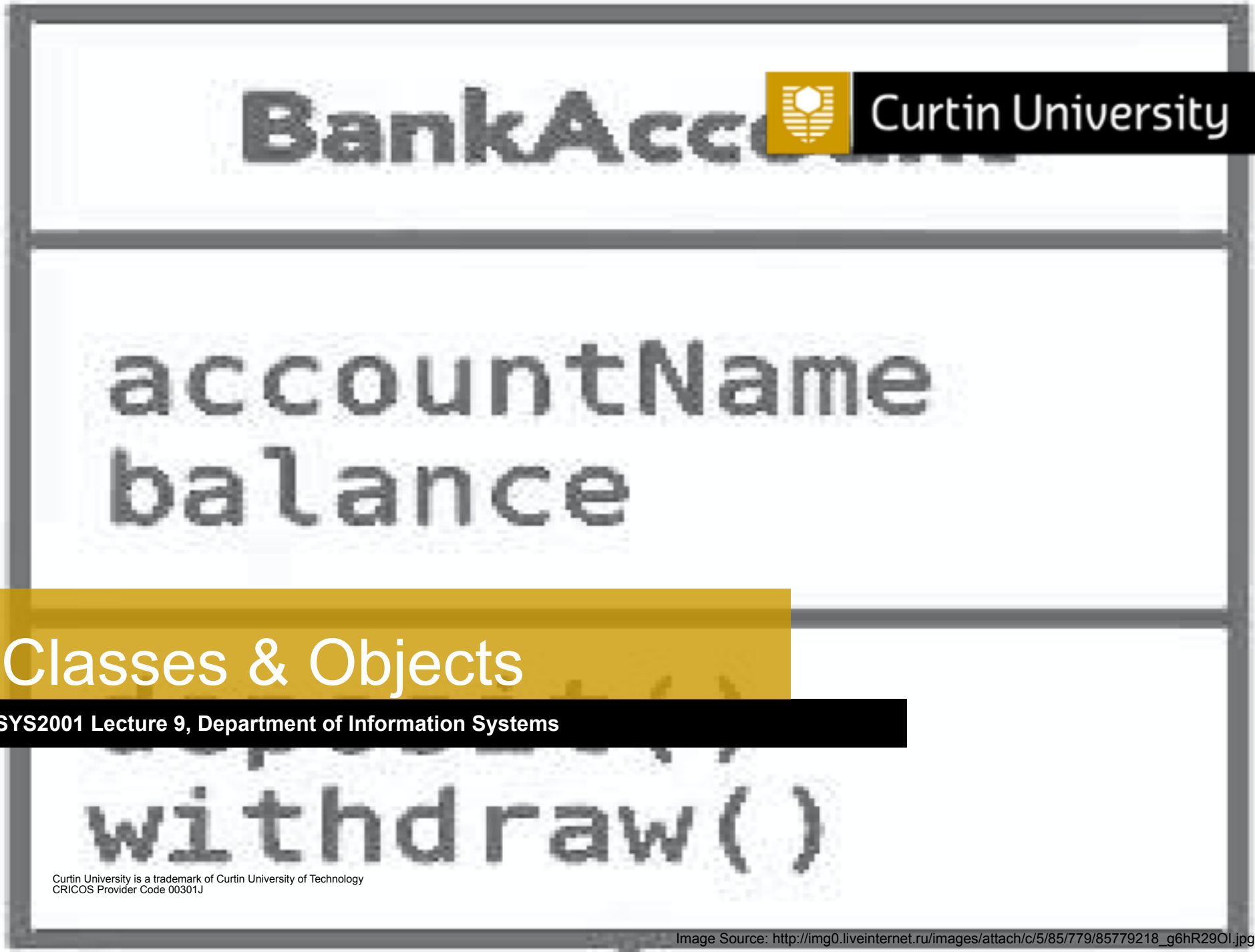


Curtin University

Classes & Objects

ISYS2001 Lecture 9, Department of Information Systems

Image Source: http://img0.liveinternet.ru/images/attach/c/5/85/779/85779218_g6hR29OI.jpg

Last Week

- List Some Useful Array Algorithms
- Describe Advanced Algorithms for Sorting and Searching Arrays
- Explain
  - Two-Dimensional Arrays
  - Jagged Arrays

Today you will be able to…

- State difference between Classes and Objects
- Use Class Properties
- Create Parameterised Constructors
- Describe Overloading
- Storing Class Type Object in Arrays and Lists
- Find Classes and Their Responsibilities in a Problem

Introduction to Classes

- A class is the blueprint for an object.

  It describes a particular type of object, yet it is not an object.

  It specifies the fields and methods a particular type of object can have.

  One or more object can be created from the class.

  Each object created from a class is called an instance of the class.

Creating a Class

- You can create a class by writing a class declaration. A generic form is:

```
class className  // class header
{
   Member declaration(s)…
}
```

- Class headers starts with the keyword class, followed by the name of the class.
- Member declarations are statements that define the class's fields, properties, and/or methods.
- A class may contains a constructor, which is special method automatically executed when an object is created.

Sample Code

```
class Coin
{
   private string sideUp; // field

   public Coin() // constructor
   {
      sideUp = “Heads”;
   }

   public void Toss() // a void method
   {
      MessageBox.Show(sideUp);
   }

   public string GetSideUp() // a value-returning method
   {
       return sideUp;
   }
}
```

Creating an Object

- Given a class named Coin, you can create a Coin object use:

```
Coin myCoin = new Coin();
```

- where,

  myCoin is a variable that references an object of the Coin class;

  the new keyword creates an instance of the Coin class; and

  the = operator assigns the reference that was returned from the new operator to the myCoin variable.

- Once a Coin object is created, you can access members of the class with it. E.g.

```
myCoin.Toss();
```

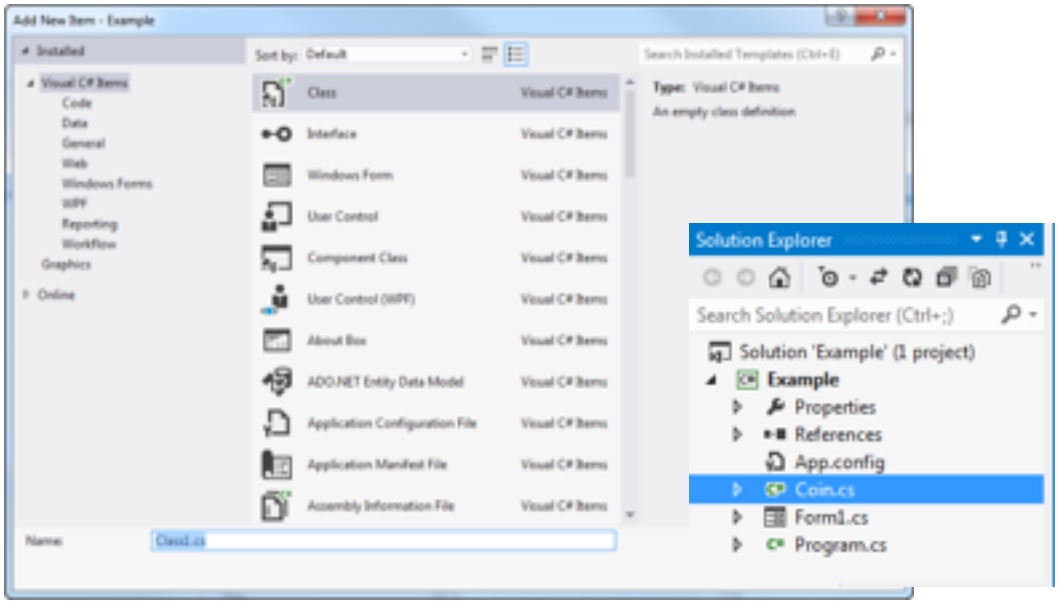
Where to Write Class Declarations

- In C# you have flexibility in choosing where to write class declarations. E.g.
- To create the Coin class, you can:

  Save the class declaration is a separated .cs file; or

  Add the Coin class next to the Form1 class inside the Form1.cs file.

Where to Write Class Declarations




```
Namespace Example
{
   public partial class Form1 :
Form
   {
      public Form1()
      {
        InitializeComponent();
    }
     ....
   }
   class Coin
   {
      ....
   }
}
```

Passing an Object to a Method

- Objects of a class can be used as parameter of a method. E.g.

```
private void ShowCoinStatus(Coin coin)
{
   MessageBox.Show("Side is " + coin.GetSideUp());
}
```

- In this example, a method named ShowCoinStatus accepts a Coin object as an argument.
- To create a Coin object and pass it as an argument to the ShowCoinStatus method, use:

```
Coin myCoin = new Coin();
ShowCoinStatus(myCoin);
```

Properties

- A property is a class member that holds a piece of data about an object.

  Properties can be implemented as special methods that set and get the value of corresponding fields.

  Both set and get methods are known

  as accessors.

  In the code, there is a private field (_name)

  which is a known as backing field and is

  used to hold any data assigned to the

  Name property.

  The value parameter of set accessor is

  automatically created by the compiler.

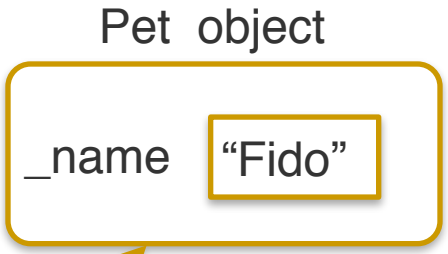
Properties

```
class Pet
{
 private string _name; // backing field
 public Pet()
 {
  _name = “”;
 }

 public string Name
 {
  get
  {
   return _name;
  }
  set
  {
   _name = value;
  }
 }
}
```

Setting the myDog object's Name Property to "Fido"

myDog.Name = "Fido";

```
// Name property
public string Name
{
  get
  {
    return _name;
  }

  set
  {
    _name = value;
  }
}
```



The Backing Field

- The private backing field is a variable that stores a value assigned to the property which the backing fields is associated with.
- It is declared to be private to protect it from accidental corruption.
- If a backing field is public, it can then be accessible directly by code outside the class without the need for accessors.

get vs set Accessors

- The get accessor, if not empty, is a method that returns the property's value because it has a return statement.

  It is executed whenever the property is read.

- The set accessor, if not empty, gets the value stored in the backing field and assigns the value to the property

  It has an implicit parameter named value.

  It is executed whenever a value is assigned to the property.

Read-Only Properties

- A read-only property can be read, but it cannot be modified.

  To set a read-only property, simply do no write a set accessor for the property. E.g.

```
// read and write
public double Diameter
{
   get { return _diameter; }
   set { _diameter = value; }
}
```

```
// read-only
public double Diameter
{
   get { return _diameter; }
}
```

Parameterized Constructor & Overloading

- A constructor that accepts arguments is known as parameterized constructor. E.g.

```
public BankAccount(decimal startingBalance) { }
```

- A class can have multiple versions of the same method known as overloaded methods.
- How does the compiler know which method to call?

  Binding relies on the signature of a method which consists of the method's name, the data type, and argument kind of the method's parameter. E.g.

```
public BankAccount(decimal startingBalance) { }
public BankAccount(double startingBalance) { }
```

  The process of matching a method call with the correct method is known as binding.

Overloading Methods

- When a method is overloaded, it means that multiple methods in the same class have the same name but use different types of parameters.

```
public void Deposit(decimal amount) { }
public void Deposit(double amount) { } // overloaded
public void Deposit(int numbers) { } // overloaded
public void Deposit(string names) { } // overloaded
```

Overloading Constructors

- Constructors are special type of methods. They can also be overloaded.

```
public BankAccount() { } // parameterless constructor
public BankAccount(decimal startingBalance) { } // overloaded
public BankAccount(double startingBalance) { } // overloaded
```

The parameterless constructor is the default constructor

- Compiler will find the matching constructors automatically. E.g.

```
BankAccount account = new BankAccount();
BankAccount account = new BankAccount(500m);
```

Storing Class Type Objects in Array & Lists

- Objects that are instances of a class can be stored in an array. E.g.

```
Const int SIZE = 4;
CellPhone[] phone = new CellPhone[SIZE];
phone[0] = new CellPhone();
phone[1] = new CellPhone();
….
```

- You can use a loop to step through the array. E.g.

```
for (int index = 0; index < phone.Length; index++)
{
   phones[index] = new CellPhone();
}
```

Initializing Array Elements

- You can initialize the array elements in the declaration statement:

```
CellPhone[] phone = {
    new CellPhone(), new CellPhone(),
    new CellPhone(), new CellPhone()
};
```

- You can also initialize an array and assign its elements with references to a class. E.g.

```
BankAccount[] accounts = {
     new BankAccount(1000),
     new BankAccount(2000),
     new BankAccount(3000),
     new BankAccount(4000),
    };
```

Lists of Class Type Objects

- You can create a List to hold a class object. E.g.

```
List<CellPhone> phoneList = new List<CellPhone>();
```

This statement creates a List object, referenced by the phoneList variable.

- Each object of the CellPhone class needs an instance of CellPhone class to hold data. E.g.

```
CellPhone myPhone = new CellPhone();
myPhone.Brand = “Acme Electronics”;
myPhone.Model = “M1000”;
myPhone.Price = 199;
```

- To add the Cellphone object to the List, use:

```
phoneList.Add(myPhone);
```

Finding the Classes & their Responsibilities in a Problem

- When developing an object-oriented program, you need to identify the classes that you will need to create.
- One simple and popular techniques involves the following steps:

  Get a written description of the problem domain.

  Identify all the nouns (including pronouns and noun phrases) in the description. Each of these is a potential class.

  Refine the list to include only the classes that are relevant to the problem.
- Once the classes have been identified, you need to identify each class's responsibilities. The responsibilities are:

  The things that the class is responsible for knowing

  The actions that the class is responsible for doing

Example

- In the textbook, there are three classes: Customer, Car, and ServiceQuote.
  - The Customer class has the following actions:
    - Create and initialize an object of the Customer class.
    - Get and set the customer's name.
    - Get and set the customer's address.
    - Get and set the customer's telephone number.
  - The Car class has the following actions:
    - Create and initialize an object of the Car class.
    - Get and set the car's make.
    - Get and set the car's model.
    - Get and set the car's year.
  - The ServiceQuote class has the following actions:
    - Create and initialize an object of the ServiceQuote class.
    - Get and set the estimated parts charges.
    - Get and set the estimated labor charges.
    - Get and set the sales tax rate.
    - Get the sales tax.
    - Get the total estimated charges.

Can You…

- State difference between Classes and Objects
- Use Class Properties
- Create Parameterised Constructors
- Describe Overloading
- Storing Class Type Object in Arrays and Lists
- Find Classes and Their Responsibilities in a Problem

Next Week

- Creating Multiple Forms in a Project
- Inheritance