OBJECT-ORIENTED PROGRAMMING WITH C#

INHERITANCE

issntt@nus.edu.sg

Objectives

At the end of this lecture, students will be able to

- Explain how inheritance promotes software reusability
- Create derived classes that inherit attributes and behaviors of a base class
- Use *protected* modifier properly
- Implement constructors in derived classes
- Use *base* reference properly
- Describe class hierarchies and the root class Object
- Design and implement maintainable solutions using Inheritance

Topics

- **Introduction**
- Base Classes and Derived Classes
- Implementing Derived Classes
- Access Modifiers and Inheritance
- Constructors in Derived Classes
- Method Overriding
- Calling Base Class Methods
- Class Hierarchies
- Inheritance Benefits

Employee Quiz

Write 2 classes: **CommissionEmployee** and ***BasePlusComissionEmployee*** as described below

Class **CommissionEmployee** has attributes *name*, *identityNumber*, *grossSales* and *commissionRate*

It also has a method *Earnings* that calculates the respective income of this type of employee, which is the multiplication of the commission rate and the gross sales

Class **BasePlusComisssionEmployee** has attributes *name*, *identifyNumber*, gr*ossSales*, *commissionRate* and *baseSalary*

It also has a method *Earnings* that calculates the respective income of this type of employee, which is the base salary plus the commission

Employee Quiz Solution 1

```
public class CommissionEmployee {
  private string name;
  private string identityNumber;
  private double grossSales;
  private double commissionRate;

  public CommissionEmployee(
       string name,
       string identityNumber,
       double grossSales,
       double commisionRate) {
    this.name = name;
    this.identityNumber = identityNumber;
    this.grossSales = grossSales;
    this.commissionRate = commisionRate;
  }

  public double Earnings() {
    return commissionRate * grossSales;
  }
}
```

Look at the similarities and differences of the two classes. What is the issue?

Hint: what if we need a new attributes for all classes, or another class for salary-only employees?

```
public class BasePlusCommissionEmployee {
  private string name;
  private string identityNumber;
  private double grossSales;
  private double commissionRate;
  private double salary;

  public BasePlusCommissionEmployee(
       string name,
       string identityNumber,
       double grossSales,
       double commisionRate,
       double salary) {
    this.name = name;
    this.identityNumber = identityNumber;
    this.grossSales = grossSales;
    this.commissionRate = commisionRate;
    this.salary = salary;
 }

 public double Earnings() {
    return salary +
       commissionRate * grossSales;
 }
}
```

Duplicate codes are BAD

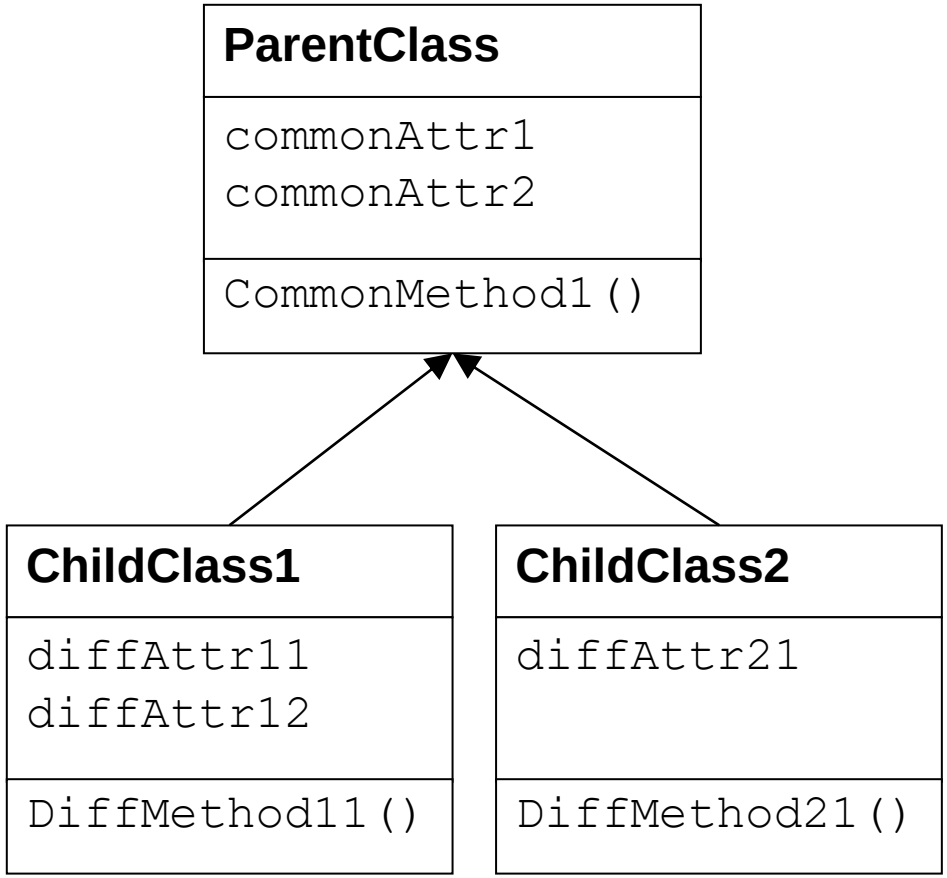
- In software development, many classes might have **common characteristics**
- **Duplicate codes lead to complicated maintenance!!!**

Is there any way to **avoid** them?

Duplicate codes https://www.codegrip.tech/productivity/what-is-duplicate-code/

Inheritance, a way to avoid duplicate code

Inheritance is to define a **general class** (i.e., a parent class) and **extend** it to **more specific classes** (i.e., child classes)



Topics

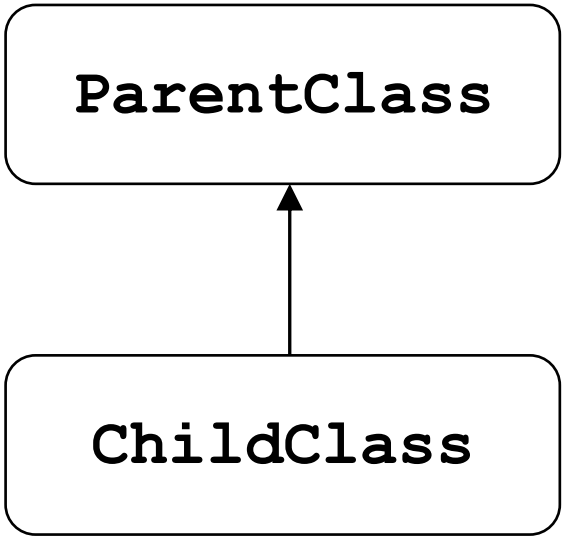
- Introduction
- **Base Classes and Derived Classes**
- Implementing Derived Classes
- Access Modifiers and Inheritance
- Constructors in Derived Classes
- Method Overriding
- Calling Base Class Methods
- Class Hierarchies
- Inheritance Benefits

Base Classes and Derived Classes

Inheritance creates an **is-a** relationship, where the child is a **more specific version** of the parent

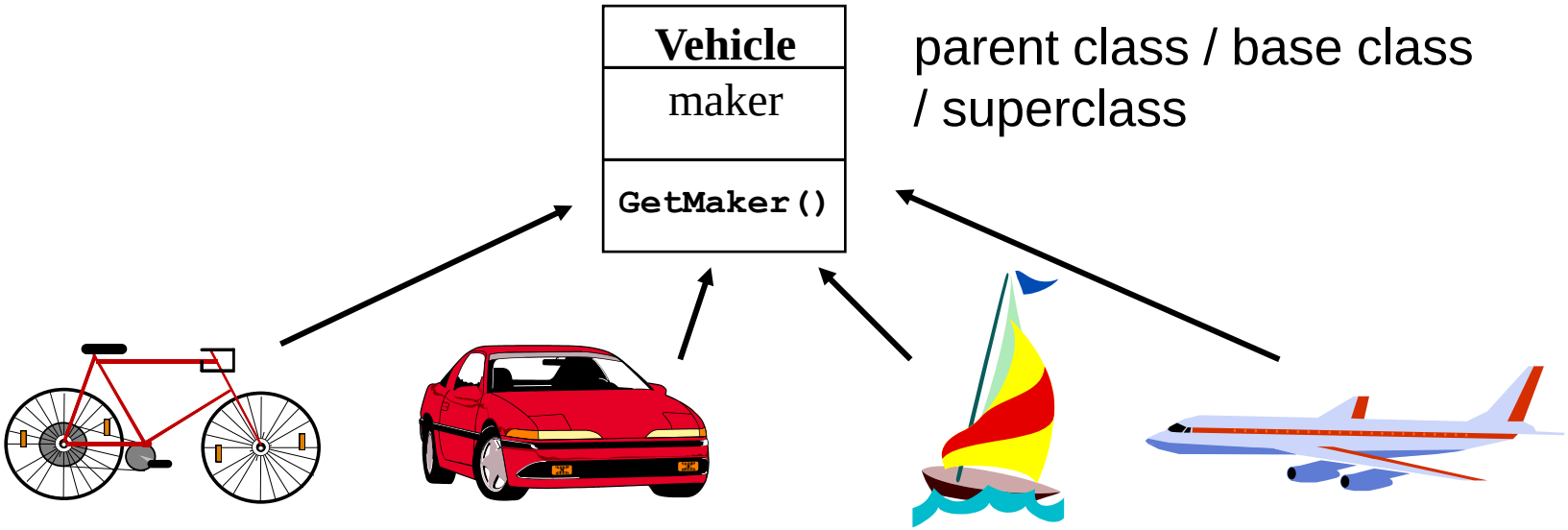
E.g.

- A horse is a mammal
- A cat is an animal
- A car is a vehicle



Base Classes and Derived Classes

Subclasses inherit the **states** and **behaviours** from the superclass



Quiz

What attributes does class **Cycle** has?

What methods does class **Cycle** has?

Topics

- Introduction
- Base Classes and Derived Classes
- **Implementing Derived Classes**
- Access Modifiers and Inheritance
- Constructors in Derived Classes
- Method Overriding
- Calling Base Class Methods
- Class Hierarchies
- Inheritance Benefits

Implementing Derived Classes

To implement a subclass, add a **colon** (`:`), followed by the **name** of the parent class

```
public class Parent
{
    // Class body
}

public class Child : Parent
{
    // Class body
}
```

Employee Quiz Solution 1 Revisit

The following is the current design

CommissionEmployee
name identityNumber grossSales commissionRate
Earnings()

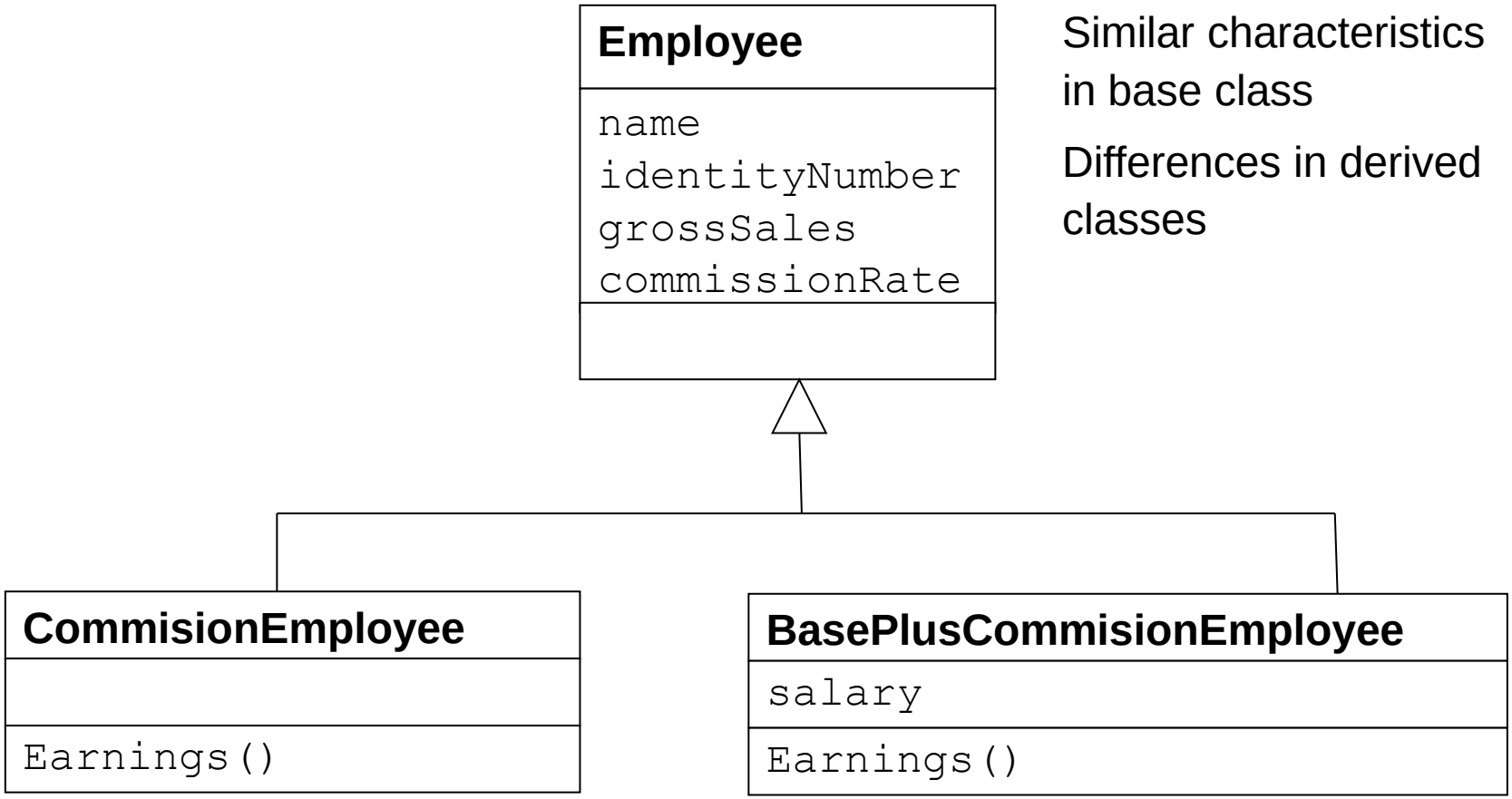
BasePlusCommissionEmployee
name identityNumber grossSales commissionRate salary
Earnings()

Re-design the Employee Quiz solution using inheritance.

***Hint*:** similarities should be in superclass and differences in subclasses

Despite the **same name**, *Earnings*() methods in the 2 classes have **different implementation**

Employee Quiz Solution 2 Design



Similar characteristics in base class

Differences in derived classes

How can we implement this?

Solution 2 - An Implementation

```
public class Employee {
  private string name;
  private string identityNumber;
  private double grossSales;
  private double commissionRate;
}
```

```
public class
  CommissionEmployee
      : Employee {
  public double Earnings()
  {
    return grossSales
          * commissionRate;
  }
}
```

```
public class
  BasePlusCommissionEmployee
      : Employee {
  private double salary;
  public double Earnings() {
    return salary +
      grossSales
          * commissionRate;
  }
}
```

But *grossSales* and *commissionRate* are **inaccessible** because they are `private` attributes. How can we make subclasses **access** attributes of superclass?

Topics

- Introduction
- Base Classes and Derived Classes
- Implementing Derived classes
- **Access Modifiers and Inheritance**
  - **The *protected* modifier**
- Constructors in Derived Classes
- Method Overriding
- Calling Base Class Methods
- Class Hierarchies
- Inheritance Benefits

Access Modifiers and Inheritance

- Up to now, we know that **class members are inherited** from the base class to their inherited classes
- But it does **not mean** they are **accessible** by inherited classes
- It depends on the **access modifiers**

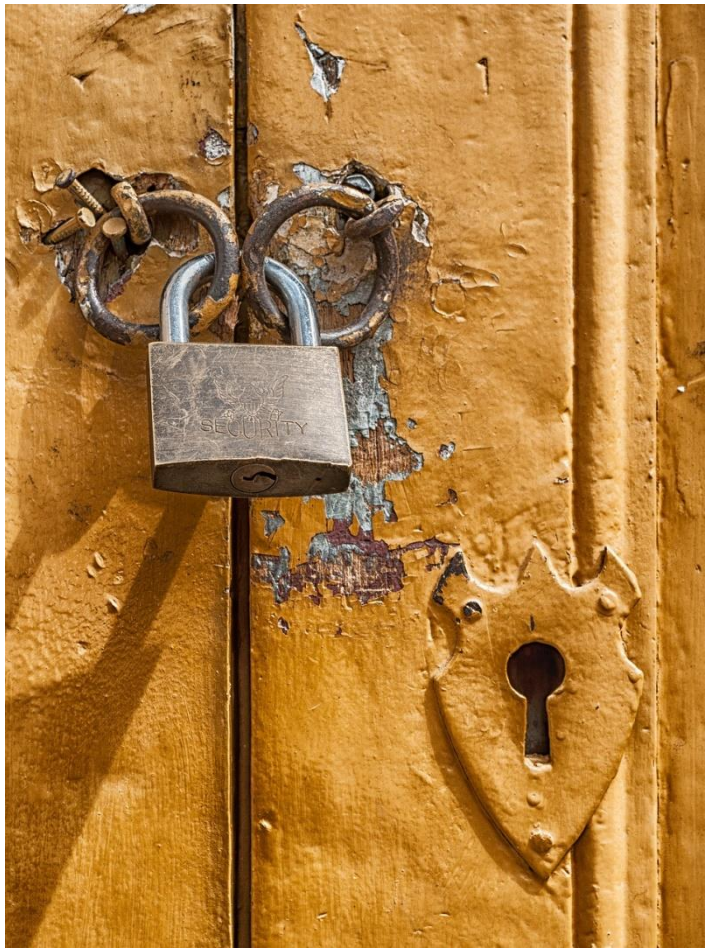


Image by David Mark from Pixabay

`private` Modifier

A `private` class members can only be accessed **within its own class**

```
class Parent {
    private int x;
    // Other code omited for brevity
}

class Child : Parent {
    public void ShowX() {
        Console.WriteLine("x is {0}", X);
    }
}
```

Error!
Attribute x is private.
Therefore, derive class cannot access it directly

How about `public`? How can we make **inherited attributes** accessible **only by inherited classes** and not others?

`protected` Modifier

Access modifier `protected` provides an accessibility level **suitable to inheritance**

```
class Parent {
   protected int x;
   // Other code omitted for brevity
}
class Child : Parent {
   public void ShowX() {
      Console.WriteLine("x is {0}", x);
   }
}
class Outside {
   public void ShowX()
   {
      Parent p = new Parent();
      Console.WriteLine("x is {0}", p.x);
   }
}
```

Attribute x can be directly accessed by **derived classes**

But **not from others**

But are we **violating** a bit on the Information Hiding?

Access Modifiers and Inheritance

The appropriate way is to **declare** the variable as `private` and provide a **property** or **method to access** it

```
class Parent {
    private int x;
    protected int X {
        get { return x; }
        set { x = value; }
    }
    // Other code omited for brevity
}

class Child : Parent {
    public void ShowX() {
        Console.WriteLine("x is {0}", X);
    }
}
```

Because x is private, we ensure data-hiding

Because X is a protected property, x is made available to derived classes but not to other classes

Employee Quiz Solution 2 – Improved

```
public class Employee
{
   private string name;
   private string identityNumber;
   private double grossSales;
   private double commissionRate;

   protected double GrossSales
   {
      get { return grossSales;  }
      set { grossSales = value; }
   }
   // Other properties omitted
}
```

```
public class CommissionEmployee
                    : Employee
{
   public double Earnings() {
      return GrossSales
            * CommissionRate;
   }
}
```

```
public class
   BasePlusCommissionEmployee
                    : Employee
{
   private double salary;
   public double Earnings()
   {
      return salary +
          GrossSales
            * CommissionRate;
   }
}
```

Next

Next, how can we **instantiate** and **initialize objects** for **subclasses?**

E.g.

ComissionEmployee and **BasePlusCommissionEmployee**?

Btw, what are different between **instantiation** and **initialization**?

Topics

- Introduction
- Base Classes and Derived Classes
- Implementing Derived Classes
- Access Modifiers and Inheritance
- **Constructors in Derived Classes**
- Method Overriding
- Calling Base Class Methods
- Class Hierarchies
- Inheritance Benefits

Object Initialization Review Quiz

What is the output of the following program?

```
public class Employee
{
  private string name;
  private string identityNumber;
  public Employee() {}
  public string Name
  {
    get { return name; }
    set { name = value; }
  }
  public string IdentityNumber
  {
    get {
      return identityNumber;
    }
    set {
      identityNumber = value;
    }
  }
}
```

```
public static void Main() {
  Employee myEmployee =
       new Employee();
  Console.WriteLine(
      myEmployee.Name +
      " and "  +
      myEmployee.IdentityNumber
  );
}
```

Now we want to make the *name* of the **myEmployee** object *"John"*, and the *identifyNumber "S123456A"*. List out **2 ways** to do that

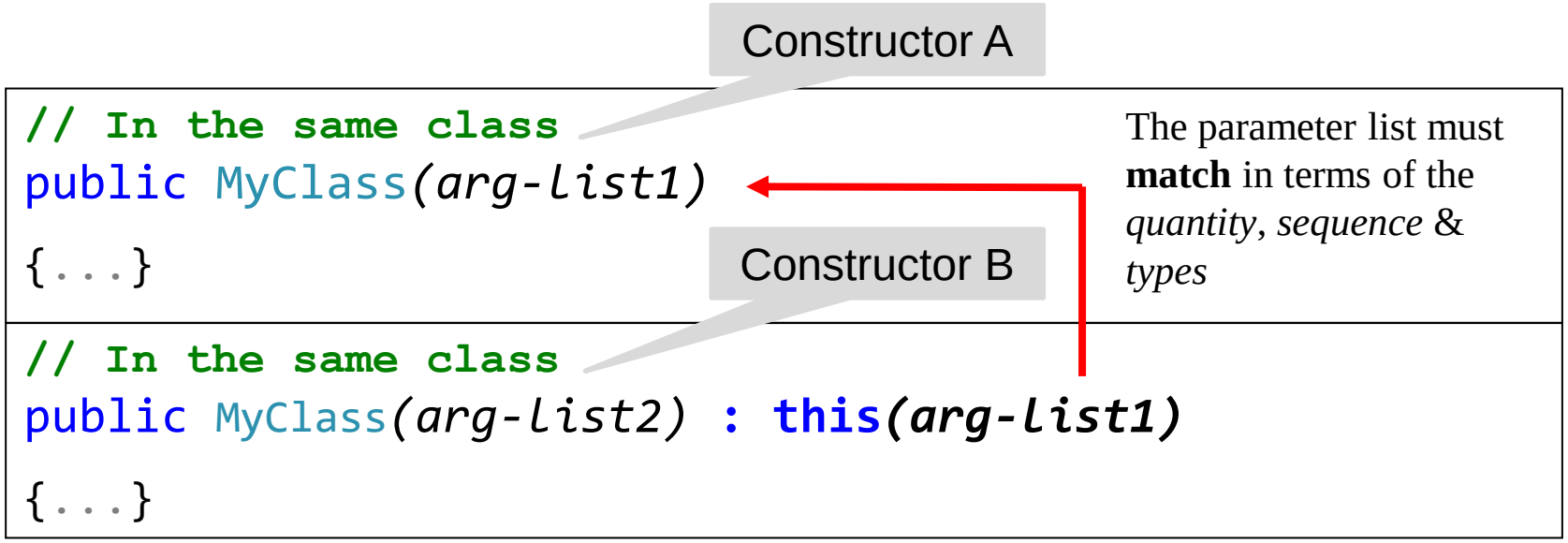
Constructors in Derived Classes

- Unlike methods, **constructors** of a base class are **NOT inherited**
- Thus, a derived class will usually need to define **its own constructors**

- However, we often wants to **make use of** the parent's constructor to **initialize the "parent's part"** of the derived object
- In order not to **re-write codes** for the parent's part of the initialization

Constructor Initializer: A Revisit

Recall that we can invoke a constructor B from a constructor A in the **same class** using the constructor initializer `: this(arg-list)`



Statements in constructor A will be executed **before** the statements in constructor B

Calling Base Class Constructor

To invoke a constructor residing in the **base class**, use keyword **`base`** instead of keyword **`this`**

```
// In base class
public Parent(arg-list1)
{...}

// In derived class
public Child(arg-list2) : base(arg-list1)
{...}
```

The parameter list must **match** in terms of the *quantity*, *sequence* & *types*

Statements in the `Parent` constructor will be executed **before** the statements in the `Child` constructor

Current Employee Quiz Solution 2 Implementation

```
public class Employee
{
   private string name;
   private string identityNumber;
   private double grossSales;
   private double commissionRate;

   protected double GrossSales
   {
      get { return grossSales;  }
      set { grossSales = value; }
   }
   // Other properties omitted
}
```

```
public class CommissionEmployee
                    : Employee
{
   public double Earnings() {
      return GrossSales
            * CommissionRate;
   }
}
```

Now, how can we **implement constructors** for these classes?

```
public class
   BasePlusCommissionEmployee
                    : Employee
{
   private double salary;
   public double Earnings()
   {
      return salary +
          GrossSales
            * CommissionRate;
   }
}
```

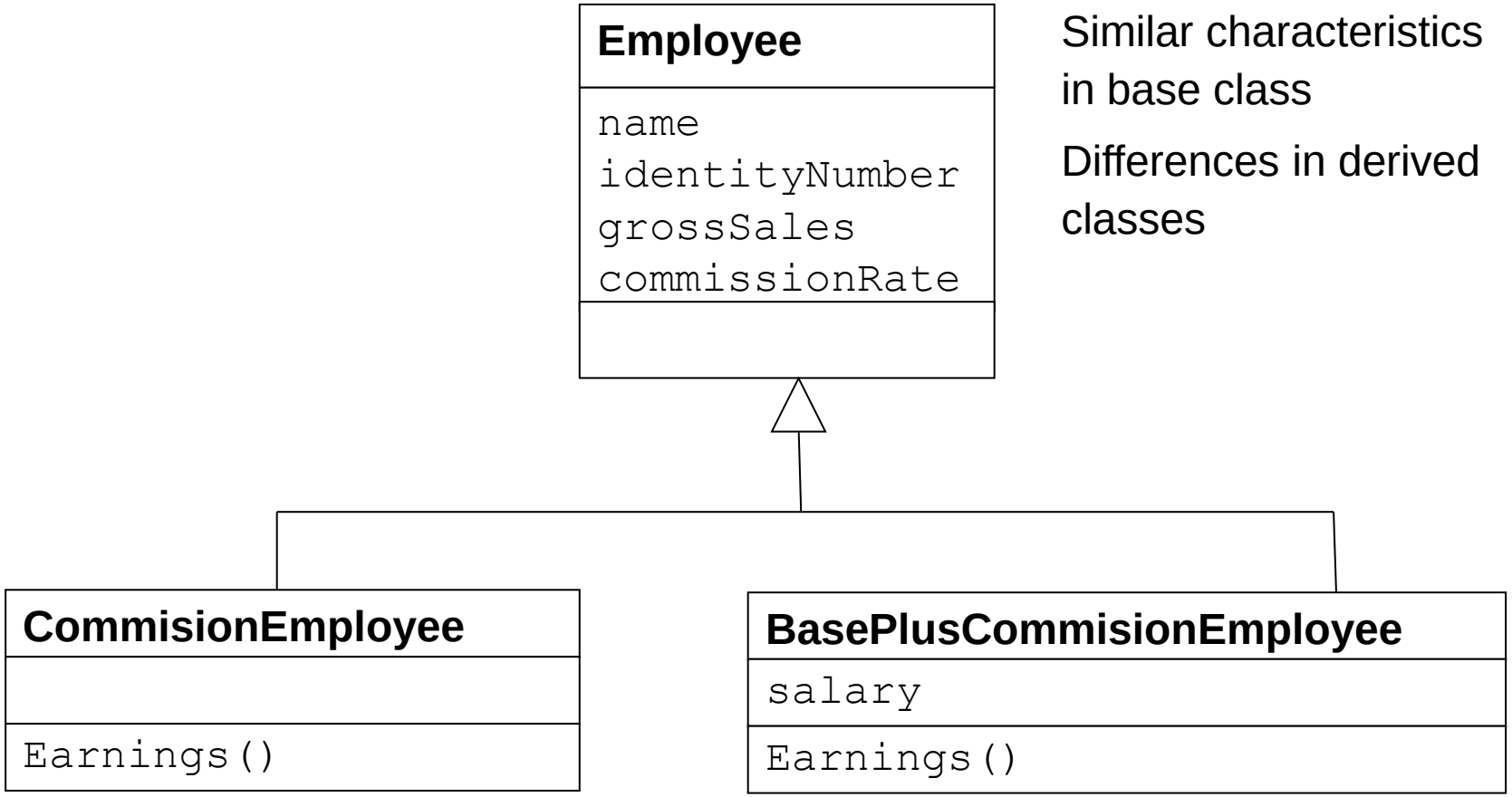
Employee Quiz Solution 2 – Improved

```
public class Employee
{
   private string name;
   private string identityNumber;
   private double grossSales;
   private double commissionRate;
1  public Employee(
        string name,
        string identityNumber,
        double grossSales,
        double commisionRate)
   {
      Name = name;
      IdentityNumber = identityNumber;
      GrossSales = grossSales;
      CommissionRate = commisionRate;
   }

   // Properties omitted for brevity
}
```

```
public class CommissionEmployee
                          : Employee {
  public CommissionEmployee(
    string name,
    string identityNumber,
    double grossSales,
    double commisionRate) :
2     base(name, identityNumber,
        grossSales, commisionRate)
   {}
   // Earnings() omitted for brevity
}
```

```
public class BasePlusCommissionEmployee
                          : Employee {
3 private double salary;
  public BasePlusCommissionEmployee(
    string name,
    string identityNumber,
    double grossSales,
    double commissionRate,
    double salary) : 4 base(name,
        identityNumber, grossSales,
          commissionRate) {
5   this.salary = salary;
  }
   // Earnings() omitted for brevity
}
```

Employee Quiz Solution 2 Design Revisit



Similar characteristics in base class

Differences in derived classes

In fact, method *Earnings()* is also **similar** and only its **implementation is different**? Can we also put it in the base class?

Topics

- Introduction
- Base Classes and Derived Classes
- Implementing Derived Classes
- Access Modifiers and Inheritance
- Constructors in Derived Classes
- **Method Overriding**
  - **Override vs Overload**
- Calling Base Class Methods
- Class Hierarchies
- Inheritance Benefits

Method Override

A subclass can **modify** the **implementation** of a method defined in its **superclass**

```
class Person {
   private string nric;
   private string name;

   public virtual void Display() {
      Console.WriteLine("NRIC: {0}; Name: {1}", nric, name);
   }
   // Properties NRIC and Name
}
class Student : Person {
   private string matricNo;

   public override void Display() {
      Console.WriteLine("NRIC: {0}; Name: {1}; MatricNo: {2}",
                                        NRIC, name, matricNo);
   }
}
```

Overriding methods

```
public virtual void
                MyMethod(int a)
{
    // Implementation omitted
}
```

Base class

```
public override void
                MyMethod(int x)
{
    // Implementation omitted
}
```

Derived class

To establish proper overriding

1. Include `virtual` keyword in the **base class** method header
2. Include `override` keyword in the overriding method in the **derived class**
3. It has the **same name**, **return type** and **signature** as the inherited method

Does the two ***`MyMethod()`*** methods above have the **same signature**?

Method override vs method overload

Overloading	Overriding
Same class, **same method name** but **different signatures**	One in **parent class**, one in **child class**, **same signature**
Similar operations in different ways for **different data**	Similar operation in different ways for **different object types**
Resolved at **compile time**	Resolved at **run-time** based on the type of object

Quiz

Which one is overriding? Which is overloading? Why?

A

```
class Number
{
  public int add(int num1,
         int num2)
  {
    return num1 + num2;
  }

  public int add(int num1,
       int num2, int num3)
  {
    return num1 + num2
                + num3;
  }
}
```

B

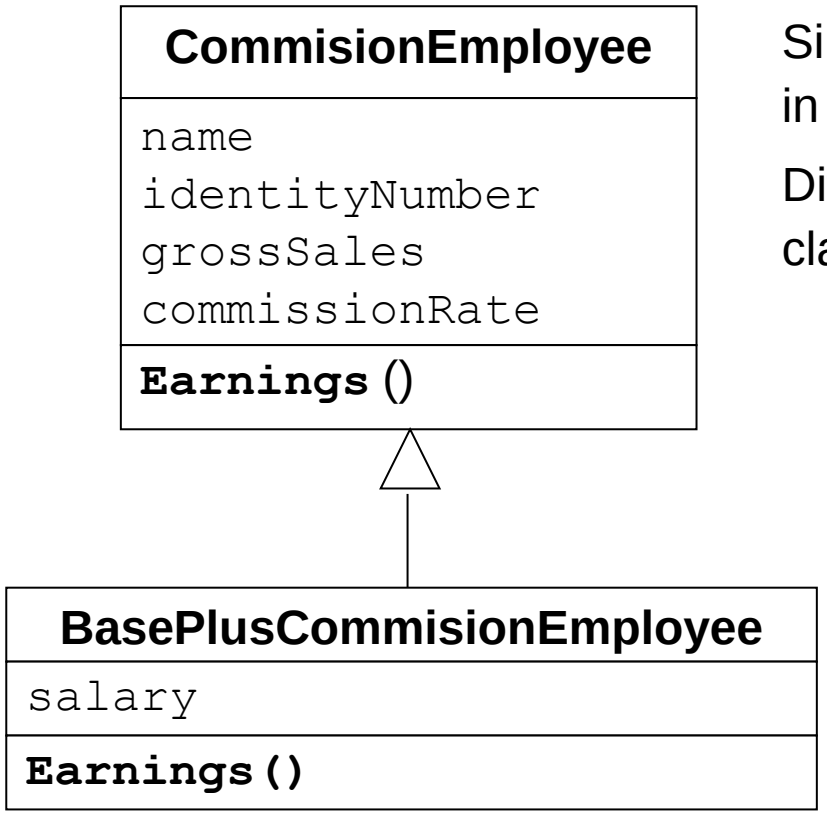
```
class Animal
{
  public virtual String sound()
  {
    return "";
  }
}

class Cat : Animal
{
  public override String sound()
  {
    return "meow";
  }
}
```

Let’s do this

Improve the Employee Quiz solution design with **method overriding**

Employee Quiz Solution 3 Design

Applying **method overriding** to the quiz



Similar characteristics in base class

Differences in derived classes

Class **Employee** and **CommissionEmployee** have been combined because class **CommissionEmployee** becomes **empty** after moving `Earnings()` method to class **Employee**

Employee Quiz Solution 3 Implementation

```
public class CommissionEmployee {
  private string name;
  private string identityNumber;
  private double grossSales;
  private double commissionRate;
  public CommissionEmployee(
       string name,
       string identityNumber,
       double grossSales,
       double commisionRate) {
     this.name = name;
     this.identityNumber = identityNumber;
     this.grossSales = grossSales;
     this.commissionRate = commisionRate;
  }
  // Properties omitted for brevity

  public 1 virtual double Earnings() {
    return CommissionRate * GrossSales;
  }
}
```

```
public class
    BasePlusCommissionEmployee :
                  CommissionEmployee {
  private double salary;
  public BasePlusCommissionEmployee(
       string name,
       string identityNumber,
       double grossSales,
       double commissionRate,
       double salary) : base (name,
         identityNumber, grossSales,
         commissionRate) {
     this.salary = salary;
  }
  // Property Salary omitted

  public 2 override double Earnings(){
    return salary +
       CommissionRate * GrossSales;
  }
}
```

In this case, implementation of *Earnings()* in the child class is actually **a part of the parent class**. Can we make use of the parent's class method?

Topics

- Introduction
- Base Classes and Derived Classes
- Implementing Derived Classes
- Access Modifiers and Inheritance
- Constructors in Derived Classes
- Method Overriding
- **Calling Base Class Methods**
- Class Hierarchies
- Inheritance Benefits

Calling Base Class Methods

The keyword `base` is also used to call a **method** or **property** of a **base class** from the derived class

```
public virtual void
                MyMethod(int a)
{
   // Implementation omitted
}
```

Base class

What happen if we don't include keyword `base`?

```
public override void
                MyMethod(int x)
{
   // ..
   base.MyMethod(x);
   // ..
}
```

Derived class

Improve the Employee Quiz implementation with the new design

Employee Quiz Solution 3 Implementation – Improved

```
public class CommissionEmployee {
    …
    public virtual double Earnings()
    {
        return CommissionRate * GrossSales;
    }
    …
}
```

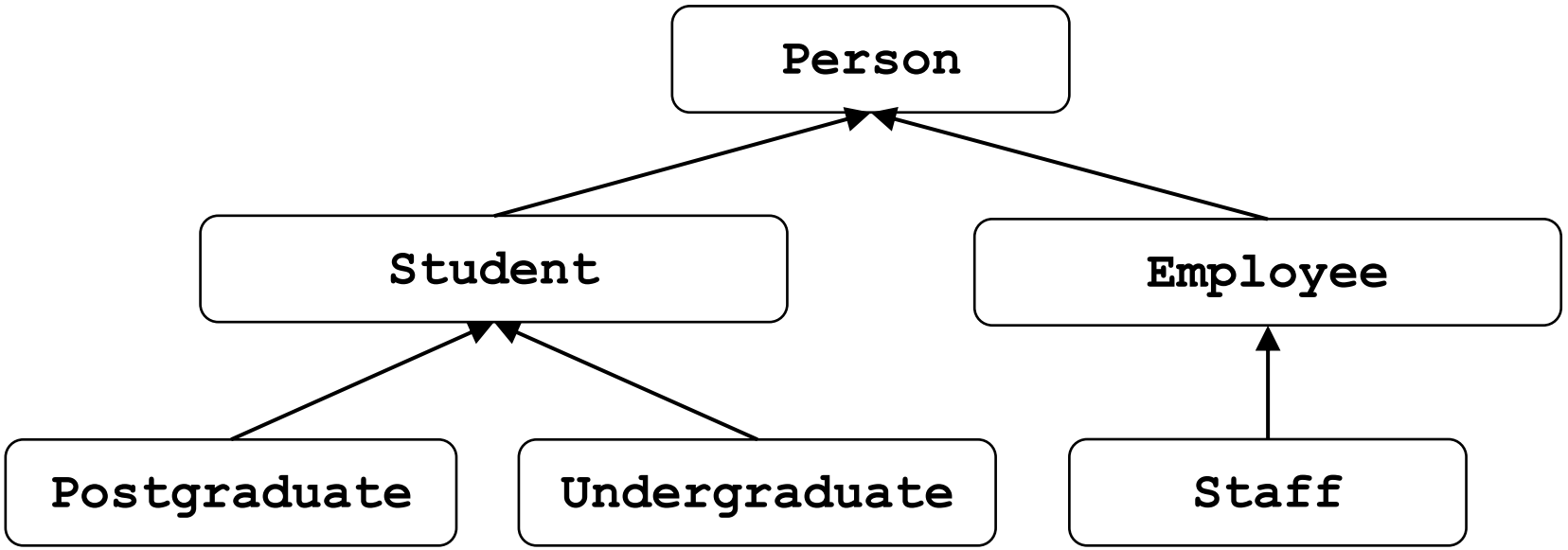
```
public class BasePlusCommissionEmployee :
                                CommissionEmployee {
    …
    public override double Earnings()
    {
        return salary + 1 base.Earnings();
    }
}
```

Topics

- Introduction
- Base Classes and Derived Classes
- Implementing Derived Classes
- Access Modifiers and Inheritance
- Constructors in Derived Classes
- Method Overriding
- Calling Base Class Methods
- **Class Hierarchies**
  - **The Ultimate Base Class: Object**
- Inheritance Benefits

Class Hierarchies

- A derived class can be further **used to derive other classes**
- Consequently, we can create **many levels** of derived classes, forming class hierarchies



Class Hierarchies

- Good class design usually **puts all common features as high as possible** in the hierarchy to **maximize re-usability** of classes
- An inherited member will be continually passed down the line, hence a member
  - may not from its **immediate parent**, but
  - may from a **few levels higher** up in the hierarchy instead

In the last example, if **Staff** has attribute **name**, which class should include this attribute?

Next

```
public class Employee
{
    private string name;
    public Employee(string name)
    {
        this.name = name;
    }
    public string Name
    {
        get { return name; }
        set { name = value; }
    }
}

public class Test
{
    public static void Main()
    {
        Employee emp = new Employee("Alex");
        Console.WriteLine(emp.ToString());
    }
}
```

Following is the output if we run the program

Output:
OOPC_Inheritance_Class_Object.Employee

Where does ***ToString()*** method come from? Why such output is generated?

The Ultimate Base Class: Object

- The topmost base/root class in C# is class `Object`
- **All** data types and other classes are **derived directly or indirectly** from class `Object`
- Everything is an `Object` in C#

https://docs.microsoft.com/en-us/dotnet/api/system.object

The Ultimate Base Class: Object

- All methods in class `Object` are **inherited in all classes**
- For example, the *`ToString()`* method is defined in class `Object`

- The *`ToString()`* in class `Object` returns a string representation of the current object, including its **namespace name** and **class name**
- This method is declared **virtual** and is almost always **overridden** to return **specific object's state**

In the last example, what to change in class ***`Employee`*** so the output becomes *"`Alex`"*?

Important Note: if we put *Console.WriteLine(obj)*, C# **automatically** calls *obj.ToString()* method

Topics

- Introduction
- Base Classes and Derived Classes
- Implementing Derived Classes
- Access Modifiers and Inheritance
- Constructors in Derived Classes
- Method Overriding
- Calling Base Class Methods
- Class Hierarchies
- **Inheritance Benefits**

Inheritance Benefits

1. Software **re-use**
   - Descendants **need not repeat** the common descriptions already specified in their ancestors
2. Ease of **maintenance**
   - If we want to modify some characteristics of class **Person**, **change in one place** (i.e. in class **Person**)
   - All the descendants automatically inherit the **new definition**

Inheritance Benefits

- However, the two benefits above can **also** be achieved by **Object Composition** (has-a relationship)
  - which is even **better** than Inheritance in many situations

- In fact, the **most important** benefit of Inheritance is **Polymorphism**

Readings

- Visual C# 2012 How to Program, 5th edition – Chapter 11, Inheritance, *Paul Deitel and Harvey Deitel*
- Head First C#, 3rd edition – Chapter 6, *Andrew Stellman and Jennifer Greene*