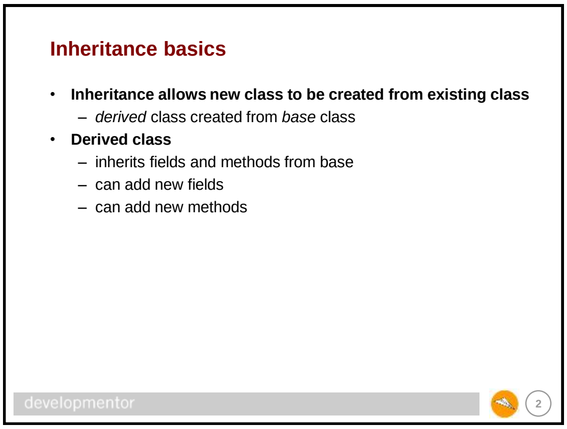
Inheritance

(modified)

developmentor

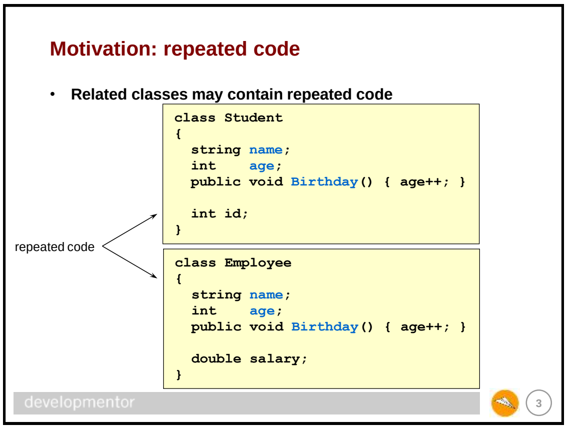
A DEVELOPER SERVICES COMPANY



Inheritance basics

- **Inheritance allows new class to be created from existing class**
  - *derived* class created from *base* class
- **Derived class**
  - inherits fields and methods from base
  - can add new fields
  - can add new methods

developmentor 2



Motivation: repeated code

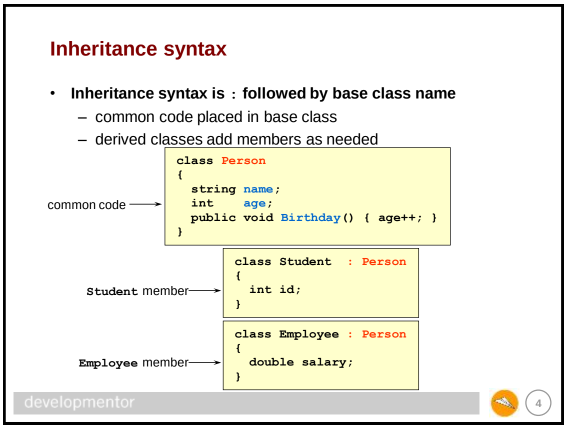
- **Related classes may contain repeated code**

repeated code

```
class Student
{
  string name;
  int    age;
  public void Birthday() { age++; }

  int id;
}
```

```
class Employee
{
  string name;
  int    age;
  public void Birthday() { age++; }

  double salary;
}
```

developmentor 3



Inheritance syntax

- **Inheritance syntax is : followed by base class name**
  - common code placed in base class
  - derived classes add members as needed

common code

```
class Person
{
  string name;
  int    age;
  public void Birthday() { age++; }
}
```

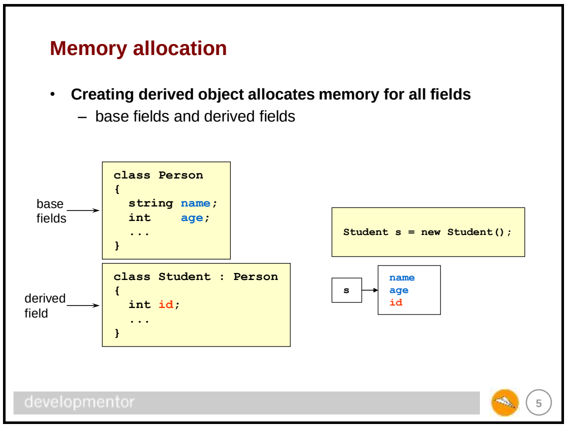
Student member

```
class Student  : Person
{
  int id;
}
```

Employee member

```
class Employee : Person
{
  double salary;
}
```

developmentor 4



Memory allocation

- **Creating derived object allocates memory for all fields**
  - base fields and derived fields

base fields

```
class Person
{
  string name;
  int    age;
  ...
}
```

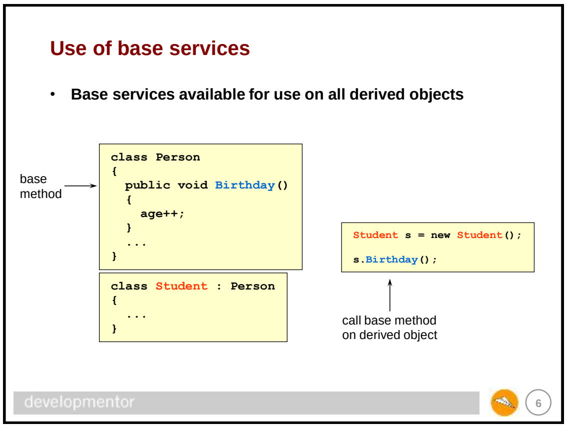
derived field

```
class Student : Person
{
  int id;
  ...
}
```

```
Student s = new Student();
```

s
name
age
id

developmentor 5



Use of base services

- **Base services available for use on all derived objects**

base method

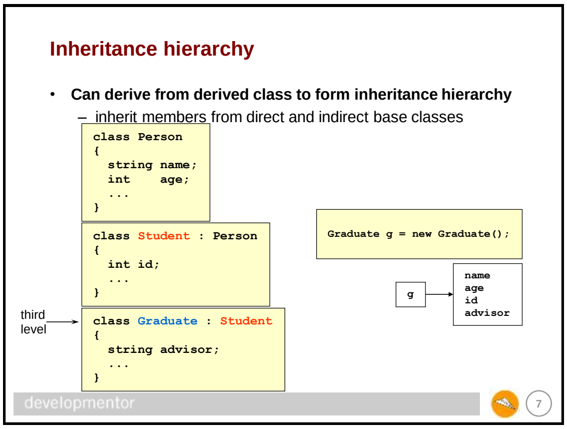
```
class Person
{
  public void Birthday()
  {
    age++;
  }
  ...
}
```

```
class Student : Person
{
  ...
}
```

```
Student s = new Student();

s.Birthday();
```

call base method on derived object

developmentor 6



Inheritance hierarchy

- **Can derive from derived class to form inheritance hierarchy**
  - inherit members from direct and indirect base classes

```
class Person
{
  string name;
  int    age;
  ...
}
```

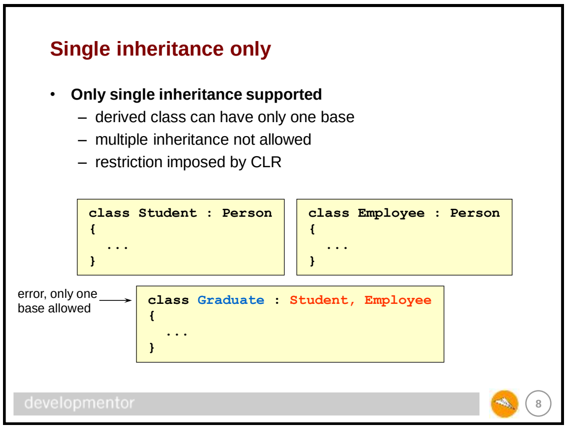
```
class Student : Person
{
  int id;
  ...
}
```

third level →

```
class Graduate : Student
{
  string advisor;
  ...
}
```

```
Graduate g = new Graduate();
```

g
name
age
id
advisor



Single inheritance only

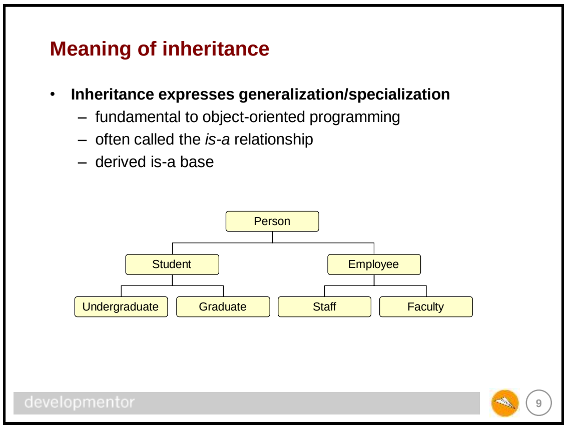
- **Only single inheritance supported**
  - derived class can have only one base
  - multiple inheritance not allowed
  - restriction imposed by CLR

```
class Student : Person
{
  ...
}
```

```
class Employee : Person
{
  ...
}
```

error, only one base allowed →

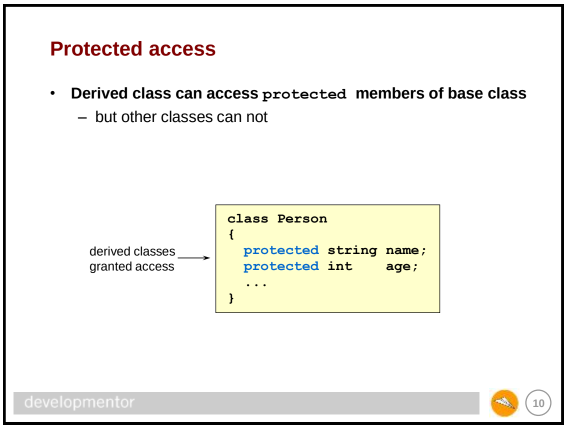
```
class Graduate : Student, Employee
{
  ...
}
```



Meaning of inheritance

- **Inheritance expresses generalization/specialization**
  - fundamental to object-oriented programming
  - often called the *is-a* relationship
  - derived is-a base

Person
Student
Employee
Undergraduate
Graduate
Staff
Faculty

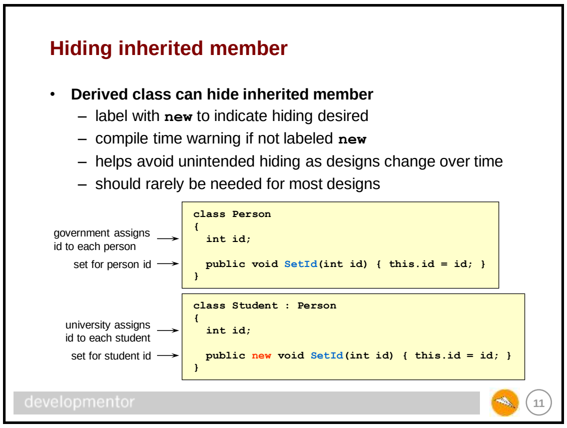


Protected access

- **Derived class can access `protected` members of base class**
  - but other classes can not

derived classes granted access →

```
class Person
{
  protected string name;
  protected int    age;
  ...
}
```



Hiding inherited member

- **Derived class can hide inherited member**
  - label with **`new`** to indicate hiding desired
  - compile time warning if not labeled **`new`**
  - helps avoid unintended hiding as designs change over time
  - should rarely be needed for most designs

government assigns id to each person →

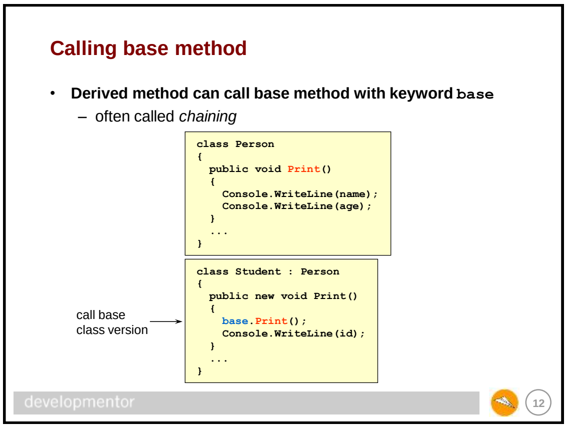
set for person id →

```
class Person
{
  int id;

  public void SetId(int id) { this.id = id; }
}
```

university assigns id to each student →

set for student id →

```
class Student : Person
{
  int id;

  public new void SetId(int id) { this.id = id; }
}
```



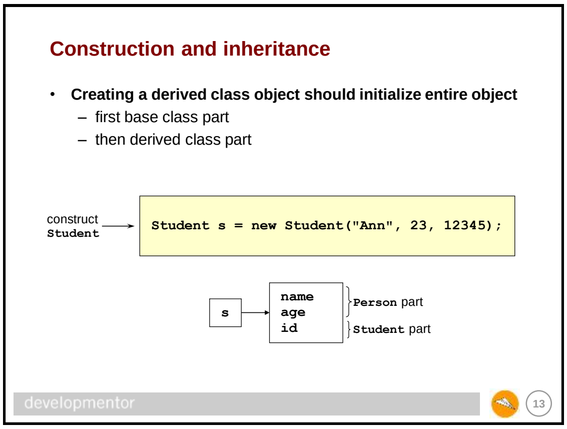
Calling base method

- **Derived method can call base method with keyword `base`**
  - often called *chaining*

```
class Person
{
  public void Print()
  {
    Console.WriteLine(name);
    Console.WriteLine(age);
  }
  ...
}
```

call base class version →

```
class Student : Person
{
  public new void Print()
  {
    base.Print();
    Console.WriteLine(id);
  }
  ...
}
```



Construction and inheritance

- **Creating a derived class object should initialize entire object**
  - first base class part
  - then derived class part

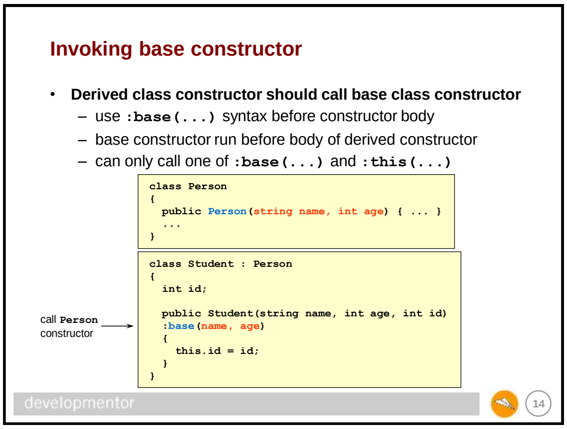
construct **Student**

```
Student s = new Student("Ann", 23, 12345);
```

s → name, age, id

Person part

Student part



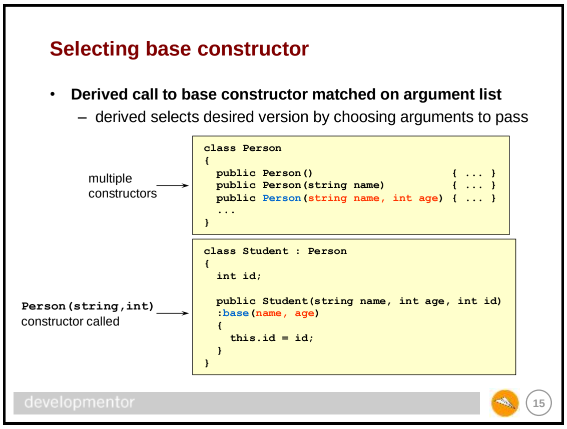
Invoking base constructor

- **Derived class constructor should call base class constructor**
  - use **:base(...)** syntax before constructor body
  - base constructor run before body of derived constructor
  - can only call one of **:base(...)** and **:this(...)**

```
class Person
{
  public Person(string name, int age) { ... }
  ...
}
```

call **Person** constructor

```
class Student : Person
{
  int id;

  public Student(string name, int age, int id)
  :base(name, age)
  {
    this.id = id;
  }
}
```



Selecting base constructor

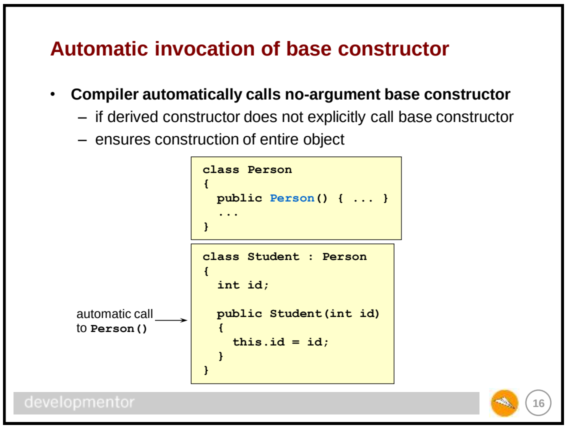
- **Derived call to base constructor matched on argument list**
  - derived selects desired version by choosing arguments to pass

multiple constructors

```
class Person
{
  public Person()                      { ... }
  public Person(string name)           { ... }
  public Person(string name, int age)  { ... }
  ...
}
```

Person(string,int) constructor called

```
class Student : Person
{
  int id;

  public Student(string name, int age, int id)
  :base(name, age)
  {
    this.id = id;
  }
}
```



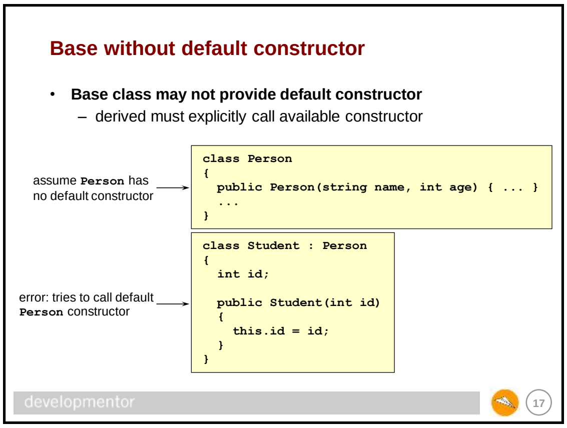
Automatic invocation of base constructor

- **Compiler automatically calls no-argument base constructor**
  - if derived constructor does not explicitly call base constructor
  - ensures construction of entire object

```
class Person
{
  public Person() { ... }
  ...
}
```

automatic call to **Person()**

```
class Student : Person
{
  int id;

  public Student(int id)
  {
    this.id = id;
  }
}
```



Base without default constructor

- **Base class may not provide default constructor**
  - derived must explicitly call available constructor

assume **Person** has no default constructor

```
class Person
{
  public Person(string name, int age) { ... }
  ...
}
```

error: tries to call default **Person** constructor

```
class Student : Person
{
  int id;

  public Student(int id)
  {
    this.id = id;
  }
}
```