Microsoft®
.NET

UNIT 4. INHERITANCE & POLYMORPHISM

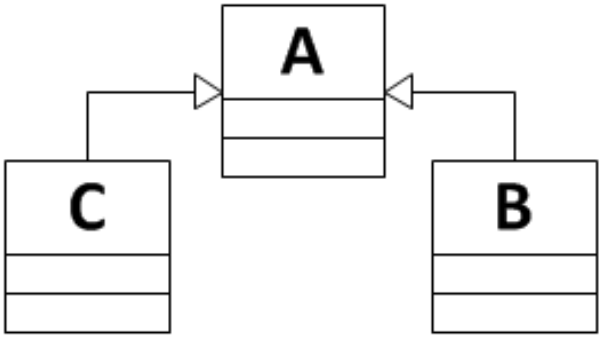
- Lession 1. Inheritance
- Lesson 2. Polymorphism

LESSION 1. INHERITANCE

- Generalization & Specialization
- Defintion
- Implement inheritance
- Sealed Classes
- Sealed Methods

Definition

- Inheritance is a way to **reuse code** of existing class (objects), or to establish a subtype from an existing class (object), or both



- A: base class (superclasses, parent class) and B: derived class (subclass, child class)

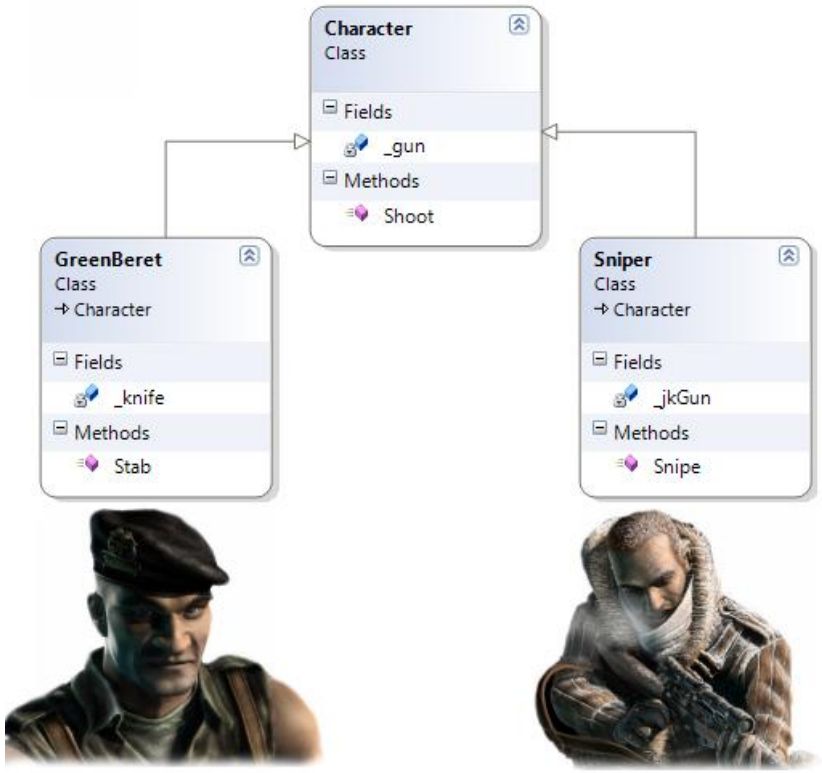
Defition

- The idea of inheritance implements the is-a relationship. EX: MiniCar is a Car, Dog is a Animal
- Don't use it to build has-a relationship
- The benefits of Inheritance?
- C# does not suppo
- Syntax:

```
<acess-specifier> class <base_class>
{
    ...
}
class <derived_class> : <base_class>
{
    ...
}
```

- Use class diagram with VS

Implement inheritance




```
public class Character
{
    private int _gun;

    public void Shoot()
    {
        Console.WriteLine("I can shoot enemies");
    }
}

public class GreenBeret : Character
{
    private int _knife;

    public void Stab()
    {
        Console.WriteLine("I can stab enemies");
    }
}
```

“base” keyword

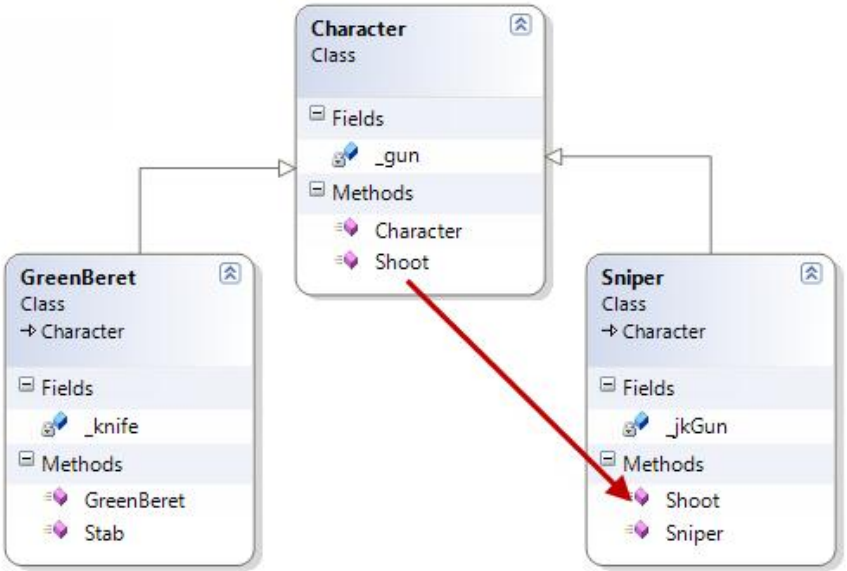
- The **base** keyword is used to access members of the base class:
  1. Inherits constructor:

```
public Sniper(int gun, int jkGun):base(gun)
{
    _jkGun = jkGun;
}
```

  2. Call a method on the base class that has been overridden by another method.

“base” and “new” keyword

```
public class Character
{
    private int _gun;

    public Character(int gun)
    {
        _gun = gun;
    }

    public void Shoot()
    {
        Console.Write("I can shoot enemies");
    }
}
public class Sniper : Character
{
    private int _jkGun;

    public Sniper(int gun, int jkGun):base(gun)
    {
        _jkGun = jkGun;
    }
    public new void Shoot()
    {
        base.Shoot();
        Console.WriteLine("by the JK gun (sniping)")
    }
}
```



Sealed class

- A sealed class cannot be used as a base class. For this reason, it cannot also be an abstract class.

C#

```
public sealed class D
{
    // Class members here.
}
```

- Prevent Inheritance
- Ensure security
- Protect from Copyright Problems

Sealed methods

- A class member, method, field, property, or event, on a derived class that is overriding a virtual member of the base class can declare that member as sealed.

C#

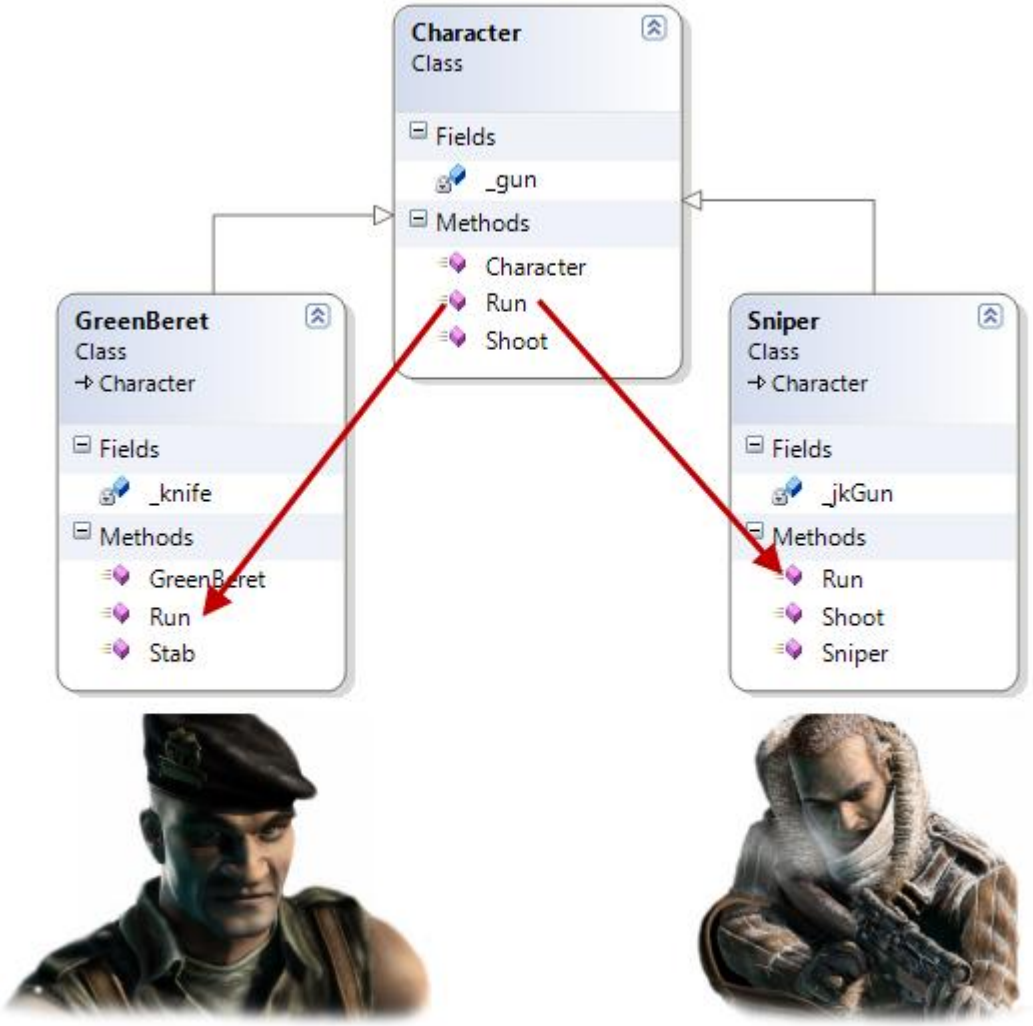
```
public class D : C
{
    public sealed override void DoWork() { }
}
```

- Prevent overriding

LESSION 2. POLYMORPHISM

- Introduction
- Defintion
- Implementation
- Casting operation

Introduction



Definition

- Polymorphism = ability to take more than one form (objects have more than one type)
  - A class can be used through its parent interface
  - A child class may override some of the behaviors of the parent class
- Polymorphism allows abstract operations to be defined and used
  - Abstract operations are defined in the base class' interface and implemented in the child classes
  - Declared as abstract or virtual

Virtual method

- Virtual method is method that can be used in the same way on instances of base and derived classes but its implementation is different
- A method is said to be a virtual when it is declared as `virtual`

```
public virtual void CalculateSurface()
```

- Methods that are declared as virtual in a base class can be overridden using the keyword `override` in the derived class

Implemention

- **Syntax**

```
class BaseClass
{
    public virtual void Method()
    {
        //...
    }
}

class DerivedClass:BaseClass
{
    public override void Method()
    {
        //...
    }
}
```

Implemention

- **Snippet**

```
public class Character
{
    private int _gun;

    public Character(int gun)
    {
        _gun = gun;
    }

    public void Shoot()
    {
        Console.Write("I can shoot enemies");
    }

    public virtual void Run()
    {
        Console.Write("I can run");
    }
}
```

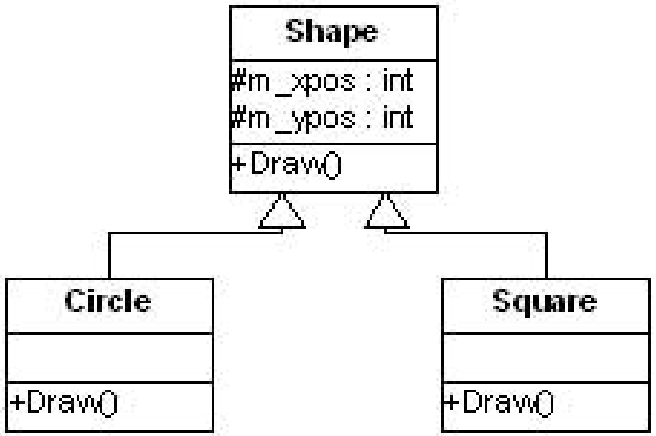
```
public class GreenBeret : Character
{
    private int _knife;

    public GreenBeret(int gun, int knife):base(gun)
    {
        _knife = knife;
    }

    public void Stab()
    {
        Console.WriteLine("I can stab enemies");
    }
    public override void Run()
    {
        base.Run();
        Console.WriteLine("really fast! ");
    }
}
```

Casting operations

- **Up-casting:**
  - Cast from the base class type to the derived type.
  - Up-casting implicit and is safe.
- **Down-casting:**
  - Cast
  - Down-casting is explicit cast and is potentially unsafe
  - Be very aware that explicit casting is evaluated at runtime, not compile time.

Casting operations

- **Ex:**




```
public class Shape
{
   protected int m_xpos;
   protected int m_ypos;

   public Shape()
   {
   }

   public Shape(int x, int y)
   {
      m_xpos = x;
      m_ypos = y;
   }

   public virtual void Draw()
   {
      Console.WriteLine("Drawing a SHAPE at {0},{1}", m_xpos, m_ypos);
   }
}
```

Casting operations

- Up-casting

```
Shape s = new Circle(100, 100);
```

- Down-casting

```
Shape s = new Circle(100, 100);
s.fillCircle();
```

```
Circle c= (Circle)s;
s.fillCircle()
```

```
//also write 2 lines above to
((Circle)s).fillCircle()
```

```
public class Circle : Shape
{
   public Circle()
   {
   }

   public Circle(int x, int y) : base(x, y)
   {
   }

   public override void Draw()
   {
      Console.WriteLine("Drawing a CIRCLE at {0},{1}", m_xpos, m_ypos);
   }
public void FillCircle()
{
   Console.WriteLine("Filling CIRCLE at {0},{1}", m_xpos, m_ypos);
}
```

as keyword

- How to make your program without a runtime exception?
- C# provides the as keyword to quickly determine at runtime whether a given type is compatible with another by checking against a null return value

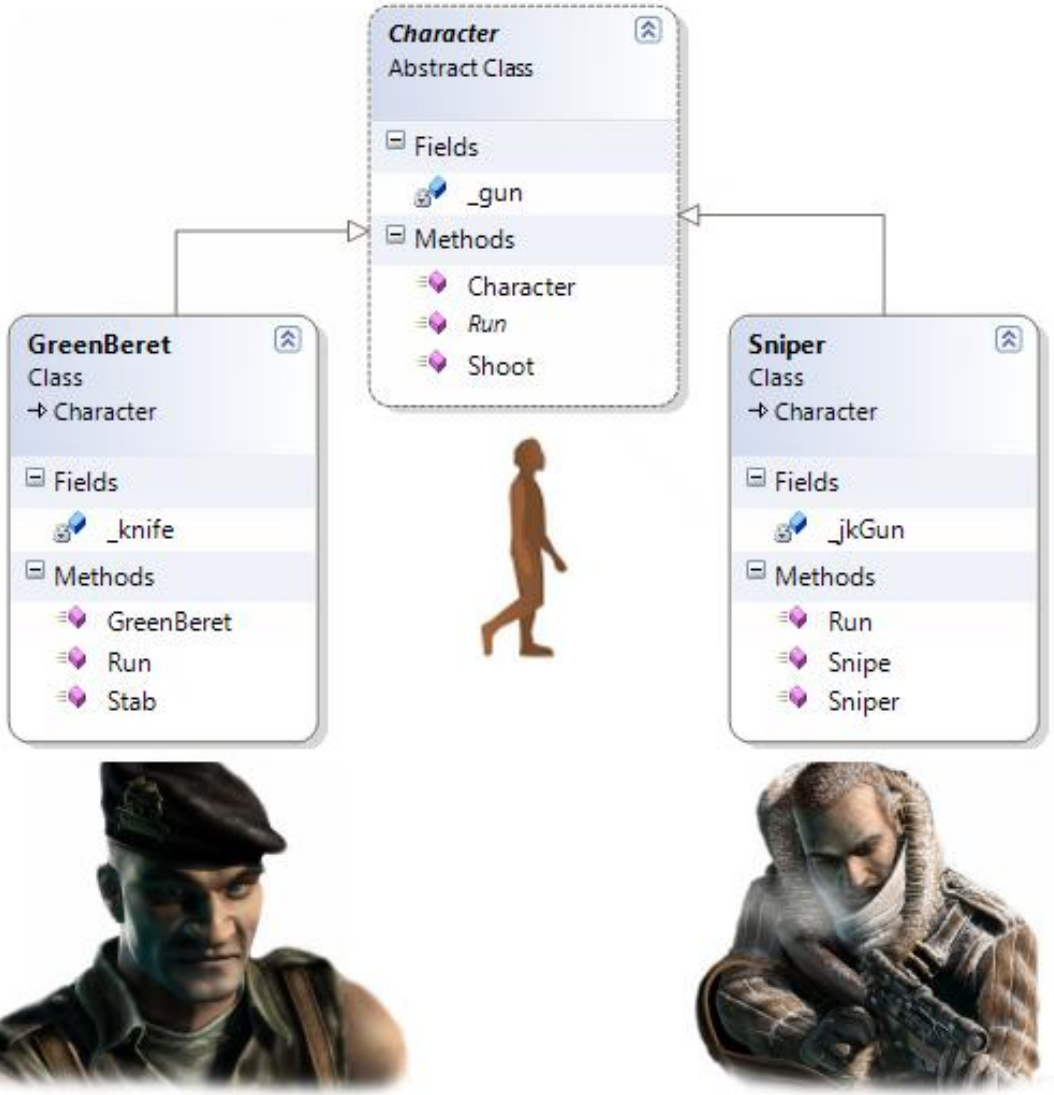
```
Circle c = shape as Circle;
If(c!=null)
  c.FillCircle();
```

C# is keyword

- is keyword to determine whether two items are compatible
- the is keyword returns false if the types are incompatible

```
foreach (Shape shape in shapes)
{
   shape.Draw();

   if (shape is Circle)
      ((Circle)shape).FillCircle();

      if (shape is Square)
         ((Square)shape).FillSquare();
}
```

Introdution to abstract class



Definition

- Classes can be declared as abstract by putting the keyword abstract before the class definition.

C#

```
public abstract class A
{
    // Class members here.
}
```

- Abstract classes may also define abstract methods that are incomplete and **must** be implemented in a derived class.

C#

```
public abstract class A
{
    public abstract void DoWork(int i);
}
```

Purpose

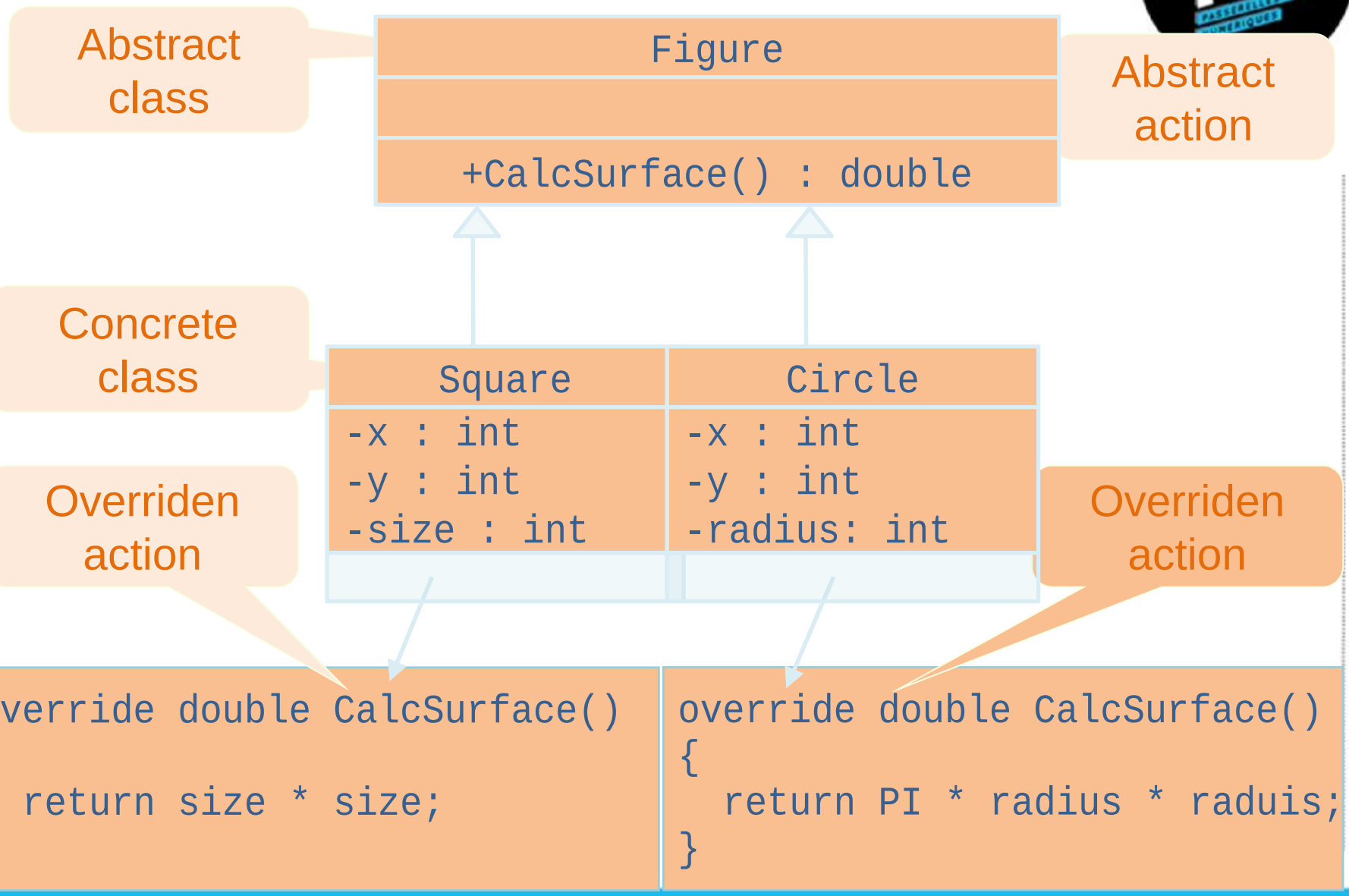
- An abstract class cannot be instantiated.
- The purpose of an abstract class is to provide a **common definition** of a base class that multiple derived classes can share.
- To implement polymorphism

Snippet


```
public abstract class Character
{
    private int _gun;

    public Character(int gun)
    {
        _gun = gun;
    }

    public void Shoot()
    {
        Console.Write("I can shoot enemies");
    }

    public abstract void Run();

}
```

```
public class GreenBeret : Character
{
    private int _knife;

    public GreenBeret(int gun, int knife):base(gun)
    {
        _knife = knife;
    }

    public void Stab()
    {
        Console.WriteLine("I can stab enemies");
    }
    public override void Run()
    {
        Console.WriteLine("I can run really fast");
    }
}
```

Example


Abstract class
Figure
+CalcSurface() : double
Abstract action
Concrete class
Square
-x : int
-y : int
-size : int
Circle
-x : int
-y : int
-radius: int
Overriden action
Overriden action
override double CalcSurface()
{
return size * size;
}
override double CalcSurface()
{
return PI * radius * raduis;
}

Exercises

- **Define class `Human` with first name and last name. Define new class `Student` which is derived from `Human` and has new field – `grade`. Define class `Worker` derived from `Human` with new field `weekSalary` and work-hours per day and method `MoneyPerHour()` that returns money earned by hour by the worker. Define the proper constructors and properties for this hierarchy. Initialize an array of 10 students and sort them by grade in ascending order. Initialize an array of 10 workers and sort them by money per hour in descending order.**