Session - 8

Polymorphism

Session Objectives

- Polymorphism and its Pre-requirements.
- Dynamic binding.
- Polymorphic Function.
- Accessing methods and field.
- Virtual methods in Java.
- Class pointers in C++.
- Polymorphism in C#.
- Dynamic v/s Static binding.

Polymorphism

- *Polymorphism* comes from Greek meaning “many forms.”
- There are three basic types of polymorphism
  1. Ad hoc polymorphism [Method Overloading]
  2. Parametric polymorphism [Template or Generic Type]
  3. Subtyping [Method Overriding]
- In Java, polymorphism refers to the dynamic binding mechanism that determines which method definition will be used when a method name has been overridden.
- Thus, polymorphism refers to dynamic binding.

Late Binding/Dynamic Binding

- Late binding or dynamic binding (run-time binding):
  - Method to be executed is determined at execution time, not compile time
- Polymorphism: to assign multiple meanings to the same method name
- Implemented using late binding
- Method overloading is resolved by the compiler (*early binding/static binding*)

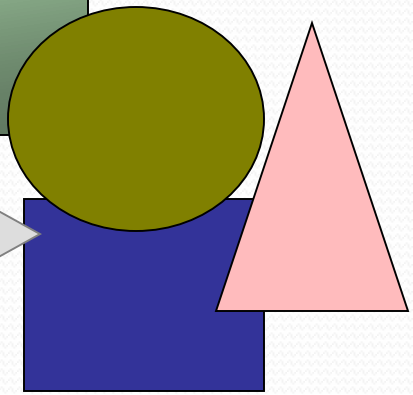
Polymorphic Function

- A **Polymorphic** function is one that has the **same name** for different classes of the same family, but has **different implementations/behaviour** for the various classes.
- In other words, polymorphism means **sending the same message (invoke/call member function)** to **different objects** of different classes in a hierarchy

Polymorphism and Method Overriding

Polymorphism is nothing but the ability of methods taking more than one form

Measuring Area of all these objects



Method overriding is one of the way to implement Polymorphism in object oriented technology

Cont….

- There are 3 pre-requisite before we can apply polymorphism:
  1. Having a hierarchy of classes/implementing inheritance
  2. Having functions with same signatures in that hierarchy of classes, but each function in each class is having different implementation (function definition)
  3. Would like to use base-class pointer that points to objects in that hierarchy

Polymorphism [Methods & Fields]

- An object of a given class can have multiple forms: either as its declared class type, or as any subclass of it
- An object of an extended class can be used wherever the original class is used
- **Question**: given the fact that an object's actual class type may be different from its declared type, then when a method accesses an object's member which gets redefined in a subclass, then which member the method refers to (subclass's or super class's)?
  - When you invoke a method through an object reference, the *actual class of the object* decides which implementation is used.
  - When you access a field, the declared type of the reference decides which field to access.

Example

```
class SuperShow {
    public String str = “SuperStr”;

    public void show( ) {
            System.out.println(“Super.show:” + str);
    }
}

class ExtendShow extends SuperShow {
    public String str = “ExtendedStr”;

    public void show( ) {
            System.out.println(“Extend.show:” + str);
    }

    public static void main (String[] args) {
      ExtendShow ext = new ExtendShow( );
      SuperShow sup = ext;
      sup.show( );
      ext.show( );
      System.out.println(“sup.str = “ + sup.str);
      System.out.println(“ext.str = “ + ext.str);
    }
}
```

Output:

Extend.show: ExtendStr
Extend.show: ExtendStr
sup.str = SuperStr
ext.str = ExtendStr

Virtual methods in Java

- In **Java**, all **non-static** methods are by default "**virtual functions.**"
- Only methods marked with the keyword **final**, which cannot be overridden, along with **private methods**, which are not inherited, are **non-virtual**.

Pointer example in c++

Pointers in classes

C++ Example

```
void main()
{
  Manager mgr;
  Employee* emp = &mgr;
  //valid: every Manager is an
    Employee

  Employee emp1;
  Manager* man = &emp1;
  //error: not every Employee
    is a Manager
}
```

C# & Java Example

```
void main()
{
    Employee emp = new Manager();
//valid: every Manager is an Employee

  Manager man = new Employee();
//error: not every Employee is a
    Manager
}
```

Polymorphism in C#

Method Overriding [using new]

- To override an existing method of the base class:
  - Declare a new method in the inherited class of the same name.
  - Prefix it with the **new** keyword.

Method Overriding [using new]

```
using System;

class A
{
public void Driver()
        {
        Console.WriteLine("This  is  the  Driver  method  of  the  class
A");
        }
}

class B : A
{
new public void Driver()
        {
        Console.WriteLine("This  is  the  Overridden  Driver  method  of
the class B");
        }
}

class Test
{
public static void Main()
        {
        B objB = new B();

        objB.Driver();
        }
}
```

```
This is the Overridden Driver method of the class B
```

Cont….

```
using System;
class A
    {
        public void driver()
        {
            Console.WriteLine("Driver from A");
        }
    }
    class B:A
    {
       new public void driver()
        {
            Console.WriteLine("Driver from B");
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            A a = new B();
            a.driver();
        }
    }
```

Driver from A

Method Overriding [using virtual & override]

- In C# & C++ Polymorphism is achieved using virtual methods.

virtual *return_type functionName(argument list);*

Method Overriding [using virtual & override]

```
class A
   {
      virtual public void driver()
      {
         Console.WriteLine("Driver from A");
      }
   }
   class B:A
   {
     override public void driver()
      {
         Console.WriteLine("Driver from B");
      }
   }
   class Program
   {
      static void Main(string[] args)
      {
         A a = new B();
         a.driver();
      }
   }
```

Driver from B

Polymorphism in C#

- In C# & C++ Polymorphism is achieved using virtual methods.

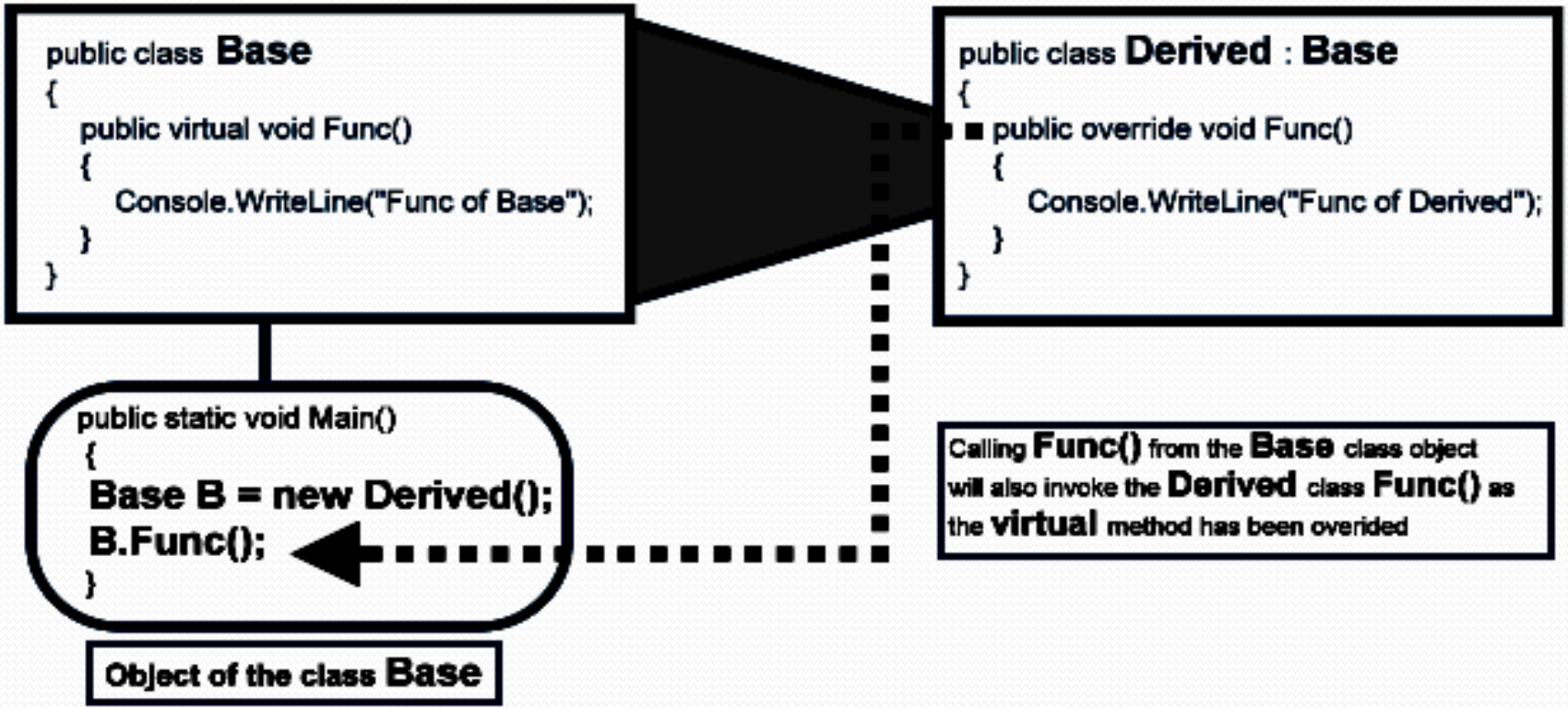
```
class DrawObj
{
    public virtual void Draw()
    {
        System.Console.WriteLine("This is the Virtual Draw
method");
    }
}
```

- Polymorphism allows us to implement the derived class methods during runtime.

- virtual -> override
- non-virtual -> redefine

Polymorphism in C#

- Virtual functions come in handy when we need to call the derived class method from an object of the base class.



Polymorphism in C#

```
public class Line : DrawObj
{
     public override void Draw()
     {
          System.Console.WriteLine("This is the Draw() method of
line");
     }
}

public class Circle : DrawObj
{
     public override void Draw()
     {
          System.Console.WriteLine("This is the Draw() method of
Circle ");
     }
}

public class Square : DrawObj
{
     public override void Draw()
     {
          System.Console.WriteLine("This is the Draw() method of
Square ");
     }
}
```

Polymorphism in C#

```
public class Test
{
    public static void Main()
    {
        DrawObj[] ObjD = new DrawObj[4];

        ObjD[0] = new DrawObj();
        ObjD[1] = new Line();
        ObjD[2] = new Circle();
        ObjD[3] = new Square();

        foreach (DrawObj IterateD in ObjD)
        {
            IterateD.Draw();
        }
    }
}
```

```
This is the Virtual Draw method
This is the Draw() method of line
This is the Draw() method of Circle
This is the Draw() method of Square
```

Polymorphism in C#

```
class A
{
public int MethodA()
      {
      return(MethodB()*MethodC());
      }

public virtual int MethodB()
      {
      return(10);
      }

public int MethodC()
      {
      return(20);
      }
}

class B : A
{
public override int MethodB()
      {
      return(30);
      }
}

class Test
{
public static void Main()
      {
B ObjB = new B();
System.Console.WriteLine(ObjB.MethodA());
      }
}
```

Output

600

Static binding

- *Static binding* means that the legality of a member function invocation is checked at the earliest possible moment: by the compiler at compile time.
- The compiler uses the static type of the pointer to determine whether the member function invocation is legal.

Dynamic binding

- *Dynamic binding* means that the legality of a member function invocation is determined at the last possible moment: based on the dynamic type of the object at run time.
- It is called "dynamic binding" because the binding to the code that actually gets called is accomplished dynamically (at run time).

Example Static / Dynamic

Polymorphic Pointers

- A reference of a parent class is allowed to point to an object of the child class. E.g.

```
class Vehicle {
    // ...
}
class Car : Vehicle {
    // ...
}
// ...
    Vehicle vp = new Car();
```

Overriding Methods

- Methods in the parent class can be redefined in the child class.

```
class Vehicle {
        void move(int i){....}
    }
    class Car : public Vehicle {
        void move(int i) {....}
    }
    // ...
        Vehicle vp = new Car();
        vp.move(100);
```

Overriding Methods

- Methods in the parent class can be redefined in the child class.

```
class Vehicle {
        void move(int i){...}
    }
    class Car : public Vehicle {
        void move(int i){...}
    }
    // ...
        Vehicle vp = new Car();
        vp.move(100);
```

BUT:

- Which of these two move() methods will be called?

Overriding Methods

- Methods in the parent class can be redefined in the child class.

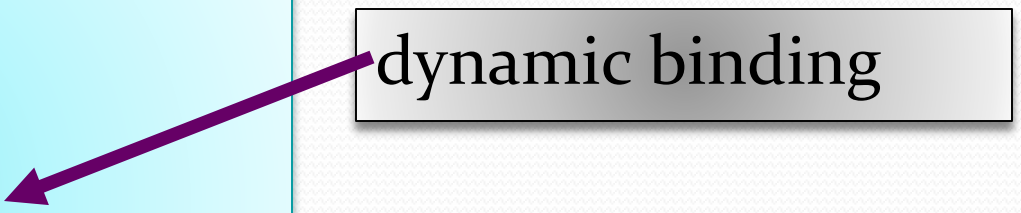
```
class Vehicle {
        void move(int i){.....}
    }
    class Car : public Vehicle {
        void move(int i){......}
    }
    // ...
        Vehicle vp = new Car();
        vp.move(100);
```

static binding

Overriding Methods

- Methods in the parent class can be redefined in the child class.

```
class Vehicle {
        void move(int i){.....}
    }
    class Car : Vehicle {
        void move(int i){......}
    }
    // ...
        Vehicle vp = new Car();
        vp.move(100);
```



Overriding Methods

- Methods in the parent class can be redefined in the child class.

```
class Vehicle {
        void move(int i){.....}
    }
    class Car : Vehicle {
       new void move(int i){.....}
    }
    // ...
        Vehicle vp = new Car();
        vp.move(100);
```

static binding!

- Without virtual keyword

As vp is of type pointer to a Vehicle, the method of the Vehicle is called.

Overriding Methods -The virtual keyword

- Methods in the parent class can be redefined in the child class.

```
class Vehicle {
        virtual  void move(int i){.....}
      }
      class Car : Vehicle {
        override void move(int i){......}
      }
      // ...
          Vehicle vp = new Car();
          vp.move(100);
```

dynamic binding!

The keyword virtual allows the use of dynamic binding.

As vp points to a Car object the method of the Car is called

Advantage of Polymorphism [Software Extension]

- Polymorphism promotes extensibility: Software that invokes polymorphic behavior is independent of the object types to which messages are sent.
- New object types that can respond to existing method calls can be incorporated into a system without requiring modification of the base system.
- Only client code that instantiates new objects must be modified to accommodate new types.
- It allows system to evolve over time, meeting the needs of a ever-changing application