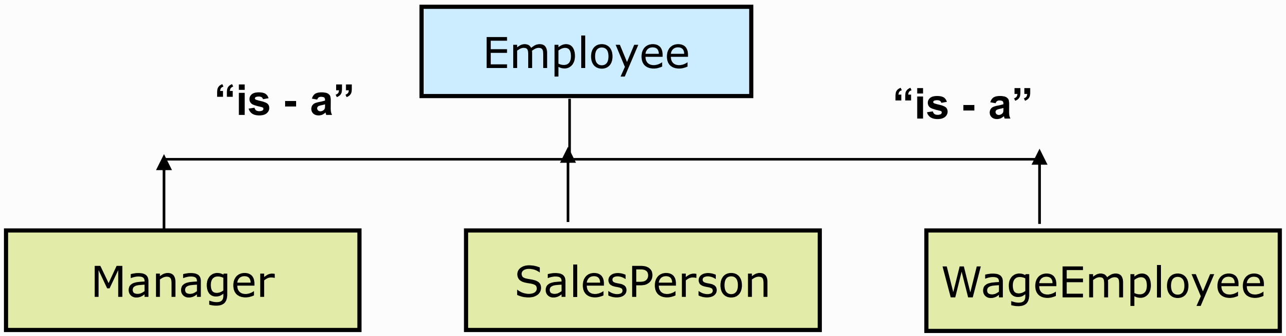
Inheritance and Polymorphism

Objectives

- On completion of this Session you will be able to
  - Implement Inheritance & Polymorphism
  - Write Shadowing methods
  - Define a sealed class
  - Differentiate abstract Class & interface
  - Implement FCL interfaces
  - Write Custom Exception classes

Inheritance

- Provides code reusability and extensibility.
- Is a property of class hierarchy whereby each derived class inherits attributes and methods of its base class.


Employee

“is - a”

“is - a”

Manager

SalesPerson

WageEmployee

Example of Inheritance

```
class Employee
 {
     public double CalculateSalary()
      {return basic_sal + hra + da ; }
 }
class Manager : Employee
{ public double CalculateIncentives()
  {
   //code to calculate incentives
  return incentives ;
   }
}
```

```
static void Main(string[] args)
{
 Manager mngr = new Manager();
 double inc=mngr.CalculateIncentives();
 double Sal=mngr.CalculateSalary();
 Console.WriteLine(“Incentives
            "+inc+".....SALARY="+Sal );
}
```

Visibility

Access Specifier In Base class	Accessibility Within Assembly	Accessibility Outside Assembly
private	X	X
public	√	√
protected	√	√
internal	√	X
protected internal	√	√

Constructor in Inheritance

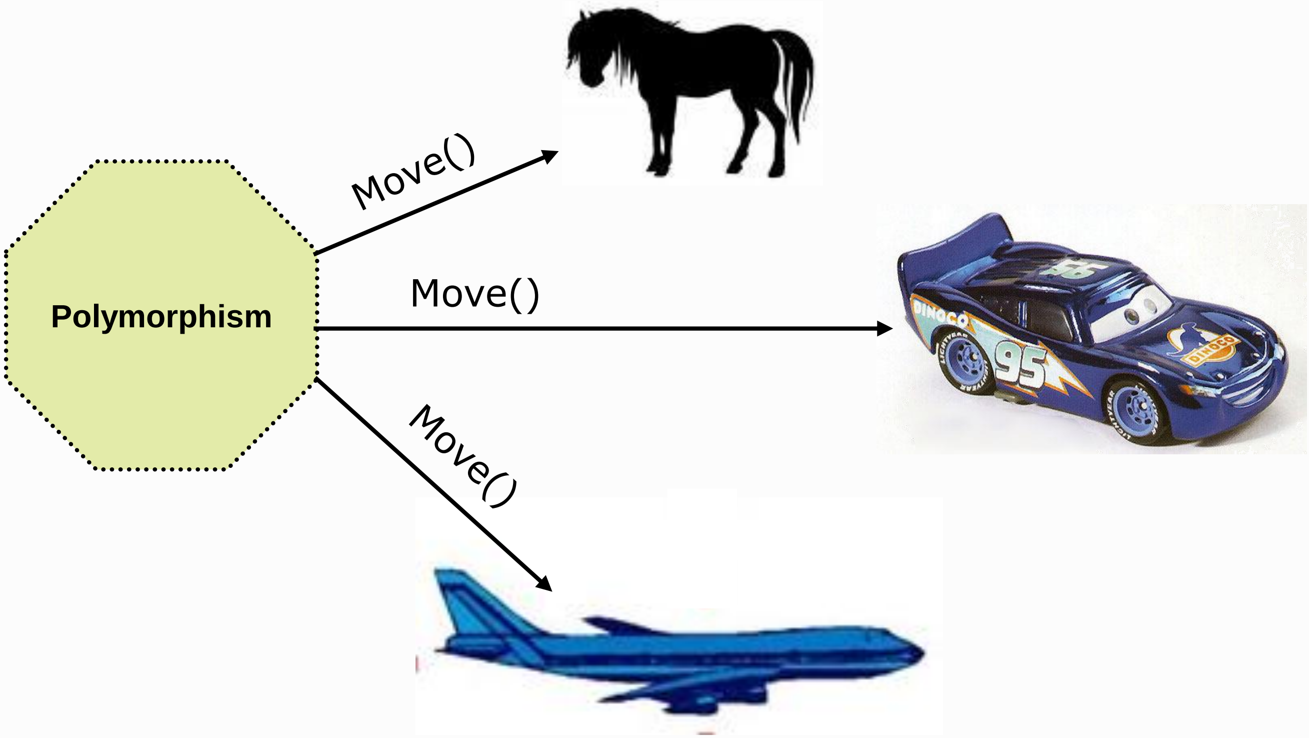
```
public class Employee
 {
   public Employee()
    { Console.WriteLine(“In default constructor"); }
   public Employee(int eid,. . .)
    { Console.WriteLine(“In parameterized constructor"); }
 }
public class Manager : Employee
{
   public Manager() : base()
   {… }
   public Manager(int id) : base(id,. . .)
   {… }
}
```

This will call base class default constructor.

This will call base class parameterized constructor

Polymorphism (Late Binding)

- Ability of different objects to responds to the same message in different way is called Polymorphism.



Virtual and Override

- Polymorphism is achieved using virtual functions and inheritance.
- `virtual` keyword is used to define a method in base class and `override` in derived class is to extend base class behavior.

```
class Employee
{
 public virtual double CalculateSalary()
 {return basic_sal + hra + da; }
}
class Manager : Employee
{  public override double CalculateSalary()
   {return (basic_sal + hra + da + allowances);}
}
```

```
static void Main(string[] args){
  Employee mngr = new Manager();
  double Salary = mngr.CalculateSalary();
  Console.WriteLine(Salary);
  Console.ReadLine();}
```

Shadowing

- Hides a base class member in the derived class by using the keyword `new`.

```
class Employee
{
 public virtual double CalculateSalary()
  { return basic_sal; }
}
class SalesPerson: Employee
{ double sales,comm;
 public new double CalculateSalary()
  {
  return basic sal+ (sales * comm);
  }
}
```

```
static void Main(string[] args)
{
   SalesPerson sper = new SalesPerson();
   double sal = sper.CalculateSalary;
   Console.WriteLine(sal );
}
```

Sealed class

- Sealed class can not be inherited.

```
sealed class SinglyList
{
 public virtual double Add()
  {
   //code to add a record in the linked list
  }
}
```

```
public class StringSinglyList:SinglyList
{
 public override double Add()
  {
    //code to add a record in the string linked list
  }
}
```

Concrete and Abstract class

- Concrete class
  - Class describes the functionality of the objects that it can be used to instantiate.
- Abstract class
  - Provides all of the characteristics of a concrete class except that it does not permit objects of the type to be created.
  - An abstract class can contain abstract and non-abstract methods.
  - Abstract methods do not have implementation.

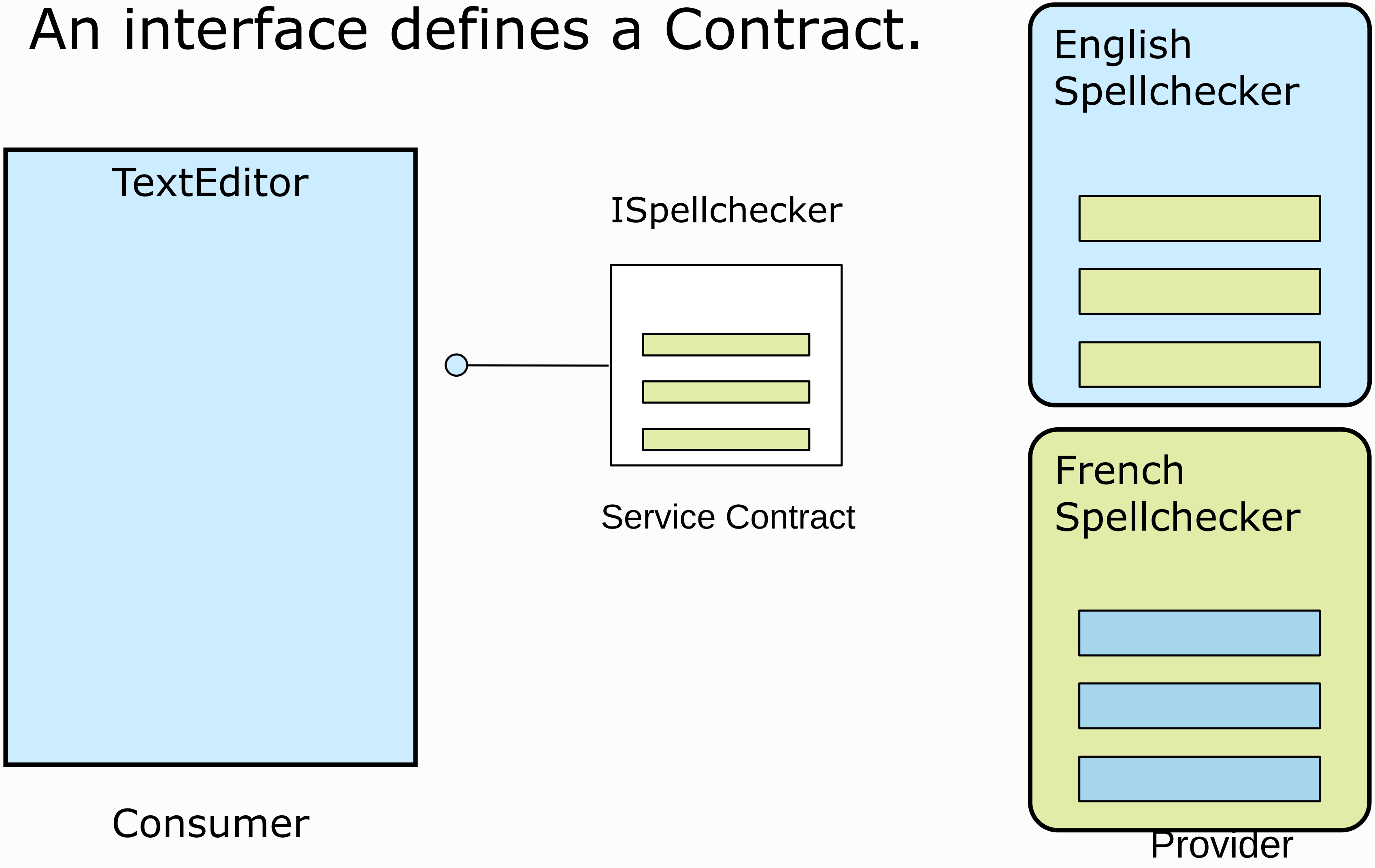
Example of Abstract class

```
 abstract  class Employee
{  public virtual double CalculateSalary()
    {return basic + hra + da;}
   public abstract double CalculateBonus();
}
class Manager:Employee
{ public override double CalculateSalary()
    {return basic+ hra + da +allowances ; }
  public override double CalculateBonus()
  {
    return basic_sal * .20;
  }
}
```

```
static void Main()
{
Manager  mngr = new Manager();
double bonus = mngr.CalculateBonus();
double Salary =mngr.CalculateSalary();
}
```

Interface

- An interface defines a Contract.



Implementing Interface

```
interface ISpellchecker
 {
   ArrayList CheckSpelling(string word);
 }
```

```
class EnglishSpellchecker : ISpellchecker
{
  ...
 ArrayList CheckSpelling(string word)
   {
     // return possible spelling suggestions.
   }
}
```

```
class FrenchSpellchecker : ISpellchecker
{
  ...
 ArrayList CheckSpelling(string word)
   {
     // return possible spelling suggestions.
   }
}
```

Using Interface

```
interface ISpellchecker
 {
   ArrayList CheckSpelling(string word);
 }
```

```
class TextEditor
{

public  static void Main ()
{
      ISpellchecker checker=new
                        EnglishSpellchecker();
      ArrayList words=
            Checker.CheckSpelling(“contract”);
…
…
}
}
```

Explicit Interface implementation

```
interface IOrderDetails { void ShowDetails(); }
interface ICustDetails { void ShowDetails(); }

class Transaction : IOrderDetails, ICustDetails
{
 void IOrderDetails.ShowDetails()
 { // implementation for interface IOrderDetails );
 }
 void ICustDetails. ShowDetails()
 {// implementation for interface ICustDetails}
 }
}
```

```
static void Main()
{
Transaction obj = new
      Transaction();
IOrderDetails OD = obj;
OD.ShowDetails();
ICustDetails CD = obj;
CD.ShowDetails();
}
```

Abstract class Vs. Interface

	Abstract class	Interface
Methods	At least one abstract method	All methods are abstract
Best suited for	Objects closely related in hierarchy.	Contract based provider model
Multiple Inheritance	Not supported	supported
Component Versioning	By updating the base class all derived classes are automatically updated.	Interfaces are immutable

Building Cloned Objects

```
class StackClass:ICloneable
{ int size;
 int[] sArr;
 public StackClass(int s)
 {
   size=s;
   sArr =  new int[size];
 }
public object Clone()
 {
   StackClass  s = new StackClass(this.size);
   this.sArr.CopyTo(s.sArr, 0);
   return s;
 }
}
```

```
public static void Main()
{
StackClass stack1 = new StackClass(3);
stack1[0] = 10;
. . .
StackClass stack2 = (StackClass)stack1.Clone();
}
```

User Defined Exception class

- Application specific exception class can be created using ApplicationException class.

```
class StackFullException:ApplicationException
{
 public string message;
 public StackFullException(string msg)
 {
    message = msg;
 }
}
```

```
public static void Main(string[] args)
{
StackClass stack1 = new StackClass(2);
  try{
      stack1.Push(10);
      stack1.Push(20);
      stack1.Push(30);
     }
 catch (StackFullException s){
     Console.WriteLine(s.message);
    }
}
```

Quick Recap…

- Reusability and extensibility are the two main advantages of inheritance.
- Polymorphism can be achieved using inheritance and `virtual` keyword
- Shadowing hides the base class implementation.
- Sealed class is a non-inheritable class.
- Abstract class is best suited for objects closely related in hierarchy whereas interface for contract based provider model.
- Any class that implements `ICloneable` interface supports cloning.
- User defined Exception class can be written by deriving it from `ApplicationException` class.