

Agenda

- Defining Polymorphism
- Method Overloading
- Method Overriding
- Using *virtual* and *override* Keywords
- Name Hiding (Using *new* Keyword)
- Early Binding and Late Binding
- Implementing Polymorphism



Defining Polymorphism

- Polymorphism is referred to as the third pillar of Object Oriented programming.
- Polymorphism provides multiple implementations of the same object.
- Polymorphism is a Greek word that means “many-shaped.”
- Polymorphism means that many classes can provide the same property or method, and a caller doesn't have to know to which class an object belongs when calling the property or method.

Method Overloading

- Overloading a method means defining multiple versions, using the same name but a different parameter list.
- The purpose of overloading is to define several closely related versions of a method without having to differentiate them by name.
- In a class multiple methods are allowed with unique different signatures.
- Overloaded methods can be implemented in the same class or in a subclass.
- .NET can figure out which method to call during compile based on the parameter types that you pass. This technique is called overloading a method.
- Method overloading is a compile-time mechanism.

Rules of Method Overloading

- There are several rules of method overloading:
 - **Same Name:** Each overloaded version method must use the same method name.
 - **Different Signature:** Each overloaded version must differ from all other overloaded versions in at least following respects:
 - Number of Parameters.
 - Order of Parameters.
 - Data type of the parameters.
 - It can use different access modifiers.

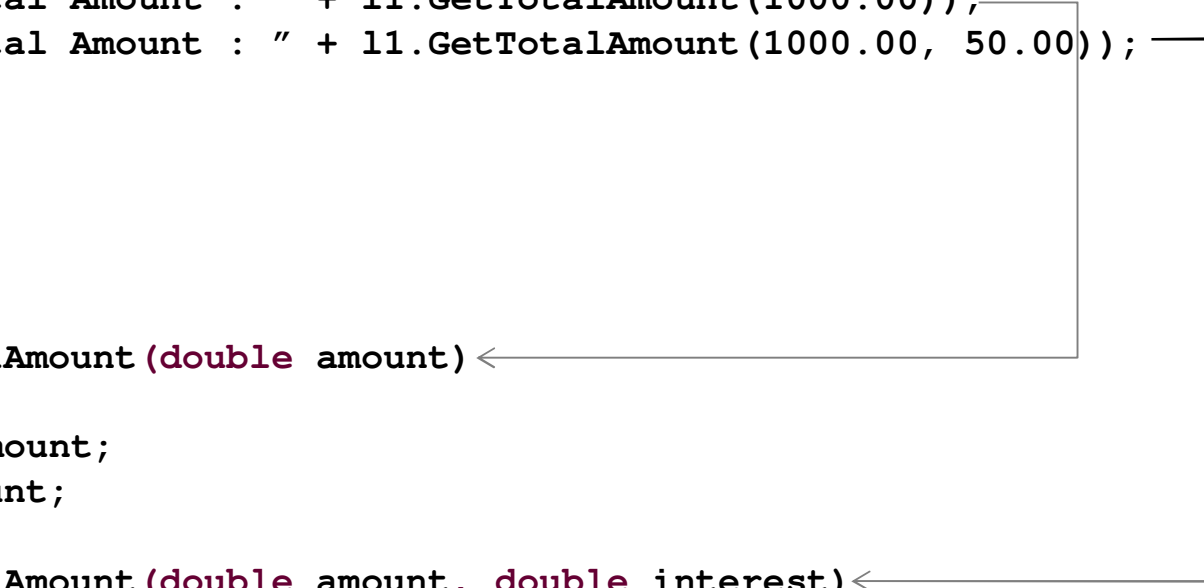
Implementing Method Overloading

```
public static void Main(string[] args)
{
    Loan l1 = new Loan();
    Console.WriteLine("Total Amount : " + l1.GetTotalAmount(1000.00));
    Console.WriteLine("Total Amount : " + l1.GetTotalAmount(1000.00, 50.00));
}

class Loan
{
    double totalamount;

    public double GetTotalAmount(double amount)
    {
        totalamount = amount;
        return totalamount;
    }

    public double GetTotalAmount(double amount, double interest)
    {
        totalamount = amount + interest;
        return totalamount;
    }
}
```



```
Total Amount : 1000.00
Total Amount : 1050.00
```

Method Overriding

- An overridden method provides new implementation of a member that is inherited from *base class*.
- The overridden base method must have the same signature as the override method.
- Overriding method call is decided at runtime.

Rules of Method Overriding

- An overridden method must have:
 - the same name.
 - the same number of parameters and types.
 - the same return type.
- Overriding a method cannot narrow the method access level defined in the overridden method.
- Methods declared as private, static, or sealed cannot be overridden.
- For a method to be overridable without any compilation error/warning, it should be marked as virtual, abstract, or override.
- A static method cannot override an instance method.

Implementing Method Overriding (Using Virtual Overrides)

```
public static void Main(string[] args)
{
    Loan l1 = new Loan();
    Loan l2 = new PersonalLoan();

    Console.WriteLine(l1.GetTotalAmount(1000.00));
    Console.WriteLine(l1.GetTotalAmount(1000.00, 50.00));
    Console.WriteLine(l2.GetTotalAmount(1000.00));
}

class Loan
{
    double totalamount;
    public virtual double GetTotalAmount(double amount)
    {
        return totalamount = amount;
    }
    public double GetTotalAmount(double amount, double interest)
    {
        totalamount = amount + interest;
        return totalamount;
    }
}

class PersonalLoan : Loan
{
    double amount;
    public override double GetTotalAmount(double amount)
    {
        this.amount = amount + 25.00;
        return this.amount;
    }
}
```

Define *Loan* reference variable containing *PersonalLoan* object.

It determined which method to run based on *object type* (*PersonalLoan*) instead of *reference type* (*Loan*).

```
1000.00
1050.00
1025.00
```


Name Hiding (Using *new*)

```
public static void Main(string[] args)
{
    Loan l1 = new Loan();
    Loan l2 = new PersonalLoan();

    Console.WriteLine(l1.GetTotalAmount(1000.00));
    Console.WriteLine(l1.GetTotalAmount(1000.00, 50.00));
    Console.WriteLine(l2.GetTotalAmount(1000.00));
}

class Loan
{
    double totalamount;
    public virtual double GetTotalAmount(double amount)
    {
        return totalamount = amount;
    }
    public double GetTotalAmount(double amount, double interest)
    {
        totalamount = amount + interest;
        return totalamount;
    }
}

class PersonalLoan : Loan
{
    double amount;
    public new double GetTotalAmount(double amount)
    {
        this.amount = amount + 25.00;
        return this.amount;
    }
}
```

Define *Sales* reference variable containing *SalesTax* object.

It determined which method to run based on *object type* (*SalesTax*) instead of *reference type* (*Sales*)

```
1000.00
1050.00
1000.00
```

Overloading vs. Overriding

- Overloaded method:
 - Overloaded methods should have different argument lists.
 - Overloaded methods can have a change in the return type.
 - Overloaded methods can have a change in the access level.
 - Overloaded method invocation is decided at compile time.
- Overridden method:
 - Overridden methods should have the same argument list.
 - Overridden methods should have the same return type.
 - Overridden methods can have the same access level and cannot be narrower.
 - Overridden method invocation is decided at run time.

Early Binding and Late Binding

- Early binding means translating operations or associating identifiers during compile time.
- Late binding means delaying translation of an operation or associating identifiers at runtime (also known as dynamic method lookup or virtual method invocation):
 - It is used in polymorphism to determine the actual method invoked (depending on the type of the actual object).

Implementing Polymorphism

```
class Loan
{
    public virtual void Display() {Console.WriteLine("Base Loan Class method
        invoked...");}
}
class PersonalLoan : Loan
{
    public override void Display() {Console.WriteLine("PersonalLoan class method
        invoked...");}
}
class HomeLoan : Loan
{
    public override void Display() {Console.WriteLine("HomeLoan class method
        invoked...");}
    public void Display(string s) {Console.WriteLine("HomeLoan class overloaded
        method invoked with argument {0}" + s);}
}
public static void Main(string[] args)
{
    Loan l1 = new Loan();
    PersonalLoan pl1 = new PersonalLoan();
    HomeLoan hl1 = new HomeLoan();
    Loan l2 = new PersonalLoan();
    Loan hl2 = new HomeLoan();
}
```

Implementing Polymorphism (cont.)

```
l1.Display();           // will this compile? Output?  
l1.Display("leftovers");  
pl1.Display();          // will this compile? Output?  
pl1.Display("chicken");  
hl1.Display();          // will this compile? Output?  
hl1.Display("grass");   // will this compile? Output?  
pl2.Display(); // will this compile? Output?  
pl2.Display("rat");  
hl2.Display(); // will this compile? Output?  
hl2.Display("carrots");  
}
```

Base Loan class method is invoked...

PersonalLoan class method is invoked...

HomeLoan class method is invoked...

PersonalLoan class method is invoked...

HomeLoan class method is invoked...

HomeLoan class overload method is invoked...