Polymorphism Using Inheritance

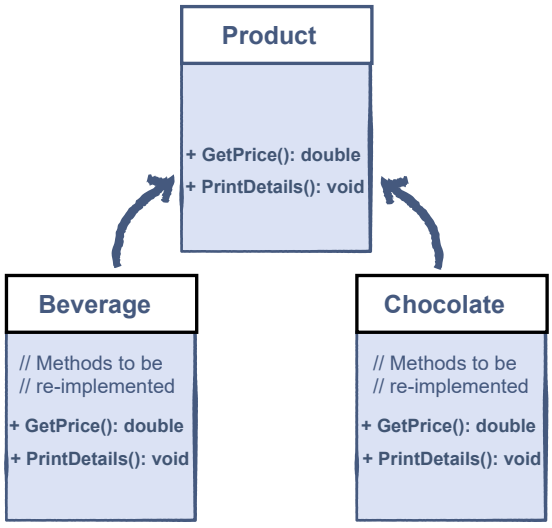
In this lesson, we will be implementing polymorphism using inheritance principles.

WE'LL COVER THE FOLLOWING

Let's move a step further and implement polymorphism using the example discussed in the previous lesson.

Example

Consider the example of a `Product` class, which is the base class. Extending from this base class are the derived classes *Beverages and Chocolates*. These classes contain the **GetPrice()** method that calculates the selling price for the respective product.



`virtual` and `override` Keywords

We already have discussed what we need our **GetPrice()** method to do.

To achieve this, we need the parent class to allow the re-implementation of this inherited method. The base class allows this by using the keyword `virtual` in its method signature like this:

```
public virtual double GetPrice(){...}
```

On the other hand, when the derived class re-implements this inherited method, it will let the compiler know that I'm overriding the inherited method by using the `override` keyword in its method signature like this:

```
public override double GetPrice(){...}
```

Don't worry too much about the method overriding concept at this point as we are going to discuss it thoroughly in the next lesson.

Implementation

Let's start by implementing the **Base** class first and then move on to the **Derived** classes.

Product Class

The `Product` class has two public methods namely, `GetPrice()` and `PrintDetails()` which are to be re-implemented by the sub-classes. So, we will allow them to be overridden by using the keyword `virtual` in their signatures.

Let's look at the implementation of the **Product** class:

```
class Product
{

    private string _name;
    private double _purchasePrice;

    // Parameterized Constructor
    public Product(string name, double purchasePrice)
    {
        this._name = name;
        this._purchasePrice = purchasePrice;
    }

    // Getter to access the purchase price of products
```

```
    public double GetPurchasePrice()
    {
        return this._purchasePrice;
    }

    // Method GetPrice() to be re-implemented in the derived classes
    public virtual double GetPrice()
    {
        return 0;
    }

    // Method to print name and to be re-implemented in the derived classes for selling price
    public virtual void PrintDetails()
    {
        Console.WriteLine("Selected Product's Name: " + this._name());
    }

}
```

Beverage Class

Now, consider the `Beverage` class which is extended from the `Product` class. It has two fields, i.e., `_refCost` and `_profit`. It calculates and returns the *selling price* of a beverage by using the `GetPrice()` method according to the following equation:

Selling price = Purchase price + 10% of Purchase price + 15% of Purchase price

Also, we have to use the `override` keyword to let the compiler know that we want to access this class' `GetPrice()` and `PrintDetails()` methods rather than the inherited one.

Let's look at the implementation of the **Beverage** class:

```
class Beverage : Product
{
    // Private fields for refregeration cost and our profit
    private double _refCost;
    private double _profit;

    // Parameterized Constructor
    public Beverage(string name, double price)
        : base(name,price)
    {
        this._refCost = GetPurchasePrice() * 0.10; // 10% of purchase price
        this._profit = GetPurchasePrice() * 0.15; // 15% of purchase price
    }

    // public method to get selling price
    public override double GetPrice()
    {   //calculating selling price, Math.round is just an ibuilt method to round off the pri
```

```
        return (GetPurchasePrice() + (int)Math.Round(this._refCost) + (int)Math.Round(this._p
    }


    // public method to call the base method for name and print the selling price from this c
    public override void PrintDetails()
    {
        base.PrintDetails();
        Console.WriteLine("Selling price: {0}", this.GetPrice());
    }

}
```

Chocolate Class

Now, consider the `Chocolate` class which inherits from the `Product` class. It has only one field, i.e., `_profit` and it returns the *selling price* of a chocolate by using the `GetPrice()` method. The only difference in this class and the `Beverage` class is the absence of `refCost` and an increased percentage of `_profit`.

Let's look at the implementation of the **Chocolate** class:

```
class Chocolate : Product
{

    private double _profit;

    // Parameterized Constructor
    public Chocolate(string name, double price)
        : base(name, price)
    {
        this._profit = base.GetPurchasePrice() * 0.20; // 20% of purchase price
    }


    // public method to get selling price
    public override double GetPrice()
    {   //calculating selling price, Math.ceiling is just an ibuilt method to round off the p
        return (base.GetPurchasePrice() + (int)Math.Round(this._profit));
    }

    // public method to call the base method for name and print the selling price from this c
    public override void PrintDetails()
    {
        base.PrintDetails();
        Console.WriteLine("Selling price: {0}", this.GetPrice());
    }

}
```

Complete Program

Now, let's see what happens by merging all the classes and calling the

`GetPrice()` method inside the `PrintDetails()` method:

```
class Product
{
    private string _name;
    private double _purchasePrice;

    // Parameterized Constructor
    public Product(string name, double purchasePrice)
    {
        this._name = name;
        this._purchasePrice = purchasePrice;
    }

    // Getters
    public string GetName()
    {
        return this._name;
    }

    public double GetPurchasePrice()
    {
        return this._purchasePrice;
    }
    // Method to calculate selling price
    public virtual double GetPrice()
    {
        return 0;
    }
    // Method to print details
    public virtual void PrintDetails()
    {
        Console.WriteLine("Selected Product's Name: " + this.GetName());
    }

}


class Beverage : Product
{


    private double _refCost;
    private double _profit;

    // Parameterized Constructor
    public Beverage(string name, double price)
        : base(name,price)
    {
        this._refCost = GetPurchasePrice() * 0.10; // 10% of purchase price
        this._profit = GetPurchasePrice() * 0.15; // 15% of purchase price
    }

     // public method to get selling price
    public override double GetPrice()
    {   //calculating selling price, Math.Round is just an inbuilt method to round off the pr
        return (GetPurchasePrice() + (int)Math.Round(this._refCost) + (int)Math.Round(this._p
    }

    public override void PrintDetails()
```

```
    {
        base.PrintDetails();
        Console.WriteLine("Selling price: {0}", this.GetPrice());

    }

}


class Chocolate : Product
{

    private double _profit;

    // Parameterized Constructor
    public Chocolate(string name, double price)
        : base(name, price)
    {
        this._profit = base.GetPurchasePrice() * 0.20; // 20% of purchase price
    }



    // public method to get selling price
    public override double GetPrice()
    {   //calculating selling price, Math.Round is just an inbuilt method to round off the pr
        return (base.GetPurchasePrice() + (int)Math.Round(this._profit));
    }


    public override void PrintDetails()
    {
        base.PrintDetails();
        Console.WriteLine("Selling price: {0}", this.GetPrice());
    }

}


class Demo
{

    public static void Main(string[] args)
    {
        // Placing the products in an array
        Product[] products = new Product[4];
        products [0] = new Beverage("Cola", 9);
        products [1] = new Chocolate("Crunch", 15);
        products [2] = new Chocolate("Kit-kat", 20);
        products [3] = new Beverage("Fanta", 8);

        // name and price of respective  product is displayed
        foreach(Product product in products)
        product.PrintDetails();

    }
}
```

Program Execution

In the `Main()` method, at **line 103**, we have declared an array of type `Product` and size **4**. Consider this array the shelf of our vending machine. The indices of the array are the marked locations of the products. So, we have placed 4 products in this array. Now, when the `PrintDetails()` method is called, the differently calculated selling price and name of the product placed at the respective location is displayed. We only have to remember this method's name and we are good to go. This is the beauty of **Polymorphism.**

In the next lesson, we'll be learning about the process of **method overriding**.