State Design Pattern in C#

The State Design Pattern falls under the category of [Behavioral Design Pattern](https://dotnettutorials.net/lesson/behavioral-design-pattern/).

What is the State Design Pattern?

Understanding the State Design Pattern with Real-Time example

Implementation of ATM Machine behavior using State Design Pattern in C#

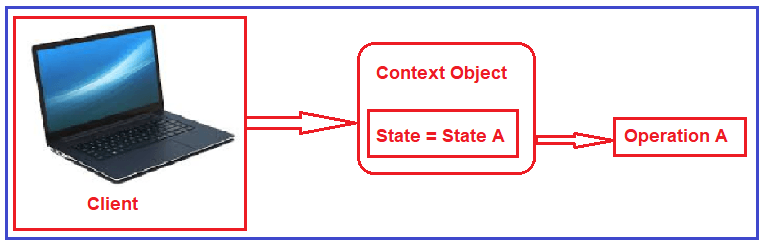
When to use the State Design Pattern in real-time applications?

What is the State Design Pattern?

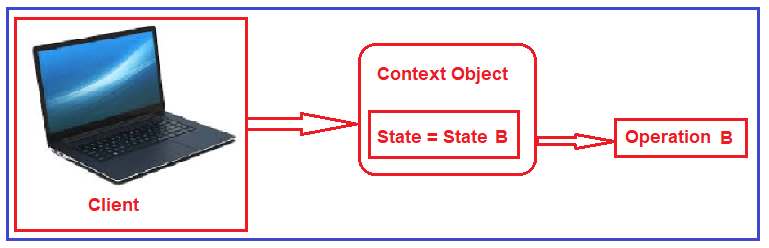
According to Gang of Four Definitions, the State Design Pattern allows an object to alter its behavior when its internal state changes. In simple words, we can say that the State Pattern is a design pattern that allows an object to completely change its behavior depending upon its current internal state.

For a better understanding of the above definition, please have a look at the following diagram. Here, you can see on the left-hand side we have the client and on the right-hand side we have the Context object and the context object internally maintains some states (such as State A and State B).

As you can see in the below diagram, the current state of the Context object is State A. So, when the client makes a request to the Context object, what the Context object will do is, it will perform Operation A.



Let us say the state of the context object is changed to State B. Now when the request is coming from the client, the context object will perform operation B as the current state us State B as shown in the following diagram.



So, the point that you need to remember is based on the internal state of the context object, the behavior will be changed.

If this is not clear at the moment then don’t worry we will try to understand this with some real-time examples.

Understanding the State Design Pattern with Real-Time example:

Let us understand the State Design Pattern with a real-time example. ATM machine behavior is the best example of the State Design Pattern. Please have a look at the following diagram. Let say the ATM machine’s internal state is Debit Card Not Inserted means the ATM card is not inserted into the slot of the ATM Machine. Then what are all the operations you can do? The operations you can do are as follows.

You can insert the debit card.

You cannot eject the debit card as the debit card is not inserted into ATM Machine.

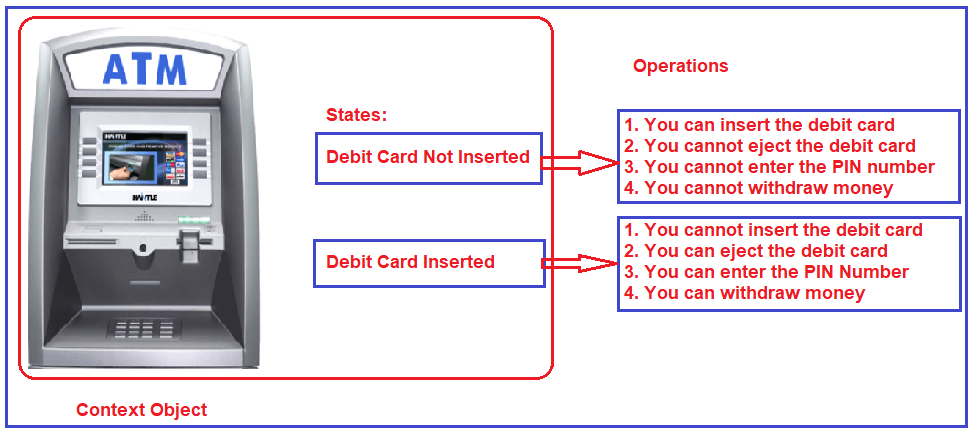
Again you cannot also enter the PIN and withdraw money. So, the only allowed operation is he can insert the debit card.

Suppose you inserted the debit card into the machine. So the state of the ATM (i.e. Context Object) is changed to Debit Card Inserted. Then what are all the operations you can do? The operations you can do are as follows.

You cannot insert the debit card as already one debit card is inserted into the machine.

It allows you to eject the Debit card.

You can enter the PIN number and withdraw the money.



The point that you need to remember is, if the state is Debit Card Not Inserted then it will allow you to perform certain operations and when the state is Debit Card Inserted then it will allow you to perform another set of operations. So, based on the internal state of the ATM machine the behavior will be changed.

Implementation of State Design Pattern in C#:

Let see the step by step procedure the implement the State Design Pattern in C#. The example that we discussed i.e. the behavior of the ATM machine will implement in this demo.

Step1: Creating the State interface

Create an interface with the name ATMState.cs and then copy and paste the following code in it. The ATMState interface defines the common methods for all the concrete states. All the concrete states will implement this interface so that they are going to be interchangeable. Here, we created the interface with the required four operations.

namespace StateDesignPattern { public interface ATMState { void InsertDebitCard(); void EjectDebitCard(); void EnterPin(); void WithdrawMoney(); } }

Step2: Creating Concrete States

As the ATM has two states, so here we are going to create two concrete states by implementing the ATMState interface.

DebitCardNotInsertedState:

Create a class file with the name DebitCardNotInsertedState.cs and then copy and paste the following code in it. The following code is very straight forward. We just implement all the methods of the ATMState interface.

using *System;*

namespace *StateDesignPattern*

{

public class DebitCardNotInsertedState : ATMState

{

public void InsertDebitCard()

{

Console.WriteLine("DebitCard Inserted");

}

public void EjectDebitCard()

{

Console.WriteLine("You cannot eject the Debit CardNo, as no Debit Card in ATM Machine slot");

}

public void EnterPin()

{

Console.WriteLine("you cannot enter the pin, as No Debit Card in ATM Machine slot");

}

public void WithdrawMoney()

{

Console.WriteLine("you cannot withdraw money, as No Debit Card in ATM Machine slot");

}

}

}

DebitCardInsertedState:

Create a class file with the name DebitCardInsertedState.cs and then copy and paste the following code in it. Here, in the following, we just implement all the methods of the ATMState interface.

using *System;*

namespace *StateDesignPattern*

{

public class DebitCardInsertedState : ATMState

{

public void InsertDebitCard()

{

Console.WriteLine("You cannot insert the Debit Card, as the Debit card is already there ");

}

public void EjectDebitCard()

{

Console.WriteLine("Debit Card is ejected");

}

public void EnterPin()

{

Console.WriteLine("Pin number has been entered correctly");

}

public void WithdrawMoney()

{

Console.WriteLine("Money has been withdrawn");

}

}

}

In this way, you can create many concrete states.

Note: The Concrete States handles requests from the context. Each concrete state provides its own implementation for a request. In this way, when the context object state changes its behavior will also change as well.

Step3: Creating Context

The Context is the class that can have a number of internal states.

Create a class file with the name ATMMachine.cs and then copy and paste the following code in it. This class also implements the ATMState interface and provide the implementation for all the interface methods. Here, we create one variable to hold the Concrete State object. Then we initialize that variable through the class constructor. As we know initially the State of the ATM Machine is DebitCardNotInsertedState so we initialize the variable with DebitCardNotInsertedState state.

Further, if you notice, when we insert the debit card, we also change the state of the context object to DebitCardInsertedState and again when we eject the debit card we also change the state of the context object to DebitCardNotInsertedState.

using System; namespace StateDesignPattern { public class ATMMachine : ATMState { public ATMState atmMachineState { get; set; } public ATMMachine() { atmMachineState = new DebitCardNotInsertedState(); } public void InsertDebitCard() { atmMachineState.InsertDebitCard(); // Debit Card has been inserted so internal state of ATM Machine // has been changed to 'DebitCardNotInsertedState' if (atmMachineState is DebitCardNotInsertedState) { atmMachineState = new DebitCardInsertedState(); Console.WriteLine("ATM Machine internal state has been moved to : " + atmMachineState.GetType().Name); } } public void EjectDebitCard() { atmMachineState.EjectDebitCard(); // Debit Card has been ejected so internal state of ATM Machine // has been changed to 'DebitCardNotInsertedState' if (atmMachineState is DebitCardInsertedState) { atmMachineState = new DebitCardNotInsertedState(); Console.WriteLine("ATM Machine internal state has been moved to : " + atmMachineState.GetType().Name); } } public void EnterPin() { atmMachineState.EnterPin(); } public void WithdrawMoney() { atmMachineState.WithdrawMoney(); } } }

Note: Internally we are calling the methods of the respective state objects.

Step4: Client

Please modify the Main method as shown below.

using *System;*

namespace *StateDesignPattern*

{

class Program

{

static void Main(string[] args)

{

// Initially ATM Machine in DebitCardNotInsertedState

ATMMachine atmMachine = new ATMMachine();

Console.WriteLine("ATM Machine Current state : "

+ atmMachine.atmMachineState.GetType().Name);

Console.WriteLine();

atmMachine.EnterPin();

atmMachine.WithdrawMoney();

atmMachine.EjectDebitCard();

atmMachine.InsertDebitCard();

Console.WriteLine();

// Debit Card has been inserted so internal state of ATM Machine

// has been changed to DebitCardInsertedState

Console.WriteLine("ATM Machine Current state : "

+ atmMachine.atmMachineState.GetType().Name);

Console.WriteLine();

atmMachine.EnterPin();

atmMachine.WithdrawMoney();

atmMachine.InsertDebitCard();

atmMachine.EjectDebitCard();

Console.WriteLine("");

// Debit Card has been ejected so internal state of ATM Machine

// has been changed to DebitCardNotInsertedState

Console.WriteLine("ATM Machine Current state : "

+ atmMachine.atmMachineState.GetType().Name);

Console.Read();

}

}

}

Output:

