Command Pattern

Estimated time for completion: 60 minutes

Overview:

The focus of this lab will be to show you that it's possible to record a series of actions and replay them at some point in the future to produce Undo logic, and Macro command behaviour. To demonstrate both these features we will use a simple Logo programming language environment.

Goals:

- Develop and put to use the command pattern to provide a simple Undo mechanism
- Use the command pattern to allow the building of macro based functionality

Lab Notes:

Part 1: Refactor to Command Pattern

For this first part you will concentrate on simply refactoring the code to use the command pattern. The command pattern comprises of two main parts, the Commands themselves and the Invoker. The invoker typically wraps additional functionality around the invocation of the command, but for this first bit you will make your invoker simply execute the command, no hidden extras. After you have refactored you will still have the same functionality you had at the start but now you will be in a better shape to layer in the necessary extra functionality to implement the undo functionality.

1. Open the solution and familiarize yourself with the project. Run the app and have a play, see what cool pictures you can make. The LogoForm.cs is the main UI form, it contains two real areas, a window pane that contains the buttons used to drive the turtle, and an area for showing the path the turtle has taken. There is an additional class library called LogoLib this contains all the logic for moving the turtle, the TurtleView control uses the observer pattern to watch for changes in the turtle location, and provides the appropriate UI update. The Undo functionality is currently not available, your first task is be to enable this using the command pattern
2. One way of creating Undo functionality would be to simply record each movement of the turtle, so that if an Undo operation is requested you simply throw away the last command in the history, clear the canvas and then repeat each remaining command in the history. Currently the Forward and Rotate methods inside LogoForm.cs make the appropriate calls to the turtle object to make it move, you need some way of recording each of these movements. For that you will encapsulate the calls to turtle into an object called a command and then create an invoker that can take a command and execute it but in addition record each command as it is executed, allowing them to be replayed later.
3. First refactor your code to use the command pattern adding no new functionality. Create a new Class library that you will use to hold all the basic command pattern functionality.

Call the Class library CommandPattern, then add a new abstract class called Command, and make it have a single abstract void method called Execute

```
public abstract class Command
{
   public abstract void Execute();
}
```

4. The command pattern calls for not only a command object but also an invoker. Create a simple command invoker, something that literally just invokes commands. Inside the CommandPattern library create a new class called the CommandInvoker. This will have a single virtual method called Execute and a command as parameter

```
public class CommandInvoker
{
   public virtual void Execute(Command command)
   {
      command.Execute();
   }
}
```

5. Compile the class library and ensure no errors. Then add a reference from the LogoApp project to the command pattern class library
6. Now you have a basic command pattern framework you can now start creating your own specific type of commands. The operations you need to encapsulate into a command are Turtle.Forward and Turtle.Rotate, so create two new classes called ForwardCommand and RotateCommand inside the LogoApp project. Each class will derive from the Command type. Remember a command encapsulates Target+Method+Parameters, the command type therefore has to take as part of its constructor the Target+Parameters the Method typically is in code.

```
public class ForwardCommand : Command
{
   private Turtle turtle;
   private int nTimes;

   public ForwardCommand( Turtle turtle , int nTimes )
   {
      this.turtle = turtle;
      this.nTimes = nTimes;
   }

   public override void Execute()
   {
      turtle.Forward( nTimes );
   }
}
```

7. Add an additional field to the LogoForm class called invoker of type CommandInvoker

```
private CommandInvoker invoker = new CommandInvoker();
```

8. Now refactor the methods Forward and Rotate in the LogoForm class to use the Forward and Rotate commands

```
private void Forward(object sender, RepeatActionEventArgs e)
{
   ForwardCommand cmd = new ForwardCommand( turtle , e.NTimes );
   invoker.Execute(cmd);
}
```

9. Re-run the application and you will still have the same level of functionality you had at the start, but now you are in a better place to add undo functionality.

Part 2: Undo Recorder

In this next part you will create a more sophisticated invoker to allow you to support undo operations

1. Now you need to create a more sophisticated invoker so that it records all the commands that have been used. Create a new class in the Command Pattern class library called CommandRecorder, and make it derive from CommandInvoker, and override the Execute method

```
public class CommandRecorder : CommandInvoker
{
   public override void Execute(Command command)
   {

   }
}
```

2. You now need to get the Execute method to execute the command and then record it. you therefore need a mechanism to record each command, you can do that using a simple List, add a List field to the CommandRecorder class, and use it to record each command

```
private List<Command>         commands = new List<Command>();

public override void Execute(Command command)
{
   base.Execute(command);
   commands.Add(command);
}
```

3. You now have a mechanism in place to record each command, you now need to think about how you can use this to build the Undo logic. The algorithm you will use is to simply
4. Clear the canvas
   - Reset the turtle
   - Replay each recorded command apart from the last one
5. One way is to provide two additional methods on the CommandRecorder class
   - Remove the last command in the list

- Return the collection of recorded commands

6. These methods will be used by the Undo method inside the LogoForm class to first reset the turtle and then replay the commands

```
public void DeleteLastCommand()
{
   if (commands.Count > 0)
   {
      commands.RemoveAt(commands.Count - 1);
   }
}

public IEnumerable<Command> GetCommands()
{
   return commands;
}
```

7. Now change the Undo method inside LogoForm to perform the necessary steps. Call ClearAndReset() method, then call DeleteLastCommand() method to remove the last command and finally using a foreach loop obtain each command that has been recorded via the GetCommands() method and execute each command directly not via the invoker

```
private void Undo(object sender, EventArgs e)
{
   ClearAndReset();

   recorder.DeleteLastCommand();

   foreach (Command command in recorder.GetCommands())
   {
      command.Execute();
   }
}
```

8. Finally update the private void ClearAndReset(object sender, EventArgs e) method to reset the history completely. By creating a new instance of the CommandRecorder. You will now have Undo functionality, compile and run the app and verify the Undo functionality works.

Part 3: Macro Command

A Macro is a way of representing multiple commands under a single command, macro command functionality will allow us to produce some funky images by simply repeating N times a series of forward and rotate commands. Users will define macros by first clicking on the Start Recording menu option and then using the Forward and Rotate commands to do a series of actions they want to repeat, when completed they hit the Stop Recording option. The user can now replay the sequence of commands by simply hitting the macro button, and entering a repeat count can generated some pretty cool pictures. However the functionality needs to be built, so in this part you will implement the macro functionality

1. The first thing you will need to do is define how to record the macro, to do this you will create a new type of Command, called a Macro Command which will derive from Command. The Macro Command will support in addition to the Execute method, and AddCommand method.

```
public class MacroCommand : Command
{
   public void AddCommand(Command command)
   {
   }

   public override void Execute()
   {
   }
}
```

2. When the user states they wish to start recording a new Macro you should create a new instance of this Macro Command, every time the user executes a forward or rotate operation the created command is added to the macro command. To do this, add a new field to the LogoForm class to hold a reference to a Macro command call it recording. Create a new instance of the MacroCommand each time a user selects start recording. When they have stated stop recording, take a copy of the reference to the macro command, and assign it to a new field called macro of type MacroCommand.

```
public partial class LogoForm : Form
{
   // ...

   private MacroCommand macro = new MacroCommand();
   private MacroCommand recordingMacro;
   // ....

   private void Forward(object sender, RepeatActionEventArgs e)
   {
      ForwardCommand cmd = new ForwardCommand(turtle, e.NTimes);

      // Tip refactor the following lines into a single method called
   Execute, and
      // use it for the Rotate Command too

      recorder.Execute(cmd);

      if (recordingMacro != null)
      {
      recordingMacro.AddCommand(command);
      }
   }

   private void StartRecording(object sender, EventArgs e)
   {
      recordingMacro = new MacroCommand();
      stopRecordingItem.Enabled = true;
      startRecordingMenuItem.Enabled = false;
   }

   private void StopRecordingMacro(object sender, EventArgs e)
   {
      macro = recordingMacro;
      recordingMacro = null;

      stopRecordingItem.Enabled = false;
      startRecordingMenuItem.Enabled = true;
   }
```

3. Make the Macro Command keep a record of each command, to do this implement the AddCommand method inside the MacroCommand type, to simply add the command to an internal list of commands.

```
private List<Command> commands = new List<Command>();

public void AddCommand(Command command)
{
   commands.Add(command);
}
```

4. The next step is to implement the Execute method on the Macro Command, you will need to iterate over the entire list of commands executing each one directly.

```
public override void Execute()
{
   foreach (Command command in commands)
   {
      command.Execute();
   }
}
```

5. Finally make the RunMacro method inside LogoForm invoke the last recorded macro the requested number of times.

```
private void RunMacro(object sender, RepeatActionEventArgs e)
{
   for (int nTime = 0; nTime < e.NTimes; nTime++)
   {
      Execute(macro);
   }
}
```

6. Now run the application and see what cool patterns you can make.