

# Template and Strategy Patterns



**DEVELOPMENTOR**

DEVELOPING PEOPLE WHO DEVELOP SOFTWARE



- Over time code changes
  - But we want to change as little as possible
- Maximise re-use
  - Change something once rather than n times
- Designing code with change in mind can lead to
  - Quicker releases
  - Fewer bugs



- You have been asked to write some code that will
- Produce a File Report for a given directory, the report will contain
  - Header, Report on Files in Directory XXX
  - Column Headings Name,Size,Last Modified
  - A row for each file with comma separated values
  - Footer, stating when the report was run and by whom

```
Report on Files in Directory D:\  
Filename,Size,Last Modified  
coolestbab552507.wmv , 745048 , 02/03/2007 13:56:57  
data.xml , 154 , 21/11/2006 07:21:21  
foo.html , 106 , 19/04/2007 23:01:13  
Report ran 02/05/2007 09:46:15 by AndyMobile\andy
```



```
public class FileReport
{
    private string dir;

    public FileReport(string directory) {
        dir= directory;
    }

    public void ProduceReport() {
        Console.WriteLine("Report on Directory {0}" , dir);
        Console.WriteLine("Filename,Size,Last Modified");
        foreach( FileInfo file in new DirectoryInfo(dir).GetFiles() )
        {
            // Write File Details
        }

        Console.WriteLine("Report ran {0} by {1}" , DateTime.Now ,
                          WindowsIdentity.GetCurrent().Name );
    }
}
```



- What we want now is same report format but for directories
- Possible Solutions
  - CUT + PASTE File Report, and modify...
    - Changes to report structure and common pieces made N times
  - Modify Report generation to take a parameter and use if/switch
    - We risk breaking existing functionality when we extend
- Or perhaps there is an OO Design Principle that may help

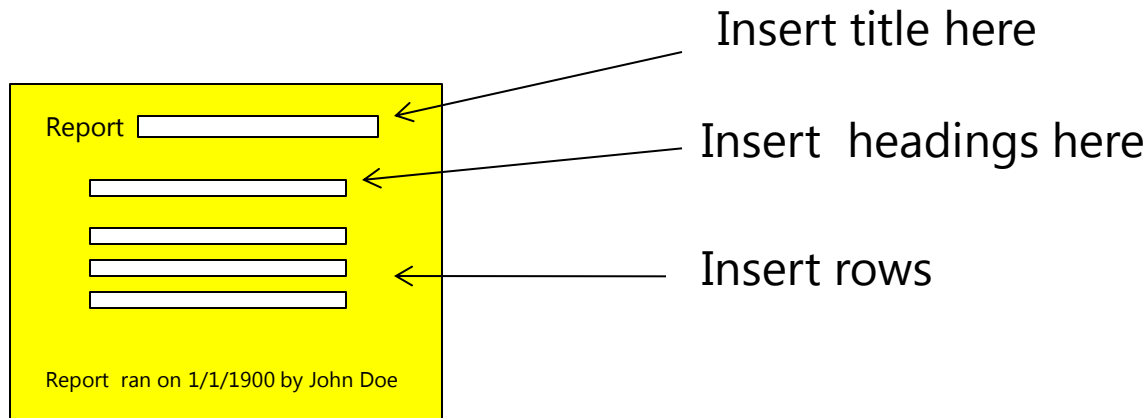
**Identify the aspects of your application that vary and separate them from what stays the same.**



- In the File and Directory Report model
  - Constant
    - Report structure: Headings , Body , Footer
  - Varies
    - Text for Headings , Rows in Body
- So we need to find some way to
  - Write the constant part once
  - But still allow us to develop new reports
- Perhaps we could think of the constant part representing a template
- The template is re-used by each of the reports



- The constant part of our code is a template
- The template defines the basic algorithm for the report
  - Title, Headings , Rows , Footer
- The template needs to call other code to fill in the gaps





```
public abstract class ReportTemplate
{
    public void ProduceReport() {
        Console.WriteLine("Report {0}" , GetTitle() );

        foreach (string column in GetColumnNames())
        {
            // Output column heading
        }

        foreach (object[] row in GetRows())
        {
            // Output Row
        }

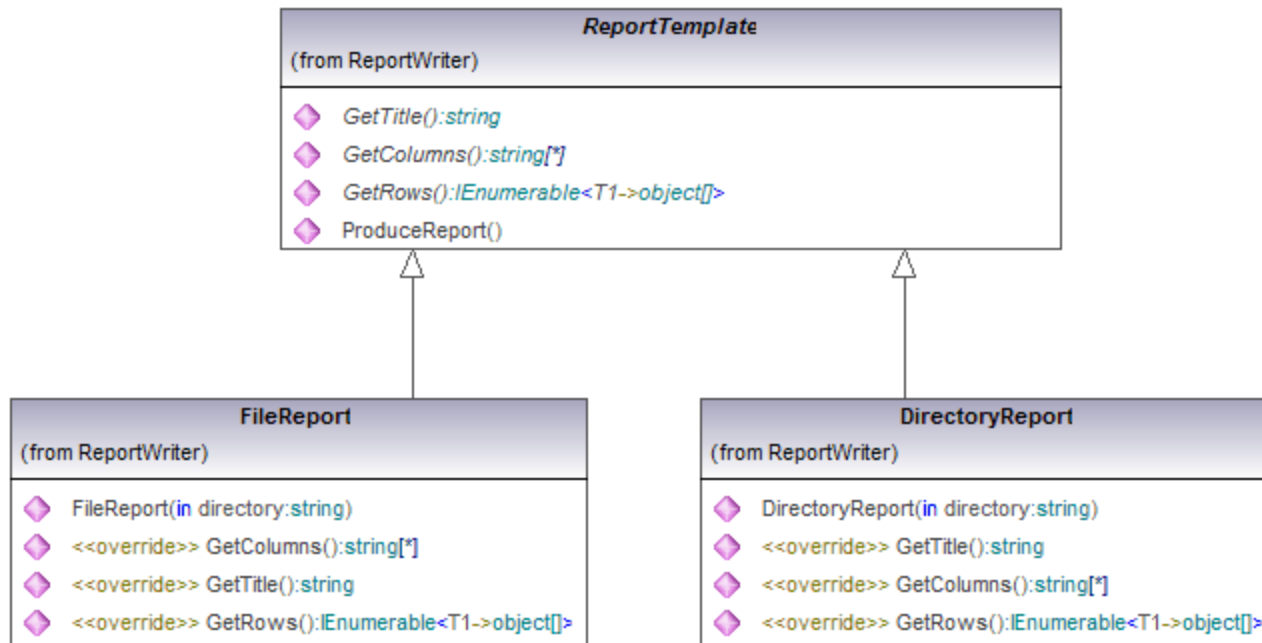
        Console.WriteLine( "Report Produced {0} by {1}",
            DateTime.Now,
            WindowsIdentity.GetCurrent().Name);
    }
}
```



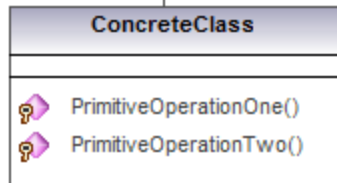
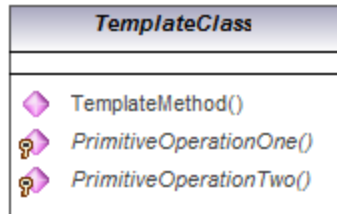
# Binding the template to code



- Place the template method into a base class
- Base class defines the template steps as abstract
- Actual report class derive from base class
- Actual report class implements the abstract methods



# Template Method Pattern



```
public abstract class TemplateClass {  
    protected abstract void TemplateStepOne();  
    protected abstract void TemplateStepTwo();  
  
    public void TemplateMethod()  
    {  
        TemplateStepOne();  
        TemplateStepTwo();  
    }  
}
```

```
public class ConcreteClass : TemplateClass {  
    protected override void TemplateStepOne() {  
        // Do Step  
    }  
    protected override void TemplateStepTwo() {  
        // Do Step  
    }  
}
```



- The ReportTemplate class controls the format of reports
  - Maximises re-use of code across all report variants
  - The report structure lives in one place, meaning code changes happen once
  - Provides a framework for new types of report to be plugged into
- Adding a new report
  - Does not affect any existing tested and working code
  - Focuses on knowledge specific for this report
- A more maintainable and scalable solution



- Console Reports are dull we want to provide the report in various output formats but keep structure the same
  - XML
  - HTML
  - XAML Flow document
- The Report base class is currently tightly coupled with console output
- How about
  - Use if/switch inside the Report base class to select output type?  
Means modifying working code. Smells bad...
- Perhaps we need a new *Strategy*....



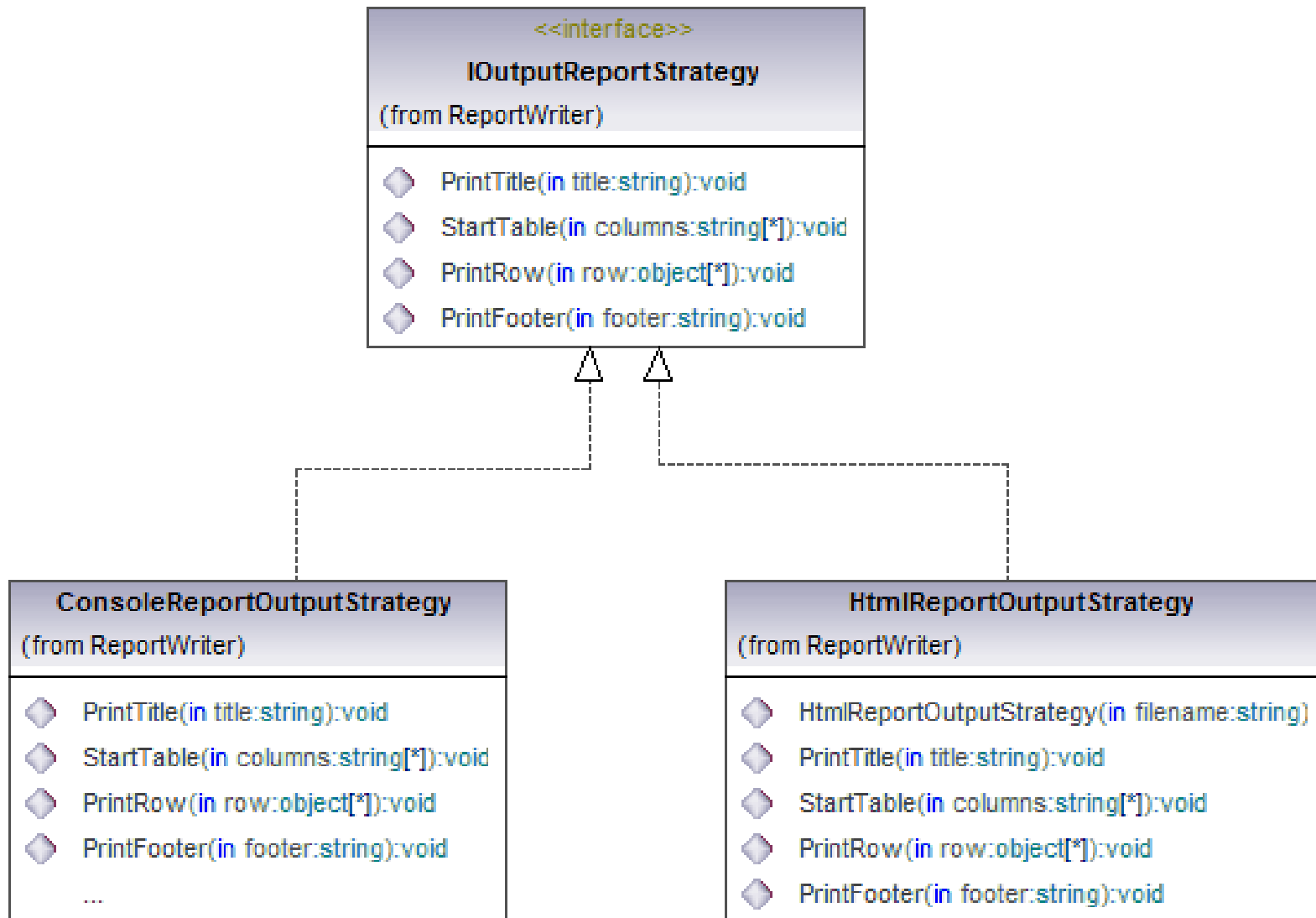
**Identify the aspects of your application that vary and separate them from what stays the same**

- Report structure code remains constant
- Report output format varies
- The question is how to separate
  - Create a HTMLReport class derived from Report using template methods for output
    - That would result in too many variants. HTMLFileReport, HTMLDirectoryReport
  - Build a separate type hierarchy for output formats
    - Supply an output format at runtime to a report instance

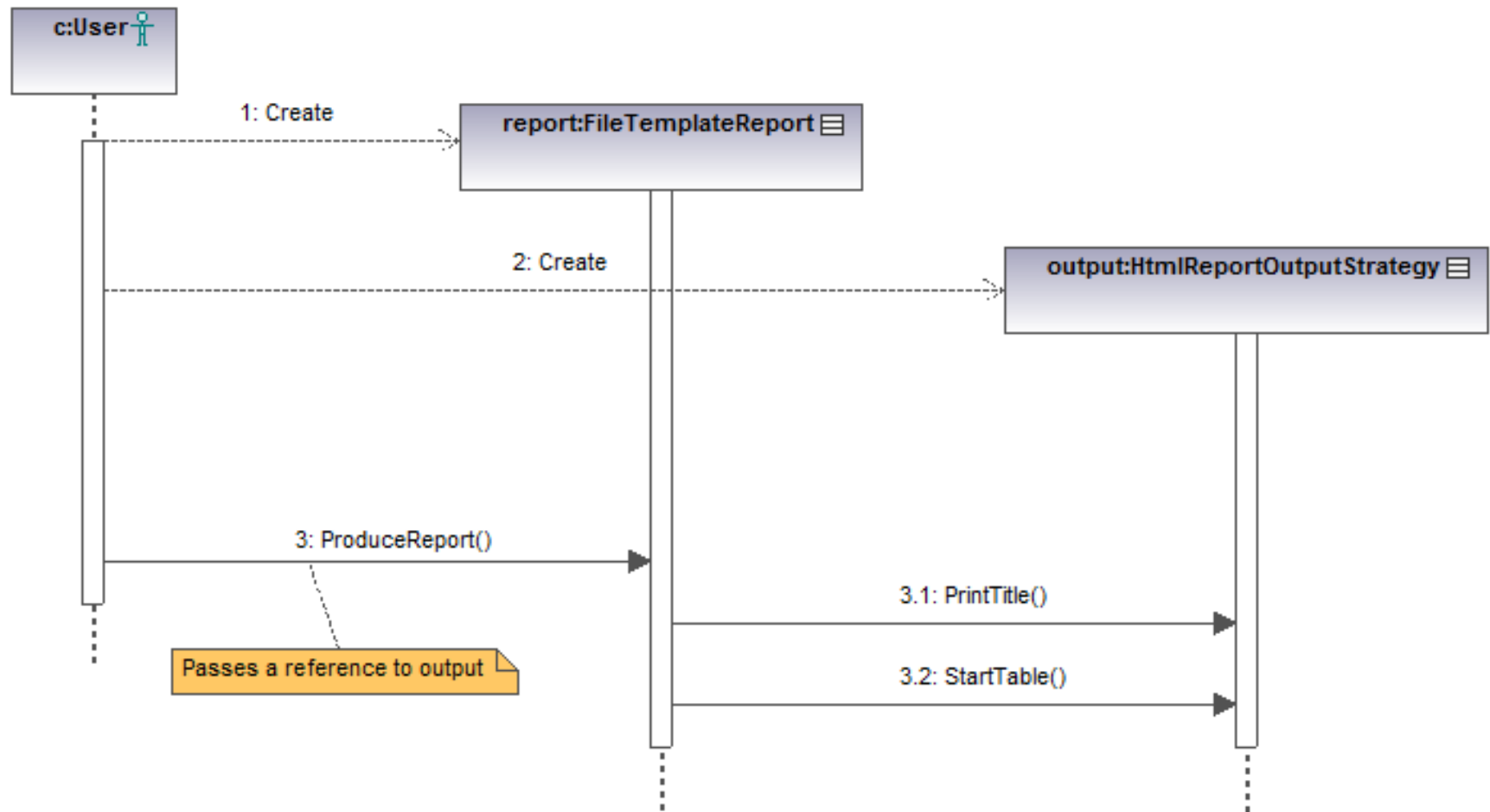


- Step 1
  - Define an interface with the required output behavior
  - Re-factor the ProduceReport method to use the new interface, as opposed to tightly coupled behaviour
- Step 2
  - Create a type that implements the interface
  - Pass an instance of this type into ProduceReport method
  - ProduceReport method will now use this object for the required output behaviour
- We have now decoupled the report definition from the output format through composition

# Refactoring the output behaviour



# Utilising the supplied strategy

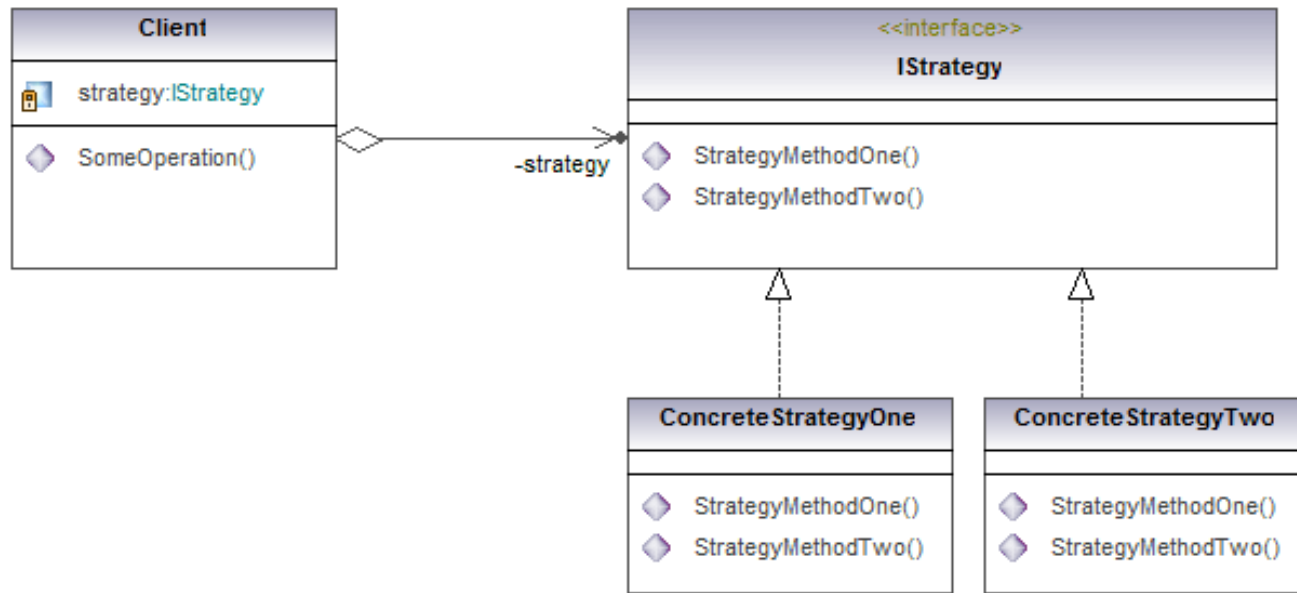


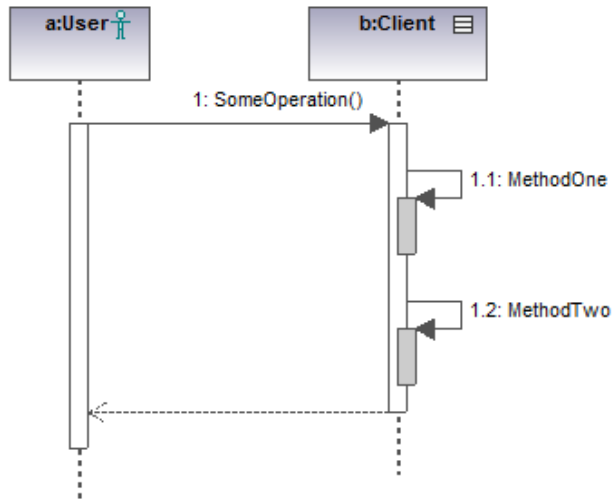


# Strategy Pattern

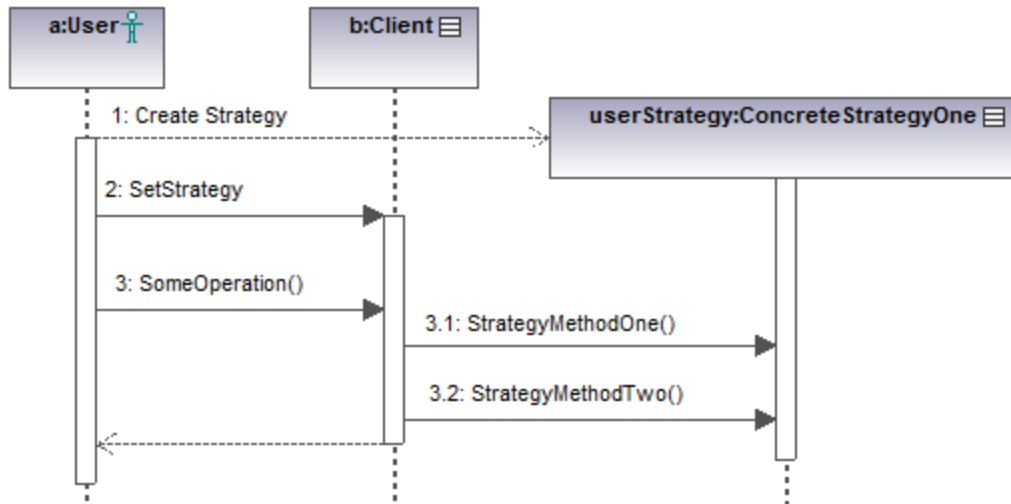


- Template method used abstract methods and inheritance to call different behaviour
- Alternatively supply implementation behaviour at run time using composition
- This is known as the Strategy pattern





Before



After



- Create new types of reports, where
  - All reports follow a common structure
  - No modification is made to existing code
- Create new types of report output format, where
  - New formats can be used without changing existing code
  - Output format is selectable at runtime
- The solution is truly scalable
  - $\text{Combinations} = \text{nReportTypes} * \text{nOutputTypes}$
  - $\text{Number of types built} = \text{nReportTypes} + \text{nOutputTypes}$
- How much work would it be for the reports to support sorting?



- The new generic list class makes extensive use of the strategy pattern
  - List.RemoveAll() , List.FindAll()
  - All these methods take an instance of a predicate delegate
  - The delegate instance is the strategy

```
public delegate bool Predicate<T>(T obj);

public class List<T> : IList<T>, ICollection<T> ...
{
    ...
    public List<T> FindAll(Predicate<T> match);
}
```



```
static void Main(string[] args)
{
    var primes = new List<int>() { 2,3,5,7,11,13 };

    foreach (int prime in primes
                .FindAll(IsValueGreaterThanSeven))
    {
        Console.WriteLine(prime);
    }
}

// The Strategy
private static bool IsValueGreaterThanSeven(int val)
{
    return val > 7;
}
```



- Use Template pattern to control the sequence of steps in the algorithm, but still allow the step implementations to vary
  - Encapsulate what doesn't change
- Use Strategy over Template
  - When algorithm needs to vary at runtime.
  - To prevent type explosion
- Use delegates for single method strategies.
  - Check for existing delegate types e.g. `Predicate<T>`