Software Architecture & Design SEC3071

Lecture No. 39

Muhammad Shahid
Department of Computer Science
National Textile University

shahid.abdullah@hotmail.com

Last Lecture Review

- Behavioral Design Patterns
- Design Principle
  - Encapsulate What Varies
  - Program to an Interface not to an Implementation
  - Favor Composition Over Inheritance
- Strategy Design Pattern
  - Applicability
  - Implementation
- Strategy Example - Animal Information Sys.

Agenda – What will you Learn Today?

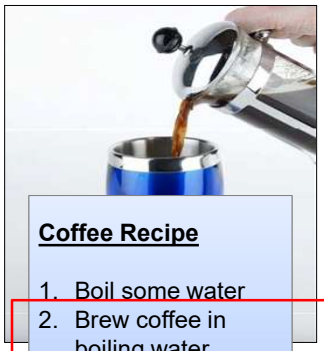
Template Method

Template Method

Behavioral Patterns

- Behavioral patterns are concerned with the assignment of responsibilities between objects, or, encapsulating behavior in an object and delegating requests to it
- They describe not just patterns of objects or classes but also the pattern of communication between them

Time for Caffeine



Coffee Recipe

1. Boil some water
2. Brew coffee in boiling water
3. Pour coffee in cup
4. Add sugar and milk

Tea Recipe

1. Boil some water
2. Steep tea in boiling water
3. Pour tea in a cup
4. Add lemon

Time for Caffeine

```
public class Coffee
{
        public void PrepareRecipe()
        {
                BoilWater()
                BrewCoffeeGrinds();
                PourInCup();
                AddSugarAndMilk();
        }

        public void BoilWater()
        {
                Console.WritleLine("Boiling water");
        }
```


Time for Caffeine

```
        public void BrewCoffeeGrinds()
        {
                Console.WritleLine("Dripping coffee through
                filter");
        }

        public void PourInCup()
        {
                Console.WritleLine("Pouring into cup");
        }

        public void AddSugarAndMilk()
        {
                Console.WritleLine("Adding sugar and milk");
        }

} // End of Coffee class
```

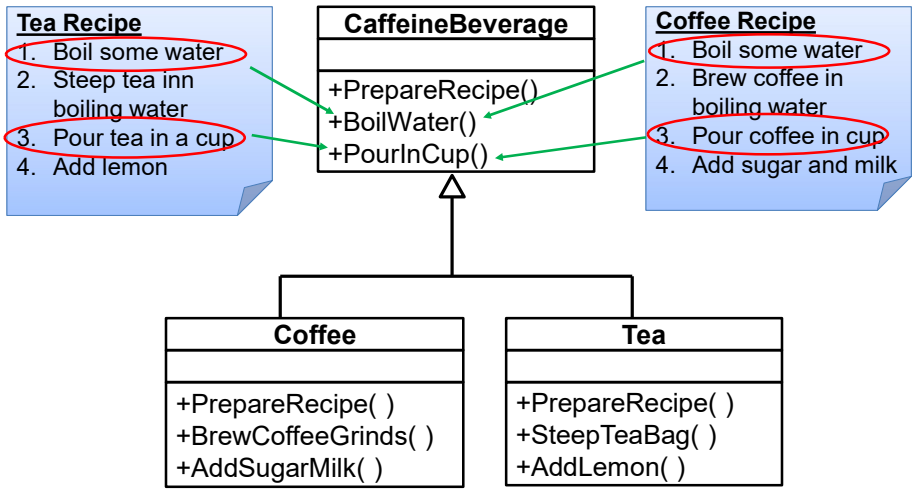

Time for Caffeine

```
public class Tea
{
        public void PrepareRecipe()
        {
                BoilWater();
                BteepTeaBag();
                PourInCup();
                AddLemon();
        }

        public void BoilWater()
        {
                Console.WritleLine("Boiling water");
        }
```

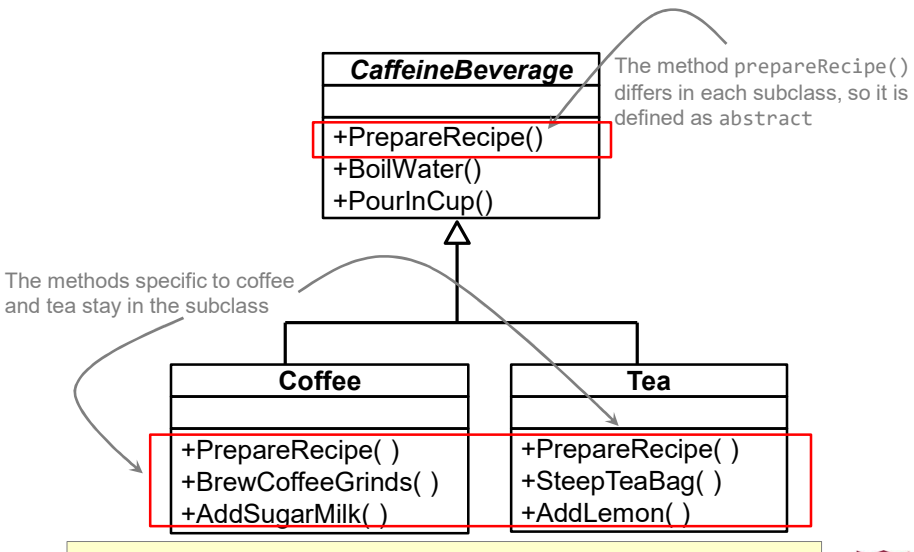

Time for Caffeine

```
        public void SteepTeaBag()
        {
                Console.WritleLine("Steeping the tea");
        }

        public void PourInCup()
        {
                Console.WritleLine("Pouring into cup");
        }

        public void AddLemon()
        {
                Console.WritleLine("Adding Lemon");
        }

}; // End of Tea class
```


Abstracting Coffee and Tea



Abstracting Coffee and Tea



It seems that we have some code duplication here.

Detecting Commonality

Coffee Recipe
1. Boil some water
2. Brew coffee in boiling water
3. Pour coffee in cup
4. Add sugar and milk

Tea Recipe
1. Boil some water
2. Steep tea in boiling water
3. Pour tea in a cup
4. Add lemon

1. Boil some water
2. Use hot water to extract the coffee or tea
3. Pour the resulting beverage into a cup
4. Add the appropriate condiments to the beverage

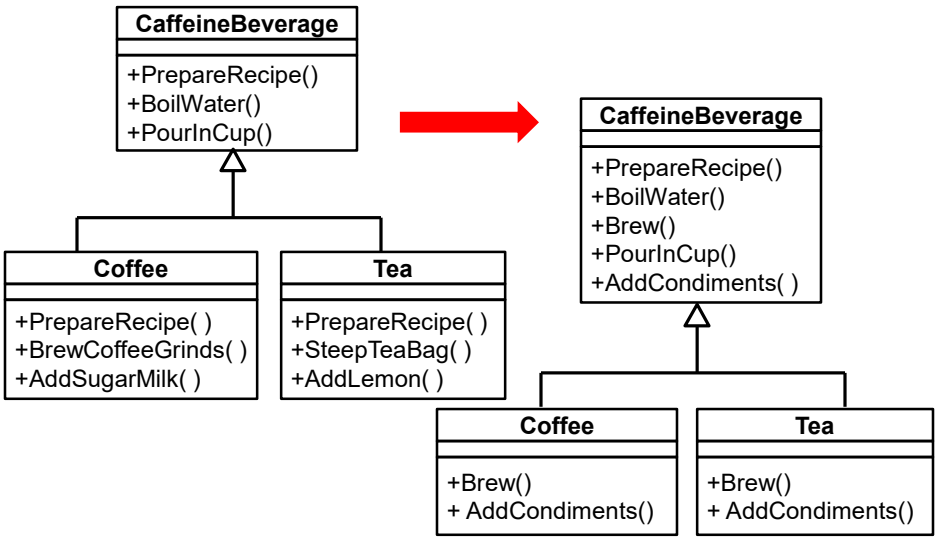
Abstracting PrepareRecipe()

```
public void prepareRecipe()
{
        BoilWater();
        BrewCoffeeGrinds();
        PourInCup();
        AddSugarAndMilk();
}
```

```
public void PrepareRecipe()
{
        BoilWater();
        SteepTeaBag();
        PourInCup();
        AddLemon();
}
```

```
public void PrepareRecipe()
{
        BoilWater();
        Brew();
        PourInCup();
        AddCondiments();
}
```


Abstracting PrepareRecipe



Intent of Template Pattern

- Define the skeleton of an algorithm in an operation, deferring some steps to subclasses. Template Method lets subclasses redefine certain steps of an algorithm without letting them to change the algorithm's structure

Motivation for Template Method

- The Template Method pattern can be used in situations when there is an algorithm, some steps of which could be implemented in multiple different ways
- Some portion of the solution is fix for all the scenarios and some portion of the solution is specific to any situation

Motivation for Template Method

- In such scenarios, the Template Method pattern suggests keeping the outline of the algorithm in a separate method referred to as a template method inside a class, which may be referred to as a template class, leaving out the specific implementations of the variant portions (steps that can be implemented in multiple different ways) of the algorithm to different subclasses of this class

Hooks or Hot Spots

- The hooks are generally empty methods that are called in base class (and does nothing because are empty), but can be implemented in derived classes
- Customization Hooks can be considered a particular case of the template method as well as a totally different mechanism

Hollywood Principle

“Don't call us, we'll call you”

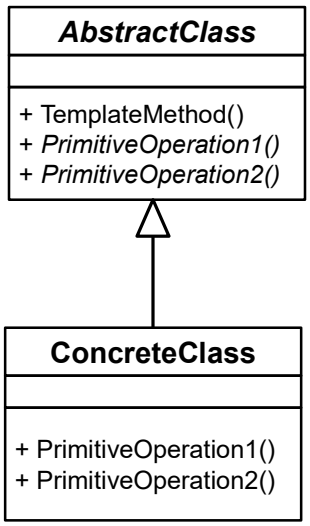
- This refers to the fact that instead of calling the methods from base class in the derived classes, the methods from derived class are called in the template method from base class

Template Method - Applicability

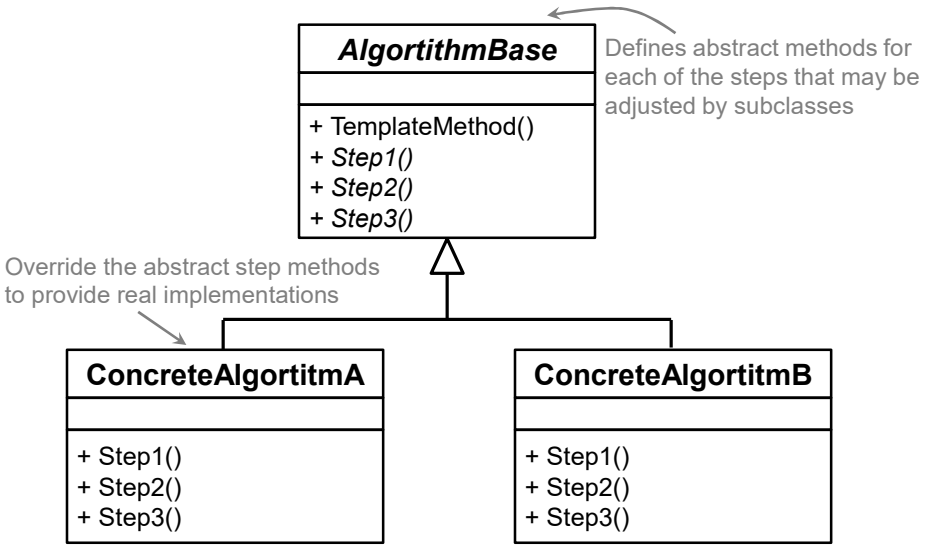
The Template Method pattern should be used:

- To implement the invariant parts of an algorithm once and leave it up to subclasses to implement the behavior that can vary
- When common behavior among subclasses should be factored and localized in a common class to avoid code duplication
- To control subclasses extensions. You can define a template method that calls "hook" operations at specific points, thereby permitting extensions only at those points

Template Method – Class Diagram



Template Method – Class Diagram



Template Method – Implementation

```
public abstract class AlgorithmBase
{
	public void TemplateMethod()
	{
		Step1( );
		Step2( );
		Step3( );
	}

	public abstract void Step1();
	public abstract void Step2();
	public abstract void Step3();

} // End of class
```

Template Method – Implementation

```
public class ConcreteAlgorithmA : AlgorithmBase
{
	public override void Step1()
	{
		Console.WriteLine("Algorithm A, Step 1");
	}
	public override void Step2()
	{
		Console.WriteLine("Algorithm A, Step 2");
	}
	public override void Step3()
	{
		Console.WriteLine("Algorithm A, Step 3");
	}
} // End of ConcreteAlgorithmA  class
```


Template Method – Implementation

```
public class ConcreteAlgorithmB : AlgorithmBase
{
	public override void Step1()
	{
		Console.WriteLine("Algorithm B, Step 1");
	}
	public override void Step2()
	{
		Console.WriteLine("Algorithm B, Step 2");
	}
	public override void Step3()
	{
		Console.WriteLine("Algorithm B, Step 3");
	}
} // End of ConcreteAlgorithmA  class
```


Template Method – Implementation

```
public class ConcreteAlgorithmC : AlgorithmBase
{
        public override void Step1()
        {
                Console.WriteLine("Algorithm C, Step 1");
        }
        public override void Step2()
        {
                Console.WriteLine("Algorithm C, Step 2");
        }
        public override void Step3()
        {
                Console.WriteLine("Algorithm C, Step 3");
        }
} // End of ConcreteAlgorithmA  class
```


Template Method – Implementation

```
static void Main(string[] args)
{
        AlgorithmBase concreteA = new ConcreteAlgorithmA();
        concreteA.TemplateMethod();

        AlgorithmBase concreteB = new ConcreteAlgorithmB();
        concreteB.TemplateMethod();
}
```

```
-----------------------------
     Algorithm A, Step 1
     Algorithm A, Step 2
     Algorithm A, Step 3
-----------------------------
```

```
-----------------------------
     Algorithm B, Step 1
     Algorithm B, Step 2
     Algorithm B, Step 3
-----------------------------
```


Task - Scoring System

Scoring System - Task

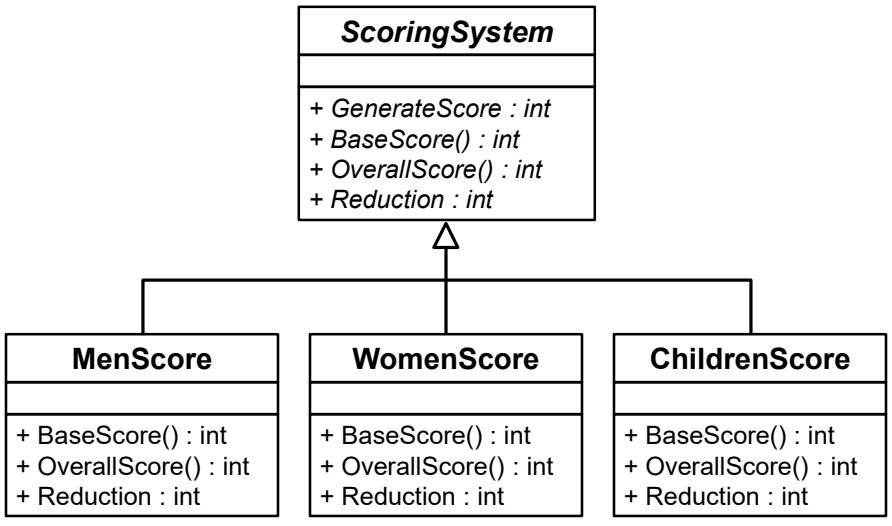
- Develop a game Scoring System. In the game being scored the players run around a circuit that includes checkpoints. At each checkpoint the player throws projectiles at a target, scoring points for each hit. The player's score is reduced if they complete the circuit in a slow time. The algorithms for calculating the score differ according to the sex and age of the player.

Scoring System - Task

The scoring rules are as follows:

- **Men:** 100 points will be awarded for every target hit. 5 points will be deducted for each second of time taken.
- **Women:** 100 points will be awarded for every target hit. 4 points will be deducted for each second of time taken.
- **Children:** 200 points will be awarded for every target hit. 2 points will be deducted for each second of time taken. Negative scores are replaced with a zero score.

Scoring System - Class Diagram



Scoring System - Implementation

```
public abstract class ScoringSystem
{
     public int GenerateScore(int hits, TimeSpan time)
     {
          int score = BaseScore(hits);
          int reduction = Reduction(time);
          return OverallScore (score, reduction);
     }
     public abstract int BaseScore(int hits);
     public abstract int Reduction(TimeSpan time);
     public abstract int OverallScore(int score, int red);
}
```


Scoring System - Implementation

```
public class MenScore : ScoringSystem
{
     public override int BaseScore(int hits)
     {
          return hits * 100;
     }
     public override int Reduction(TimeSpan time)
     {
          return ((int)time.TotalSeconds/5);
     }
     public override int OverallScore(int score, int red)
     {
          return score - red;
     }
}
```


Scoring System - Implementation

```
public class WomenScore : ScoringSystem
{
     public override int BaseScore(int hits)
     {
          return hits * 100;
     }
     public override int Reduction(TimeSpan time)
     {
          return ((int)time.TotalSeconds/4);
     }
     public override int OverallScore(int score, int red)
     {
          return score - red;
     }
}
```


Scoring System - Implementation

```
public class ChildrenScore : ScoringSystem
{
     public override int BaseScore(int hits)
     {
          return hits * 200;
     }
     public override int Reduction(TimeSpan time)
     {
          return ((int)time.TotalSeconds/2);
     }
     public override int OverallScore(int score, int red)
     {
          if (score > red) return score - red;
          else return 0;
     }
}
```


Scoring System - Implementation

```
Console.Write("Man Score:");
ScoringSystem algorithm = new MenScore();
Console.WriteLine(algorithm.GenerateScore(8, new
TimeSpan(0, 1, 50)));

Console.Write("Woman Score:");
ScoringSystem algorithm = new WomenScore();
Console.WriteLine(algorithm.GenerateScore(9, new
TimeSpan(0, 1, 20)));

Console.Write("Child Score:");
ScoringSystem algorithm = new ChildrenScore();
Console.WriteLine(algorithm.GenerateScore(5, new
TimeSpan(0, 3, 0)));
```

```
Man Score:          778
Woman Score:        880
Child Score:        910
```


Recap

- Behavioral Patterns
- Template Pattern
  - Definition
  - Intent
  - Motivation
  - Applicability
  - Class Diagram
  - Implementation
- Hollywood Principle
- Template Method Example – Scoring System

Questions