Introduction To Design Patterns

You will learn about design techniques that have been successfully applied to different scenarios.

What Is A Design Pattern?

- A general and reusable solution to a commonly occurring problem in the design of software.
- IT IS a template for how to solve a problem that has been used in many different situations.
- IT IS NOT a finished algorithm that can be directly translated into program code.
- The various Object-Oriented design patterns show interactions between classes and objects without being tied to the specific the program code that implements the pattern (language independent)
  - e.g., Information hiding, inheritance etc.

James Tam

Some General Resources (Last Visited 2021)

- Microsoft:
  - https://docs.microsoft.com/en-us/archive/msdn-magazine/2001/july/design-patterns-solidify-your-csharp-application-architecture-with-design-patterns
- Oracle
  - https://www.oracle.com/java/technologies/design-patterns-catalog.html
- The original "Gang of Four" resource
  - https://springframework.guru/gang-of-four-design-patterns/
- Another book authored by Gamma et al.
  - https://www.amazon.ca/Design-Patterns-Elements-Reusable-Object-Oriented/dp/0201633612

James Tam

Origin Of Design Patterns

- The foundation for design patterns come from the original patterns specified in the book "*Design Patterns: Elements of Reusable Object-Oriented Software*"
- Authors: "The gang of four" (Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides).
- Although examples of the patterns were provided in the C++ and SmallTalk programming languages the patterns can be applied to any Object-Oriented language.

James Tam

The Model-View-Controller Pattern[1]

- Sometimes the same data may have to be accessed under different contexts e.g., powerful desktop, web, mobile device.
- Each context may require a different interface (e.g., web page on a mobile device, software on a computer).
- Even the context of a single program running on a single device there may be a desire to see different views of the data:
  - Financial analysts may want to see details (actual numbers in a spreadsheet and/or financial statement)
  - Shareholders or management may focus on overviews (graphical representations)

1 Some additional sources that describe the Model-View Controller pattern (last visited 2021):

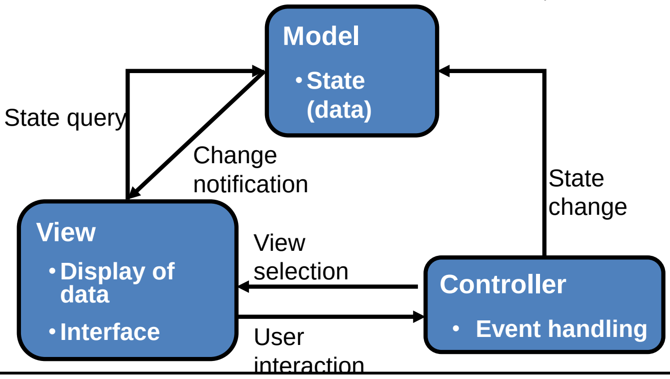
I. Sun Microsystems, now via the Oracle link (last visited 2021):
   - https://www.oracle.com/java/technologies/design-patterns-catalog.html

II. Microsoft:
   - http://msdn.microsoft.com/en-us/library/ms978748.aspx

James Tam

The Model-View-Controller Pattern[1]

- With this pattern, different parts are separate and independent:
  - Model: The data (database, text file):
  - View: How the data appears or the perspective under which it is viewed (graph, numerical)
  - Controller: How the data can be interacted with (GUI, command line).



James Tam

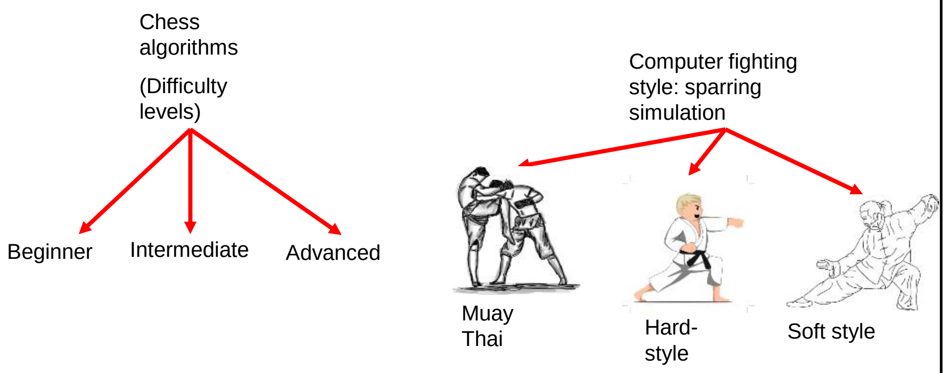
Model-View-Controller Pattern (2)

- Implementing different parts that are decoupled (minimized dependencies) provides many benefits:
  - One part may be changed independent of the other parts e.g., updates to the interface can have minimal impact on the data.
  - It's seldom that one person will have a deep understanding of all parts (e.g., knowledge of Accounting to create the financial statements vs. knowledge of web design to create the web interface).
  - Different people with different areas of expertise can work on the different parts.
  - One version of the data can be created and maintained and as needed different ways of interacting and viewing data can be developed.

James Tam

The Strategy Pattern

- The algorithm is determined at run time.



Images: colourbox.com

James Tam

The Strategy Pattern (2)

- One object contains a reference to another object.
- The second object determines the algorithm to execute.

James Tam

The Strategy Algorithm: Example

Name of the folder containing the complete example: `designPatterns/strategy`

```
public class Driver {
   public static void main (String [] args) {
       MyContainer aContainer = null;

       // First algorithm
       aContainer = new MyContainer (new AddAlgorithm());
       System.out.println(aContainer.executeAlgorithm(2,5));

       // Second algorithm
       aContainer = new MyContainer (new MultiplyAlgorithm());
       System.out.println(aContainer.executeAlgorithm(2,5));
   }
}
```

James Tam

The Strategy Algorithm: An Example (2)

```
public class MyContainer
{
   private Algorithm anAlgorithm;
   public MyContainer (Algorithm anAlgorithm)
   {
      this.anAlgorithm = anAlgorithm;
   }
   public int executeAlgorithm (int x, int y)
   {
      return(anAlgorithm.execute(x,y));
   }
}
```

James Tam

The Strategy Algorithm: An Example (3)

```
public interface Algorithm {
   public int execute (int x, int y);
}

public class AddAlgorithm implements Algorithm {
   public int execute (int x, int y) {
      return (x+y);
   }
}

public class MultiplyAlgorithm implements Algorithm {
   public int execute (int x, int y) {
      return (x*y);
   }
}
```

James Tam

Advantages Of The Strategy Pattern

- It decouples the context/container from the algorithm used by the context/container.
  - For the container it may allow the context/container to easily substitute additional algorithms.
    - ‘Expansion packs’
  - For the algorithm, the algorithm may be used in a number of different contexts/containers (e.g., sorting algorithms).

James Tam

Side Note: Static Attributes

- Static attributes of a class are initialized when the Java virtual machine (“java”) loads a class into memory.
- This must be done before any of the methods of the class can be called (even the constructor).
- **Name of the folder containing the complete example**:

```
designPatterns/static
```

James Tam

Static Attributes: Driver Class

```
public class Driver
{
    public static void main (String [] args)
    {
        Foo aFoo = new Foo();
    }
}
```

James Tam

Static Attributes: Class Foo & Bar

```
public class Foo
{
    private static Bar aBar =
        new Bar();
    public Foo()
    {
        System.out.println(">>>
          Trace only: constructor
          Foo() <<<");
    }
}
```

```
public class Bar
{
    public Bar()
    {
        System.out.println(">>>
          Trace only: constructor
          Bar() <<<");
    }
}
```

The Singleton Pattern

- Singleton class: there is only one instance of the class (one object).
- That object provides a common set of operations for the rest of the program and globally accessible (variable) data.
- It is not the same as a purely static class.
  – Static methods but no variable attributes.
- The Singleton pattern is enforced by making the constructor private.
- Example singleton class: Random number generator.
  – For testing/debugging it is desirable to generate the same sequence of random numbers.

James Tam

Singleton Example

- **Name of the folder containing the complete example**:

`designPatterns/singleton`

James Tam

Singleton: Driver

```
public class DriverSingleRandom
{
    public static void main(String [] args)
    {
        SingleRandom aSingleRandom = SingleRandom.getInstance();
        aSingleRandom.setSeed(1);
        for (int i = 0; i < 10; i++)
            System.out.println(i + ": " +
              aSingleRandom.nextInt());
        System.out.println();
    }
}
```

James Tam

Class SingleRandom

```
                                                            1st
public class SingleRandom
{
    private Random generator;
    private static SingleRandom instance = new SingleRandom();

    private SingleRandom()                                       2nd
    {
        System.out.println(">>> Trace only: this.SingleRandom()
          <<<");
        generator = new Random();                 3rd
    }

    public static SingleRandom getInstance()
    {
        System.out.println(">>> Trace only:
                SingleRandom.getInstance() <<<");
        return(instance);
    }
```

James Tam

Class SingleRandom (2)

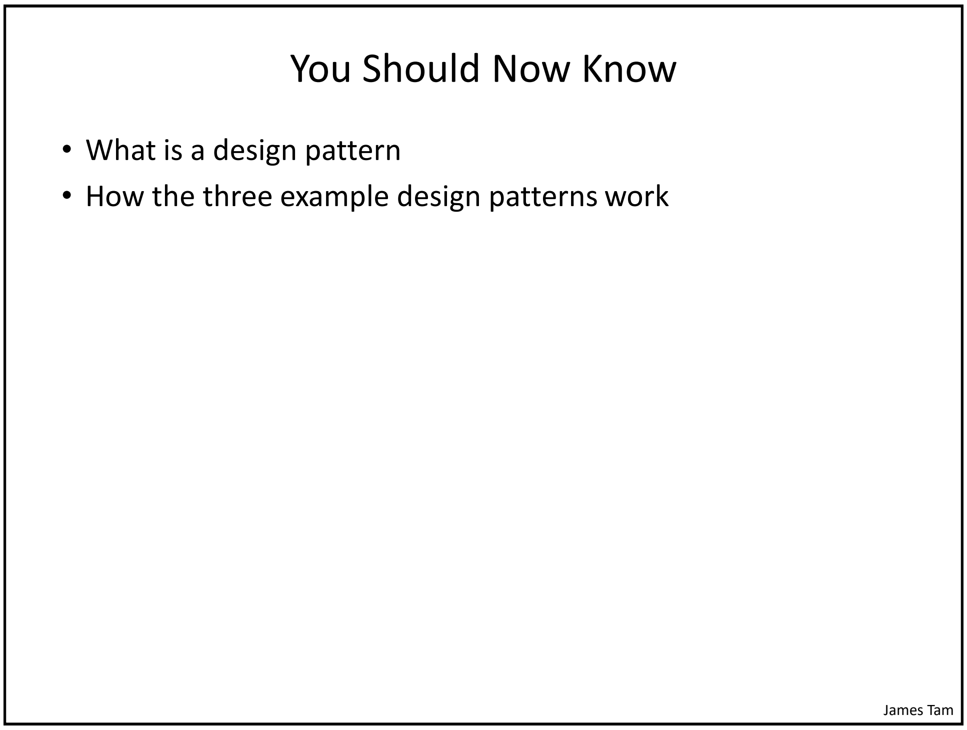
```
    public void setSeed(int seed)
    {
        System.out.println(">>> Trace only: ref.setSeed() <<<");
        generator.setSeed(seed);
    }


    public int nextInt()
    {
        System.out.println(">>> Trace only: ref.nextInt() <<<");
        return (generator.nextInt());
    }

}
```

James Tam

Discussions/Resources: Singleton Pattern

- (Last visited 2021):
  - http://msdn.microsoft.com/en-us/library/ee817670.aspx

James Tam



You Should Now Know

- What is a design pattern
- How the three example design patterns work

James Tam