Final Exam

Thursday, Apr 23

3:30-5:30pm

ES 443 (Here)

Design Patterns Summary

- Introduction
- Structure of Design Patterns
- Categories of Design Patterns

- Singleton Pattern
- Strategy Pattern

By the end of this lecture, you will be able to describe what a design pattern is.

You will also be able to identify a singleton design pattern and a strategy design pattern.

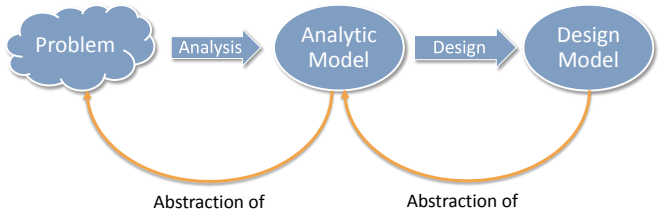
"Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use the solution a million times over."

Christopher Alexander (Architect)
A Pattern Language, Oxford University Press, 1977

Discussion:

What is knowledge and understanding?

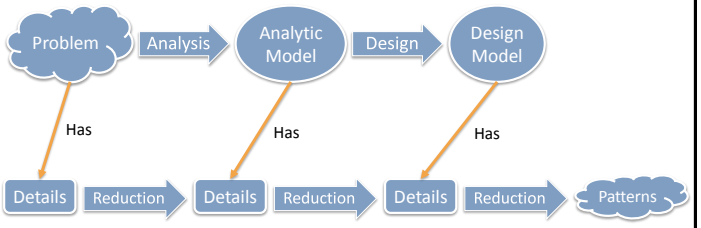
How does it relate to analysis and Design?



Discussion: How does this quote apply to our understanding of analysis and design?

"Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use the solution a million times over."

Christopher Alexander (Architect)
A Pattern Language, Oxford University Press, 1977



Discussion:

How can Design Patterns be applied within multiple contexts?

What is your responsibility as a designer/programmer when you use design patterns?

What work has been done for you, what work has not been done?

Structure of Design Patterns

- Name
- Problem they solve
- Solution
  - elements, relationships, responsibilities, collaborations
- Consequences
  - constraints, tradeoffs

From *Design Patterns* Textbook

- Pattern name and Classification
- Intent
- Also Known As
- Motivation
- Applicability
- Structure
- Participants
- Collaborators
- Consequences
- Implementation
- Sample Code
- Known Uses
- Related Patterns

Categories of Design Patterns

- Creational
- Structural
- Behavioural

Singleton Design Pattern

```
public class Singleton
{
        private static Singleton theInstance = null;

        // instance variables defined here

        public static Singleton getInstance()
        {
                if (theInstance == null)
                {
                        theInstance = new Singleton();
                }
                return theInstance;
        }

        private Singleton()
        {
        }

        // instance methods implemented here.
}
```


Strategy Design Pattern

```
public abstract class AssertIA
{
  private static AssertIA current
                      = new AssertIANormal();

  public static void set(AssertIA assertion)
  {
    if (assertion != null)
    {
      current = assertion;
    }
  }

  // cont'd on next slide
```


Strategy Design Pattern (cont'd)

```
  public static void assertTrue(boolean assertion,
                                String message)
  {
    current.assertTrueImpl(assertion, message);
  }

  public static void assertFalse(boolean assertion,
                                 String message)
  {
    current.assertFalseImpl(assertion, message);
  }

  public abstract void assertTrueImpl(boolean cond,
                                      String message);
  public abstract void assertFalseImpl(boolean cond,
                                       String message);
}
```


Strategy Design Pattern (cont'd)

```
public class AssertIANormal extends AssertIA
{
  public void assertTrueImpl(boolean assertion,
                             String errorMessage)
  {
    if (!assertion)
      throw new IllegalArgumentException(errorMessage);
  }

  public void assertFalseImpl(boolean assertion,
                              String errorMessage)
  {
    if (assertion)
      throw new IllegalArgumentException(errorMessage);
  }
}
```

Strategy Design Pattern (cont'd)

```
public class AssertIADebug extends AssertIA
{
  public void assertTrueImpl(boolean assertion,
                             String errorMessage)
  {
    System.out.println("Assert True: " +
                       assertion + " " + errorMessage);
    if (!assertion)
      throw new IllegalArgumentException(errorMessage);
  }

  public void assertFalseImpl(boolean assertion,
                              String errorMessage)
  {
    System.out.println("Assert False: " +
                       assertion + " " + errorMessage);
    if (assertion)
      throw new IllegalArgumentException(errorMessage);
  }
}
```


References

The material in these notes is based on *Design Patterns: Elements of Reusable Object-Oriented Software* by Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. Addison-Wesley, 1995.

Assignment #4 Discussion

Next Class

- HCI
- Demo