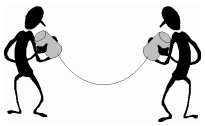
Software Development in Teams & Organizations

Design Patterns

For an overview of the various patterns, see pages.cpsc.ucalgary.ca/~kremer/patterns/

Design Patterns

- A pattern is a tested solution to a common problem within a specific context.
- Advantages:
  - Shared language of design
  - Decreases misunderstandings
  - No reinventing - learn from others' experiences
  - Good first step to understand good designs to innovate new ones.



http://www.cs.cmu.edu/~aldrich/courses/413/

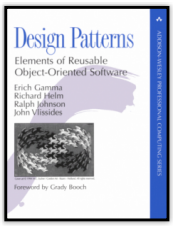
Elements of Design Patterns

- Name
  - becomes part of design vocabulary & improves communication
- Problem
  - What kind of problem can this pattern solve?
- Solution
  - The needed design elements and their relationships
- Consequences
  - Tradeoffs of applying the pattern (e.g. flexibility, extensibility, performance)
  - There may be variations in the pattern with different consequences.

http://www.cs.cmu.edu/~aldrich/courses/413/

Get the book!

- Brought design patterns into the mainstream.
- Authors known as the Gang of Four (GoF).
- Contains descriptions of communicating objects and classes that are customized to solve a general design problem in a particular context.



http://www.cs.cmu.edu/~aldrich/courses/413/

Types of Patterns

- Creational patterns
- Structural patterns
- Behavioral patterns.

http://www.freejavaguide.com/java-design-patterns.pdf

Creational Patterns

- Prescribe the way objects are created.
- Used when a decision must be made at the time a class is instantiated.
- Details of the instantiated classes are encapsulated by an abstract superclass and hidden from the client class.
- Client class knows only about the abstract class or the interface it implements.
- The specific type of the concrete class is typically unknown to the client class.

http://www.freejavaguide.com/java-design-patterns.pdf

Example: Singleton Pattern

- Encapsulates the creation of an object to maintain control over it.

- Usually used to:
  - Ensure that only one instance is created.
  - Allow lazy instantiation: the instantiation of the object can be delayed until it is actually needed.

http://www.freejavaguide.com/java-design-patterns.pdf

Example: Singleton Pattern

provide unique sequential numbers, such as might be required for use as p database:

```
// Sequence.java
 public class Sequence {
   private static Sequence instance;
   private static int counter;
   private Sequence()
   {
     counter = 0; // May be necessary to obtain
                  // starting value elsewhere...
   }
   public static synchronized Sequence getInstance()
   {
     if(instance==null) // Lazy instantiation
     {
       instance = new Sequence();
     }
     return instance;
   }
   public static synchronized int getNext()
   {
     return ++counter;
   }
 }
```

http://www.freejavaguide.com/java-design-patterns.pdf

Structural Patterns

- Prescribe the organization of classes and objects.
- Concerned with how classes inherit from each other or how they are composed from other classes.
- They introduce a level of indirection between a client class and a class it wants to use.

http://www.freejavaguide.com/java-design-patterns.pdf

Example: Adapter Pattern

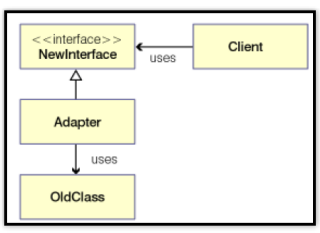
- Used to allow the reuse of a class that is similar, but not the same, as the class the client class would like to see.

- Typically the original class is capable of supporting the behavior the client class needs, but does not have the interface the client class expects.
- It may not possible or practical to alter the original class.

http://www.freejavaguide.com/java-design-patterns.pdf

Example: Adapter Pattern

```
public class OldClassAdapter implements NewInterface {
  private OldClass ref;
  public OldClassAdapter(OldClass oc)
  {
    ref = oc;
  }
   public void NewMethod()
  {
     ref.OldMethod();
  }
}
```



http://www.freejavaguide.com/java-design-patterns.pdf

Behavioral Patterns

- Prescribe the way objects interact with each other.
- Help make complex behavior manageable by specifying the responsibilities of objects and the ways they communicate with each other.

http://www.freejavaguide.com/java-design-patterns.pdf

Example: Observer Pattern

- Typically used when implementing an application with a presentation layer and a model (data) layer.
- Decouples the presentation of data from the data itself.
- Must know!! Very powerful.

http://www.freejavaguide.com/java-design-patterns.pdf

Example: Observer Pattern

```
import java.util.*;
class Sensor extends Observable {
private int temp = 68;
  void takeReading()
  {
   double d;
   d =Math.random();
   if(d>0.75)
   {
     temp++;
     setChanged();
   }
   else if (d<0.25)
   {
     temp--;
     setChanged();
   }
   System.out.print("[Temp: " + temp + "]");
}
public int getReading()
{
  return temp;
}
}
```

http://www.freejavaguide.com/java-design-patterns.pdf

Example: Observer Pattern

```
public class Display implements Observer {
  public void update(Observable o, Object arg)
  {
     System.out.print("New Temp: " +  ((Sensor) o).getReading());
  }
  public static void main(String []ac)
  {
     Sensor sensor = new Sensor();
     Display display = new Display();
     // register observer with observable class
     sensor.addObserver(display);
     // Simulate measuring temp over time
     for(int i=0; i < 20; i++)
     {
        sensor.takeReading();
        sensor.notifyObservers();
        System.out.println();
     }
  }
}
```

http://www.freejavaguide.com/java-design-patterns.pdf

Sample Exam Question

- The composite pattern is defined as a tree structure of objects where every object has the same interface. Is this a…
  - A. Structural pattern?
  - B. Behavioral pattern?
  - C. Creational pattern?
  - D. Anti-pattern?