Our First Pattern

Façade
Façade Pattern

**Purpose**: Provide a simplified interface to another system.

**Motivation**: Have to interact with a complex interface, but only need a subset. So you write a new interface that just talks to the parts you need.

Makes original interface simpler and easier to use.
Interface?

Remember that the interface is the publicly accessible parts of the code.
- Frequently this is the GUI!

So generally we are
- hiding some of the methods.
- hiding some of the GUI.
- or both.
Example

I have written an internet security program with a fancy “click and point” user interface.

- can scan for viruses
- can send secure emails
- can setup a firewall
- etc.

I need to write an installation program for my security software.

But I’m afraid that someone will write a virus that hurts the installation.
Example (cont. 1)

So I need to have my installer program run the anti-virus part of my program.

BUT I don’t want to give the user all of the interface options during installation.

Solution: I provide a simplified interface to the anti-virus check

- no interface for firewall
- no interface for secure email
- etc.
Example (cont. 2)

This is a façade – a new simplified interface to the whole complex interface.
Façade Consequences

From example we can see that

1. Façade is easier to use (than whole interface).
2. Façade hides some functionality.

Better be sure these consequences are ok!
- (especially #2)
Another Example

I have written software that accesses the NCAR database, the NOAA database, and various satellite database and feeds the info into a weather prediction model.

The user can choose which satellites, which databases, how many days to predict, etc.
Another Example (cont. 1)

I am now writing software that lets high school students run the model over the web.

- only one day prediction
- only one satellite
- basically point and click.

Very simplified.
Another Example (cont. 2)

Original interface

- NOAAInterface
- NCARInterface
- SatelliteInterface
Another Example (cont. 3)

Façade interface

Composition and/or aggregation is very common
Another Example (cont. 4)

Or possibly...

```
GenericInterface

NOAAInterface
NCARInterface
SatelliteInterface

Pro
Amateur
Kiddie
```
Façade Implementation

- Create new class(es) with simplified interface.

- Have new class use other existing classes.
  - Probably aggregation.
    - The existing classes have a reason to be there.

- Code solution will depend on the existing code, so hard to give a single example.
  - Evaluate existing code and decide how to simplify the interface.
    - The interface is made of the public methods.
    - The interface is often the GUI you see on the screen.
  - Frequently uses composition of the class that is being “hidden”.
Example With Linked List

The Java LinkedList API has many, many methods including peekFirst, peekLast.

- Many of these are not part of the standard LinkedList.
- They renamed the standard “insert” and “delete” – called them “add” and “remove”.

Let’s write a façade to make it easier and more inline with a basic linked list.

- Only use methods insert and delete.
- Only store “ints” instead of general Objects.
import java.util.LinkedList;

/**<n
* A facade for the linked list api. Reduces the number of methods to only the most common and most important. Renames the methods to insert and delete, which is what most users expect.
* @author David Bahr
*/
public class LinkedListFacade
{
    // the fancy non-traditional Java linked list
    private LinkedList list = null;

    /**
     * Create a traditional linked list.
     */
    public LinkedListFacade()
    {
        list = new LinkedList();
    }

    /**
     * Inserts the specified value at the given position.
     * @param index  The position where the value will be inserted
     * @param value  The value that will be inserted.
     */
    public void insert(int index, int value)
    {
        list.add(index, new Integer(value));
    }

    /**
     * Removes and returns the value at the given position.
     * @param index  The position where the value will be deleted.
     * @return The value contained by the node being removed.
     */
    public int delete(int index)
    {
        return ((Integer) list.remove(index)).intValue();
    }
}
Ok, Your Turn

Create teams of 2.

Create a façade for the Stack class in the Java API.
- Only allow access to the push, pop, and top methods.

Draw the UML diagram for your façade.