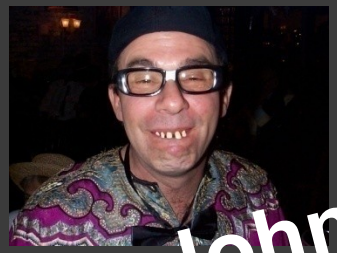
ADAPTER PATTERN

By SmartBoard team

Agenda

- Problem & Exercises
- Definition of Adapter
- Object Adaptor
- Class Adaptor
- Pros and Cons
- References

Problems...



Mr. John

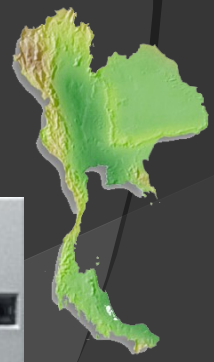
110 volts

220 volts

How to fix it ?

- Please help Mr. John..

110 volts



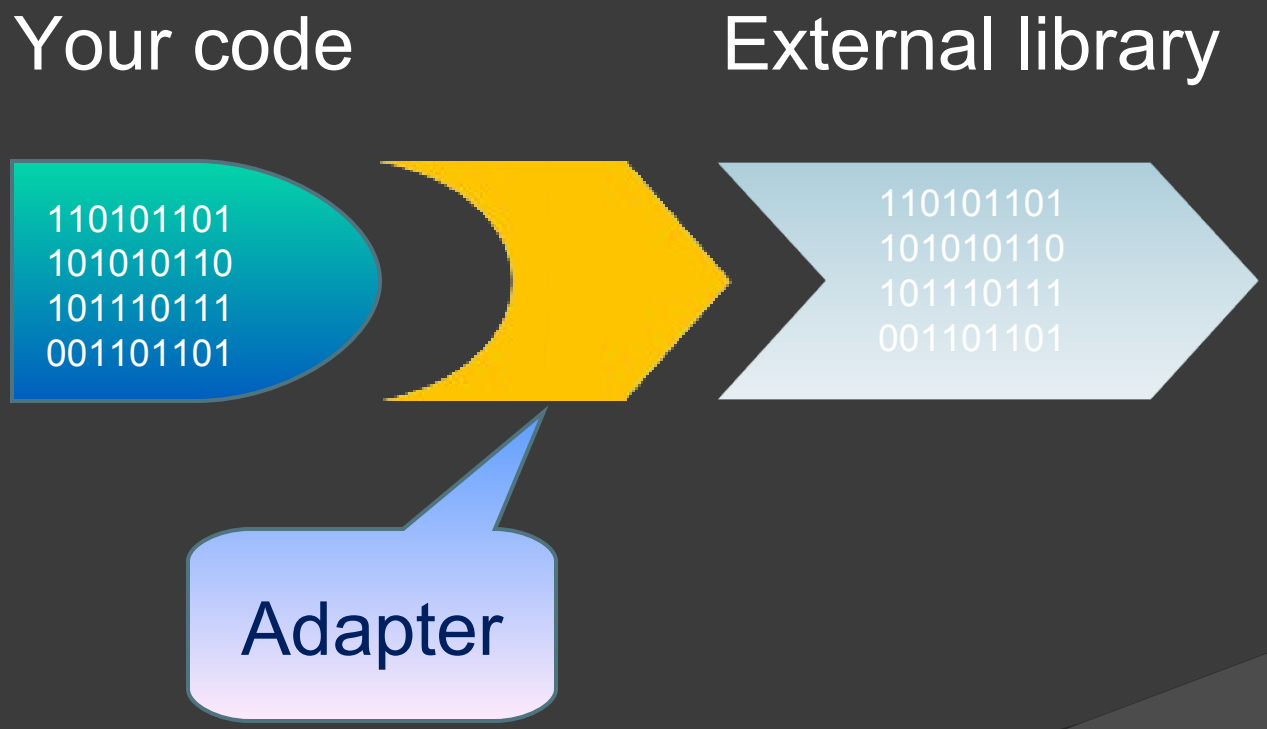
220 volts

Definition

Wrapper pattern

- An **Adapter pattern** convert the interface of a class into another interface client expect
- It allows classes to work together that normally could not because of incompatible interfaces by wrapping its incompatible interface with another interface client expect
- The adapter pattern is used so that two or more unrelated interfaces can work together

For more understandable

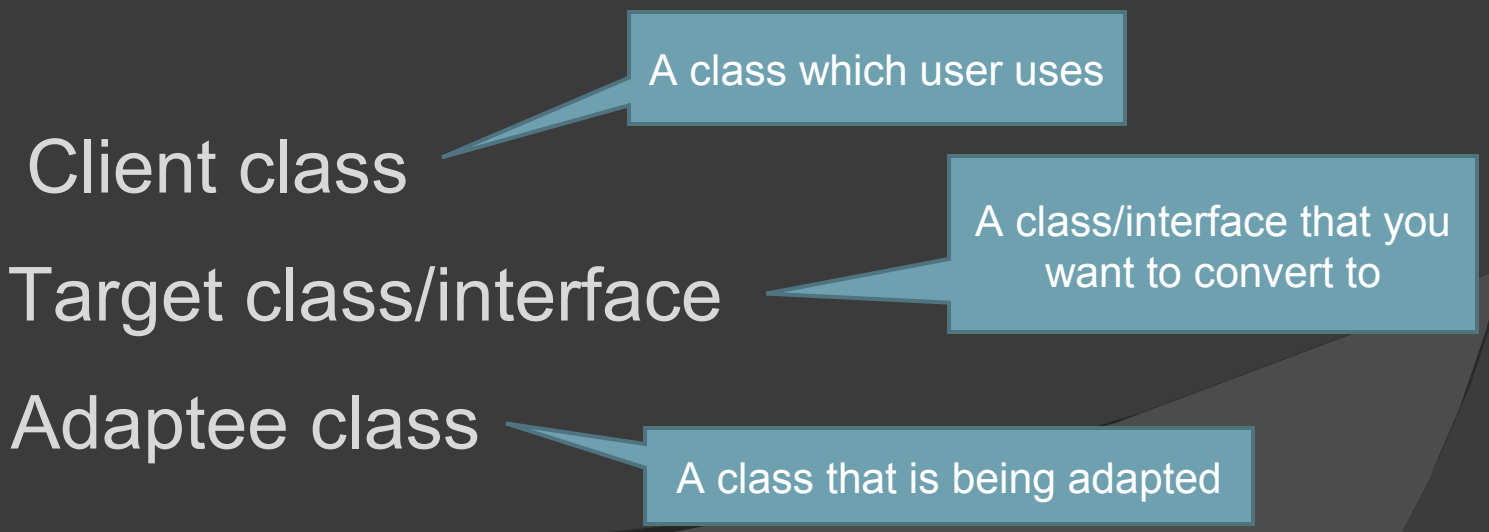


When to use Adapter ?

- You want to use an existing class, and its interface doesn't match with the others you need
- You want to create a reusable co-op class that cooperates with unrelated classes with incompatible interfaces

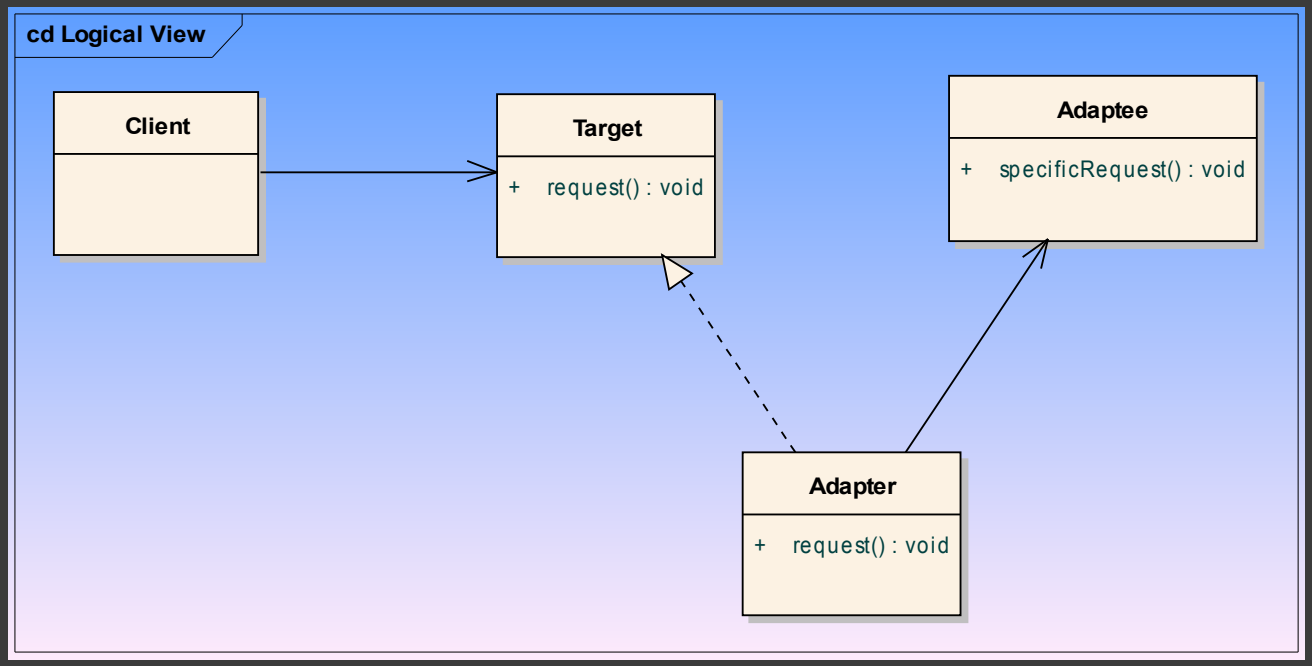
Ways to implement adaptor

- We have 2 ways to implement
  - Object Adaptor
  - Class Adaptor



Object Adaptor

- Or we can call Object Composition



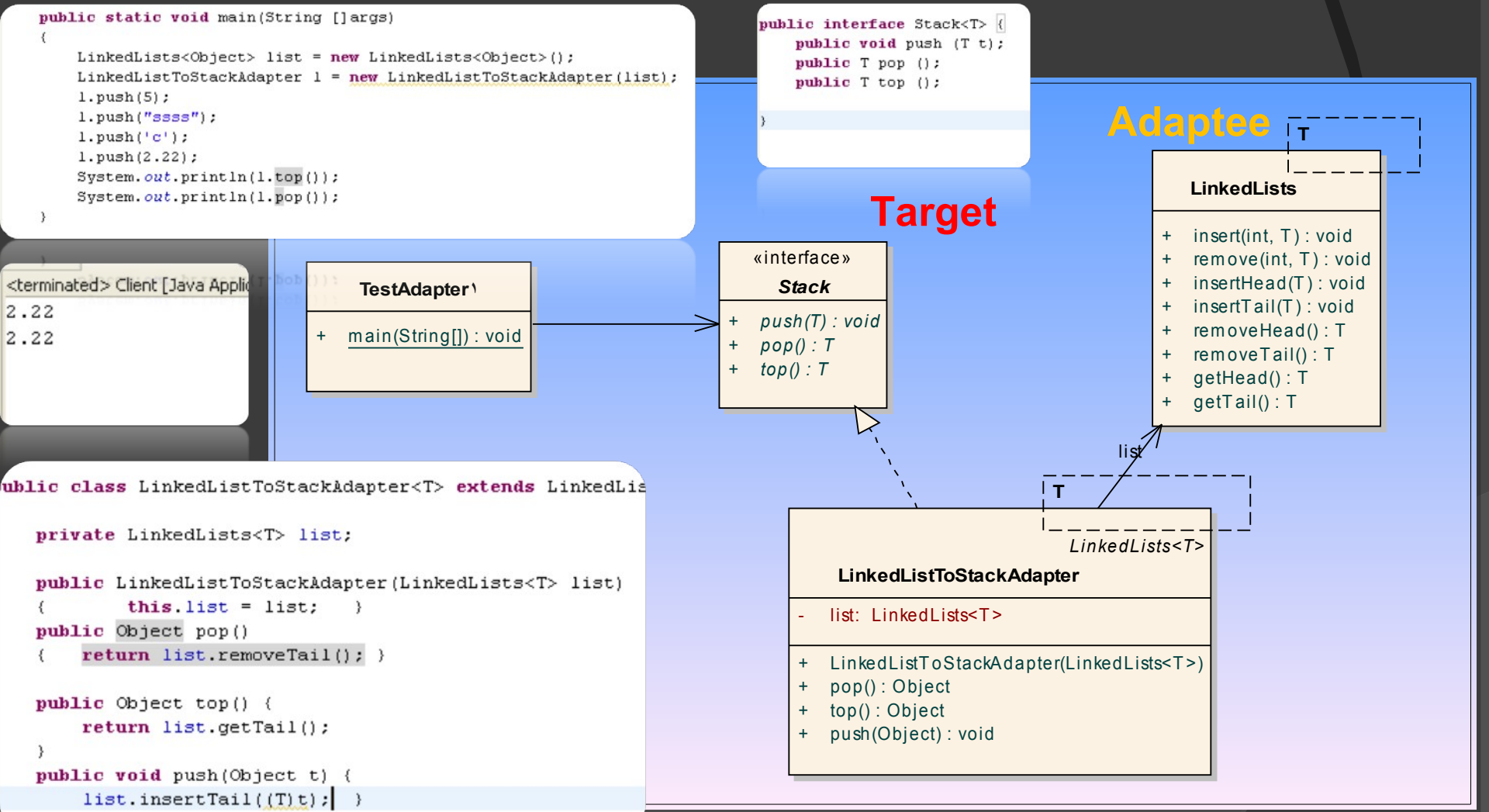
Example : Object Adapter

- We need to implement **Stack**
- But only library we have is **Linked Lists**

```
public static void main(String []args)
{
    LinkedLists<Object> list = new LinkedLists<Object>();
    LinkedListToStackAdapter l = new LinkedListToStackAdapter(list);
    l.push(5);
    l.push("ssss");
    l.push('c');
    l.push(2.22);
    System.out.println(l.top());
    System.out.println(l.pop());
}
```

```
<terminated> Client [Java Appli
2.22
2.22
```

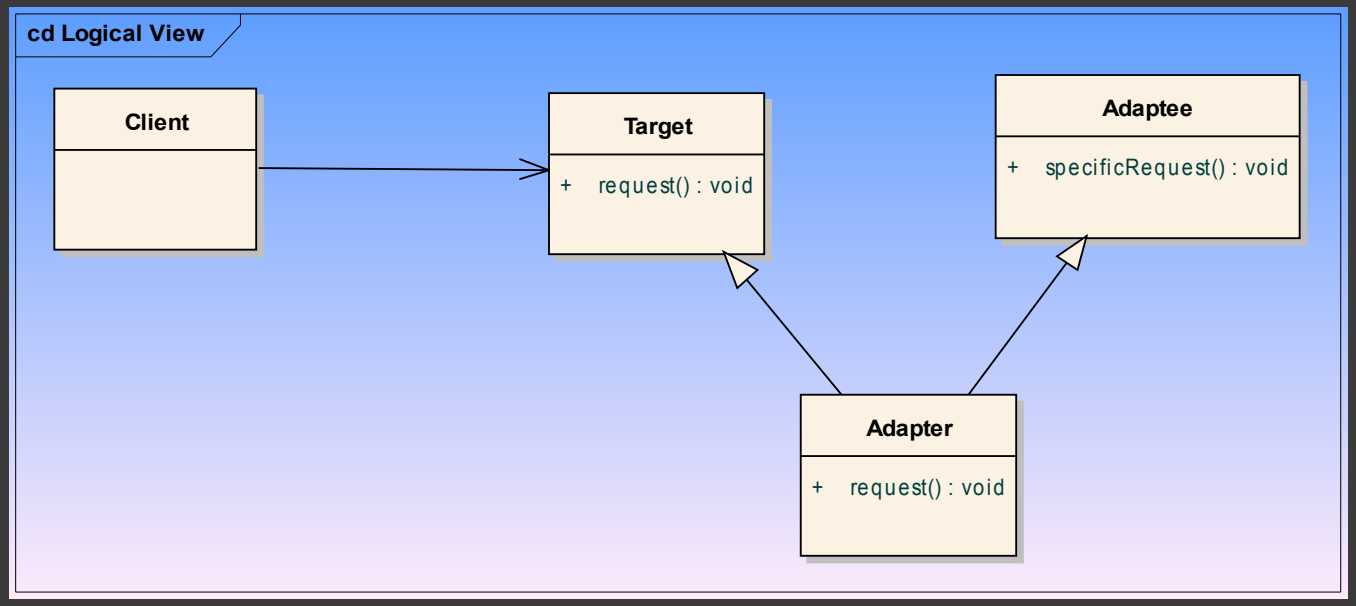
```
public interface Stack<T> {
    public void push (T t);
    public T pop ();
    public T top ();
}
```




```
ublic class LinkedListToStackAdapter<T> extends LinkedLis

   private LinkedLists<T> list;

   public LinkedListToStackAdapter(LinkedLists<T> list)
   {       this.list = list;    }
   public Object pop()
   {    return list.removeTail(); }

   public Object top() {
       return list.getTail();
   }
   public void push(Object t) {
       list.insertTail((T)t);  }
```

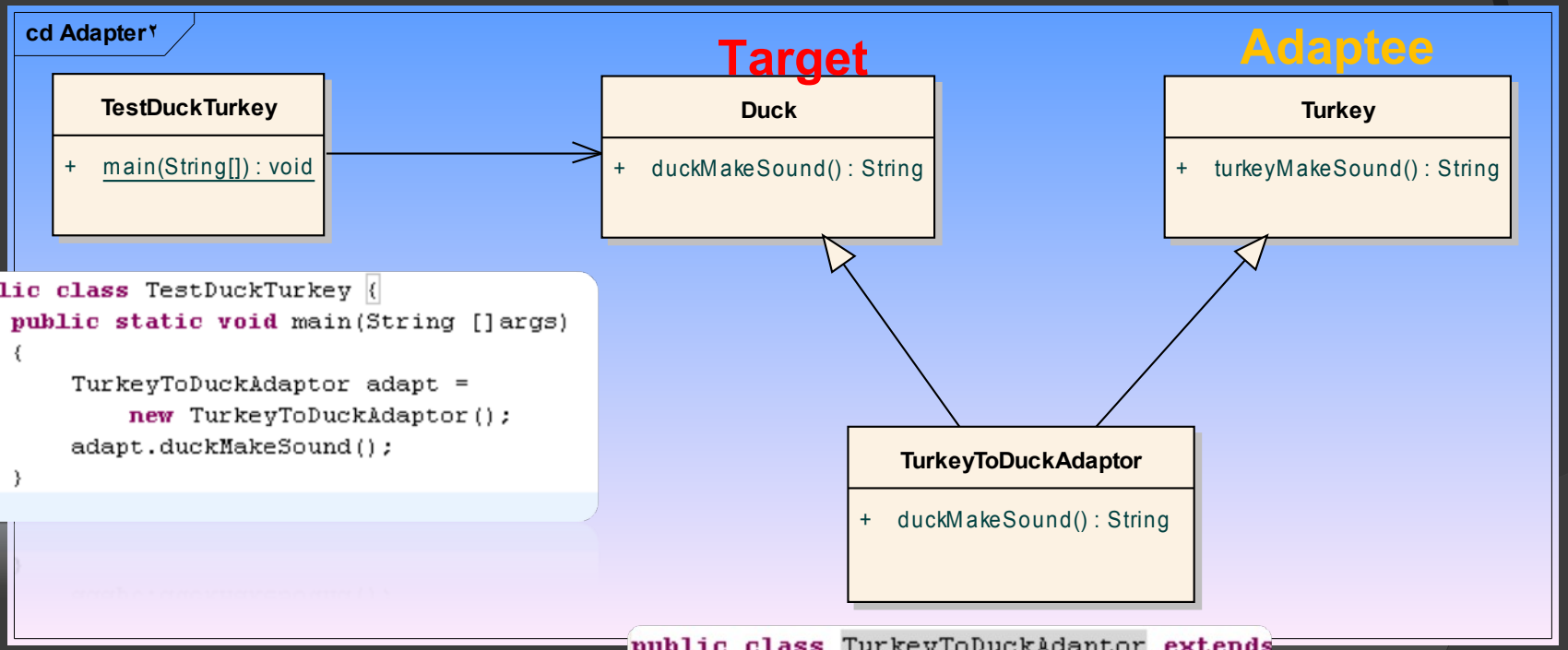
Class Adaptor

Multiple Inheritance
Can't do in JAVA



Example : Class Adaptor

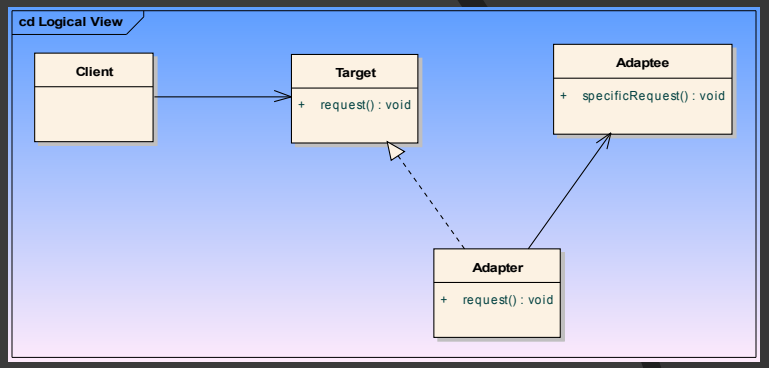
- We have **Turkey** and **Duck**
- And we want **Duck** to make sounds like **Turkey**




```
public class TestDuckTurkey {
    public static void main(String []args)
    {
        TurkeyToDuckAdaptor adapt =
            new TurkeyToDuckAdaptor();
        adapt.duckMakeSound();
    }
}
```

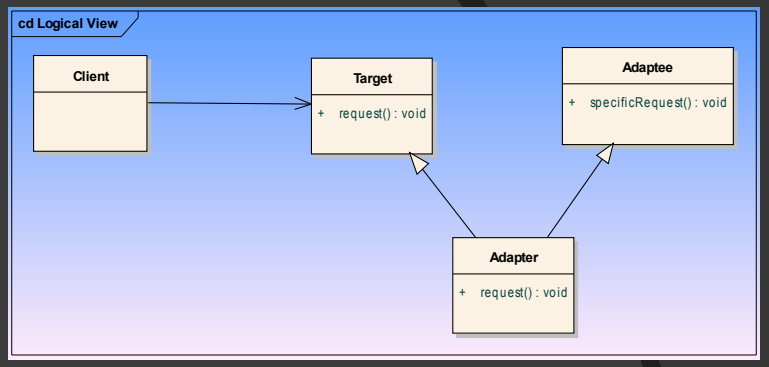
```
public class TurkeyToDuckAdaptor extends
    public String duckMakeSound()
    {
        return turkeyMakeSound();
    }
```

Object Adaptor



- Pros
  - More flexible than class Adapter
  - Doesn't require sub-classing to work
  - Adapter works with Adaptee and all of its subclasses
- Cons
  - Harder to override Adaptee behaviour
  - Requires more code to implement properly

Class Adaptor



- Pros
  - Only 1 new object, no additional indirection
  - Less code required than the object Adapter
  - Can override Adaptee's behaviour as required
- Cons
  - Requires sub-classing (tough for single inheritance)
  - Less flexible than object Adapter

Pros and Cons

- Pros
  - Let the different interface classes can work together
  - Easy maintainable for adaptor class
  - Increase the ability of the particular class from adapting the other classes
- Cons
  - Required multiple inheritances in class adaptor, some prog. languages are not supported
  - If adaptee class is huge and some part of it is not used, adaptor class will be big unnecessarily.

Facade pattern

Wh facade?

One Happy Meal


I need Set 1
I need Happy meal 2 sets
I need set 4
I need lunch special

(cons.)

Get these commands to our pos machine.

et's use acade ern to olve it.

Set 1

Happy Meal

Set A

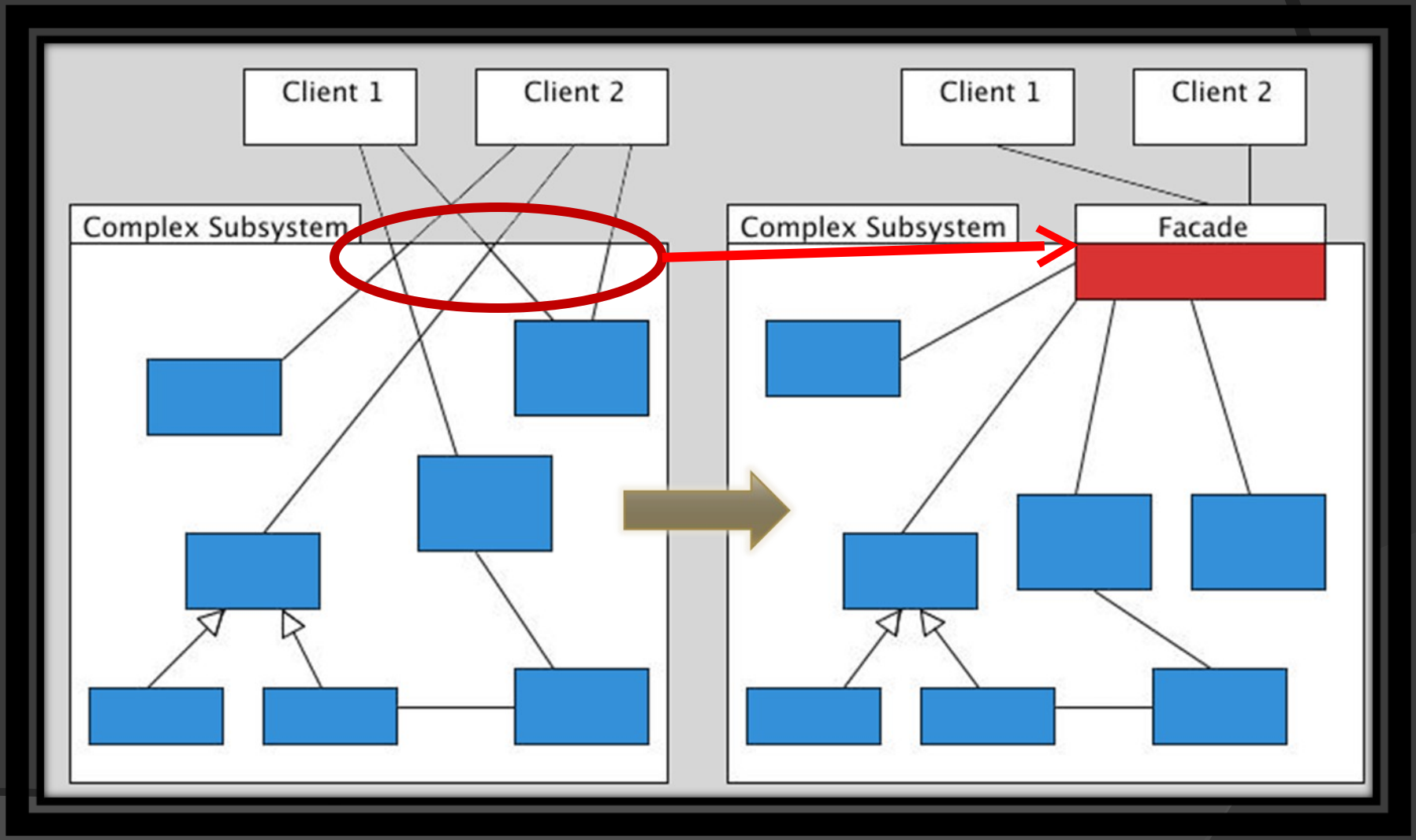
Special Lunch

Set 2

Definition

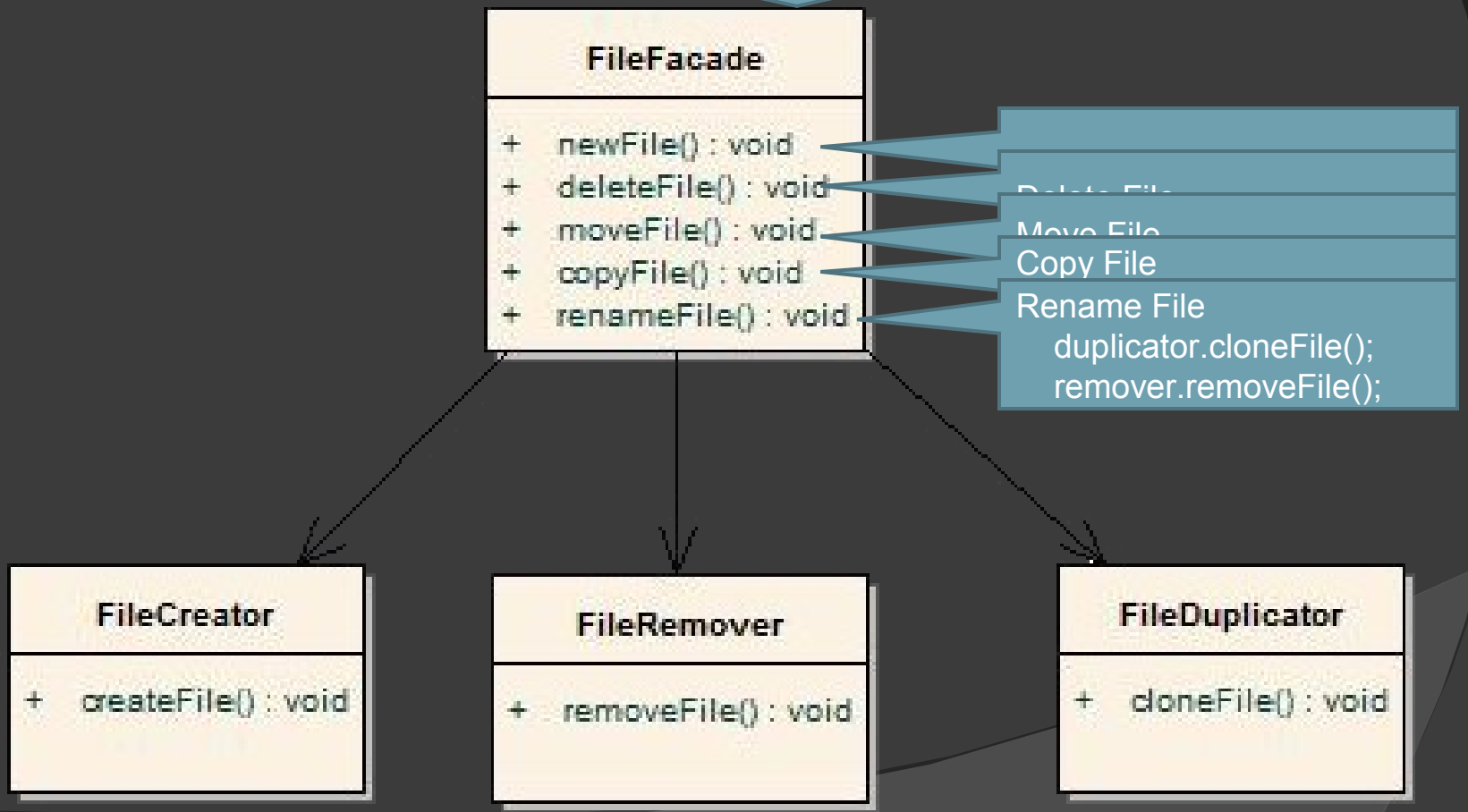
- Make a complex system simpler by providing a unified or general interface, which is a higher layer to these subsystems
- make a software library easier to use and understand, since the facade has convenient methods for common tasks
- reduce dependencies of outside code on the inner workings of a library, since most code uses the facade, thus allowing more flexibility in developing the system;
- wrap a poorly designed collection of APIs with a single well-designed API (As per task needs).

Facade Structure

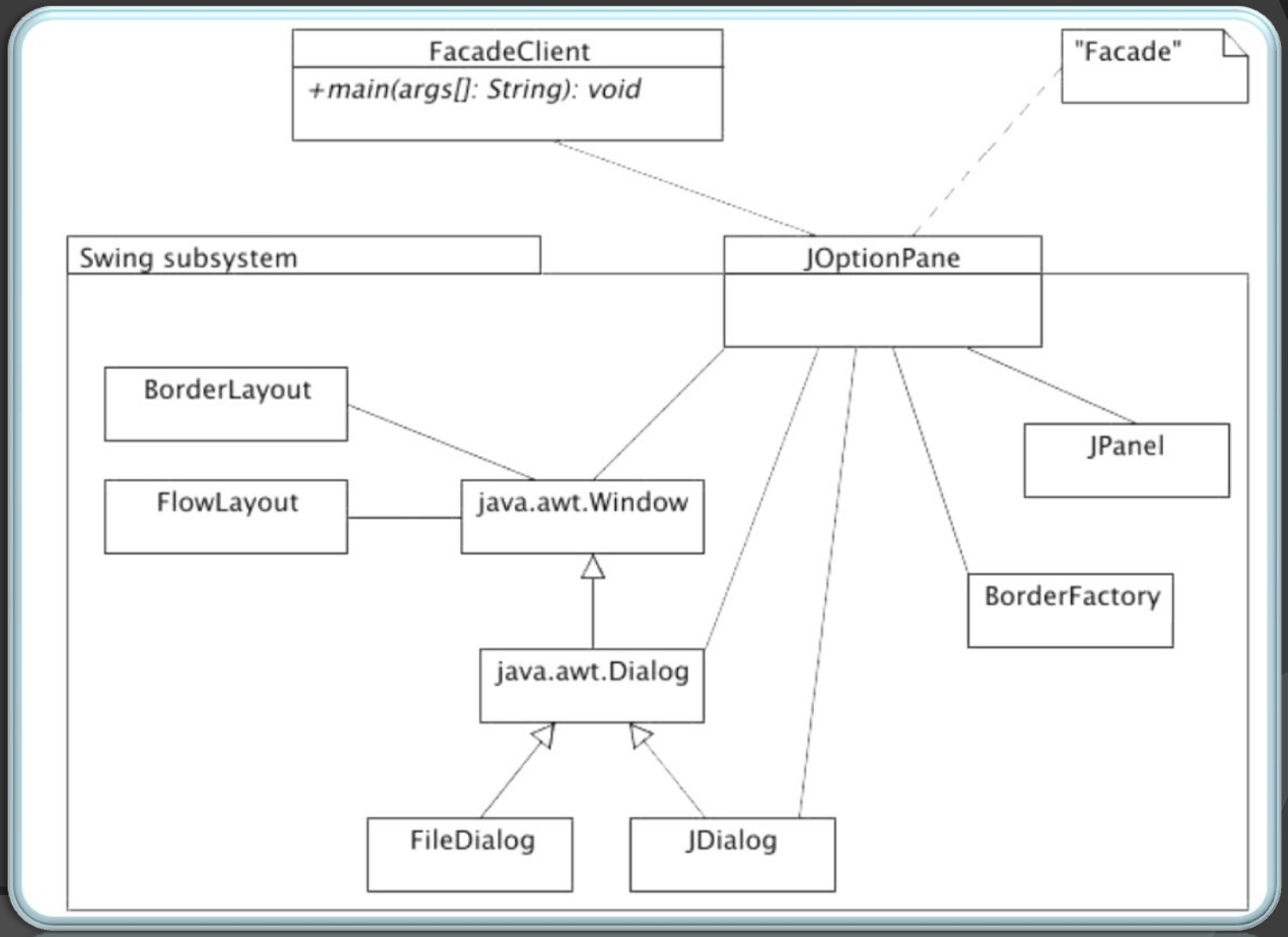


Easy Example FileFacade

```
private FileCreator creator   =  new FileCreator();
private FileRemover remover   = new FileRemover();
private FileDuplicator duplicator = new FileDuplicator();
```



Complex Example SwingFacade



Pros & Cons

- Pros
  - Hides the implementation from clients,
  - Reduces class dependencies in large systems
  - Easier to reuse or maintain if the routine is changed, or even there's a new routine.
- Cons
  - The subsystem class is not encapsulated, clients still can access it.

References

- http://en.wikipedia.org/wiki/Wrapper_pattern
- http://developerlife.com/tutorials/?p=19#\
- http://en.wikipedia.org/wiki/Fa%C3%A7ade_pattern
- http://c2.com/cgi/wiki?AdapterPattern
- http://userpages.umbc.edu/~tarr/dp/lectures/Adapter-2pp.pdf