CS 525 - ASD

Advanced Software Development

MS.CS Program

Department of Computer Science

Rene de Jong, MsC.

Maharishi University
OF MANAGEMENT

CS 525 - ASD

Advanced Software Development

Maharishi University
OF MANAGEMENT

Lesson 9 Decorator pattern

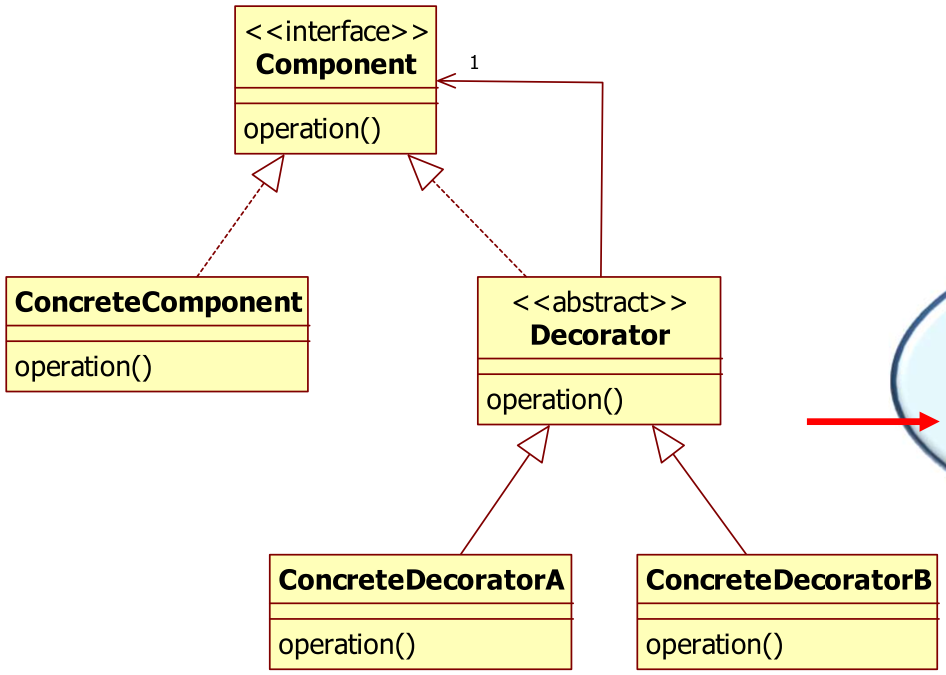
Decorator pattern

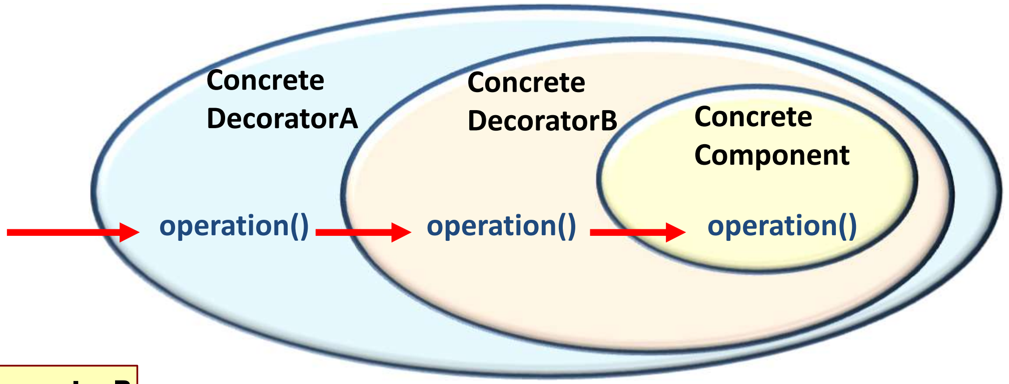
- Allows to dynamically add new behavior to an existing object.

Plain pizza crust

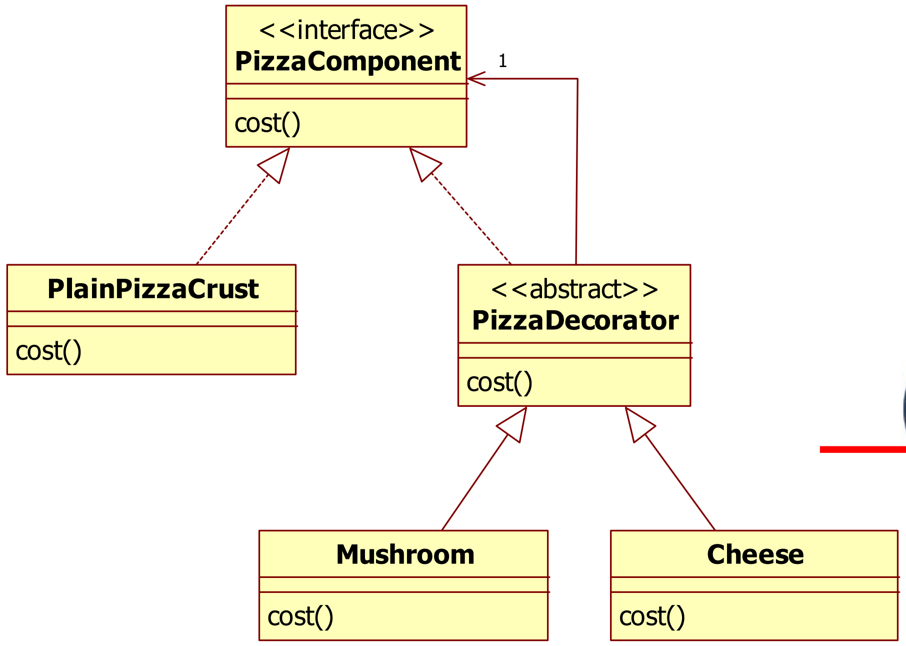
Pizza toppings

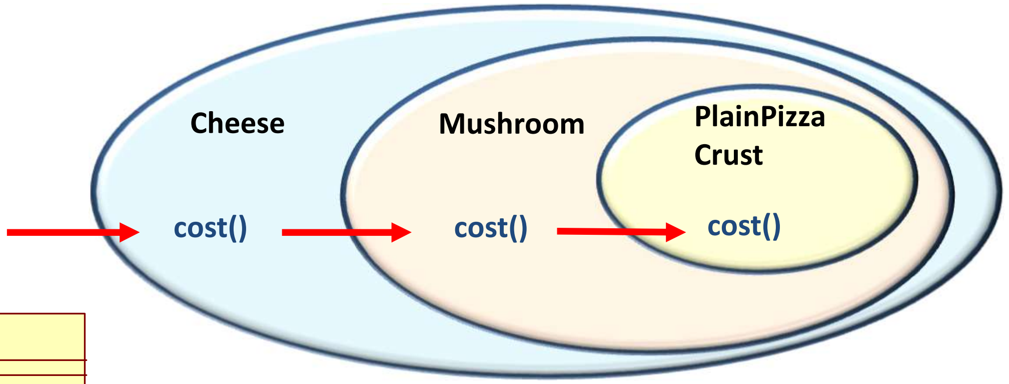
Decorator pattern



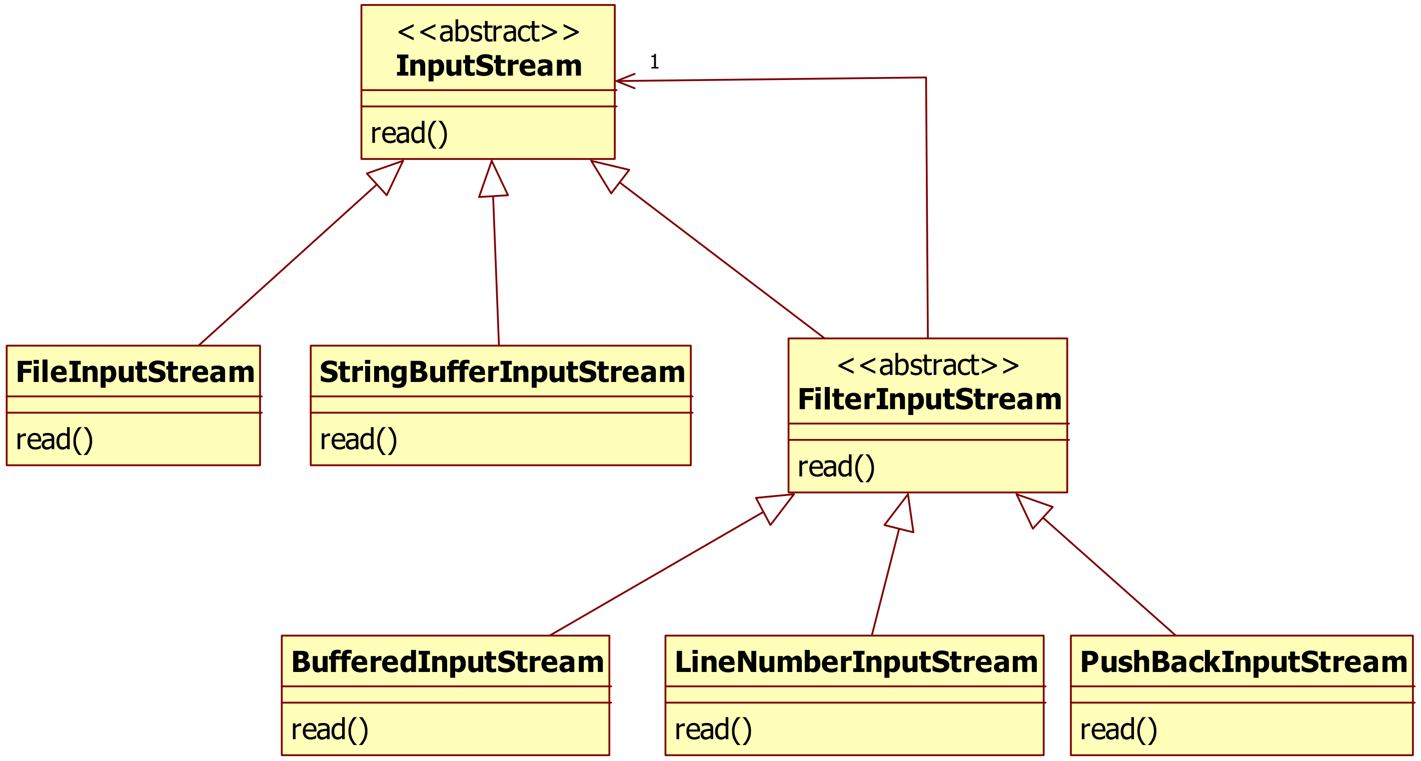


Decorating a pizza





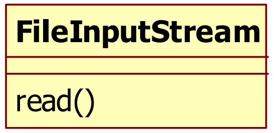
Java.io

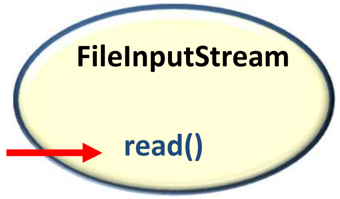

<<abstract>>
InputStream
read()
1
FileInputStream
read()
StringBufferInputStream
read()
<<abstract>>
FilterInputStream
read()
BufferedInputStream
read()
LineNumberInputStream
read()
PushBackInputStream
read()

FileInputStream

```java
public class Application {

  public static void main(String[] args) {
    int c;
    String rootPath = Thread.currentThread().getContextClassLoader().getResource("").getPath();
    try {
      InputStream inputStream = new FileInputStream(rootPath + "/input.txt");

      while ((c = inputStream.read()) >= 0) {
        System.out.print((char) c);
      }

      inputStream.close();
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
}
```

Reads a byte of data

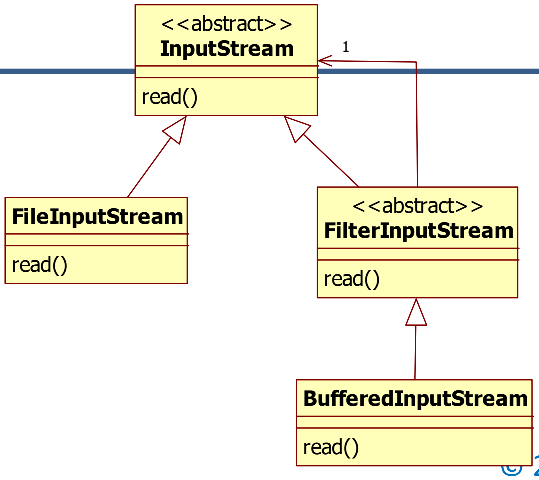


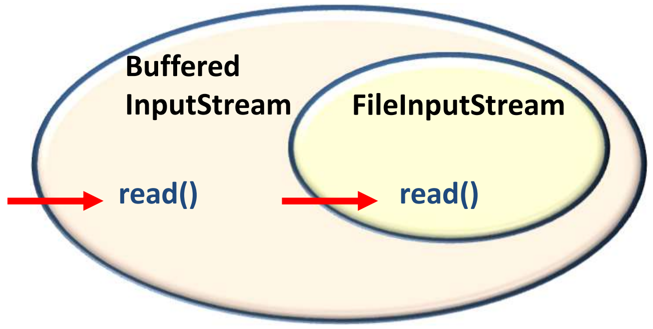


BufferedInputStream

```
public class Application {

  public static void main(String[] args) {
    int c;
    String rootPath = Thread.currentThread().getContextClassLoader().getResource("").getPath();
    try {
      InputStream inputStream =
            new BufferedInputStream(new FileInputStream(rootPath + "/input.txt"));

      while ((c = inputStream.read()) >= 0) {
        System.out.print((char) c);
      }

      inputStream.close();
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
}
```

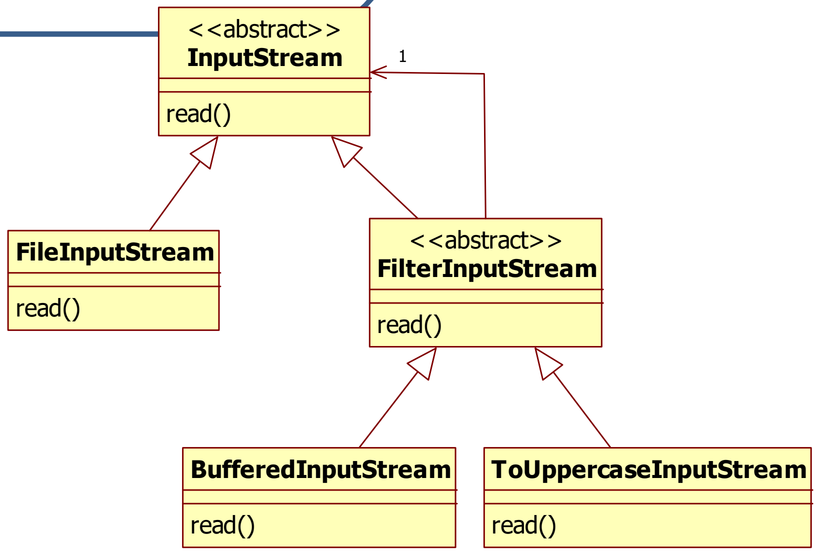
Reads 8 kilobytes of data and buffers them





Write your own decorator

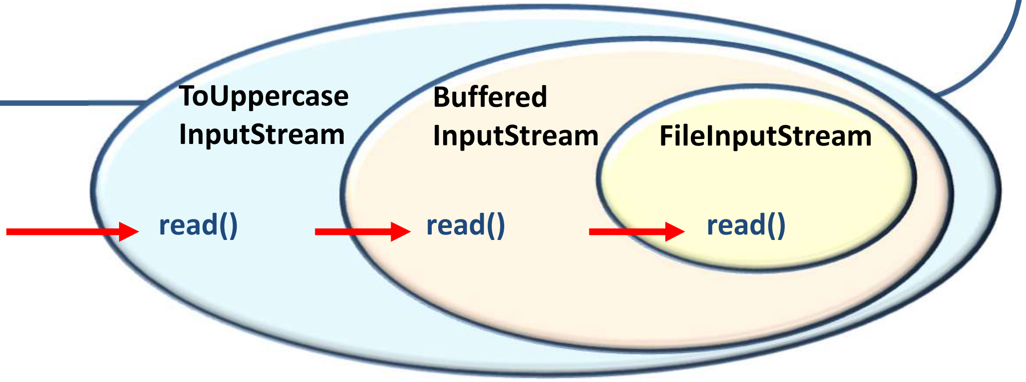
```
public class ToUppercaseInputStream extends FilterInputStream {

  protected ToUppercaseInputStream(InputStream in) {
    super(in);
  }

  @Override
  public int read() throws IOException {
    int c = super.read();
    if (c != -1)
      c = Character.toUpperCase((char)c);
    return c;
  }
}
```



ToUppercaseInputStream

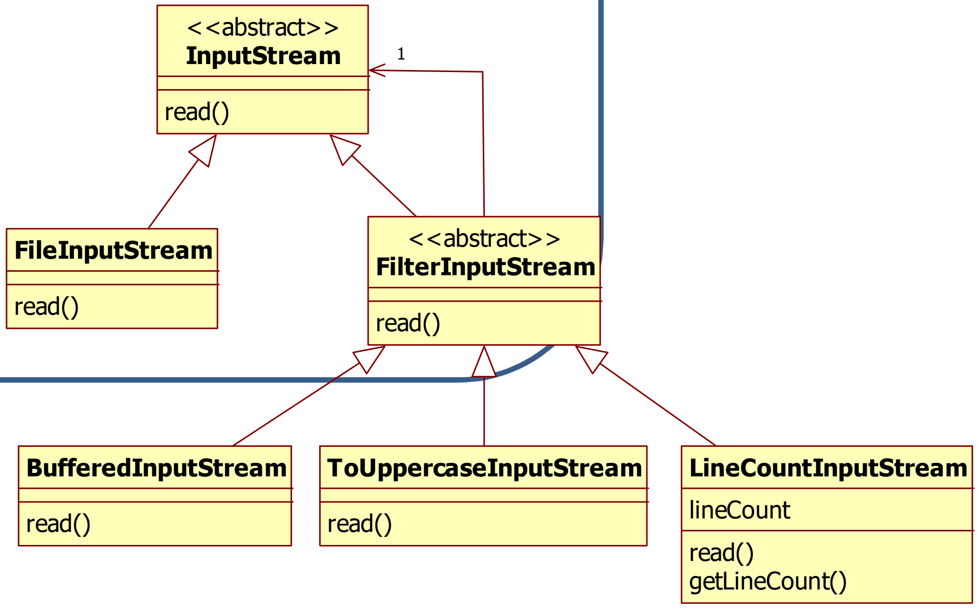
```
public class Application {

  public static void main(String[] args) {
    int c;
    String rootPath = Thread.currentThread().getContextClassLoader().getResource("").getPath();
    try {
      InputStream inputStream =
            new ToUppercaseInputStream(new BufferedInputStream(
                new FileInputStream(rootPath + "/input.txt")));
      while ((c = inputStream.read()) >= 0) {
        System.out.print((char) c);
      }

      inputStream.close();
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
}
```

Add decorators to the FileInputStream



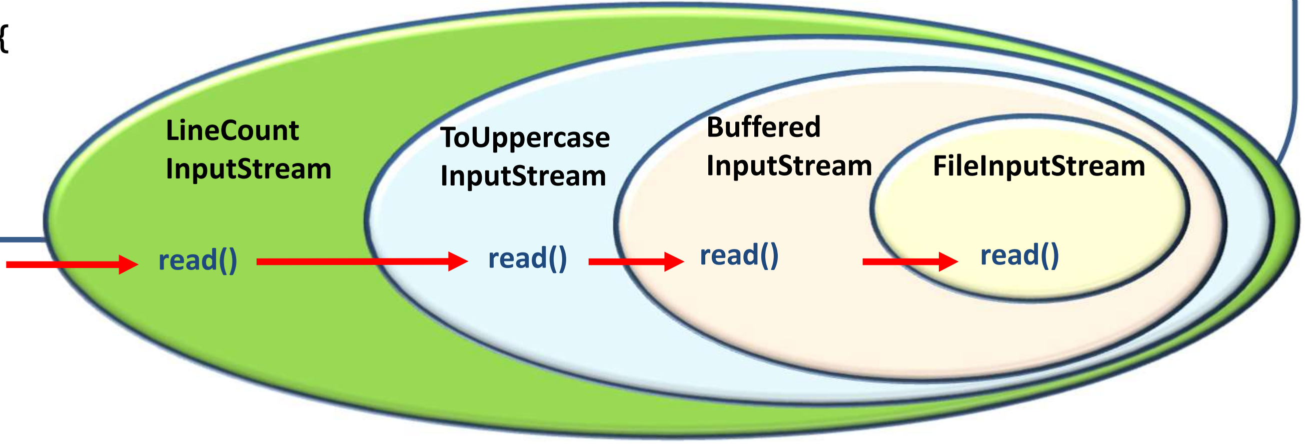
Write your own decorator

```
public class LineCountInputStream extends FilterInputStream {
  int lineCount = 0;

  protected LineCountInputStream(InputStream in) {
    super(in);
  }

  @Override
  public int read() throws IOException {
    int c = super.read();
    if (c != -1 && c==10 ) //carriage return = 10
      lineCount++;
    return c;
  }

  public int getLineCount() {
    return lineCount;
  }
}
```

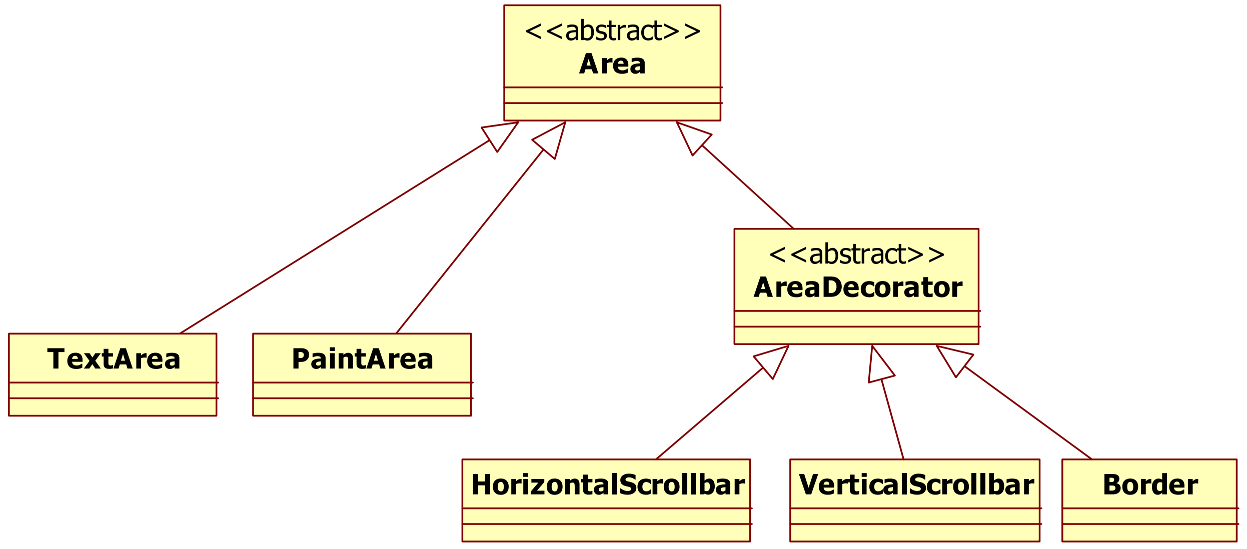


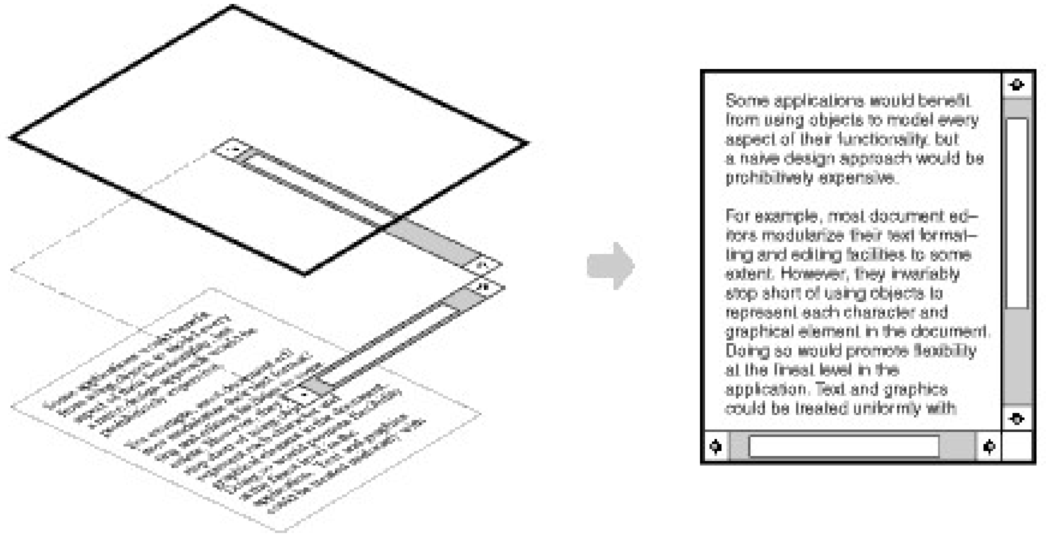
LineCountInputStream

```
public class Application {

  public static void main(String[] args) {
    int c;
    String rootPath = Thread.currentThread().getContextClassLoader().getResource("").getPath();
    try {
      LineCountInputStream inputStream =
        new LineCountInputStream(new ToUppercaseInputStream(new BufferedInputStream(
          new FileInputStream(rootPath + "/input.txt"))));

      while ((c = inputStream.read()) >= 0) {
        System.out.print((char) c);
      }
      System.out.println("");
      System.out.println("This file contains "+inputStream.getLineCount()+" lines");
      inputStream.close();
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
}
```



Decorator example





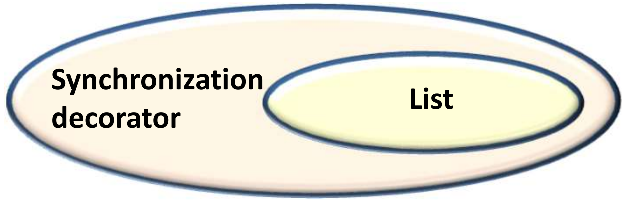
Some applications would benefit
from using objects to model every
aspect of their functionality, but
a naive design approach would be
prohibitively expensive.

For example, most document ed–
itors modularize their text format–
ting and editing facilities to some
extent. However, they invariably
stop short of using objects to
represent each character and
graphical element in the document.
Doing so would promote flexibility
at the finest level in the
application. Text and graphics
could be treated uniformly with

Decorator in Java collections

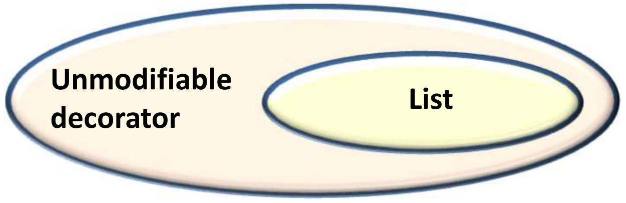
```
public static <T> Collection<T> synchronizedCollection(Collection<T> c);
public static <T> Set<T> synchronizedSet(Set<T> s);
public static <T> List<T> synchronizedList(List<T> list);
```

Factory methods that return a decorated collection



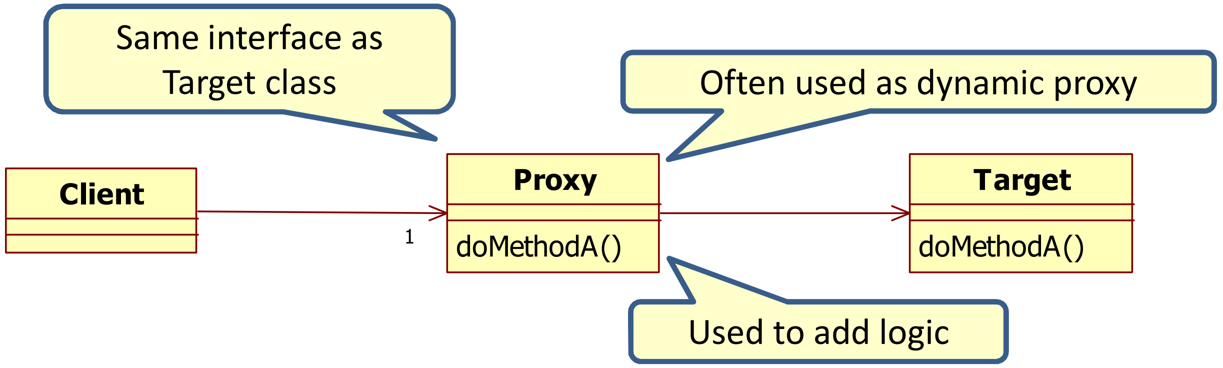
```
public static <T> Collection<T> unmodifiableCollection(Collection<? extends T> c);
public static <T> Set<T> unmodifiableSet(Set<? extends T> s);
public static <T> List<T> unmodifiableList(List<? extends T> list);
```

Factory methods that return an unmodifiable (immutable) collection

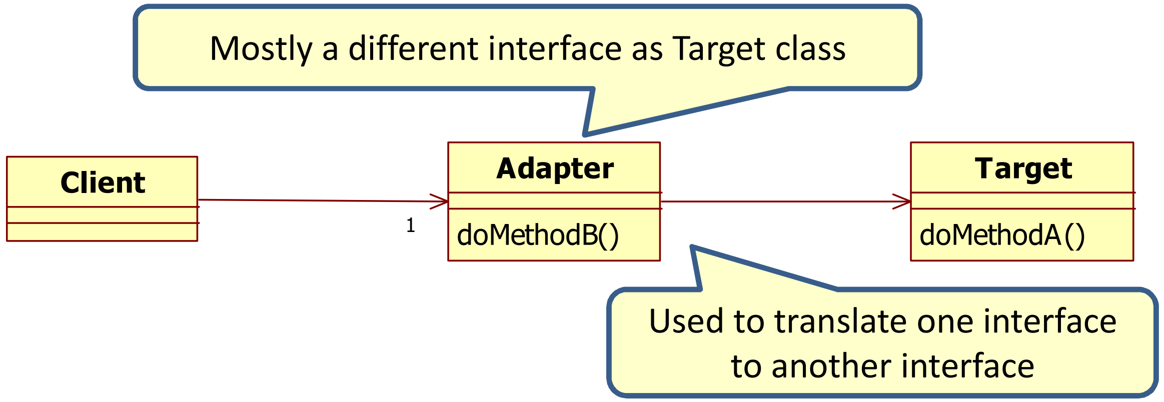


Wrappers

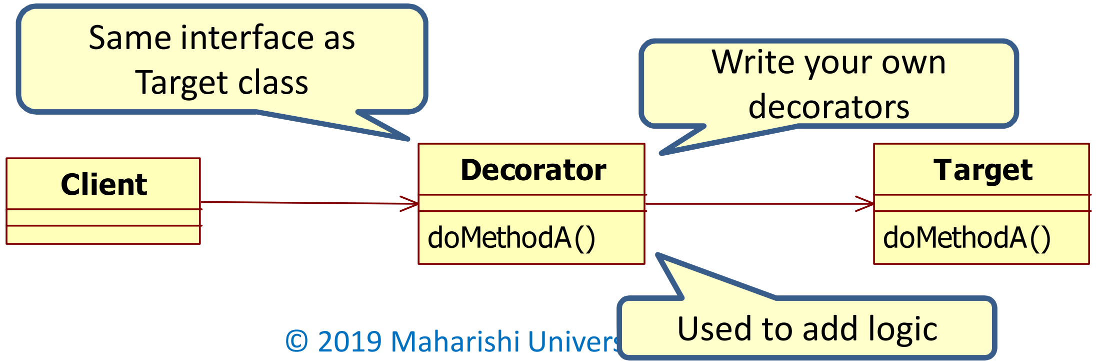
- Proxy



- Adapter


Mostly a different interface as Target class
Client
1
Adapter
doMethodB()
Target
doMethodA()
Used to translate one interface to another interface

- Decorator


Same interface as Target class
Write your own decorators
Client
Decorator
doMethodA()
Target
doMethodA()
Used to add logic