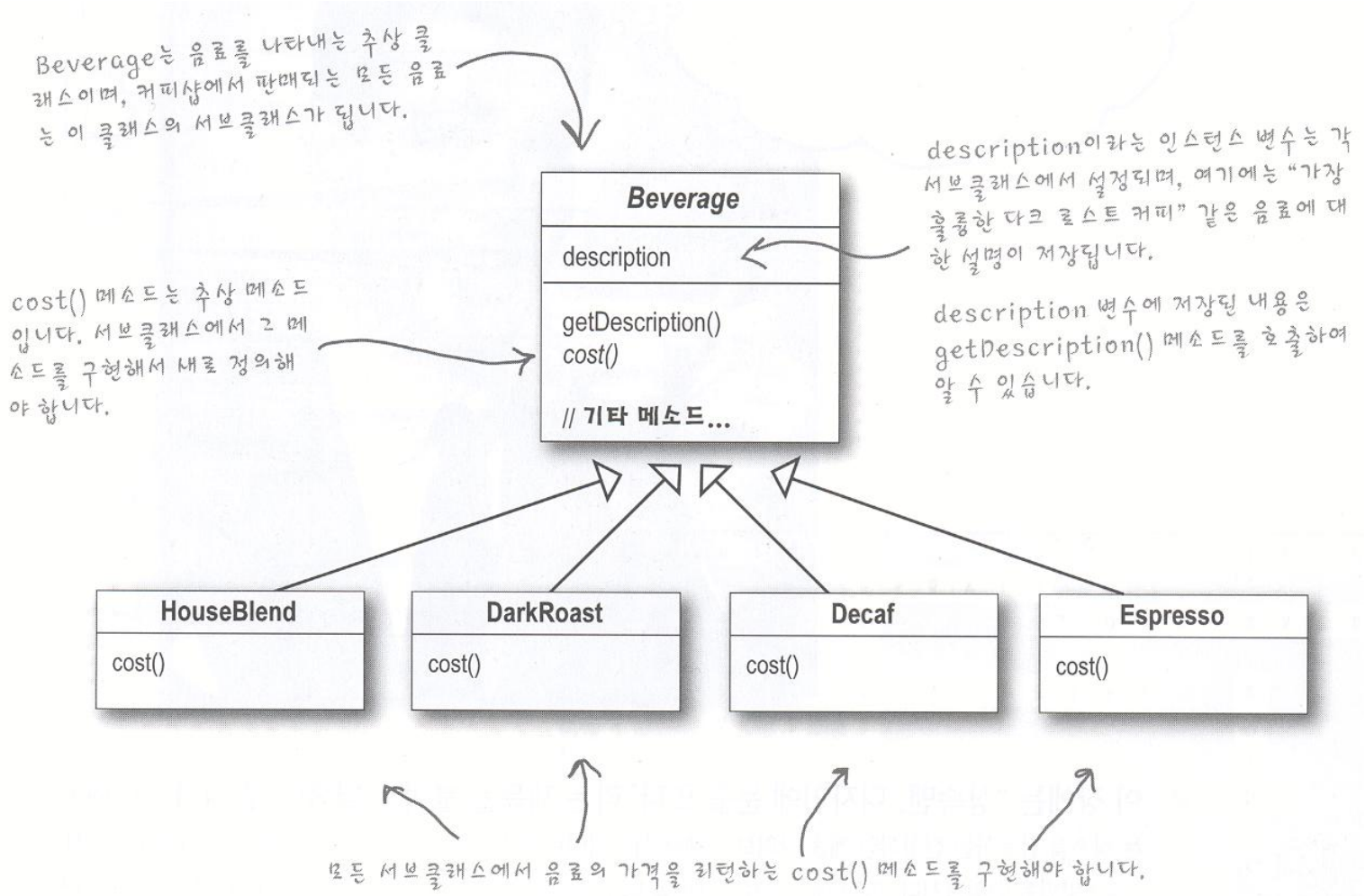
3. DECORATOR PATTERN

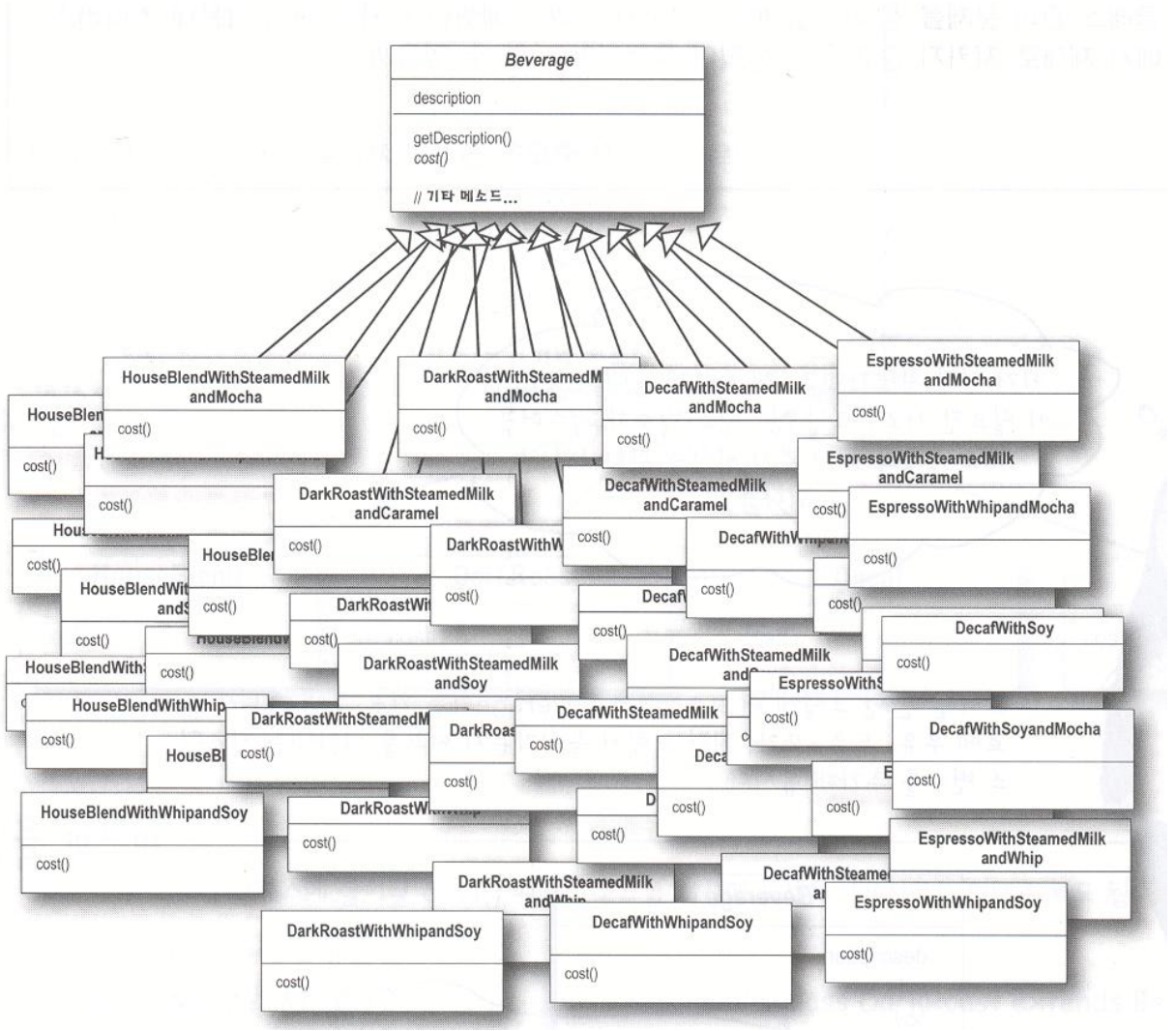
동의대학교 컴퓨터소프트웨어공학과

이종민 교수

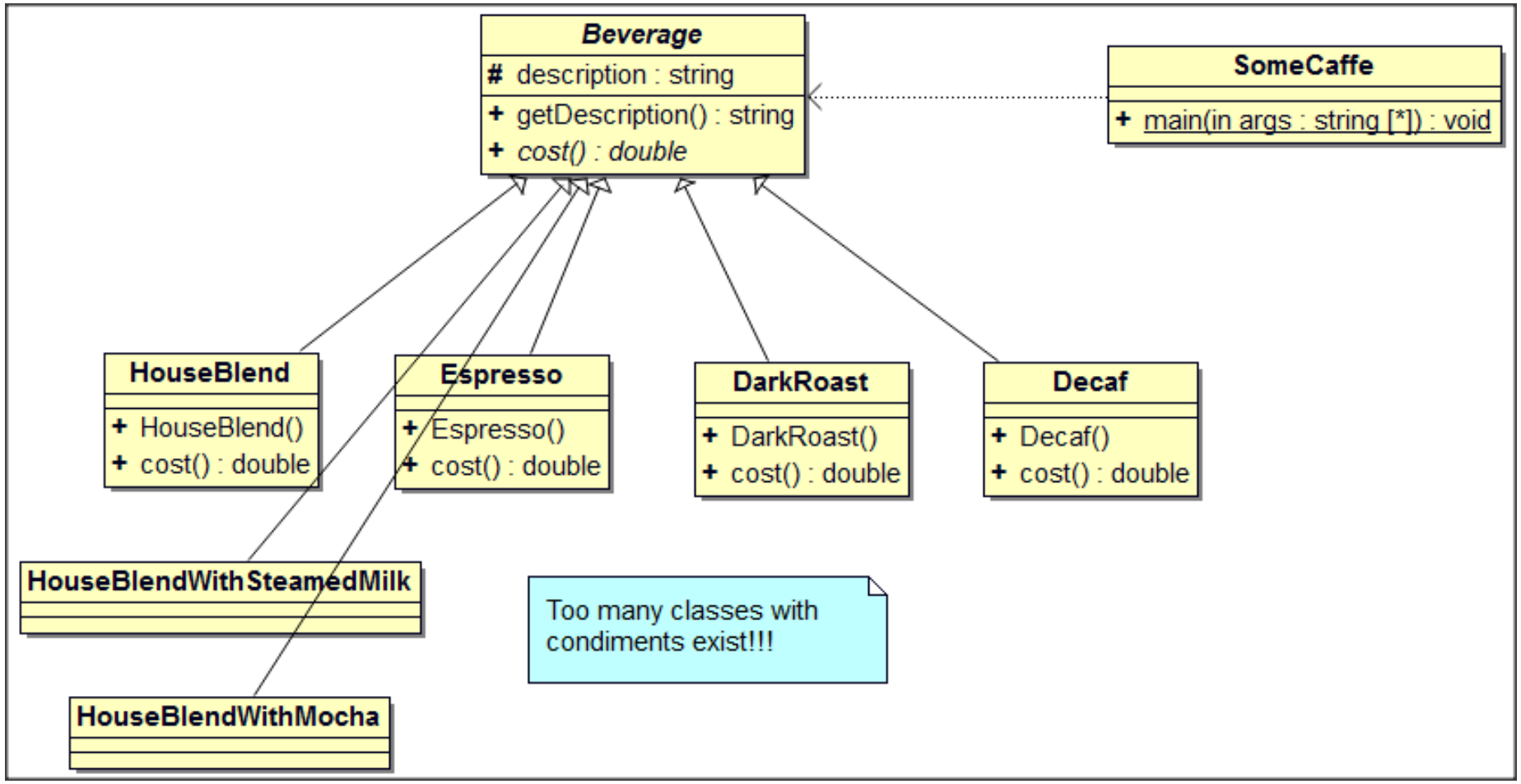
스타버즈 기본 음료 클래스 (basic)



추가 항목을 고려한 클래스



실습용 클래스 다이어그램(basic)



Beverage.java

```
6   package cse.design_pattern.ch03.basic;
7
8   /**...4 lines */
    public abstract class Beverage {
13      protected String description;
14
15      public String getDescription() {
16          return description;
17      }
18
        public abstract double cost();
20  }
```

HouseBlend.java

```
6   package cse.design_pattern.ch03.basic;
7
8   /**...4 lines */
12  public class HouseBlend extends Beverage {
13
14      public HouseBlend() {
15          description = "House Blend Coffee";
16      }
17
18      @Override
19      public double cost() {
20          return .89;
21      }
22
23  }
```

Espresso.java

```
6   package cse.design_pattern.ch03.basic;
7
8   /**...4 lines */
12  public class Espresso extends Beverage {
13
14      public Espresso() {
15          description = "Espresso";
16      }
17
18      @Override
        public double cost() {
20          return 1.99;
21      }
22  }
```

HouseBlendWithSteamedMilk.java

```
6   package cse.design_pattern.ch03.basic;
7
8   /**...4 lines */
12  public class HouseBlendWithSteamedMilk extends Beverage {
13
14      public HouseBlendWithSteamedMilk() {
15          description = "House Blend Coffee with Steamed Milk";
16      }
17
18      /**
19       * 우유 추가한 하우스 블렌드 커피 가격
20       * 새로운 첨가물이 있을 때마다 새로운 클래스를 만들어야 합니다!
21       *
22       * @return 우유 값 0.1 추가
23       */
24      @Override
        public double cost() {
26          return .89 + .1;
27      }
28
29  }
```

HouseBlendWithMocha.java

```
6   package cse.design_pattern.ch03.basic;
7
8   /**...4 lines */
12  public class HouseBlendWithMocha extends Beverage {
13
14      public HouseBlendWithMocha() {
15          description = "House Blend Coffee with Mocha";
16      }
17
18      /**
19       * 모카 추가한 하우스 블렌드 커피 가격
20       * 새로운 첨가물이 있을 때마다 새로운 클래스를 만들어야 합니다!
21       *
22       * @return 모카 값 0.2 추가
23       */
24      @Override
        public double cost() {
26          return .89 + .2;
27      }
28
29  }
```

SomeCaffe.java

```
6   package cse.design_pattern.ch03.basic;
7
8   /**...4 lines */
12  public class SomeCaffe {
13
14      /**...3 lines */
17      public static void main(String[] args) {
18          HouseBlend hb = new HouseBlend();
19          Espresso esp = new Espresso();
20
21          System.out.println("HouseBlend 커피는 얼마인가요?");
22          System.out.println(String.format("HouseBlend는 %.2f입니다.", hb.cost()));
23
24          System.out.println("Espresso 커피는 얼마인가요?");
25          System.out.printf("Espresso %.2f입니다.%n", esp.cost());
```

```
27          System.out.println("HouseBlend 커피에 우유를 추가하면 얼마인가요?");
28          System.out.println("HouseBlend 클래스만 있어 "
29                  + "새로 HouseBlendWithSteamedMilk 클래스를 만들어야 합니다.");
30          System.out.printf("우유 추가한 HouseBlend는 %.2f입니다.%n",
31                  new HouseBlendWithSteamedMilk().cost());
32
33          System.out.println("HouseBlend 커피에 모카를 추가하면 얼마인가요?");
34          System.out.println("HouseBlend 클래스만 있어 "
35                  + "새로 HouseBlendWithMocha 클래스를 만들어야 합니다.");
36          System.out.printf("모카 추가한 HouseBlend는 %.2f입니다.%n",
37                  new HouseBlendWithMocha().cost());
38      }
39
40  }
```

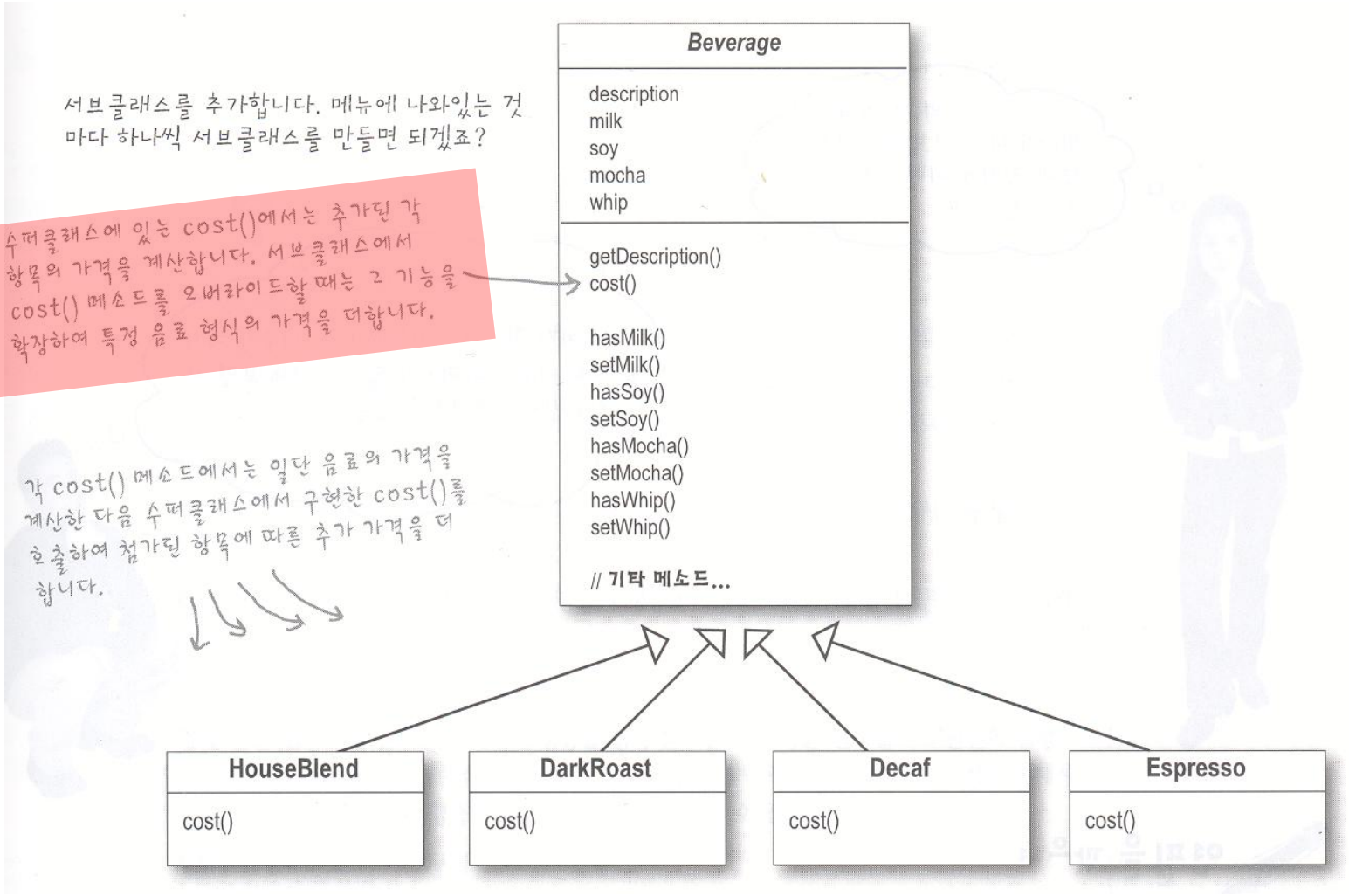
실행 결과

```
--- exec-maven-plugin:1.2.1:exec (default-cli) @ desing_pattern ---
HouseBlend 커피는 얼마인가요?
HouseBlend는 0.89입니다.
Espresso 커피는 얼마인가요?
Espresso 1.99입니다.
HouseBlend 커피에 우유를 추가하면 얼마인가요?
HouseBlend 클래스만 있어 새로 HouseBlendWithSteamedMilk 클래스를 만들어야 합니다.
우유 추가한 HouseBlend는 0.99입니다.
HouseBlend 커피에 모카를 추가하면 얼마인가요?
HouseBlend 클래스만 있어 새로 HouseBlendWithMocha 클래스를 만들어야 합니다.
모카 추가한 HouseBlend는 1.09입니다.
```

문제점

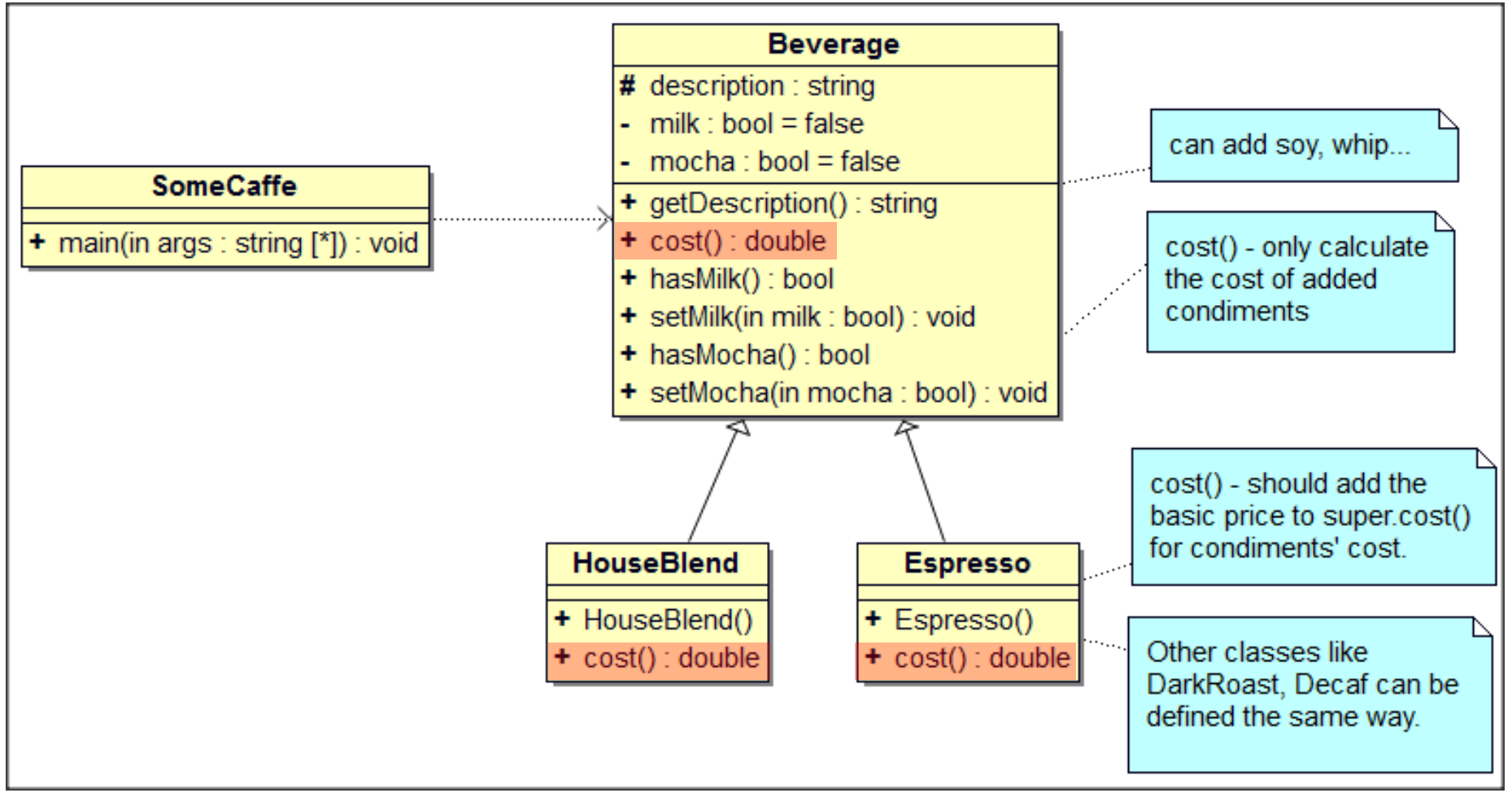
- 첨가물 추가에 따라서 cost() 메서드가 변경되어야 하므로 새로운 클래스 필요
- cost() 메서드 관점에서 중복된 코드 많이 존재

개선 1 (enhancement1)



실습용 클래스 다이어그램

- enhancement01



Beverage.java

```
6   package cse.design_pattern.ch03.enhancement1;
7
8   /** 첫 번째 개선 ...5 lines */
    public abstract class Beverage {
14      protected String description;
15      private boolean milk = false;
16      private boolean mocha = false;
17      // soy, whip 등 필요 항목 더 추가 가능
18
19      public String getDescription() {
20          return description;
21      }
22
23      /**
24       * 추가된 항목의 가격만 계산함.
25       * 서브 클래스에서 cost() 메소드를 오버라이드할 때는
26       * 그 기능을 확장하여 특정 음료 형식의 가격을 더함.
27       *
28       * @return 추가된 항목의 가격
29       */
        public double cost() {
31          double cost = 0.0;
32
33          if (hasMilk()) {
34              cost += 0.1;
35          }
36          if (hasMocha()) {
37              cost += 0.2;
38          }
39          return cost;
40      }
```

```
42      public boolean hasMilk() {
43          return milk;
44      }
45
46      public void setMilk(boolean milk) {
47          this.milk = milk;
48      }
49
50      public boolean hasMocha() {
51          return mocha;
52      }
53
54      public void setMocha(boolean mocha) {
55          this.mocha = mocha;
56      }
57
58  }
```

HouseBlend.java

```
6   package cse.design_pattern.ch03.enhancement1;
7
8
9   /**...4 lines */
    public class HouseBlend extends Beverage {
14
15      public HouseBlend() {
16          description = "House Blend Coffee";
17      }
18
19      /**
20       *
21       * @return super.cost() 추가
22       */
23      @Override
        public double cost() {
25          return .89 + super.cost();
26      }
27
28  }
```

Espresso.java

```
6   package cse.design_pattern.ch03.enhancement1;
7
8   /**...4 lines */
12  public class Espresso extends Beverage {
13
14      public Espresso() {
15          description = "Espresso";
16      }
17
18      /**
19       *
20       * @return super.cost() 추가
21       */
22      @Override
        public double cost() {
24          return 1.99 + super.cost();
25      }
26  }
```

SomeCaffe.java

```
6   package cse.design_pattern.ch03.enhancement1;
7
8   /**...4 lines */
12  public class SomeCaffe {
13
14      /**
15       * @param args the command line arguments
16       */
17      public static void main(String[] args) {
18          HouseBlend hb = new HouseBlend();
19          Espresso esp = new Espresso();
20
21          System.out.println("HouseBlend 커피는 얼마인가요?");
22          System.out.println(String.format("HouseBlend는 %.2f입니다.", hb.cost()));
23
24          System.out.println("Espresso 커피는 얼마인가요?");
25          System.out.println(String.format("Espresso %.2f입니다.", esp.cost()));
26
27          System.out.println("HouseBlend 커피에 우유를 추가해 주세요.");
28          hb.setMilk(true);
29          System.out.println(String.format("우유 추가한 HouseBlend는 %.2f입니다.",
30                  hb.cost()));
31
32          System.out.println("HouseBlend 커피에 우유를 한 번 두 추가하면 얼마인가요?");
33          System.out.println("현재 클래스 구조로는 더 이상 추가할 수 없습니다.");
34      }
35
36  }
```

실행 결과

```
--- exec-maven-plugin:1.2.1:exec (default-cli) @ desing_pattern ---
HouseBlend 커피는 얼마인가요?
HouseBlend는 0.89입니다.
Espresso 커피는 얼마인가요?
Espresso 1.99입니다.
HouseBlend 커피에 우유를 추가해 주세요.
우유 추가한 HouseBlend는 0.99입니다.
HouseBlend 커피에 우유를 한 번 두 추가하면 얼마인가요?
현재 클래스 구조로는 더 이상 추가할 수 없습니다.
------------------------------------------------------------------------
```

문제점

- 첨가물 가격 변동 시 Beverage 클래스 수정해야 함.
- 첨가물 종류가 많아지면 Beverage 클래스에
  - 새로운 메소드 추가: set~(), has~()
  - cost() 메소드 수정 필요
- Tea와 같은 새로운 음료 출시될 때 Beverage를 상속하면
  - hasWhip() 과 같은 메소드 상속받아 휘핑 크림 추가 가능
    → 별로 좋은 생각 아닌 듯!
- 동일 첨가물을 두 번 추가하려면?
  - 기존의 set~() 메소드로는 반영 불가

p.118 (1판 p.122)

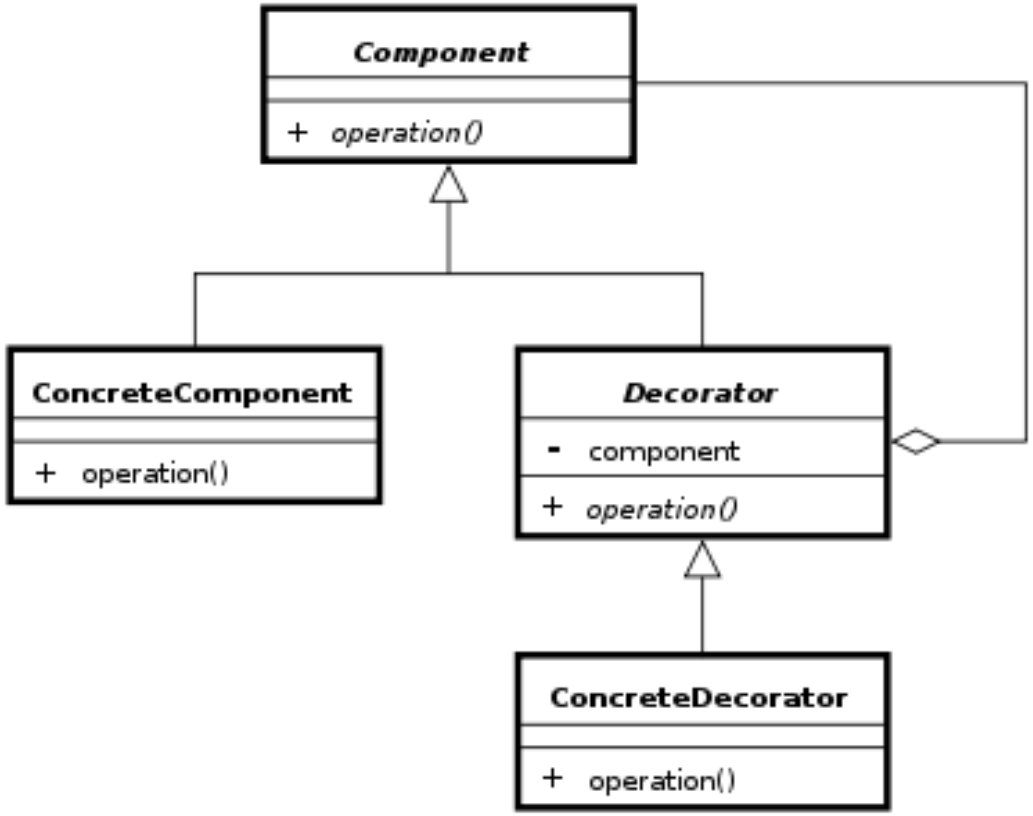
Open-Closed Principle

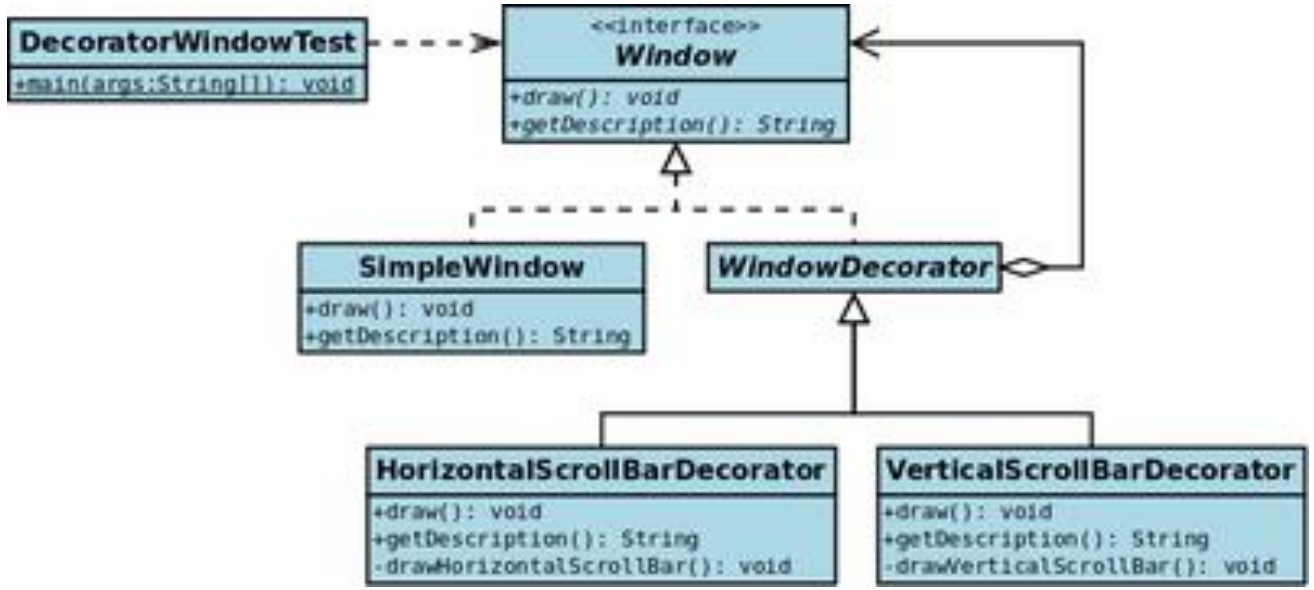
- Software entities (classes, modules, functions, etc.) should be **open for extension**, but **closed for modification**.
- 확장에는 열려 있고 수정에는 닫혀 있는 코드 구조 필요
- **기존 코드는 건드리지 않은 채로 확장을 통해서 새로운 행동을 간단하게 추가할 수 있다면!!!**

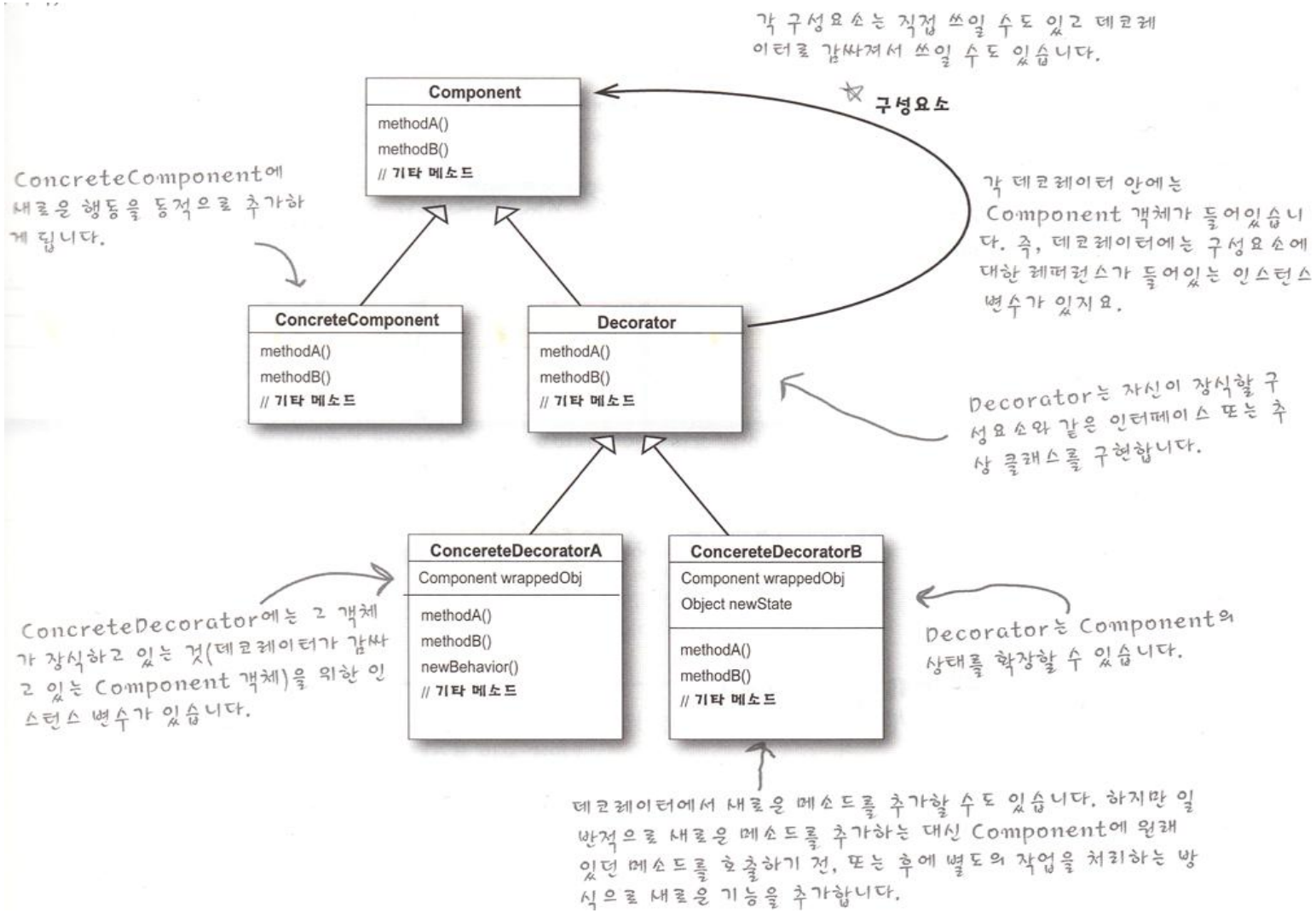
- 참고: http://butunclebob.com/ArticleS.UncleBob.PrinciplesOfOod

Decorator Pattern

- A design pattern that allows new/additional behavior to be added to an existing class dynamically.

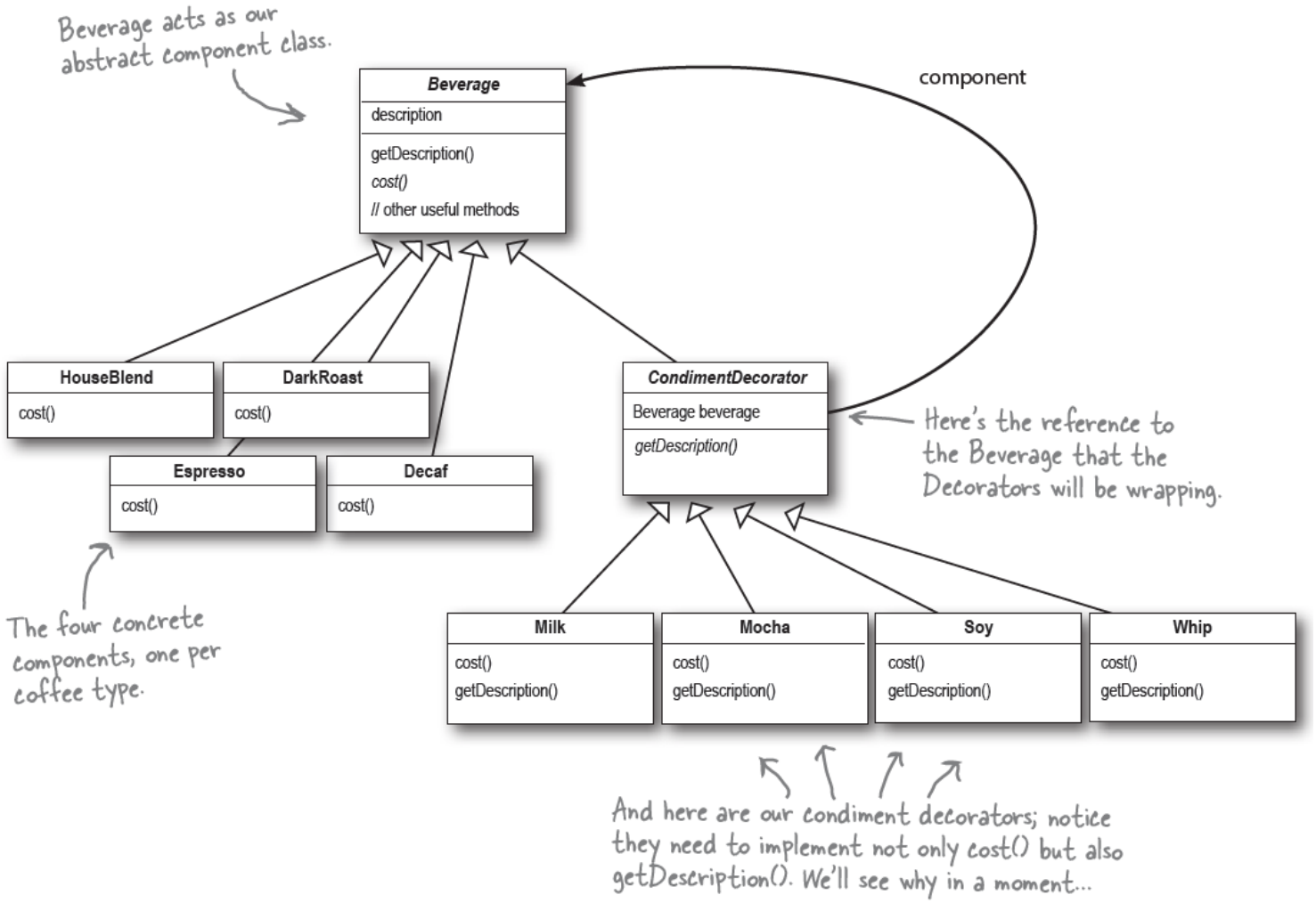





각 구성요소는 직접 쓰일 수도 있고 데코레이터로 감싸져서 쓰일 수도 있습니다.
구성요소
Component
methodA()
methodB()
// 기타 메소드
ConcreteComponent에 새로운 행동을 동적으로 추가하게 됩니다.
각 데코레이터 안에는 Component 객체가 들어있습니다. 즉, 데코레이터에는 구성요소에 대한 레퍼런스가 들어있는 인스턴스 변수가 있지요.
ConcreteComponent
methodA()
methodB()
// 기타 메소드
Decorator
methodA()
methodB()
// 기타 메소드
Decorator는 자신이 장식할 구성요소와 같은 인터페이스 또는 추상 클래스를 구현합니다.
ConcereteDecoratorA
Component wrappedObj
methodA()
methodB()
newBehavior()
// 기타 메소드
ConcereteDecoratorB
Component wrappedObj
Object newState
methodA()
methodB()
// 기타 메소드
ConcreteDecorator에는 그 객체가 장식하고 있는 것(데코레이터가 감싸고 있는 Component 객체)을 위한 인스턴스 변수가 있습니다.
Decorator는 Component의 상태를 확장할 수 있습니다.
데코레이터에서 새로운 메소드를 추가할 수도 있습니다. 하지만 일반적으로 새로운 메소드를 추가하는 대신 Component에 원래 있던 메소드를 호출하기 전, 또는 후에 별도의 작업을 처리하는 방식으로 새로운 기능을 추가합니다.

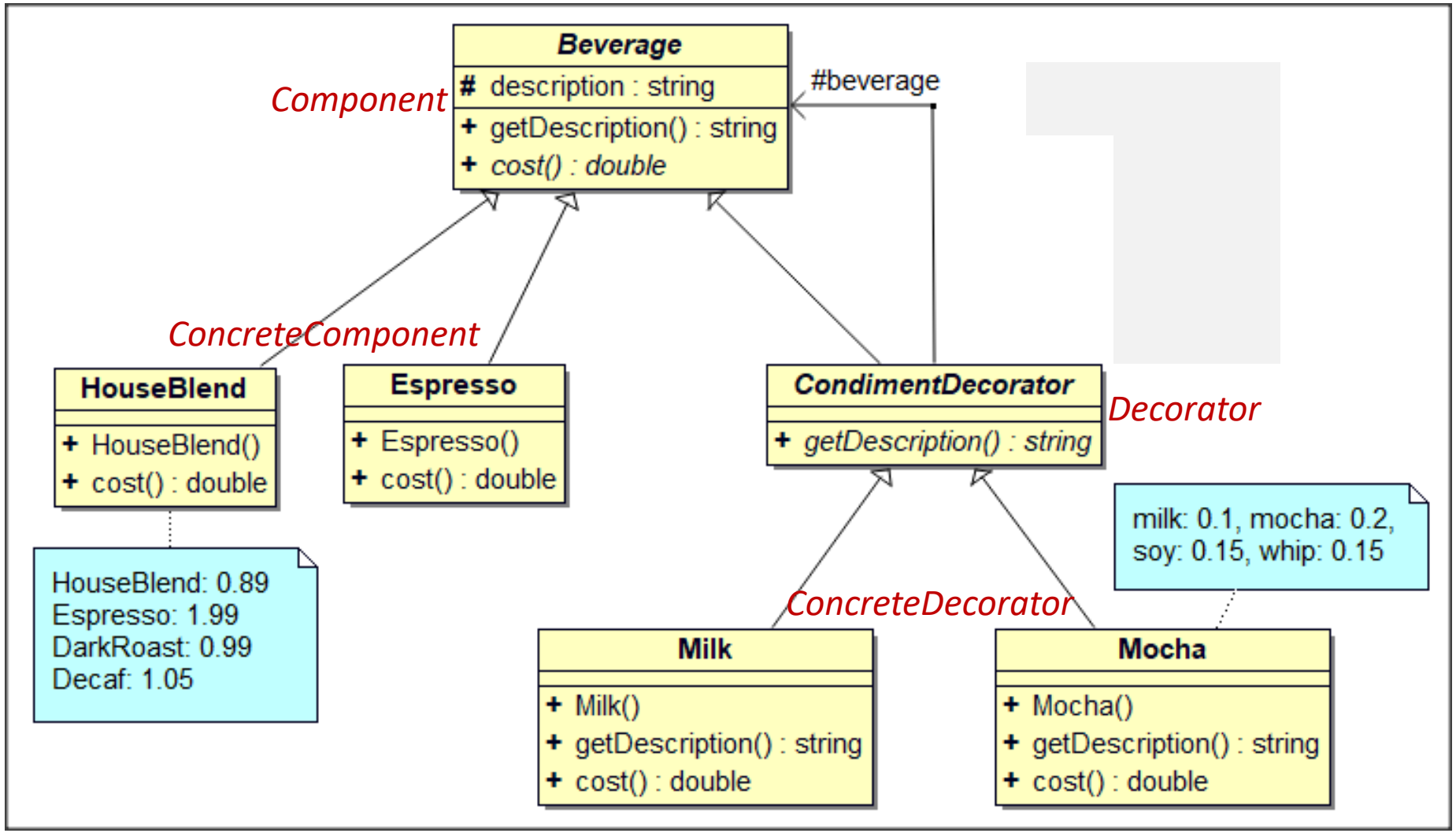
p.125

음료수 꾸미기 (Decorating our Beverages)



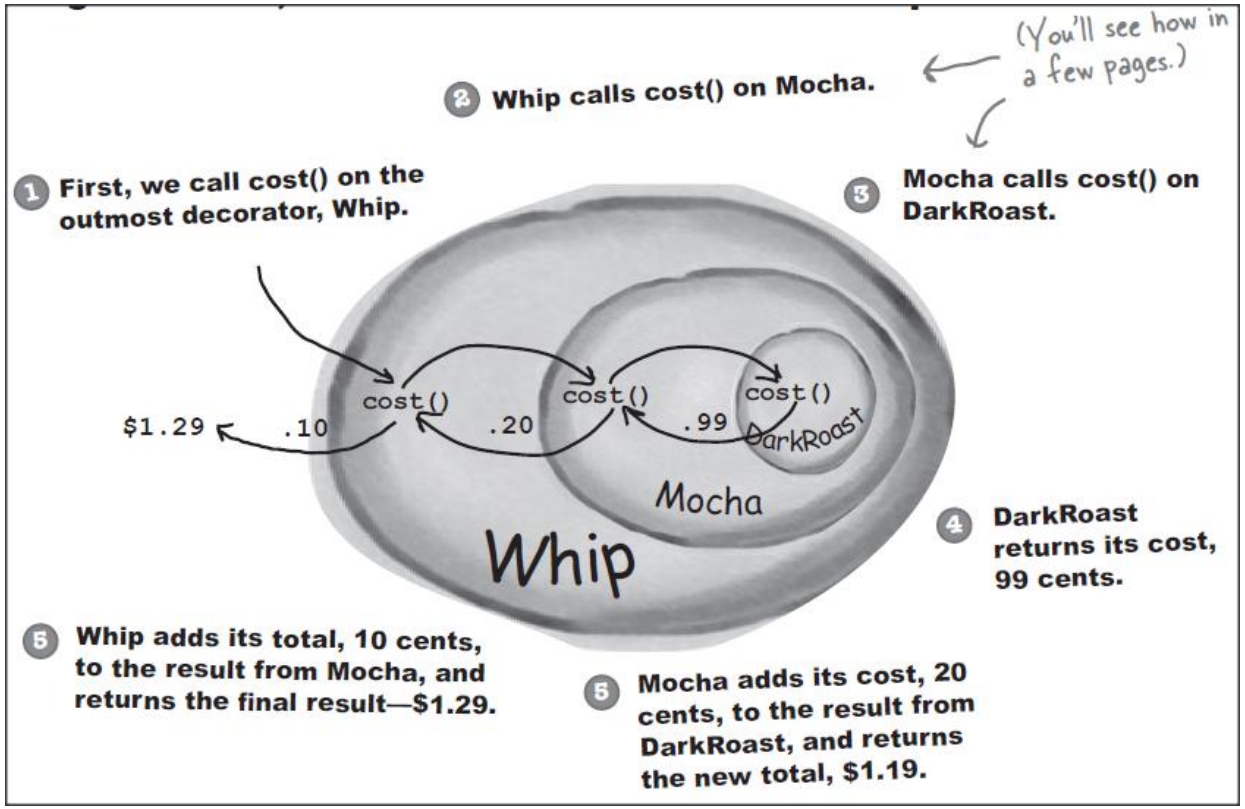
p.126

개선 2 (enhancement2)



데코레이터 사용 방법

- new Whip(new Mocha(new DarkRoast())).cost()



p.128

Beverage.java

- Component에 해당

```
1   package cse.design_pattern.ch03.enhancement2;
2
    public abstract class Beverage {
4
5       protected String description = "Unknown Beverage";
6
        public String getDescription() {
8           return description;
9       }
10
        public abstract double cost();
12  }
```

CondimentDecorator.java

- Decorator에 해당

```
2   package cse.design_pattern.ch03.enhancement02;
3
    public abstract class CondimentDecorator extends Beverage {
5     protected Beverage beverage;
6
7     @Override
      public abstract String getDescription();
9
10  }
```

HouseBlend.java

- ConcreteComponent에 해당

```
1   package cse.design_pattern.ch03.enhancement2;
2
3   public class HouseBlend extends Beverage {
4
5       public HouseBlend() {
6           description = "House Blend Coffee";
7       }
8
9       public double cost() {
10          return .89;
11      }
12  }
```

Espresso.java

- ConcreteComponent에 해당

```
1   package cse.design_pattern.ch03.enhancement2;
2
3   public class Espresso extends Beverage {
4
5       public Espresso() {
6           description = "Espresso";
7       }
8
9       public double cost() {
10          return 1.99;
11      }
12  }
```

Milk.java

- ConcreteDecorator에 해당

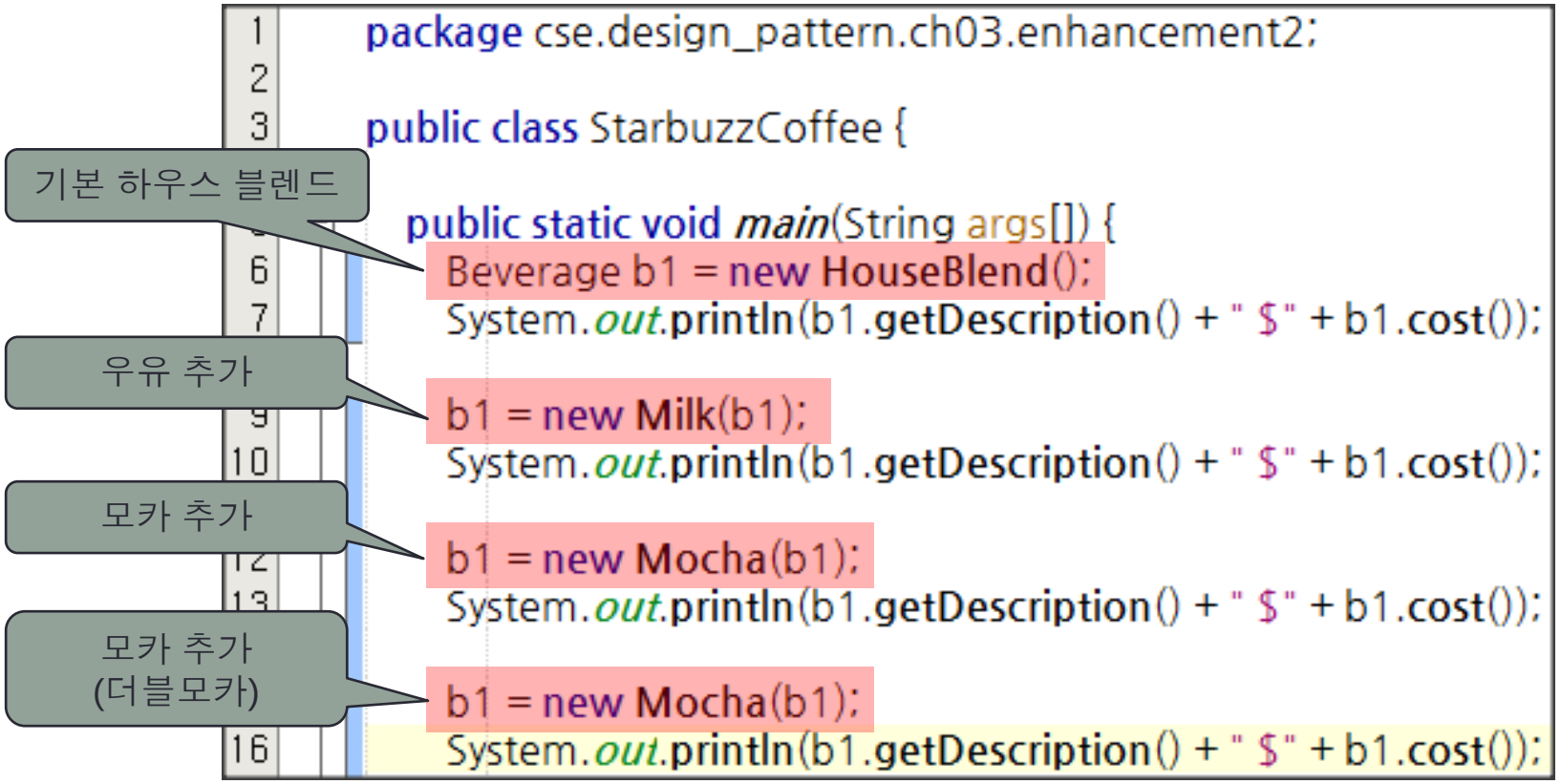
```
1   package cse.design_pattern.ch03.enhancement02;
2
3   public class Milk extends CondimentDecorator {
4
5       public Milk(Beverage beverage) {
6           this.beverage = beverage;
7       }
8
9       public String getDescription() {
10          return beverage.getDescription() + ", Milk";
11      }
12
13      public double cost() {
14          return .10 + beverage.cost();
15      }
16  }
```

Mocha.java

- ConcreteDecorator에 해당

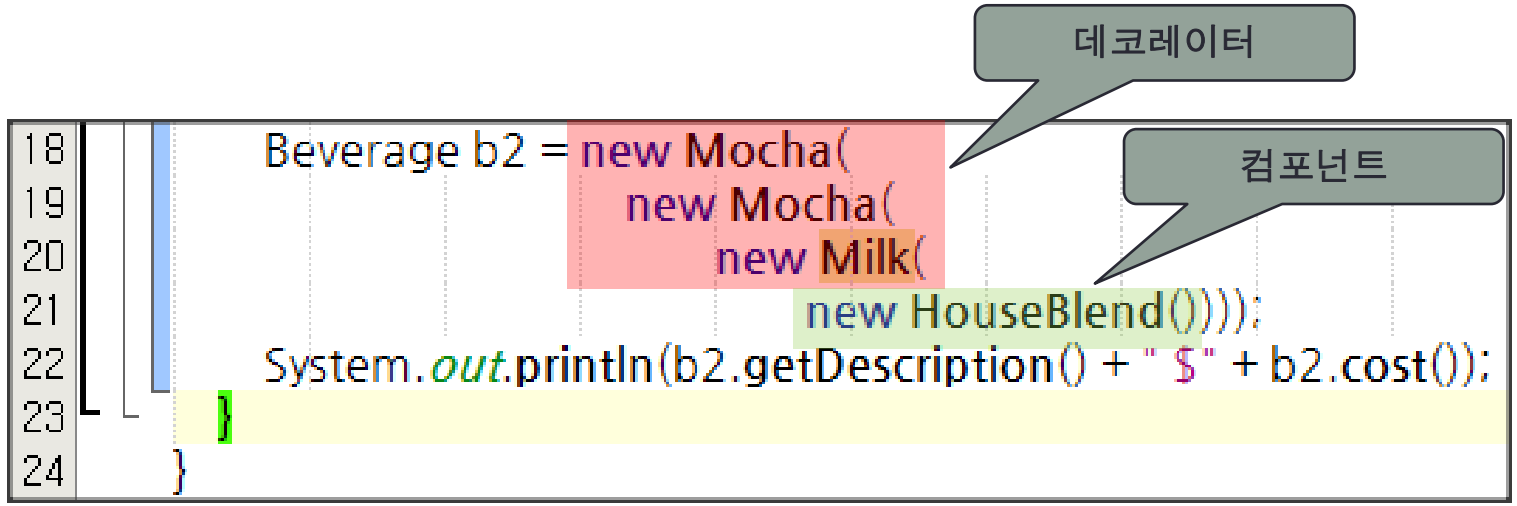
```
1   package cse.design_pattern.ch03.enhancement02
2
3   public class Mocha extends CondimentDecorator {
4
5       public Mocha(Beverage beverage) {
6           this.beverage = beverage;
7       }
8
9       public String getDescription() {
10          return beverage.getDescription() + ", Mocha";
11      }
12
13      public double cost() {
14          return .20 + beverage.cost();
15      }
16  }
```

StarbuzzCoffee.java




```
1   package cse.design_pattern.ch03.enhancement2;
2
3   public class StarbuzzCoffee {
4
5       public static void main(String args[]) {
6           Beverage b1 = new HouseBlend();
7           System.out.println(b1.getDescription() + " $" + b1.cost());
8
9           b1 = new Milk(b1);
10          System.out.println(b1.getDescription() + " $" + b1.cost());
11
12          b1 = new Mocha(b1);
13          System.out.println(b1.getDescription() + " $" + b1.cost());
14
15          b1 = new Mocha(b1);
16          System.out.println(b1.getDescription() + " $" + b1.cost());
```

```
House Blend Coffee $0.89
House Blend Coffee, Milk $0.99
House Blend Coffee, Milk, Mocha $1.19
House Blend Coffee, Milk, Mocha, Mocha $1.39
```

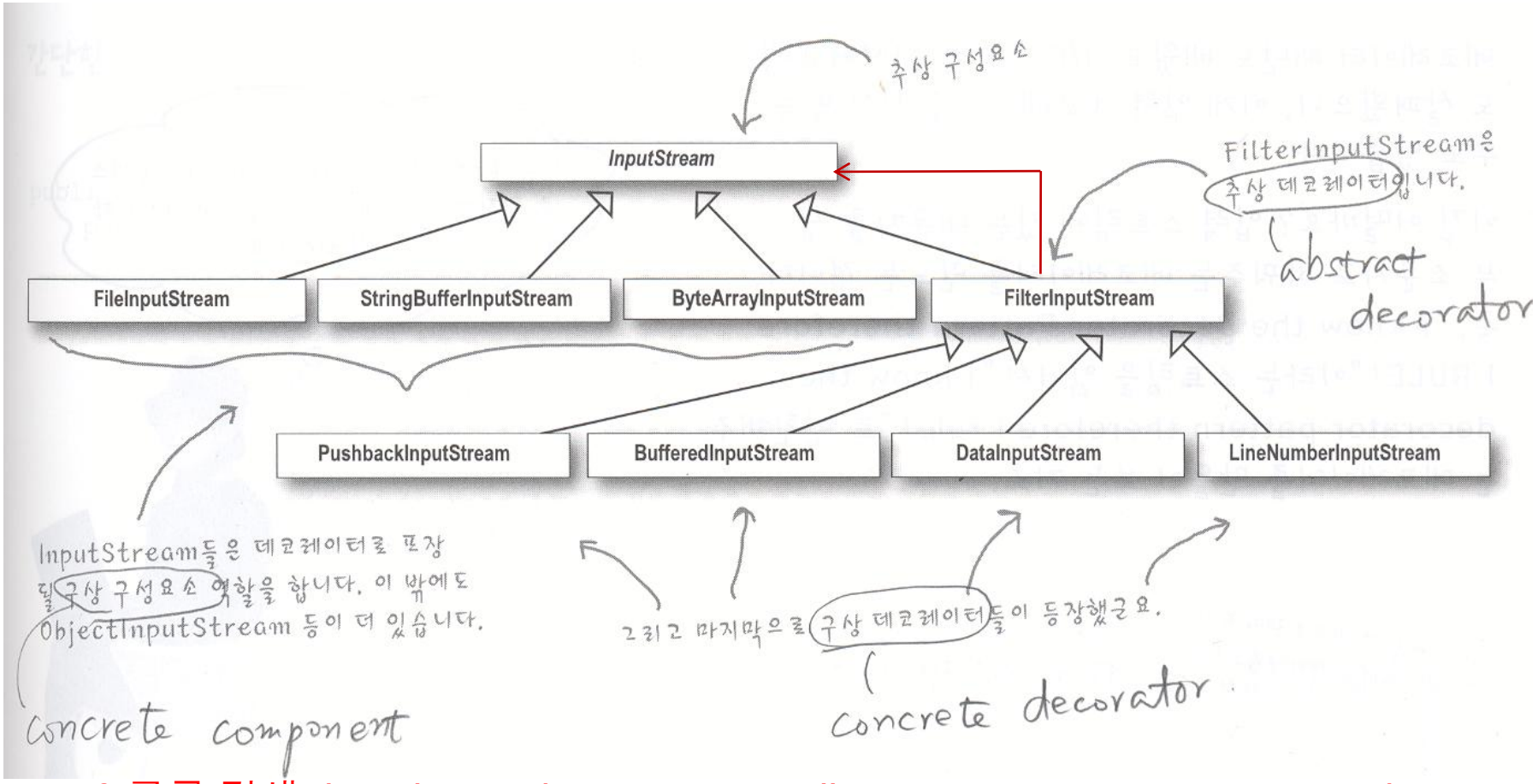



```
House Blend Coffee, Milk, Mocha, Mocha $1.39
```

java.io 클래스

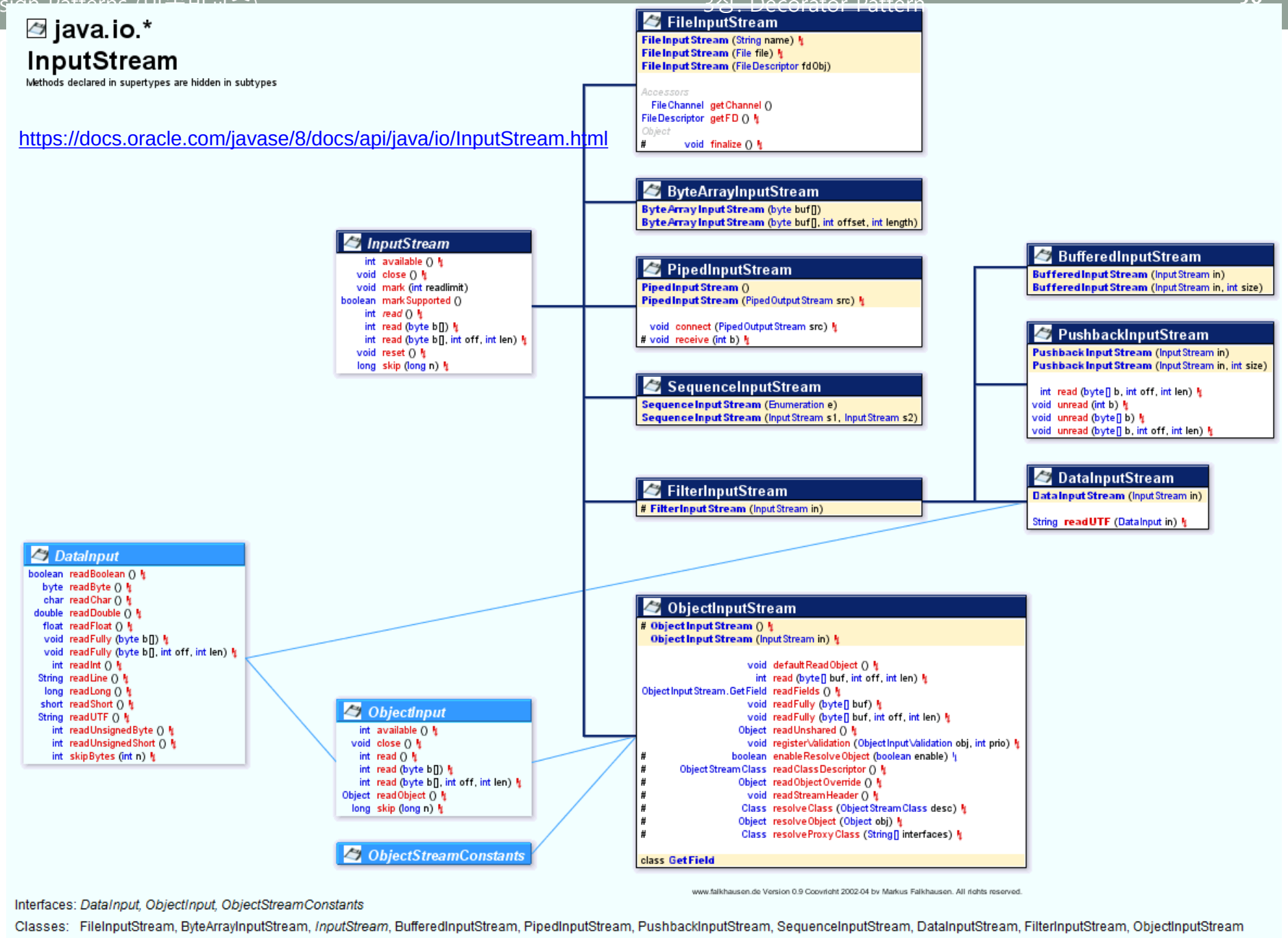
Constructor Summary	
protected	FilterInputStream(InputStream in) Creates a FilterInputStream by assigning the argument in to the field this.in so as to remember it for later use.

- Decorator pattern이 적용된 예



p.135

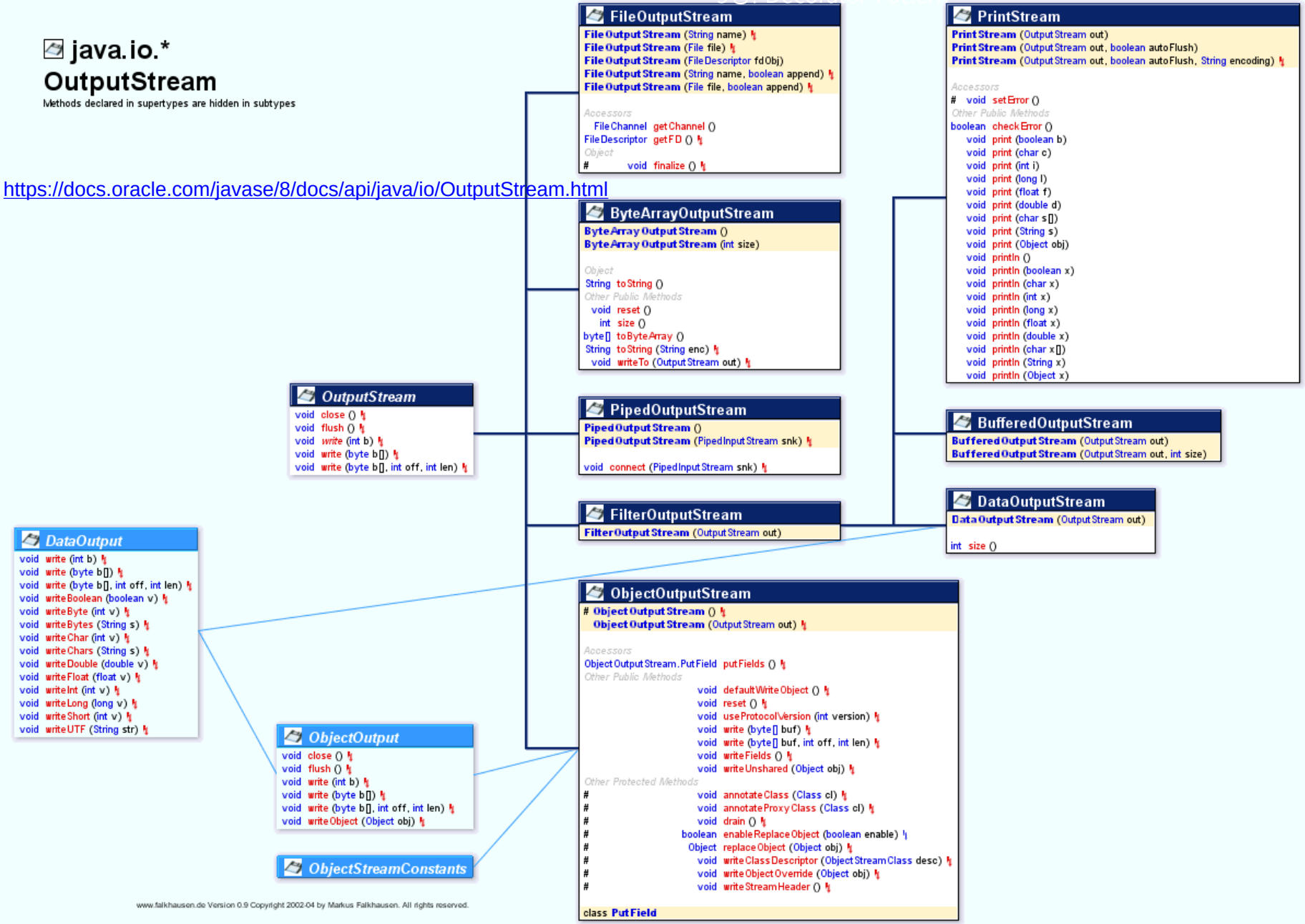
cf. 구글 검색: javadoc 21 (InputStream, FilterInputStream, OutputStream)


java.io.*
InputStream
Methods declared in supertypes are hidden in subtypes
https://docs.oracle.com/javase/8/docs/api/java/io/InputStream.html
FileInputStream
FileInputStream (String name)
FileInputStream (File file)
FileInputStream (FileDescriptor fdObj)
Accessors
FileChannel getChannel ()
FileDescriptor getFD ()
Object
void finalize ()
ByteArrayInputStream
ByteArrayInputStream (byte buf[])
ByteArrayInputStream (byte buf[], int offset, int length)
InputStream
int available ()
void close ()
void mark (int readlimit)
boolean markSupported ()
int read ()
int read (byte b[])
int read (byte b[], int off, int len)
void reset ()
long skip (long n)
PipedInputStream
PipedInputStream ()
PipedInputStream (PipedOutputStream src)
void connect (PipedOutputStream src)
void receive (int b)
BufferedInputStream
BufferedInputStream (InputStream in)
BufferedInputStream (InputStream in, int size)
PushbackInputStream
PushbackInputStream (InputStream in)
PushbackInputStream (InputStream in, int size)
int read (byte[] b, int off, int len)
void unread (int b)
void unread (byte[] b)
void unread (byte[] b, int off, int len)
SequenceInputStream
SequenceInputStream (Enumeration e)
SequenceInputStream (InputStream s1, InputStream s2)
FilterInputStream
FilterInputStream (InputStream in)
DataInputStream
DataInputStream (InputStream in)
String readUTF (DataInput in)
DataInput
boolean readBoolean ()
byte readByte ()
char readChar ()
double readDouble ()
float readFloat ()
void readFully (byte b[])
void readFully (byte b[], int off, int len)
int readInt ()
String readLine ()
long readLong ()
short readShort ()
String readUTF ()
int readUnsignedByte ()
int readUnsignedShort ()
int skipBytes (int n)
ObjectInput
int available ()
void close ()
int read ()
int read (byte b[])
int read (byte b[], int off, int len)
Object readObject ()
long skip (long n)
ObjectStreamConstants
ObjectInputStream
ObjectInputStream ()
ObjectInputStream (InputStream in)
void defaultReadObject ()
int read (byte[] buf, int off, int len)
ObjectInputStream.GetField readFields ()
void readFully (byte[] buf)
void readFully (byte[] buf, int off, int len)
Object readUnshared ()
void registerValidation (ObjectInputValidation obj, int prio)
boolean enableResolveObject (boolean enable)
ObjectStreamClass readClassDescriptor ()
Object readObjectOverride ()
void readStreamHeader ()
Class resolveClass (ObjectStreamClass desc)
Object resolveObject (Object obj)
Class resolveProxyClass (String[] interfaces)
class GetField
www.falkhausen.de Version 0.9 Copyright 2002-04 by Markus Falkhausen. All rights reserved.
Interfaces: DataInput, ObjectInput, ObjectStreamConstants
Classes: FileInputStream, ByteArrayInputStream, InputStream, BufferedInputStream, PipedInputStream, PushbackInputStream, SequenceInputStream, DataInputStream, FilterInputStream, ObjectInputStream

java.io.*
OutputStream

Methods declared in supertypes are hidden in subtypes

https://docs.oracle.com/javase/8/docs/api/java/io/OutputStream.html



Interfaces: *DataOutput, ObjectOutput, ObjectStreamConstants*

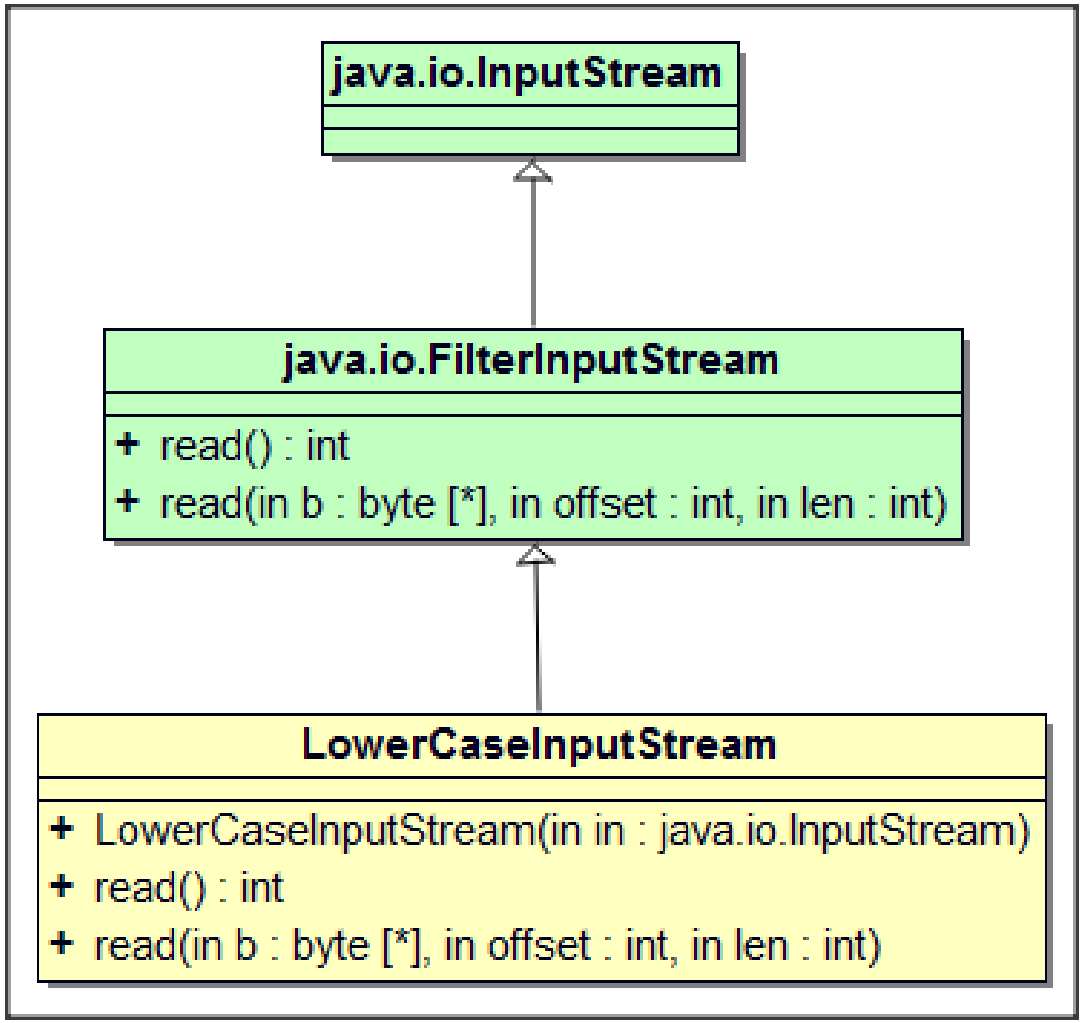
Classes: FileOutputStream, PrintStream, ByteArrayOutputStream, *OutputStream*, PipedOutputStream, BufferedOutputStream, DataOutputStream, FilterOutputStream, ObjectOutputStream

자바 I/O DECORATOR 실습

cse.design_pattern.ch03.io 패키지

AI: JDK21에서 decorator pattern이 적용된 클래스나 인터페이스를 알려줘.

실습 클래스 다이어그램 (io)

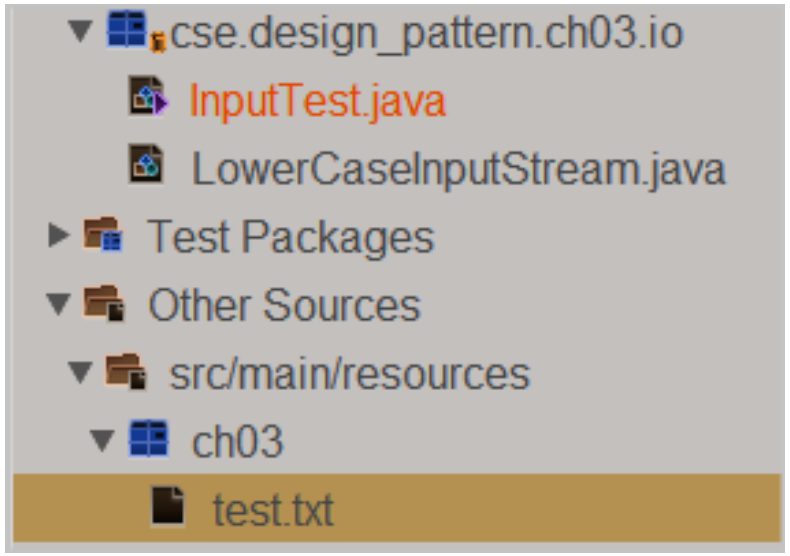


LowerCaseInputStream.java

```
1   package cse.design_pattern.ch03.io;
2
3   import java.io.*;
4
5   public class LowerCaseInputStream extends FilterInputStream {
6
7       public LowerCaseInputStream(InputStream in) {
8           super(in);
9       }
10
11      @Override
12      public int read() throws IOException {
13          int c = super.read();
14          return (c == -1 ? c : Character.toLowerCase((char) c));
15      }
16
17      @Override
18      public int read(byte[] b, int offset, int len) throws IOException {
19          int result = super.read(b, offset, len);
20          for (int i = offset; i < offset + result; i++) {
21              b[i] = (byte) Character.toLowerCase((char) b[i]);
22          }
23          return result;
24      }
25  }
```

test.txt

- Other Sources > **src/main/resources (src/main 폴더 아래에 resources 폴더 생성해야 함)**
  - ch03 폴더 생성 → Empty File... 생성



- test.txt 내용

```
1  I know the Decorator Pattern therefore I RULE!
```

InputTest.java

```
1  package cse.design_pattern.ch03.io;
2
3  import java.io.*;
4
5  public class InputTest {
6
7      public static void main(String[] args) throws IOException {
8
9          System.out.println("File.getPath(): " + new java.io.File(".").getPath());
10         System.out.println("File.getAbsolutePath(): " + new java.io.File(".").getAbsolutePath());
11         System.out.println("File.getCanonicalPath(): " + new java.io.File(".").getCanonicalPath());
12
13         System.out.printf("Working Directory = %s₩n", System.getProperty("user.dir"));
14         System.out.println("---------------------------------");
```

```
File.getPath(): .
File.getAbsolutePath(): D:₩NetBeansProjects₩design_pattern₩.
File.getCanonicalPath(): D:₩NetBeansProjects₩design_pattern
Working Directory = D:₩NetBeansProjects₩design_pattern
---------------------------------
```

```
16          try {
            InputStream in
18                  = new LowerCaseInputStream(
19                      new BufferedInputStream(
20                          new FileInputStream("./src/main/resources/ch03/test.txt")));
21
22          int c;
23          while ((c = in.read()) >= 0) {
24            System.out.print((char) c);
25          }
26
27          in.close();
28        } catch (IOException e) {
            e.printStackTrace();
30        }
31      }
32    }
```

i know the decorator pattern therefore i rule!

참고. InputTest2.java

```java
package cse.design_pattern.ch03.io;

import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.net.URL;
import java.util.logging.Level;
import java.util.logging.Logger;

public class InputTest2 {

    public void method01() {
        InputStream resource = getClass().getClassLoader().getResourceAsStream("ch03/test.txt");
        try (InputStream is = new LowerCaseInputStream(resource)) {
            while (is.available() > 0) {
                System.out.print((char) is.read());
            }
        } catch (IOException e) {
            System.out.println("오류 발생: " + e.getMessage());
        }
    }
```

```
24      /** getFile() 사용 ...3 lines */
27      public void method02() {
28          URL url = this.getClass().getClassLoader().getResource("ch03/test.txt");
29          if (url != null) {
30              try (InputStream is = new LowerCaseInputStream(
31                      new FileInputStream(url.getFile()))) {
32                  while (is.available() > 0) {
33                      System.out.print((char) is.read());
34                  }
35              } catch (IOException ex) {
36                  Logger.getLogger(InputTest2.class.getName()).log(Level.SEVERE, null, ex);
37              }
38          } else {
39              System.out.println("url이 null입니다.");
40          }
41      }
42
43      public static void main(String[] args) throws IOException {
44          new InputTest2().method01();
45          new InputTest2().method02();
46      }
47  }
```

```
--- exec-maven-plugin:1.5.0:exec (default-cli) @
i know the decorator pattern therefore i rule!
i know the decorator pattern therefore i rule!
```