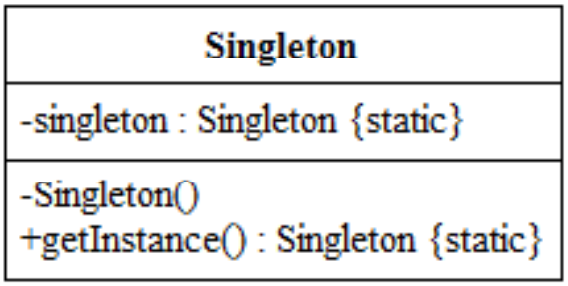
5장. 객체지향 구현

디자인 패턴(Design Pattern)

- Erich Gamman, Richard Helm, Ralph Johnson, John Vlissides 4명이 개발자의 '경험'이나 '내적인 축적'을 '디자인 패턴'으로 정리
  - 위 4명은 The Gang of Four (GoF)라 불림
  - 자주 사용하는 23개의 디자인 패턴 정리
- 참고도서
  - "Java 언어로 배우는 디자인 패턴 입문", Yuki Hiroshi 저, 영진닷컴, 2010

Singleton

- 인스턴스가 1개 밖에 존재하지 않음을 보증하는 디자인 패턴
  - 예: 컴퓨터 자체를 표현한 클래스, 윈도우 시스템을 표현한 클래스 등



- 외부에서 생성자 호출하지 못하도록 private 설정
- Static 필드의 초기화
  - getInstance()를 최초로 호출했을 때
  - Singleton 클래스 로드 시 1회만 실행됨

```
1  public class Singleton {
2      private static Singleton singleton = new Singleton();
3      private Singleton() {
4          System.out.println("인스턴스를 생성했습니다.");
5      }
6      public static Singleton getInstance() {
7          return singleton;
8      }
9  }
```

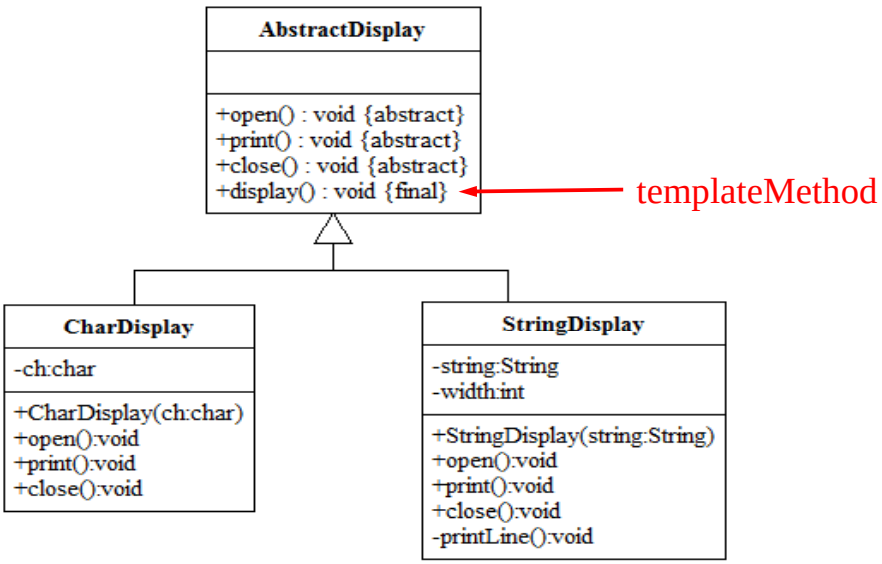
```java
public class Main {
    public static void main(String[] args) {
        System.out.println("Start.");
        Singleton obj1 = Singleton.getInstance();
        Singleton obj2 = Singleton.getInstance();
        if (obj1 == obj2) {
            System.out.println("obj1과 obj2는 같은 인스턴스입니다.");
        } else {
            System.out.println("obj1과 obj2는 같은 인스턴스가 아닙니다.");
        }
        System.out.println("End.");
    }
}
```

실행결과

```
Start.
인스턴스를 생성했습니다.
obj1과 obj2는 같은 인스턴스입니다.
End.
```

Template Method

- 상위클래스에서 뼈대를 결정하고, 하위클래스에서 구체적인 내용을 결정하는 디자인 패턴
- 예 : 문자나 문자열을 5회 반복해서 표시하기



```java
public abstract class AbstractDisplay {  // 추상 클래스 AbstractDisplay
    public abstract void open();         // 하위 클래스에 구현을 맡기는 추상 메소드 (1) open
    public abstract void print();        // 하위 클래스에 구현을 맡기는 추상 메소드 (2) print
    public abstract void close();        // 하위 클래스에 구현을 맡기는 추상 메소드 (3) close
    public final void display() {        // 추상 클래스에서 구현되고 있는 메소드 display
        open();                          // 우선 open하고…
        for (int i = 0; i < 5; i++) {    // 5번 print을 반복하고…
            print();
        }
        close();     // … 마지막으로 close한다. 이것이 display 메소드에서 구현되고 있는 내용.
    }
}
```

```java
public class CharDisplay extends AbstractDisplay {  // CharDisplay는 AbstractDisplay의
// 하위 클래스.
    private char ch;                                 // 표시해야 할 문자
    public CharDisplay(char ch) {                    // 생성자에서 전달된 문자 ch을
        this.ch = ch;                                // 필드에 기억해 둔다.
    }
    public void open() {                             // 상위 클래스에서는 추상 메소드였다.
                        // 여기에서 오버라이드해서 구현.
        System.out.print("<<");                      // 개시 문자열"<<"을 표시한다.
    }
    public void print() {                            // print 메소드도 여기에서 구현한다.
// 이것이 display에서 반복해서 호출된다.
        System.out.print(ch);                        // 필드에 기억해 둔 문자를 1개 표시한다.
    }
    public void close() {                            // close 메소드도 여기에서 구현.
        System.out.println(">>");                    // 종료 문자열 ">>"을 표시.
    }
}
```

```java
public class StringDisplay extends AbstractDisplay {    // StringDisplay도
// AbstrctDisplay의 하위 클래스.
    private String string;                           // 표시해야 할 문자열.
    private int width;                               // 바이트 단위로 계산한 문자열의 「폭」.
    public StringDisplay(String string) {            // 생성자에서 전달된 문자열 string을
        this.string = string;                        // 필드에 기억.
        this.width = string.getBytes().length;       // 그리고 바이트 단위의 폭도 필드에
                        // 기억해 두고 나중에 사용한다.
    }
    public void open() {                             // 오버라이드해서 정의한 open 메소드.
        printLine();                                 // 이 클래스의 메소드 printLine에서
                                                     // 선을 그리고 있다.
    }
    public void print() {                            // print 메소드는
        System.out.println("|" + string + "|");     // 필드에 기억해 둔 문자열의
                        // 전후에 "|"을 붙여서 표시.
    }
    public void close() {                            // close 메소드는
        printLine();                                 // open 처럼 printLine 메소드에서
                                                     // 선을 그리고 있다.
    }
    private void printLine() {                       // open과 close에서 호출된 printLine 메소드이다.
                    // private이기 때문에 이 클래스 안에서만 사용된다.
        System.out.print("+");                       // 테두리의 모서리를 표현하는"+" 마크를 표시.
        for (int i = 0; i < width; i++) {            // width개의 "-"을 표시하고
            System.out.print("-");                   // 테두리 선으로 이용한다.
        }
        System.out.println("+");                     // 테두리의 모서리를 표현하는 "+" 마크를 표시.
    }
}
```

```java
public class Main {
    public static void main(String[] args) {
    // 'H'을 가진 CharDisplay 인스턴스를 1개 만든다>
        AbstractDisplay d1 = new CharDisplay('H');
    // "Hello, world."을 가진 StringDisplay의 인스턴스를 1개 만든다.
        AbstractDisplay d2 = new StringDisplay("Hello, world.");
        // "안녕하세요."를 가진 StringDisplay의 인스턴스를 1개 만든다.
        AbstractDisplay d3 = new StringDisplay("안녕하세요.");
        d1.display();   // d1, d2, d3 모두 AbstractDisplay의 하위클래스의 인스턴스이기 때문에
        d2.display();   // 상속한 display메소드를 호출할 수 있다.
        d3.display();   // 실제 동작은 CharDisplay나 StringDisplay에서 결정한다.
    }
}
```

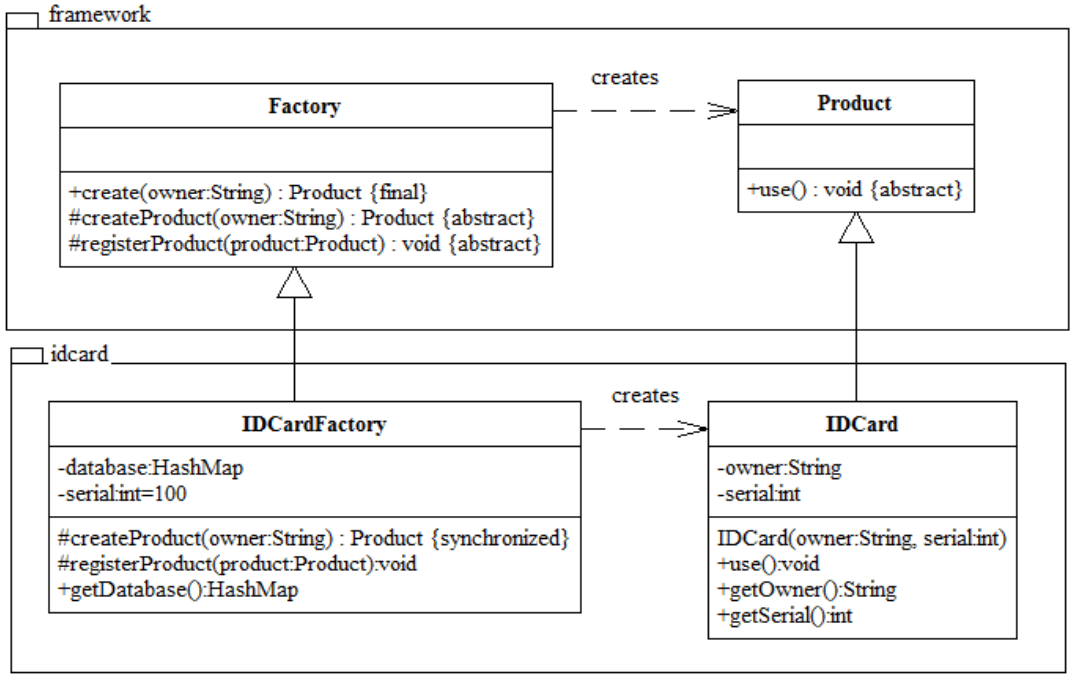
실행결과

```
<<HHHHH>>
+-------------+
|Hello, world.|
|Hello, world.|
|Hello, world.|
|Hello, world.|
|Hello, world.|
+-------------+
+-----------+
|안녕하세요.|
|안녕하세요.|
|안녕하세요.|
|안녕하세요.|
|안녕하세요.|
+-----------+
```

상위클래스의 **templateMethod**에 알고리즘이 기술되어 있으므로, 하위클래스에서 일일이 기술할 필요가 없음

Factory Method

- 인스턴스 생성을 위한 골격 클래스와 실제의 인스턴스를 생성하는 클래스를 분리하는 디자인 패턴



```java
package framework;

public abstract class Product {
    public abstract void use();
}
```

```java
package framework;

public abstract class Factory {
    public final Product create(String owner) {
        Product p = createProduct(owner);
        registerProduct(p);
        return p;
    }
    protected abstract Product createProduct(String owner);
    protected abstract void registerProduct(Product product);
}
```

```java
package idcard;
import framework.*;

public class IDCard extends Product {
    private String owner;
    private int serial;
    IDCard(String owner, int serial) {
        System.out.println(owner + "(" + serial + ")" + "의 카드를 만듭니다.");
        this.owner = owner;
        this.serial = serial;
    }
    public void use() {
        System.out.println(owner + "(" + serial + ")" + "의 카드를 사용합니다.");
    }
    public String getOwner() {
        return owner;
    }
    public int getSerial() {
        return serial;
    }
}
```

```java
package idcard;
import framework.*;
import java.util.*;

public class IDCardFactory extends Factory {
    private HashMap database = new HashMap();
    private int serial = 100;
    protected synchronized Product createProduct(String owner) {
        return new IDCard(owner, serial++);
    }
    protected void registerProduct(Product product) {
        IDCard card = (IDCard)product;
        database.put(new Integer(card.getSerial()), card.getOwner());
    }
    public HashMap getDatabase() {
        return database;
    }
}
```

```java
import framework.*;
import idcard.*;

public class Main {
    public static void main(String[] args) {
        Factory factory = new IDCardFactory();
        Product card1 = factory.create("홍길동");
        Product card2 = factory.create("이순신");
        Product card3 = factory.create("강감찬");
        card1.use();
        card2.use();
        card3.use();
    }
}
```

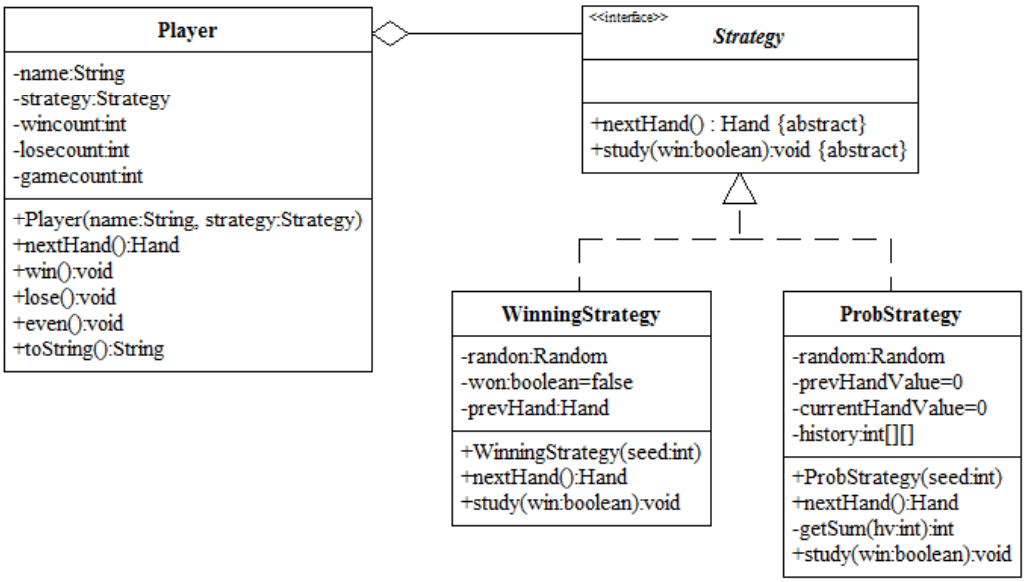
실행결과

```
홍길동(100)의 카드를 만듭니다.
이순신(101)의 카드를 만듭니다.
강감찬(102)의 카드를 만듭니다.
홍길동(100)의 카드를 사용합니다.
이순신(101)의 카드를 사용합니다.
강감찬(102)의 카드를 사용합니다.
```

Framework 패키지의 내용을
수정하지 않아도
전혀 다른 제품과 공장을
만들 수 있음

Strategy

- 알고리즘 구현 부분을 쉽게 교체할 수 있도록 하는 디자인 패턴
- 예 : 가위바위보 게임



- Hand 클래스
  - 가위바위보의 '손'을 표시하는 클래스

```
1  public class Hand {
2      public static final int HANDVALUE_GUU = 0;  // 주먹을 표시하는 값
3      public static final int HANDVALUE_CHO = 1;  // 가위를 표시하는 값
4      public static final int HANDVALUE_PAA = 2;  // 보를 표시하는 값
5      public static final Hand[] hand = {         // 가위바위보의 손을 표시하는 3개의 인스턴스
6          new Hand(HANDVALUE_GUU),
7          new Hand(HANDVALUE_CHO),
8          new Hand(HANDVALUE_PAA),
9      };
10     private static final String[] name = {        // 가위바위보의 손의 문자열 표현
11         "주먹", "가위", "보",
12     };
13     private int handvalue;                      // 가위바위보의 손의 값
14     private Hand(int handvalue) {
15         this.handvalue = handvalue;
16     }
17     public static Hand getHand(int handvalue) {  // 값에서 인스턴스를 얻는다
18         return hand[handvalue];
19     }
20     public boolean isStrongerThan(Hand h) {     // this가 h를 이긴 경우 true
21         return fight(h) == 1;
22     }
```

```
23      public boolean isWeakerThan(Hand h) {       // this가 h에게 질 경우 true
24          return fight(h) == -1;
25      }
26      private int fight(Hand h) {                 // 무승부는 0, this의 승이면 1, h의 승이면 -1
27          if (this == h) {
28              return 0;
29          } else if ((this.handvalue + 1) % 3 == h.handvalue) {
30              return 1;
31          } else {
32              return -1;
33          }
34      }
35      public String toString() {                  // 문자열 표현으로 변환
36          return name[handvalue];
37      }
38  }
```

```
1   public interface Strategy {
2       public abstract Hand nextHand();
3       public abstract void study(boolean win);
4   }
```

```
1   import java.util.Random;
2
3   public class WinningStrategy implements Strategy {
4       private Random random;
5       private boolean won = false;
6       private Hand prevHand;
7       public WinningStrategy(int seed) {
8           random = new Random(seed);
9       }
10      public Hand nextHand() {
11          if (!won) {
12              prevHand = Hand.getHand(random.nextInt(3));
13          }
14          return prevHand;
15      }
16      public void study(boolean win) {
17          won = win;
18      }
19  }
```

```java
import java.util.Random;

public class ProbStrategy implements Strategy {
    private Random random;
    private int prevHandValue = 0;
    private int currentHandValue = 0;
    private int[][] history = {
        { 1, 1, 1, },
        { 1, 1, 1, },
        { 1, 1, 1, },
    };
    public ProbStrategy(int seed) {
        random = new Random(seed);
    }
    public Hand nextHand() {
        int bet = random.nextInt(getSum(currentHandValue));
        int handvalue = 0;
        if (bet < history[currentHandValue][0]) {
            handvalue = 0;
        } else if (bet < history[currentHandValue][0] + history[currentHandValue][1]) {
            handvalue = 1;
        } else {
            handvalue = 2;
        }
        prevHandValue = currentHandValue;
        currentHandValue = handvalue;
        return Hand.getHand(handvalue);
    }
```

// **history[이전에 낸 손] [이번에 낼 손]**
// 과거의 승패를 반영한 확률계산 표

```java
    private int getSum(int hv) {
        int sum = 0;
        for (int i = 0; i < 3; i++) {
            sum += history[hv][i];
        }
        return sum;
    }
    public void study(boolean win) {
        if (win) {
            history[prevHandValue][currentHandValue]++;
        } else {
            history[prevHandValue][(currentHandValue + 1) % 3]++;
            history[prevHandValue][(currentHandValue + 2) % 3]++;
        }
    }
}
```

```java
public class Player {
    private String name;
    private Strategy strategy;
    private int wincount;
    private int losecount;
    private int gamecount;
    public Player(String name, Strategy strategy) {         // 이름과 전략을 할당받는다
        this.name = name;
        this.strategy = strategy;
    }
    public Hand nextHand() {                                // 전략의 지시를 받는다
        return strategy.nextHand();
    }
    public void win() {                 // 승
        strategy.study(true);
        wincount++;
        gamecount++;
    }
    public void lose() {                // 패
        strategy.study(false);
        losecount++;
        gamecount++;
    }
    public void even() {                // 무승부
        gamecount++;
    }
    public String toString() {
        return "[" + name + ":" + gamecount + " games, " + wincount + " win, " + losecount
        + " lose" + "]";
    }
}
```

```java
public class Main {
    public static void main(String[] args) {
        if (args.length != 2) {
            System.out.println("Usage: java Main randomseed1 randomseed2");
            System.out.println("Example: java Main 314 15");
            System.exit(0);
        }
        int seed1 = Integer.parseInt(args[0]);
        int seed2 = Integer.parseInt(args[1]);
        Player player1 = new Player("두리", new WinningStrategy(seed1));
        Player player2 = new Player("하나", new ProbStrategy(seed2));
        for (int i = 0; i < 10000; i++) {
            Hand nextHand1 = player1.nextHand();
            Hand nextHand2 = player2.nextHand();
            if (nextHand1.isStrongerThan(nextHand2)) {
                System.out.println("Winner:" + player1);
                player1.win();
                player2.lose();
            } else if (nextHand2.isStrongerThan(nextHand1)) {
                System.out.println("Winner:" + player2);
                player1.lose();
                player2.win();
            } else {
                System.out.println("Even...");
                player1.even();
                player2.even();
            }
        }
        System.out.println("Total result:");
        System.out.println(player1.toString());
        System.out.println(player2.toString());
    }
}
```

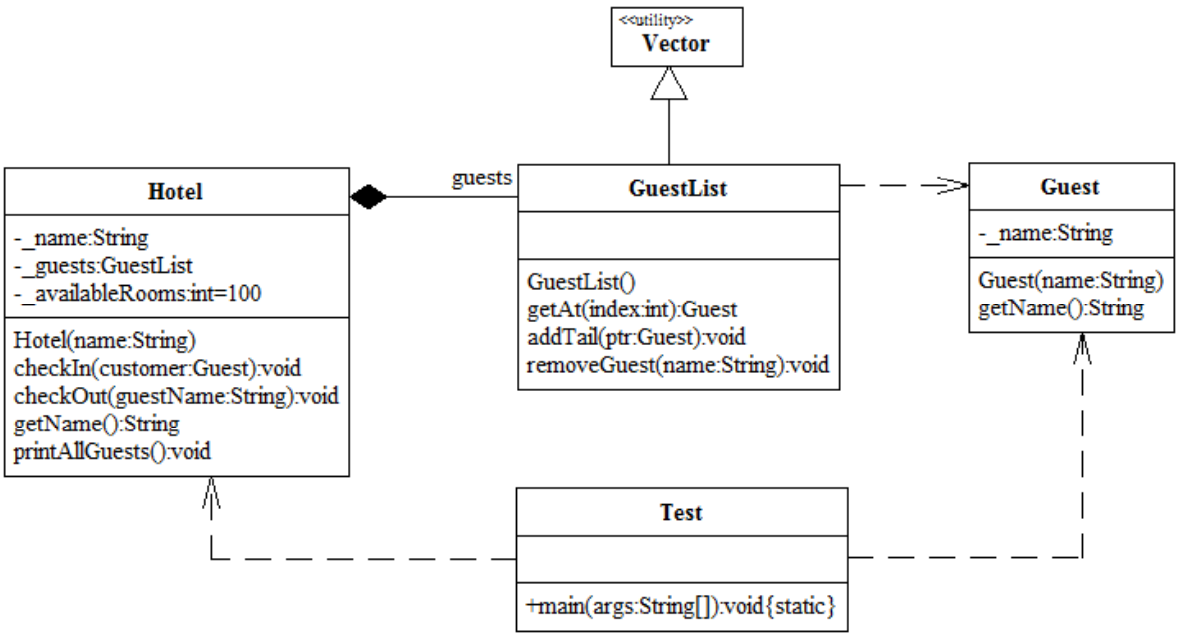
실행결과

```
Even...
Winner:[두리:9992 games, 3164 win, 3488 lose]
Winner:[하나:9993 games, 3488 win, 3165 lose]
Winner:[두리:9994 games, 3165 win, 3489 lose]
Winner:[두리:9995 games, 3166 win, 3489 lose]
Winner:[하나:9996 games, 3489 win, 3167 lose]
Even...
Even...
Even...
Total result:
[두리:10000 games, 3167 win, 3490 lose]
[하나:10000 games, 3490 win, 3167 lose]
```

알고리즘 부분을 다른 코드 부분과 분리해서 구현하는 방법

4장 예제 리뷰

- [예1] 호텔에 고객이 투숙하고 있는 관계
  - "Hotel"과 "Guest"간 1대다 관계



```
class Guest {
// Attributes
     private String _name;

// Operations
     Guest(String name) {
          _name = name;
     }
     String getName() {
          return _name;
     }
}
```

```
import java.util.Iterator;
import java.util.Vector;

class GuestList extends Vector {
    // Attributes
    static final long serialVersionUID = -5507543155571264736L;

    // Operations
    GuestList() {
        super();
    }

    Guest getAt(int index) {
        Object obj = super.get(index);
        return (Guest) obj;
    }

    void addTail(Guest ptr) {
        super.add(ptr);
    }

    void removeGuest(String name) {
        Guest tmp;

        for (int i = 0; i < this.elementCount; i++) {
            tmp = (Guest) this.elementAt(i);
            if (name.equals(tmp.getName()))
                super.removeElementAt(i);
        }
    }
}
```

```java
import java.util.Iterator;

class Hotel {
// Attributes
    private String _name;
    private GuestList _guests;
    private int _availableRooms = 100;

// Operations
    Hotel(String name) {
        _name = name;
        _guests = new GuestList();
    }
    void checkIn(Guest customer) {
        _guests.addTail(customer);
        _availableRooms--;
    }
    void checkOut(String guestName) {
        _guests.removeGuest(guestName);
        _availableRooms++;
    }
    String getName() {
        return _name;
    }
    void printAllGuests() {
        Iterator<Guest> i = _guests.iterator();

        System.out.println("\n>>> " + getName() + " 투숙객");
        while (i.hasNext())
            System.out.print(i.next().getName() + " ");
    }
}
```

```java
class Test {
// Operations
    public static void main(String[] args) {
        Hotel h1 = new Hotel("인터불고호텔");
        Hotel h2 = new Hotel("현대호텔");
        Guest guest;

        for (int i = 0; i < 10; i++) {
            guest = new Guest("아무개" + i);
            h1.checkIn(guest);
        }
        h1.printAllGuests();

        h1.checkOut("아무개1");
        h1.checkOut("아무개9");
        h1.printAllGuests();

        h2.checkIn(new Guest("홍길동"));
        h2.checkIn(new Guest("김철수"));
        h2.checkIn(new Guest("이영희"));
        h2.printAllGuests();

        h2.checkOut("홍길동");
        h2.printAllGuests();
    }
}
```

- 실행결과

```
>>> 인터불고호텔 투숙객
아무개0 아무개1 아무개2 아무개3 아무개4 아무개5 아무개6 아무개7 아무개8 아무개9
>>> 인터불고호텔 투숙객
아무개0 아무개2 아무개3 아무개4 아무개5 아무개6 아무개7 아무개8
>>> 현대호텔 투숙객
홍길동 김철수 이영희
>>> 현대호텔 투숙객
김철수 이영희
```

<실행조건>

호텔객체를 3개 이상 만들고, 임의의 고객들을 각각 추가한 후 실행

❖수정1

- 단방향 의존관계를 그대로 두고, 특정 고객이 숙박하고 있는 호텔이름을 출력하는 기능 구현

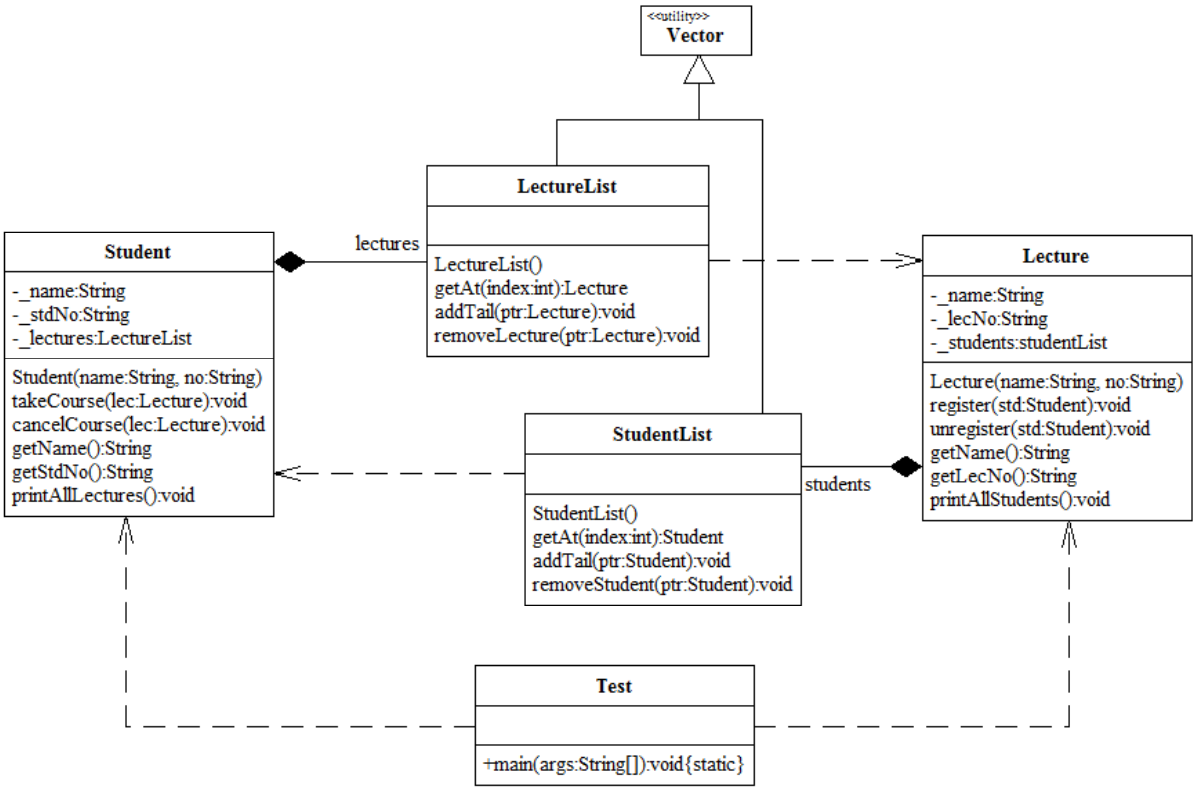
– 예) “이영희”가 숙박하는 호텔은?

- “현대호텔” 출력

❖ 수정2

- 양방향 의존관계로 수정하여, 특정 고객이 숙박하고 있는 호텔이름을 출력하는 기능 구현

- [예2] “Student”와 “Lecture”간 다대다 관계



객체 지향적으로 프로그래밍 하는 예제

- 에덴동산 : GUI가 없는 프로그램
  - 자료추상화, 상속, 동적 바인딩, 다형 개념 적용한 예
  - Java, C++ 로 구현
- 그래픽 편집기 : GUI가 있는 프로그램
  - VC++ Drawer : MFC 사용
  - Java Drawer : Swing 사용

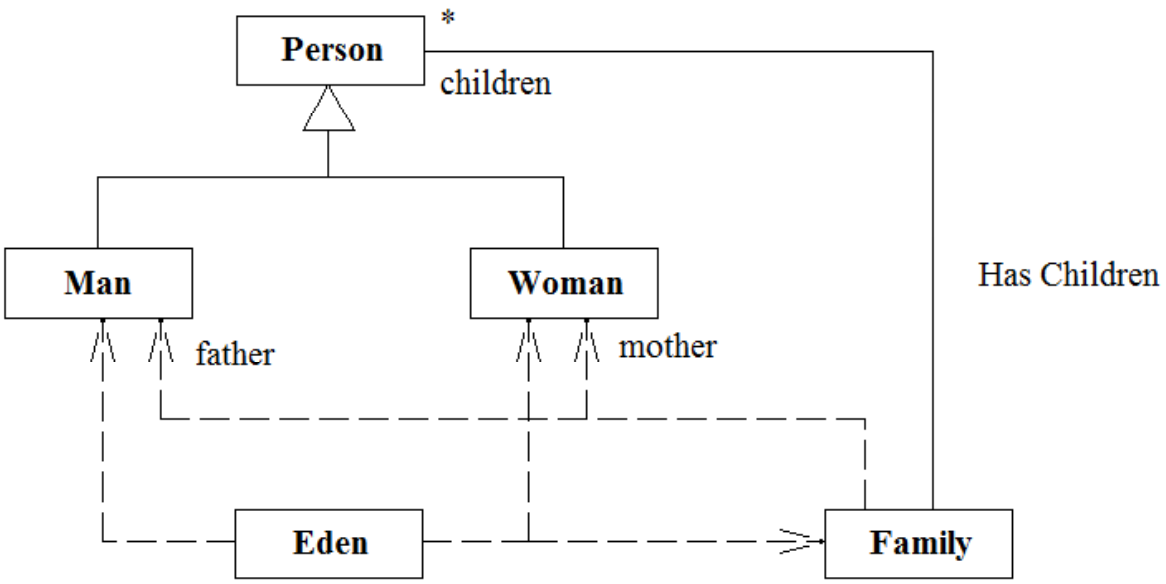
5.1 에덴 동산 예제

❑ 요구 사항 명세서 : 교재 p.306

이 프로그램의 작성 목적은 객체 지향 구현 방식을 교육하기 위한 것이다. 이 프로그램의 이름을 에덴 동산이라 하자. 이 프로그램이 수행하는 기능은 가상의 세계인 에덴 동산에서 생활하는 사람 객체들의 행위와 상태 변화를 시뮬레이션 하는 것이다. 에덴 동산에는 남자와 여자로 분류되는 두 종류의 사람 객체가 존재한다. 남자 객체의 경우는 이름, 몸무게, 힘이라는 상태 값을 가지며 여자 객체의 경우는 이름, 몸무게, 행복도라는 상태 값을 갖는다.

...

개략 설계



❑ 메인 함수

- Man, Woman, Family 클래스가 이미 존재한다는 가정하에 작성된 main() 함수

```
1  public class Eden {
2      public static void main(String[] args) {
3          int SOME_DURATION = 10;
4          // 옛날 옛적에 지오디가 아담을 창조하셨다.
5          Man adam = new Man("아담");
6
7          int clock;
8          // 아담은 한 동안 밥 먹고 화장실 가는 일만을 반복했다.
9          for (clock = 0; clock < SOME_DURATION; clock++) {
10             adam.eat(Person.BANANA); // 아침에는 바나나를 먹고
11             adam.urinate(Person.BIG); // 화장실에서 큰걸 해결한 후
12             adam.eat(Person.MEAT); // 점심으로 불고기를 먹은 후
13             adam.eat(Person.APPLE); // 저녁에는 사과를 먹었다.
14         }
```

```
16     /* 매일 밥 먹고 화장실 가는 일만 반복하던 그는 매우 심심하다고 느꼈다.
17        그래서 아담은 지오디에게 "여자"라고 분류되는 동반자를 만들어 달라고 부탁하였다.
18        다행히 지오디는 그 부탁을 들어 주었고 이브를 창조하였다. */
19        Woman eve = new Woman("이브");
20
21        // 아담은 이브와 함께 밥 먹고 화장실 가는 일을 반복하며 생활했다.
22        for (clock = 0; clock < SOME_DURATION; clock++) {
23            // 아침 식사
24            adam.eat(Person.BANANA); // 아담은 아침에 바나나를 먹고
25            eve.eat(Person.APPLE); // 이브는 아침에 사과를 먹었다.
26            // 화장실 가기
27            adam.urinate(Person.BIG); // 아담은 화장실에서 큰걸 보고
28            eve.urinate(Person.SMALL); // 이브는 작은 걸 본다.
29            // 점심 식사
30            adam.eat(Person.MEAT); // 아담은 점심에 불고기를 먹고
31            eve.eat(Person.APPLE); // 이브는 점심에도 사과를 먹었다.
32            // 저녁 식사
33            adam.eat(Person.APPLE); // 아담은 저녁에 사과를 먹고
34            eve.eat(Person.APPLE); // 이브는 저녁에도 사과를 먹었다.
35        }
```

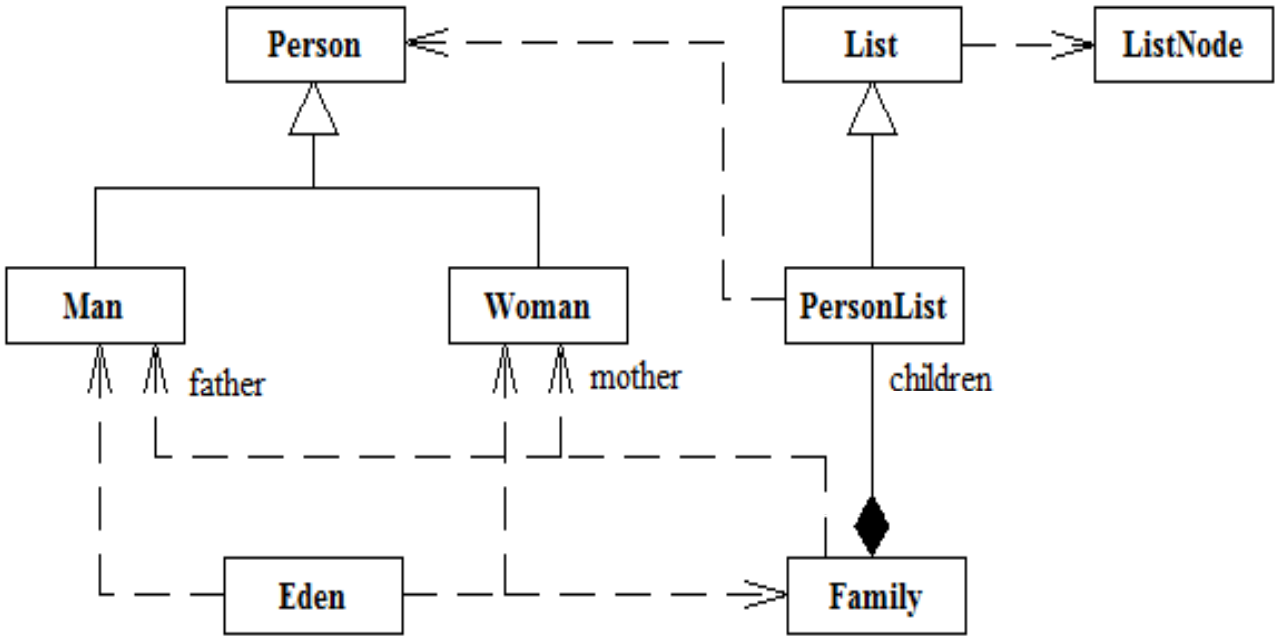
```
37      /* 함께 식사하고 밥 먹는 일을 반복하며 생활하던 아담과 이브는
38         좀 더 즐거운 시간을 보내기 위한 놀이를 생각해냈다.
39         그 놀이의 이름은 "그걸 한다"인데 하루 일과가 끝난 후에 하기에 적당한 놀이였다.
40         그래서 그 둘은 매일 밤에 그걸 하기로 했다. */
41         for (clock = 0; clock < SOME_DURATION; clock++) {
42            // 아침 식사
43            adam.eat(Person.BANANA); // 아담은 아침에 바나나를 먹고
44            eve.eat(Person.APPLE); // 이브는 아침에 사과를 먹는다.
45            // 화장실 가기
46            adam.urinate(Person.BIG); // 아담은 화장실에서 큰걸 보고
47            eve.urinate(Person.SMALL); // 이브는 작은 걸 본다.
48            // 점심 식사
49            adam.eat(Person.MEAT); // 아담은 점심에 불고기를 먹고
50            eve.eat(Person.APPLE); // 이브는 점심에도 사과를 먹는다.
51            // 저녁 식사
52            adam.eat(Person.APPLE); // 아담은 저녁에 사과를 먹고
53            eve.eat(Person.APPLE); // 이브는 저녁에도 사과를 먹는다.
54            // 해지고 난 후 한 밤중에
55            adam.doingX(eve); // 아담과 이브는 그걸 한다.
56         }
```

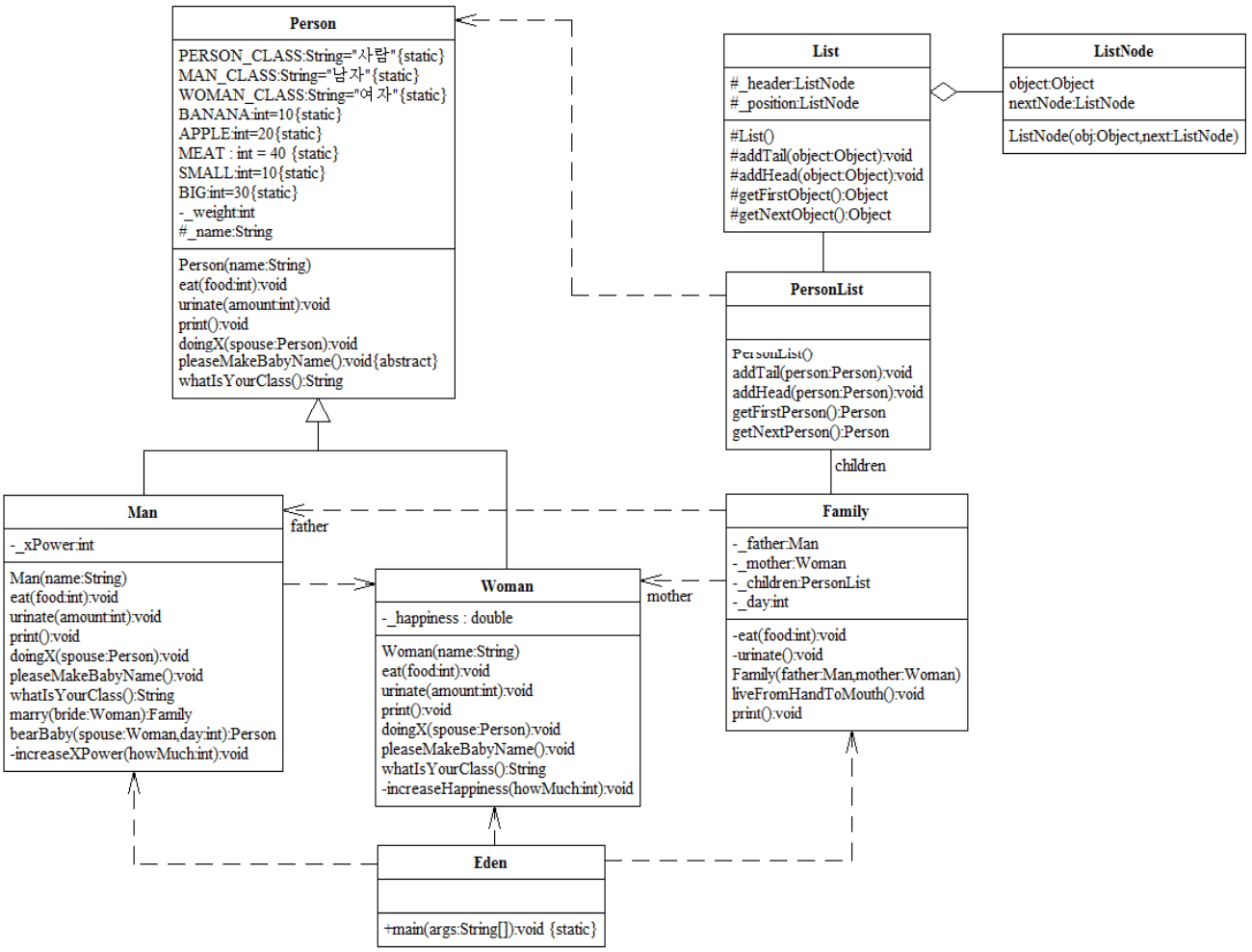
```
57      /* 정확한 이유는 모르겠지만 지오디는 이브가 자꾸 사과만 먹는 것 때문에 화가 났다.
58         그래서 지오디는 이들에게 놀이를 하기 위한 의무를 지우기로 결정했다.
59         그걸 하기 위한 놀이의 의무의 명칭은 "결혼한다"는 것이며 그 의무의 부담으로서
60         결혼하는 사람들은 가족을 구성해야하고 그걸 365회 할 때마다 아들이나 딸이 새로
61         태어나야 한다는 것이다.
62         아담과 이브는 이 의무를 따르기로 결정하고 둘이 결혼하여 새 가족을 구성하였다. */
63         Family aFamily = adam.marry(eve);
64         // 아담과 이브는 가정을 꾸린 후 아이들을 낳으면서 그럭저럭 살아간다.
65         for (clock = 0; clock < 200*SOME_DURATION; clock++) {
66            aFamily.liveFromHandToMouth();
67         }
68         // 자, 이 가족 구성원의 상태 값(성별,몸무게,힘,행복도)들을 알아보자.
69         aFamily.print();
70      }
71   }
```

- Person 클래스

```
1   abstract class Person
2   {
3   // Attributes
4       static String PERSON_CLASS = "사람";
5       static String MAN_CLASS = "남자";
6       static String WOMAN_CLASS = "여자";
7       static int BANANA = 10;
8       static int APPLE = 20;
9       static int MEAT = 40;
10      static int SMALL = 10;
11      static int BIG = 30;
12      private int _weight;
13      protected String _name;
```

상세 설계





❑ 실행 화면

```
안녕하세요 지오디 ! 제 이름(딸 이름) 좀 정해주세요 : a
안녕하세요 지오디 ! 제 이름(아들 이름) 좀 정해주세요 : b
안녕하세요 지오디 ! 제 이름(딸 이름) 좀 정해주세요 : c
안녕하세요 지오디 ! 제 이름(아들 이름) 좀 정해주세요 : d
안녕하세요 지오디 ! 제 이름(딸 이름) 좀 정해주세요 : e

                << 가족의 상태 값 >>
+--------+------+--------+--------+-----------+
|  이름  | 성별 | 몸무게 |   힘   |   행복도  |
+--------+------+--------+--------+-----------+
|   아담 | 남자 |  61200 |    200 |           |
|   이브 | 여자 |  81000 |        | 235678.40 |
|      a | 여자 |  98260 |        | 107027.12 |
|      b | 남자 |  50940 |  12710 |           |
|      c | 여자 |  54460 |        |  59270.52 |
|      d | 남자 |  21740 |   5410 |           |
|      e | 여자 |  10660 |        |  11513.92 |
+--------+------+--------+--------+-----------+
```

C++ 버전을 자바 버전으로 변환하기

- 잘 짜여진 C++ 프로그램인 경우에는 기계적인 변환이 가능함
- 파일 이름 변환
- 매크로 정리
- 포인터 타입의 변수 정리
- 가시성 정리
- 내부 클래스 사용 방식 정리
- 소멸자 삭제
- 가상함수 정리

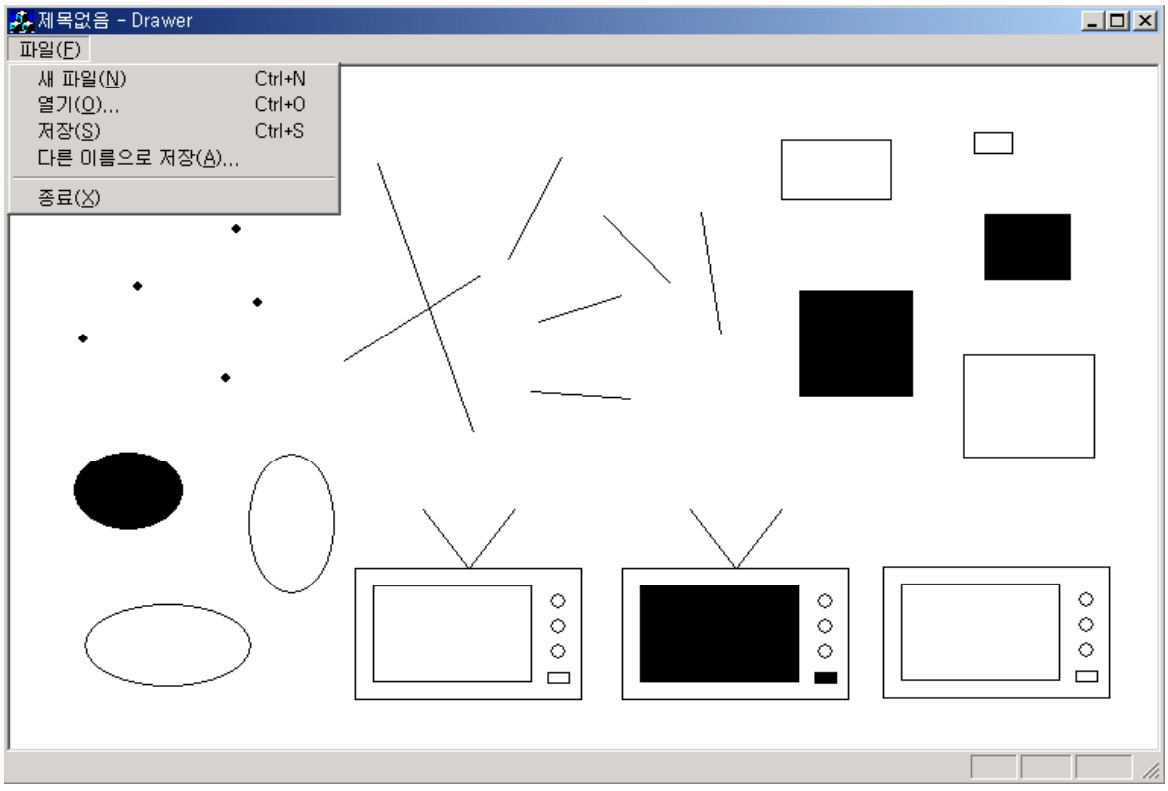
- 화살표를 이용한 함수 호출 정리
- NULL 매크로 정리
- 상속 명시 방법 정리
- 상위 클래스의 생성자 호출 정리
- 순수 가상 함수 정리
- 메인 프로그램 정리
- 문자열 처리 방식 정리
- 스트링 함수 정리
- i/o 함수 정리

5.2 간단한 그래픽 편집기 예제

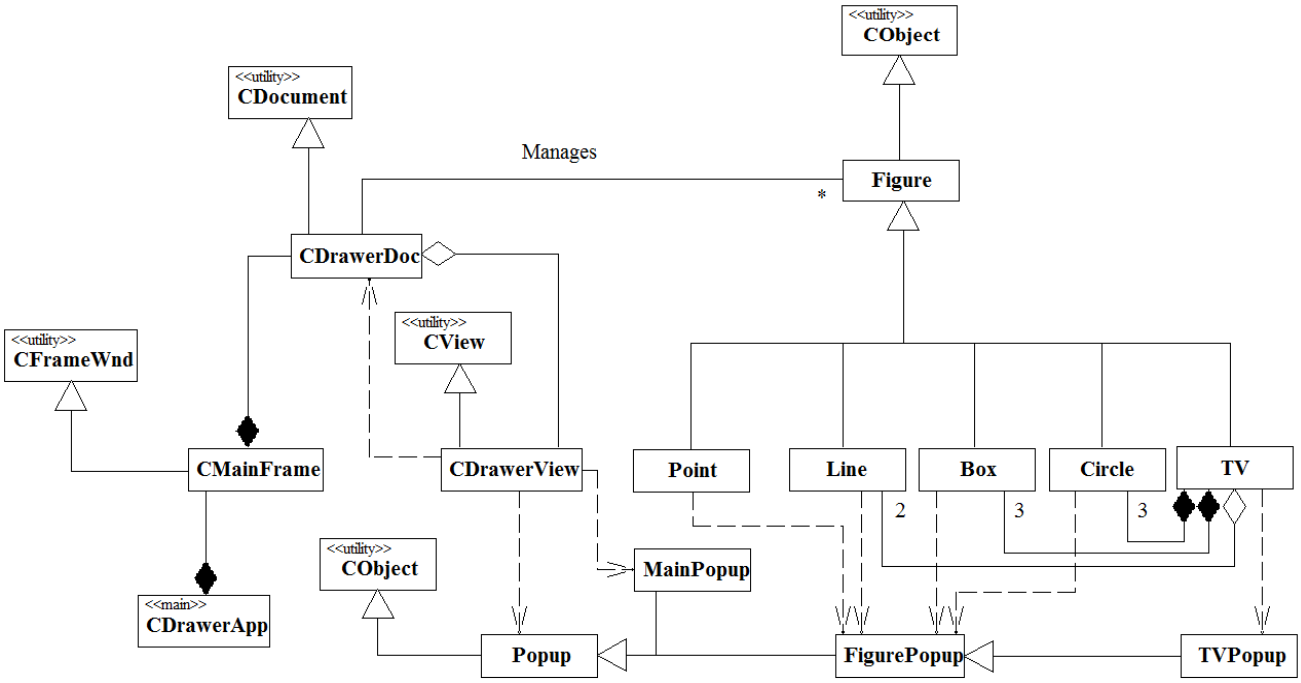
❑ 요구 사항 명세서

이 프로젝트를 통해 만들고자 하는 소프트웨어는 가칭 Drawer라 불리는 간단한 그래픽 편집기이다. 이 편집기는 윈도 환경에서 작동되며 사용자 인터페이스의 모습이 이렇게 저렇게 생겼는데 이를 그림으로 나타내면 (그림 5.5)와 같다. 이 소프트웨어는 풀다운 메뉴와 팝업 메뉴를 통하여 작동되는데 풀다운 메뉴에 속하는 아이템은 이런 저런 것이 있으며, 팝업 메뉴는 이런 저런 형태로 만들어지고 이런 저런 경우에 사용된다. 이 에디터가 그릴 수 있는 그림은 **점**, **선**, **사각형** 등이고, ...

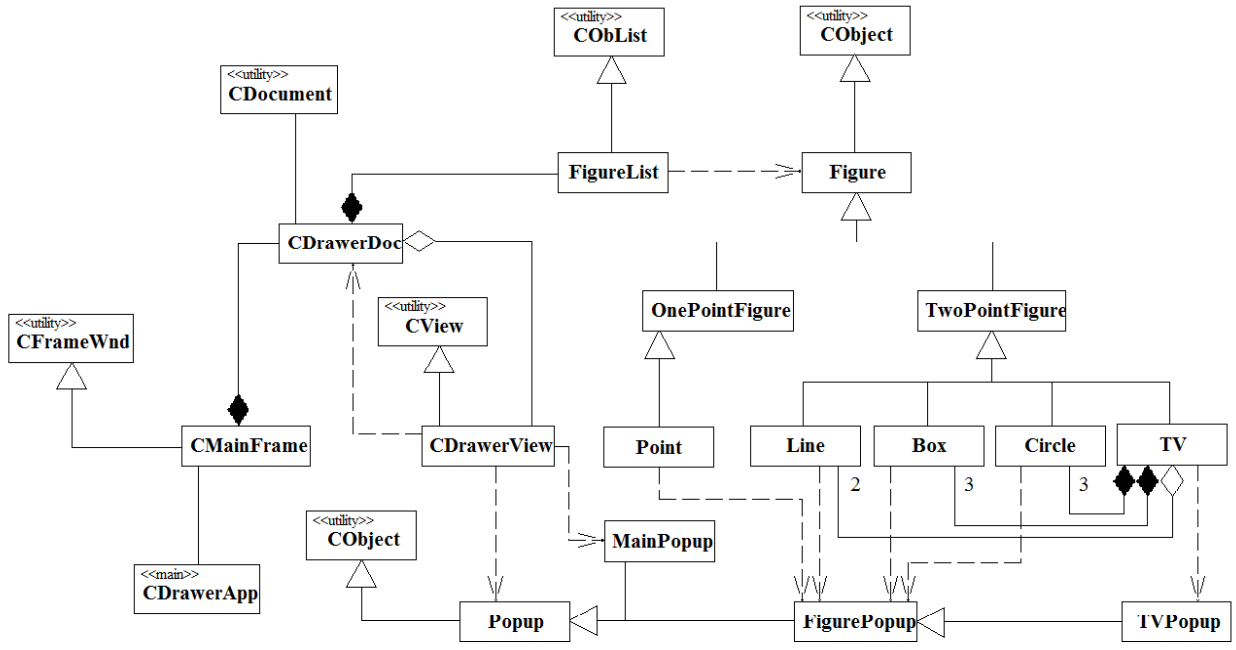
❑ 실행 화면



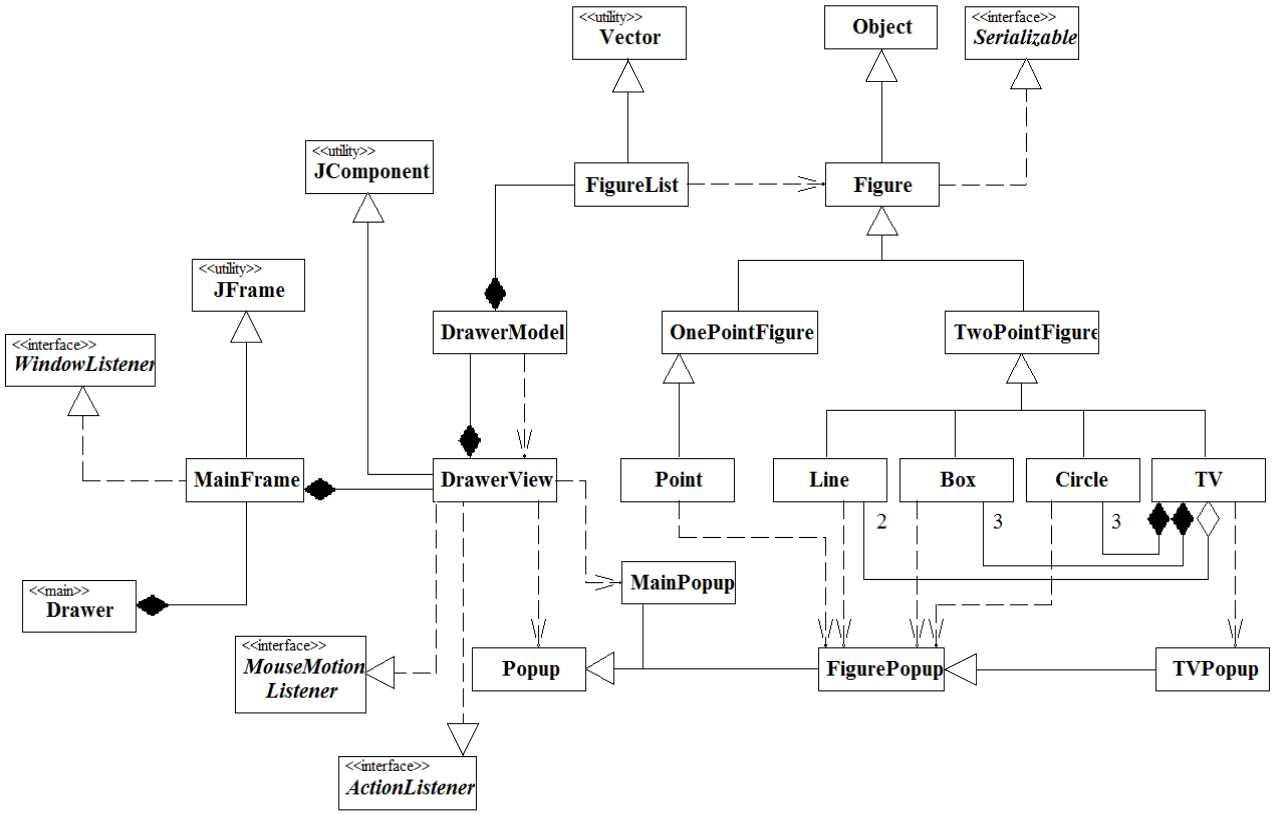
❑ 개략 설계


<<utility>>
CObject
<<utility>>
CDocument
Manages
Figure
*
CDrawerDoc
<<utility>>
CFrameWnd
<<utility>>
CView
CMainFrame
CDrawerView
Point
Line
Box
Circle
TV
2
3
3
<<utility>>
CObject
MainPopup
<<main>>
CDrawerApp
Popup
FigurePopup
TVPopup

❑ 상세 설계


<<utility>>
CObList
<<utility>>
CObject
<<utility>>
CDocument
FigureList
Figure
CDrawerDoc
OnePointFigure
TwoPointFigure
<<utility>>
CFrameWnd
<<utility>>
CView
CMainFrame
CDrawerView
Point
Line
Box
Circle
TV
2
3
3
<<utility>>
CObject
MainPopup
<<main>>
CDrawerApp
Popup
FigurePopup
TVPopup

❑ 자바 버전 상세 설계


<<utility>>
Vector
Object
<<interface>>
Serializable
<<utility>>
JComponent
FigureList
Figure
<<utility>>
JFrame
DrawerModel
OnePointFigure
TwoPointFigure
<<interface>>
WindowListener
MainFrame
DrawerView
Point
Line
Box
Circle
TV
2
3
3
MainPopup
<<main>>
Drawer
<<interface>>
MouseMotion
Listener
Popup
FigurePopup
TVPopup
<<interface>>
ActionListener