7
자료구조: 배열

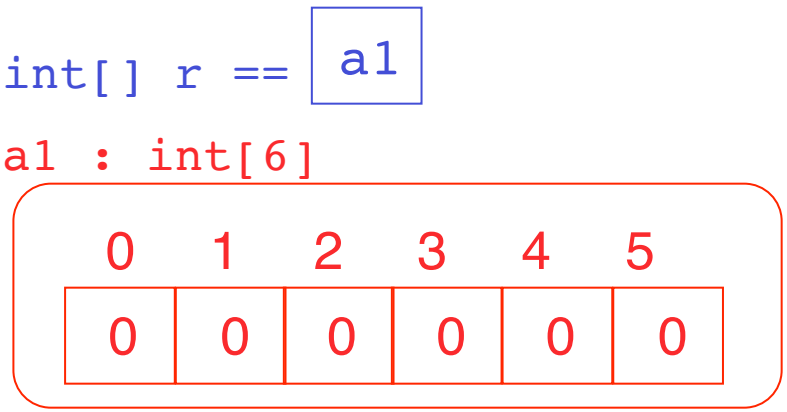
배열이 필요한 이유

- 6명의 학생의 점수를 받아서 각 학생에게 최고점과 얼마나 차이가 나는지 알려주려고 한다.
  - int score1, score2, score3, score4, score5, score6;
  - int high_score = score1;
  - if(score2 > high_score) high_score = score2;
  - ...
  - if(score6 > high_score) high_score = score6;

- 이렇게 하면 좋겠는데
  - for(i=2; i<=6; i++)
    { if(score $_i$ > high_score) high_score = score $_i$; }

배열

- 정의
  - 동일한 타입의 자료(원소)를 고정된 개수만큼 갖고 있는 자료 구조
- 타입에 [ ]를 붙이면 배열 타입이 된다.
  - `int[]`, `String[]`
- Java의 배열은 객체이다.
  - new로 생성한다.
  - `int[] r = new int[6];`



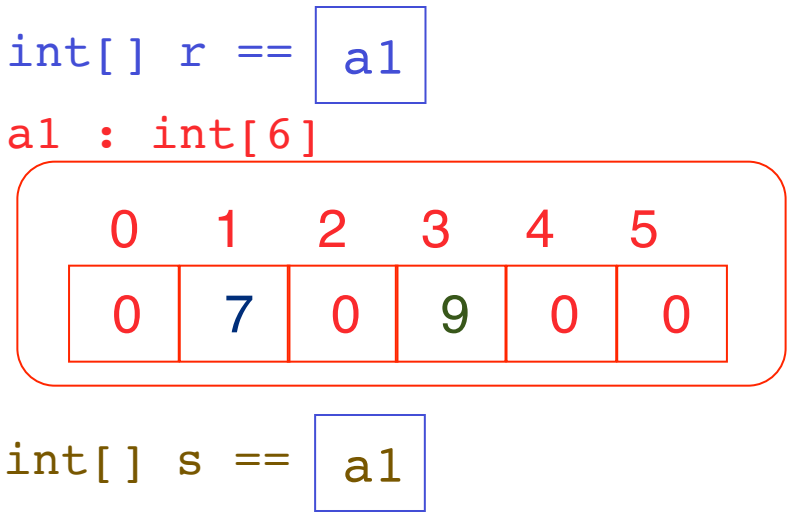
배열 다루기

```
int[] r = new int[6];
int x = 6;

r[1] = 7;
r[3] = r[x-5] + 2;
int[] s = r;
```

인덱스(index)는 임의의 정수 계산식 가능

같은 배열을 가리킴



배열의 초기화

- 배열의 원소들은 생성될 때 초기화된다.
  - 필드 변수 초기화 되는 것과 동일
    - int: 0, double: 0.0, boolean: false, 객체타입: null
- 생성 및 사용자 초기화
  - `int[] r = {1, 2, 4, 8, 16, 32};`

루프를 사용하여 초기화하기

```
int[] r = new int[12];

r[0] = 1; r[1] = 1;
for (int i=2; i<r.length; i=i+1) {
  r[i] = r[i-1] + r[i-2];
}

vs

int[] r = { 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144 };
```

배열의 크기는 length 필드에 저장됨

메소드에 인수로 전달, 결과로 반환

- 배열의 내용이 뒤집힌 배열을 반환하는 메소드를 작성하여라.

```
int[] reverse(int[] r) {
   int size = r.length;
   int[] answer = new int[size];
   for(i=0; i<size; i++) {
      answer[size-1-i] = r[i];
   }
   return answer;
}
```

배열이 복사되어 전달되는 것이 아니라 주소가 전달되는 것이다.

배열이 복사되어 반환되는 것이 아니라 주소가 반환되는 것이다.

예제, 단순치환 암호

- 공백과 알파벳 소문자로 구성된 문자열을 변환표에 따라 암호화하고 복호화하는 프로그램을 작성해라.
  - 편의상 공백=0, 'a'=1, ..., 'z'=26 코드를 사용하겠다.

- 변환표는 다음과 같이 만든다.
  - code[0] = seed
  - code[i] = (code[i-1]+4) % 27

명세: class TranslateTable

생성자	
`TranslateTable(int seed)`	seed를 사용하여 암호화 테이블, 복호화 테이블 구축
속성	
`int[] encode`	암호화 테이블
`int[] decode`	복호화 테이블
메소드	
`encode(char c): char` `decode(char c): char`	문자를 암호화/복호화
`encode(String s): String` `decode(String s): String`	문자열을 암호화/복호화

변환표 생성

```
class TranslateTable {

    private int[] encode;
    private int[] decode;

    public TranslateTable(int seed) {
        encode = new int[27];
        decode = new int[27];
        encode[0] = seed;
      decode[seed] = 0;
        for(int i=1; i<27; i++) {
            int new_code = (encode[i-1]+4) % 27;
            encode[i] = new_code;
            decode[new_code] = i;
        }
    }
```

문자 코드 간 변환

```
private int c2i(char c) {
   if(c==' ') return 0;
   if('a'<=c && c<='z') return c-'a'+1;
   throw new RuntimeException("c2i: invalid character " + c);
}

private char i2c(int i) {
   if(i==0) return ' ';
   if(0<i && i<27) return (char)('a'+i-1);
   throw new RuntimeException("i2c: invalid code " + i);
}
```

암호화

```
    public char encode(char c) {
       return i2c(encode[c2i(c)]);
    }

    public String encode(String s) {
       String answer = "";
    for(int i=0; i<s.length(); i++)
           answer = answer + encode(s.charAt(i));
       return answer;
    }

  // 복호화도 유사하게 작성할 수 있다.
}
```

구동 클래스

```
import javax.swing.*;

class TranslateString {
    public static void main(String[] args) {
       TranslateTable m = new TranslateTable(1);
       String original = JOptionPane.showInputDialog
       ("암호화할 문장을 입력하세요.");
       String encoded = m.encode(original);
       String decoded = m.decode(encoded);
       JOptionPane.showMessageDialog(null,
       "원본: " + original + "\n암호화: " + encoded +
       "\n복호화: " + decoded);
    }
}
```

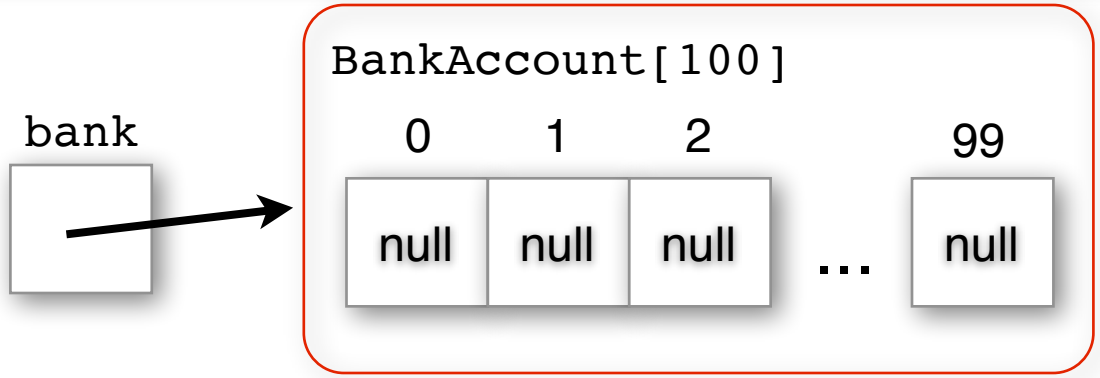
예제, 계좌 관리 프로그램

- 6장에서 다루었던 계좌 관리 프로그램을 두 계좌가 아닌 100 계좌를 관리할 수 있도록 하자.

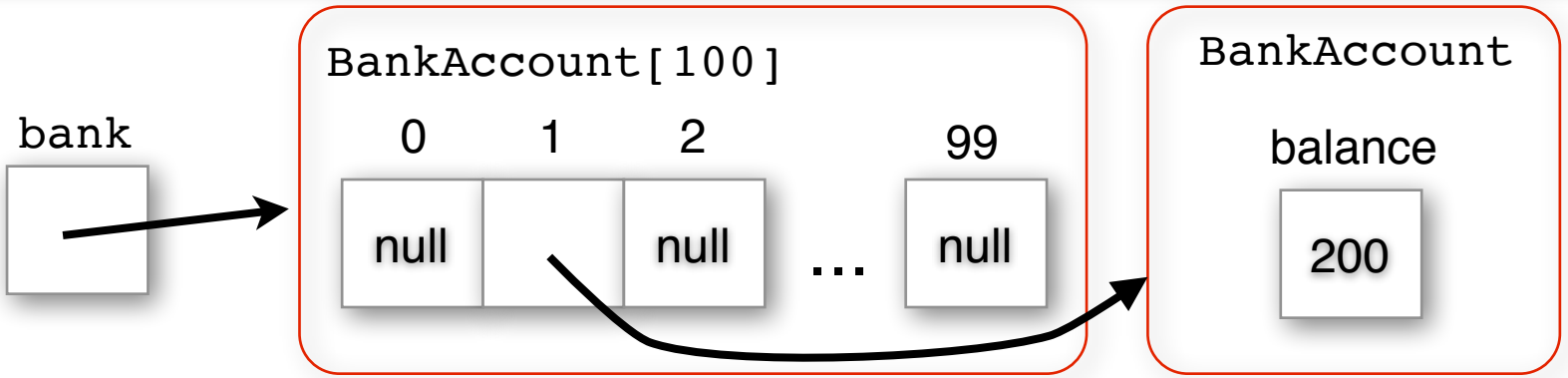
- 객체타입 배열 가능
  - `BankAccount[] bank = new BankAccount[100];`
  - 100개의 계좌가 만들어진 것이 아니라, 100개의 계좌를 저장할 수 있는 배열이 생성된 것.
  - `bank[75] = new BankAccount(200);`
  - 이제 2.00$가 들어 있는 계좌가 생성된 것이다.

계좌 배열

```
BankAccount[] bank = new BankAccount[100];
```




```
bank[1] = new BankAccount(200);
```



객체 배열 다루기

- bank[1].deposit(600);
  - bank[1] 객체에 deposit 메시지를 보낸다.
- bank[1] = null;
  - bank[1] 계좌를 없앤다.
- 객체 배열의 속성 상 모든 원소가 다 있어야 하는 것은 아님

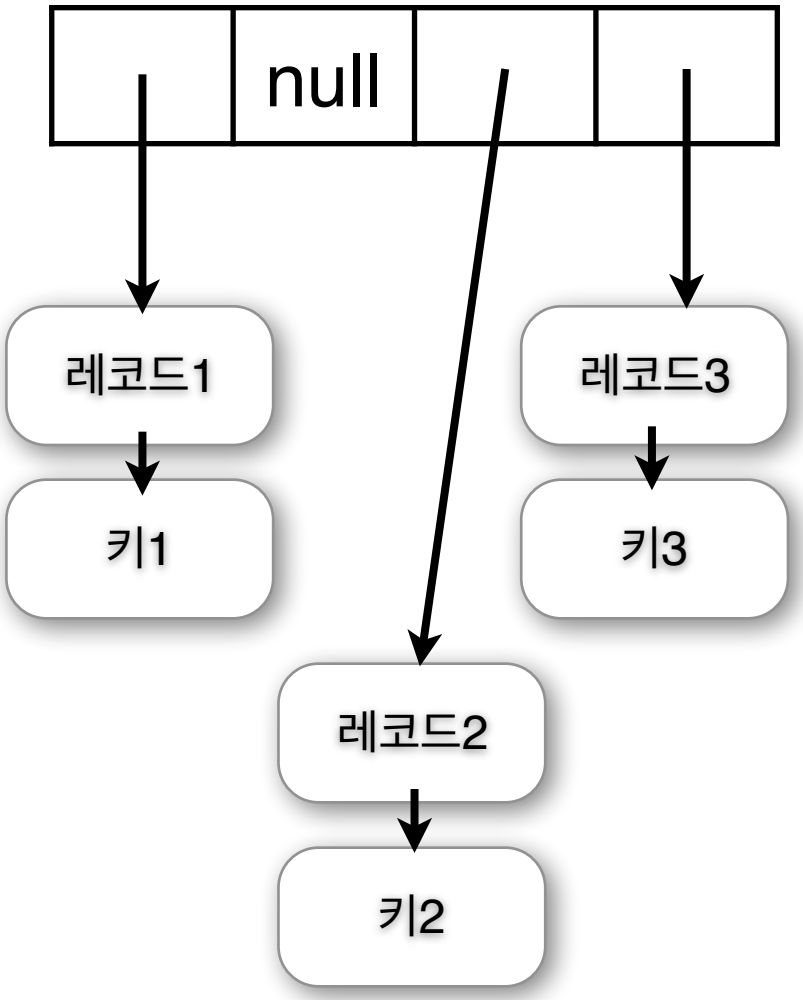
설계 예제, 데이터베이스

- 데이터베이스 (database)
  - 정보를 많이 모아둔 것
  - 예, 도서관의 소장도서 정보, 학교의 학생정보, 회사의 매출 정보 등

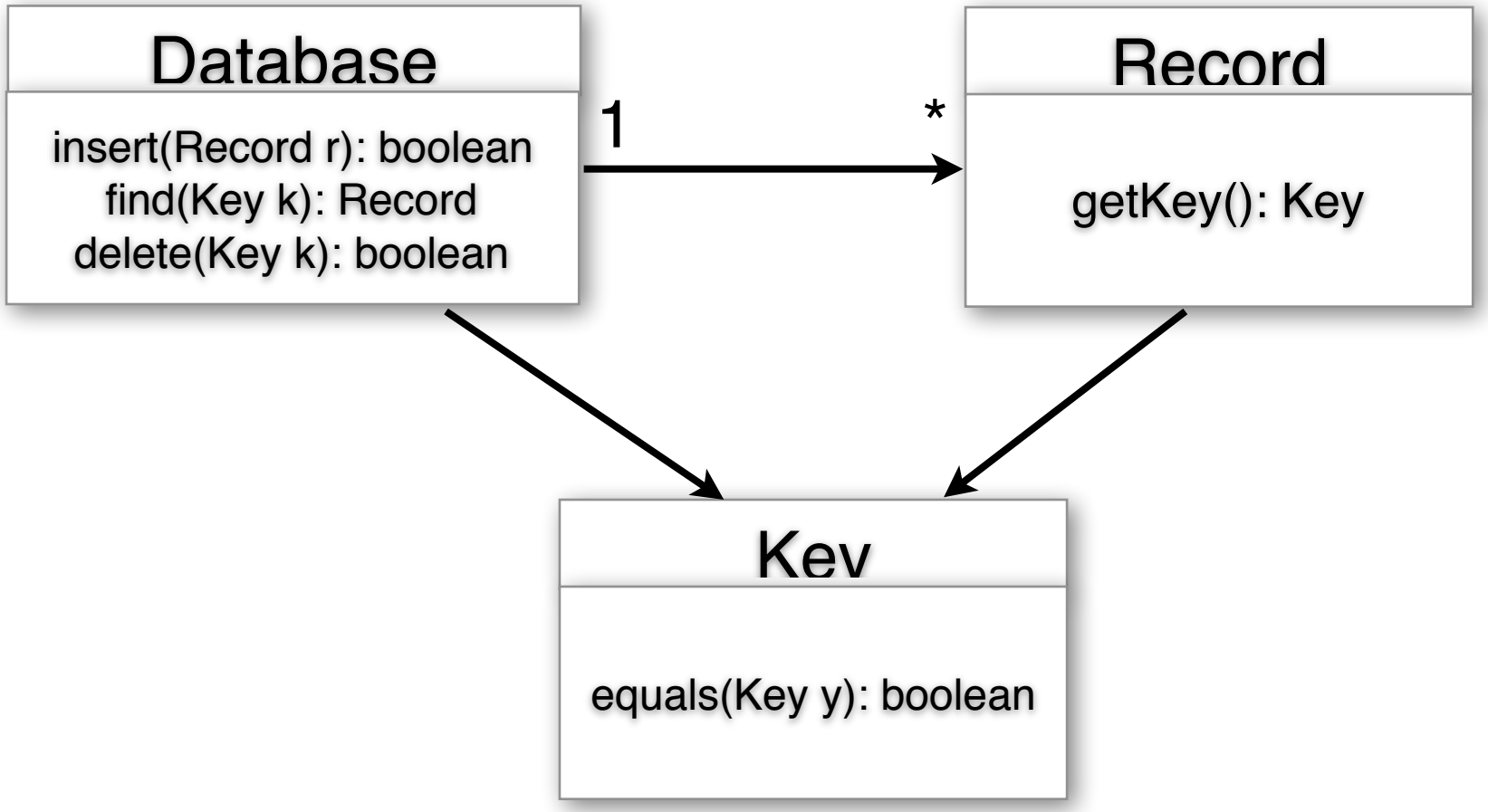
- 레코드 (record)
  - 데이터베이스에서 저장하는 정보의 한 단위
  - 키(key)를 통해 레코드들을 구별한다.

Java에서 단순 데이터베이스 만들기

- 키는 객체다.
- 레코드는 객체다. 키 객체를 갖고 있어야 한다.
- 데이터베이스는 레코드의 "배열"로 볼 수 있다. 레코드를 넣고, 찾고, 없앨 수 있어야 한다.
  - insert: 레코드를 데이터베이스에 추가
  - find: 키를 통해 레코드를 검색
  - delete: 키를 통해 레코드를 검색하여 삭제



소프트웨어 구조


Database
insert(Record r): boolean
find(Key k): Record
delete(Key k): boolean
1
*
Record
getKey(): Key
Key
equals(Key y): boolean

명세

`class Database`	레코드를 저장하는 컨테이너
메소드	
`insert(Recrd r): boolean`	r을 데이터베이스에 추가한다. 성공하면 true, 아니면 false를 반환한다.
`find(Key k): Record`	k 키를 가지는 레코드를 찾는다. 실패하면 null을 반환한다.
`delete(Key k): boolean`	k 키를 가지는 레코드를 삭제한다. 성공하면 true, 실패하면 false를 반환한다.

`class Record`	데이터베이스의 자료 단위
메소드	
`keyOf(): Key`	레코드의 키를 반환한다.

`class Key`	레코드의 식별자, 키
메소드	
`equals(Key m): boolean`	자기와 m을 비교한다. 같으면 true, 틀리면 false를 반환한다

필드, 생성 메소드

```
public class Database {

  private Record[] base;
    private int NOT_FOUND = -1;

  public Database (int initial_size) {
       if (initial_size <= 0)
          initial_size = 1;
       base = new Record[initial_size];
    }
```

위치 찾기 메소드

```
private int findLocation(Key k)
{
    for (int i=0; i<base.length; i++)
        if(base[i]!=null && base[i].getKey().equals(k))
            return i;
    return NOT_FOUND;
}
private int findEmpty()
{
    for (int i=0; i<base.length; i++)
        if(base[i]==null)
            return i;
    return NOT_FOUND;
}
```

find & delete

```
public Record find(Key k) {
    int index = findLocation(k);
    if(index != NOT_FOUND)
        return base[index];
    else
        return null;
}
public boolean delete(Key k) {
    int index = findLocation(k);
    if(index != NOT_FOUND) {
        base[index] = null;
    return true;
  }
  else
        return false;
}
```

insert

```
public boolean insert(Record r) {
    if(findLocation(r.getKey()) != NOT_FOUND)
        return false;
    int index = findEmpty();
    if(index != NOT_FOUND)
        base[index] = r;
    else {
        Record[] temp = new Record[base.length * 2];
        for(int i=0; i<base.length; i++)
            temp[i] = base[i];
    temp[base.length] = r;
        base = temp;
    }
    return true;
}}
```

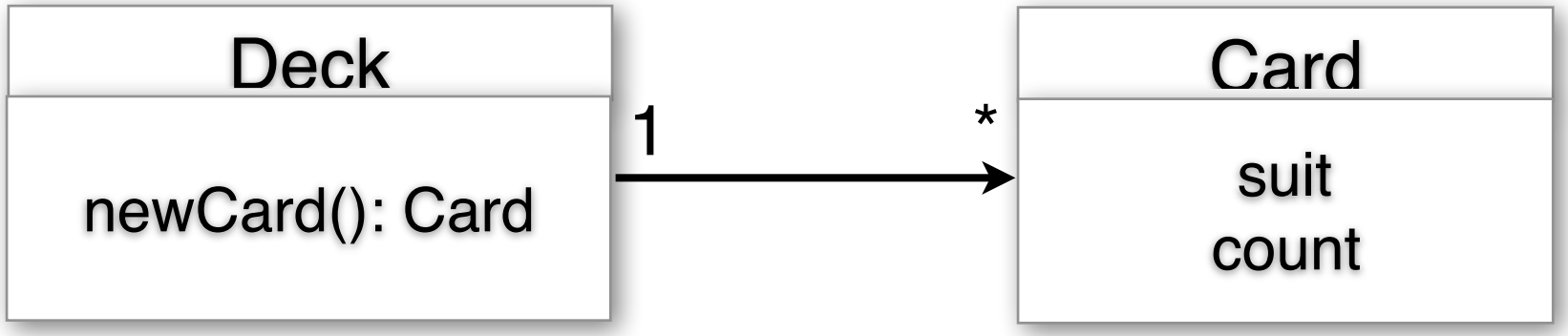
설계 예제, 카드 게임

- 카드 통에서 카드를 한 장씩 주는 프로그램을 작성해 보자.

- 카드
  - 모양(suit): 다이아몬드(diamonds), 하트(hearts), 클로버(clubs), 스페이드 (spades)
  - 숫자: A(1), 2~10, 잭(11), 여왕(12), 왕(13)

- 카드 통 (deck)
  - 카드의 배열

소프트웨어 구조



명세

class CardDeck	카드 통
속성	
private Card[] deck	남은 카드를 갖고 있다.
메소드	
newCard(): Card	카드 한 장을 준다. 통이 비었을 때는 null을 반환한다.
moreCards(): boolean	남은 카드가 있는지 반환한다.

class Card	카드
속성	
private suit: String	모양
private int count;	숫자
메소드	
getSuit(): String	모양 반환
getCount(): int	숫자 반환

카드

변경할 수 없는 필드 변수의 경우 final 한정자를 붙인다.

```
public class Card {
    public static final String SPADES = "spades";
    public static final String HEARTS = "hearts";
    public static final String DIAMONDS = "diamonds";
    public static final String CLUBS = "clubs";

    public static final int ACE = 1;
    public static final int JACK = 11;
    public static final int QUEEN = 12;
    public static final int KING = 13;
    public static final int SIZE_OF_ONE_SUIT = 13;

    private String suit;
    private int count;

    public Card(String s, int c)
        { suit = s; count = c; }
    public String getSuit() { return suit; }
    public int getCount() { return count; }
}
```

카드 통

```
public class CardDeck {
    private int card_count; // 남은 카드 수
    private Card[] deck = new Card[4*Card.SIZE_OF_ONE_SUIT];
    // 불변식: deck[0]...deck[card_count-1]에는 카드가 있다.

  private void createSuit(String which_suit) {
    for(int i=1; i<=Card.SIZE_OF_ONE_SUIT; i++) {
            deck[card_count] = new Card(which_suit, i);
            card_count++;
        }
    }
    public CardDeck() {
        createSuit(Card.SPADES); createSuit(Card.HEARTS);
        createSuit(Card.CLUBS); createSuit(Card.DIAMONDS);
    }
```

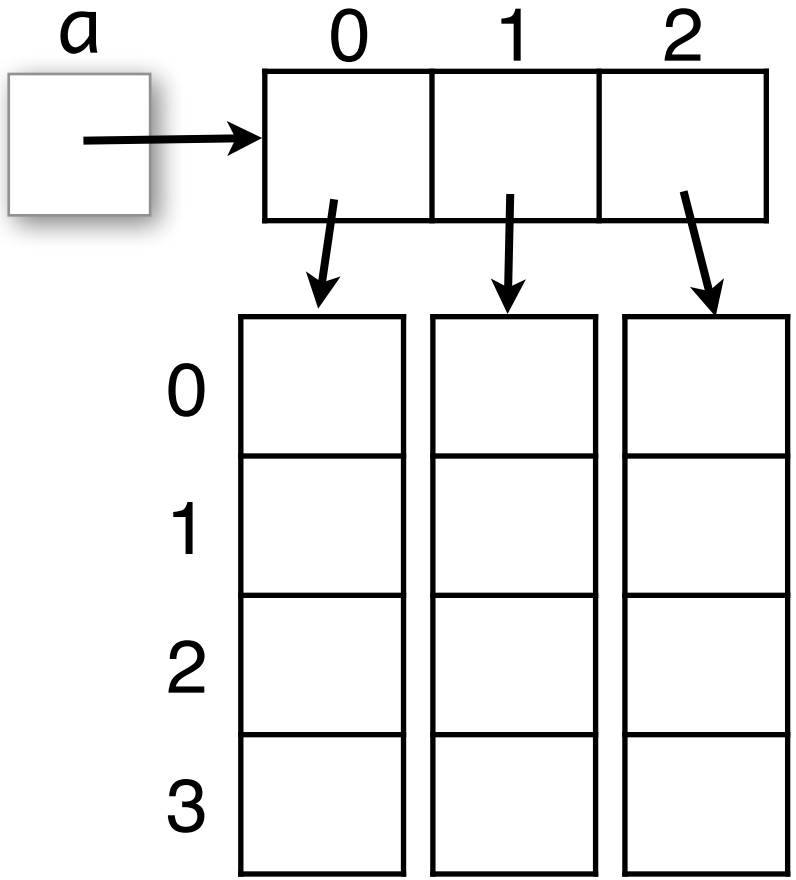
카드 통에서 카드 꺼내 주기

```
    public Card newCard() {
        Card next_card = null;
        if(card_count != 0 ) {
            int index = (int)(Math.random() * card_count);
            next_card = deck[index];
      // 카드를 뽑은 위치부터 앞으로 당겨 준다.
            for(int i=index+1; i<card_count; i++)
                deck[i-1] = deck[i];
            card_count--;
        }
        return next_card;
    }

    public boolean moreCards() { return card_count > 0; }
}
```

2차원 배열

- 정의: 배열을 원소로 하는 배열
- 문법
  - 생성: `int[][] a = new int[3][4];`
  - 열(column)의 수: `a.length`
  - 행(row)의 수: `a[0].length`



0,0	1,0	2,0
0,1	1,1	2,1
0,2	1,2	2,2
0,3	1,3	2,3

Java에서는 배열의 배열로 구현된다.

예, 대선 통계


```
int[ ][ ] election = new int[3][4];

for(int j=0; j<4; j++) {
    int votes = 0;
    for(int i=0; i<3; i++)
        votes = votes + election[i][j];
    System.out.println
       ("기호" + (j+1) + "번은 " +
        votes + "표 받았습니다.");
}

for(int  i=0; i<3; i++) {
    int votes = 0;
    for(int j=0; j<4; j++)
        votes = votes + election[i][j];
    System.out.println
       ((i+1) + " 지역은 " + votes +
        "표 행사했습니다.");
}
```

들쭉날쭉 배열 (Ragged Array)

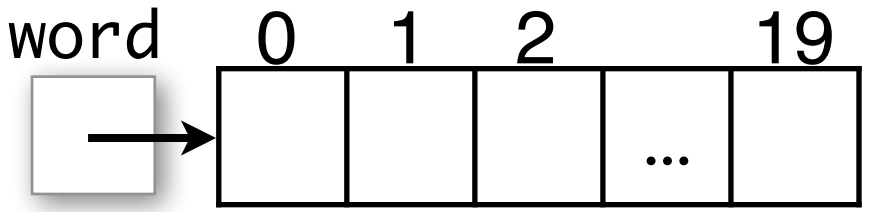
- 이차원 배열이 배열의 배열이기 때문에, 각 원소 배열의 크기를 다르게 할 수 있다.

```
int max_words = 20;
char[][] word = new char[max_words][];
int count = 0;
boolean processing = true;
while (processing && count < max_words) {
    String s = JOptionPane.showInputDialog("Please type a word: ");
    if (s.equals("")) processing = false;
    else {
        word[count] = new char[s.length()];
        for (int i=0; i<s.length(); i++)
            word[count][i] = s.charAt(i);
        count++;
    }
}
```

이차원 배열의 첫 크기만 지정하면 원소 배열이 생성되지 않는다.

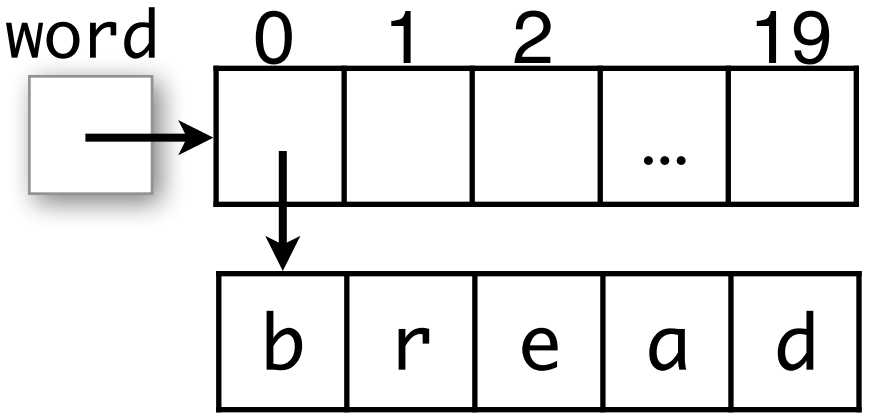
실행

```
char[][] word = new char[max_words][];
```



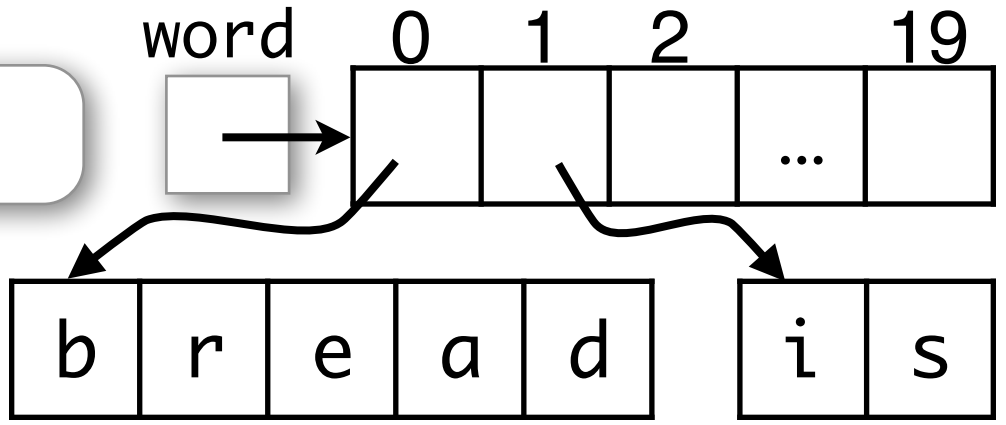

```
word[count] = new char[s.length()];
```

s="bread"

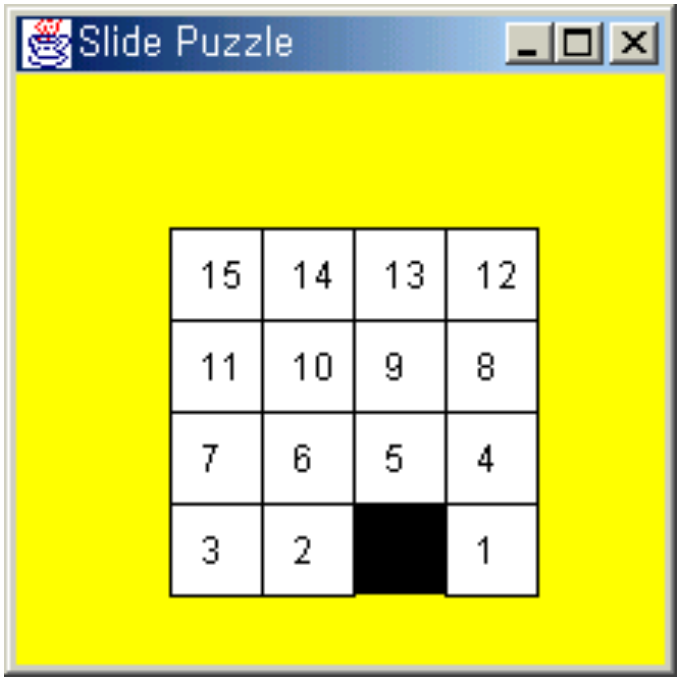


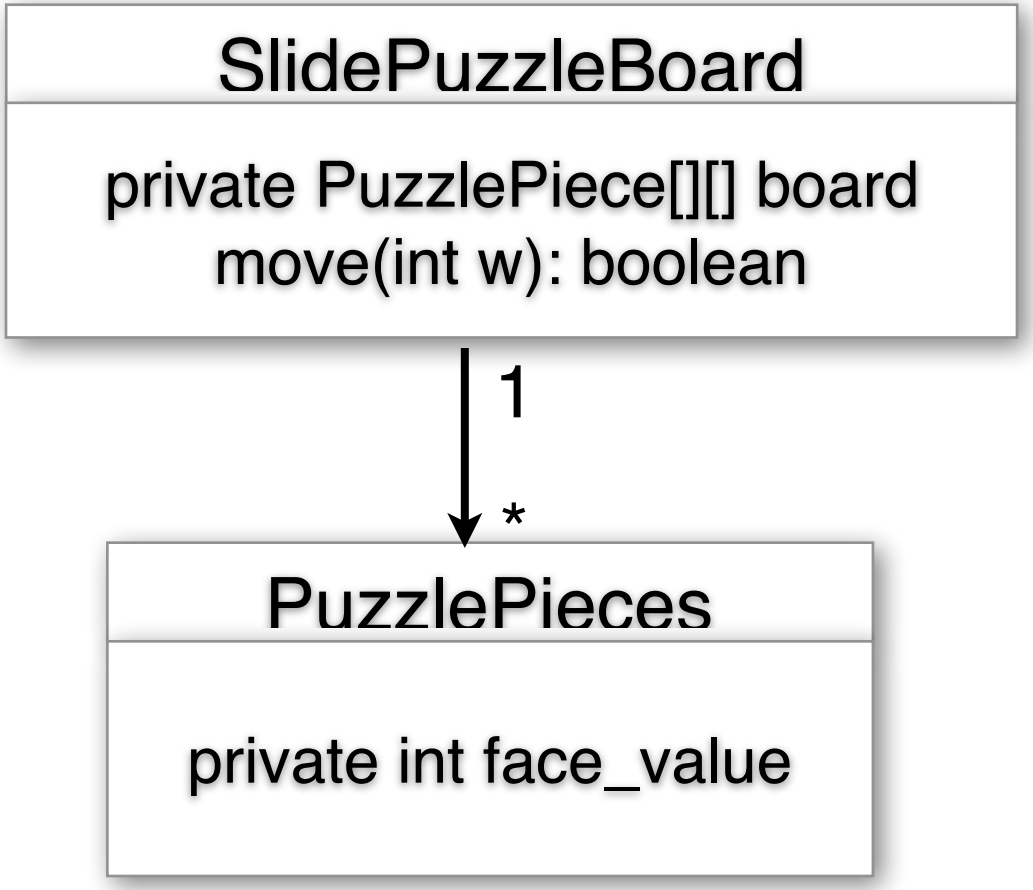

```
word[count] = new char[s.length()];
```

s="is"



예제, 퍼즐 게임


Slide Puzzle
15 14 13 12
11 10 9 8
7 6 5 4
3 2 1


SlidePuzzleBoard
private PuzzlePiece[][] board
move(int w): boolean
1
*
PuzzlePieces
private int face_value

퍼즐 조각

```
public class PuzzlePiece {
    private int face_value;
    public PuzzlePiece(int value) { face_value = value; }
    public int valueOf() { return face_value; }
}
```

퍼즐 판

```
public class SlidePuzzleBoard {
    private PuzzlePiece[][] board;
    int empty_x, empty_y, size;

    private boolean has_face_value(int x, int y, int w) {
        return (0<=x && x<size && 0<=y && y<size && board[x][y].valueOf() == w);
    }
    public boolean move(int w) {
        int x = 0, y = 0;
        boolean found = false;
        int[] neighbour_x = { -1, 0, 0, 1 }; int[] neighbour_y = { 0, -1, 1, 0 };

        for(int i=0; !found && i<4; i++) {
            x = empty_x + neighbour_x[i]; y = empty_y + neighbour_y[i];
            found = has_face_value(x, y, w);
        }
        if(found) {
            board[empty_x][empty_y] = board[x][y];
            board[x][y] = null;
            empty_x = x; empty_y = y;
        }
        return found;
    }}
```

산술 연산과 변수 1번

- 한 사무실의 벽들과 바닥과 천장을 페인트 칠하는데 필요한 페인트의 양을 계산하는 함수를 설계한 후 작성하라. 사무실은 가로와 세로와 높이를 가진다. 1통의 페인트로 500평방미터를 칠할수 있다고 가정하라. 사무실의 가로와 세로와 높이를 키보드를 통해 입력 받아 칠해야 할 총 면적을 계산하여 필요한 페인트의 통수를 계산한다. 페인트의 통수는 실수일 수 있다. 페인트의 통수를 출력하라.

```
public class Paint {
    public static void main(String args[]) {
        Scanner s = new Scanner(System.in);
        int width = s.nextInt();
        int height = s.nextInt();
        int depth = s.nextInt();
        double answer = (width*height*2 +
                         height*depth*2 +
                         width*depth*2 ) / 500.0;

        System.out.println(answer);
    }
}
```

산술 연산과 변수 2번

- 돼지 저금통에 들어 있는 동전들의 총액을 계산하여 출력하는 함수를 설계한 후 작성하라. 저금통에 들어 있는 500원짜리 100원짜리 50원짜리 동전들의 수를 나타내는 정수 값들을 키보드를 통해 입력 받아야 한다.

```
public class Coins {
    public static void main(String args[]) {
        Scanner s = new Scanner(System.in);
        int s500 = s.nextInt();
        int s100 = s.nextInt();
        int s50 = s.nextInt();
        System.out.println(s500*500 + s100*100 + s50*50);
    }
}
```

산술 연산과 변수 3번

- 자동판매기로부터 물건을 사면 거스름돈을 계산하여 알려주는 함수를 설계한 후 작성하라. 자동판매기안에 있는 물건들의 가격은 모두 1,000원 이하이다. 자동판매기는 물건 값으로 1,000원짜리 지폐만을 받는다. 거스름돈은 500원, 100원, 50원, 10원, 5원, 1원짜리 동전들로만 주어야 한다. 거스름돈에 포함된 동전들의 개수는 최소가 되어야 한다.

```
public class Vending {
    public static void main(String args[]) {
          int price; //물건 금액
        int change; //거스름 돈
        System.out.print("물건의 가격을 입력하세요:");
        Scanner price_Class = new Scanner(System.in);
        price = price_Class.nextInt();

        System.out.print("거스름돈 : “+ (change = 1000-price) + "\n");

        System.out.print("500원 짜리 동전갯수 = " + change / 500 + "\n");
        change %= 500;

        System.out.print("100원 짜리동전 갯수 = " + change / 100 + "\n");
        change %= 100;

        System.out.print("50원 짜리 동전 갯수 = " + change / 50 + "\n");
        change %= 50;

        System.out.print("10원 짜리 동전 갯수 = " + change / 10 + "\n");
        change %= 10;

        System.out.print("5원짜리 동전갯수 = " + change / 5 + "\n");
        change %= 5;

        System.out.print("1원 짜리 동전 갯수 = " + change);
    }
}
```

클래스와 객체 이용 1번

다음은 위도와 경도로 표현하는 위치 클래스이다.

```java
public class Location {

  private static double RadiusEarth = 6371.0;    // 지구의 반지름
  private double latitude;    // 위도
  private double longitude;    // 경도
  public Location (double lat, double lon) {
    this.latitude = lat;
    this.longitude = lon;
  }
  public double getLatitude() {
    return this.latitude;
  }
  public double getLongitude() {
    return this.longitude;
  }
  public double distance (Location other) {
    // fill here!
  }
}
```

Java:latest

RUN RESET

위의 클래스에서 distance 함수 내부를 정의하라. 두 위도 $lat1$, $lat2$ 두 경도 $lon1$, $lon2$ 가 주어졌을 때 두 지점 사이의 거리(km)은 다음과 같이 구할 수 있다:

$$R \times \arccos(\cos(\text{rad}(90 - lat1)) \times \cos(\text{rad}(90 - lat2)) + \sin(\text{rad}(90 - lat1)) \times \sin(\text{rad}(90 - lat2)) \times \cos(\text{rad}(lon1 - lon2)))$$

여기서 R은 지구의 반지름 (이미 정의된 RadiusEarth 변수)이고 rad 는 decimal degree 로 받은 위도와 경도를 radian 으로 바꿔주는 함수이다.

```
public class Location {
    private static double RadiusEarth = 6371.0;    // 지구의 반지름
    private double latitude;    // 위도
    private double longitude;    // 경도
    public Location (double lat, double lon) {
        this.latitude = lat;
        this.longitude = lon;
    }
    public double getLatitude() {
        return this.latitude;
    }
    public double getLongitude() {
        return this.longitude;
    }
    public double distance (Location other) {
     double cos1 = Math.cos(Math.toRadians(90 - getLatitude()));
     double cos2 = Math.cos(Math.toRadians(90 - other.getLatitude()));
     double sin1 = Math.sin(Math.toRadians(90 - getLatitude()));
     double sin2 = Math.sin(Math.toRadians(90 - other.getLatitude());
     double cos3 = Math.cos(Math.toRadians(getLongitude() - other.getLongitude()));
     return RadiusEarth * Math.acos(cos1 * cos2 + sin1 * sin2) * cos3;
    }
}
```

클래스와 객체 이용 2번

다음 원뿔의 클래스를 정의하고 부피와 표면적을 계산하는 프로그램을 작성하라.

```java
public class Cone {
  private static double PI = 3.14;
  private double radius;  // 원뿔 밑의 원의 반지름
  private double height;  // 원뿔의 높이
  public Cone(double radius, double height) {
    // fill here
  }
  public double getVolume() {  // 현재 원뿔의 부피 반환
    // fill here
  }
  public double getArea() {  // 현재 원뿔의 표면적 반환
    // fill here
  }
}
```

Java:latest

RUN RESET

원뿔의 부피(volume)와 표면적(area)을 계산하는 공식은 다음과 같다:

$$volume = \frac{1}{3}\pi r^2 h$$

$$area = \pi r\sqrt{r^2 + h^2} + \pi r^2$$

여기서 r 은 원뿔 밑의 원의 반지름, h 는 원뿔의 높이를 의미한다.
π 는 이미 정의된 상수 PI를 사용한다.

```
public class Cone {
    private static double PI = 3.14;
    private double radius;  // 원뿔 밑의 원의 반지름
    private double height;  // 원뿔의 높이
    public Cone(double radius, double height) {
        this.radius = radius;
        this.height = height;
    }
    public double getVolume() {  // 현재 원뿔의 부피 반환
        return (PI*radius*radius*height)/3;
    }
    public double getArea() {  // 현재 원뿔의 표면적 반환
        return (PI * radius *
                Math.sqrt((radius*radius)+(height*height)))
               + (PI*radius*radius);
    }
}
```

클래스와 객체 이용 3번

유리수 클래스 Fraction 을 작성하라. Fraction 객체는 유리수 값을 가지고 있어야 한다.
(힌트: 유리수는 정수 두개로 표현 가능: 분자(numerator)와 분모(denominator))

생성자

- public Fraction(int numerator, int denominator)

다음과 같은 메소드를 통해 객체의 값을 변경할 수 있어야 함.

- public Fraction add(Fraction i): 자기와 유리수 i를 더한 유리수 객체를 반환
- public Fraction add(int i): 자기와 정수 i를 더한 유리수 객체를 반환
- public Fraction negate(): 자기와 부호가 다른 유리수 객체를 반환
- public Fraction inverse(): 자기의 역수인 유리수 객체를 반환
- public Fraction multiply(Fraction i): 자기에 유리수 i를 곱한 객체를 반환
- public Fraction multiply(int i): 자기에 정수 i를 곱한 객체를 반환
- public String toString(): 자기를 문자열로 변환해서 반환. 예, "3/5".
- public void setNumerator(int numIn): 분자 수정
- public void setDenominator(int numIn): 분모 수정

```
public class Fraction {
    private int numerator;
    private int denominator;

   private static int findGCD(int number1, int number2) {
        //base case
        if(number2 == 0) {
            return number1;
        }
        return findGCD(number2, number1 % number2);
    }

    public Fraction(int numerator, int denominator) {
        int gcd = findGCD(numerator, denominator);
        this.numerator = numerator / gcd;
        this.denominator = denominator / gcd;
    }
```

```
public Fraction add(Fraction i) {
    int new_numerator = numerator * i.denominator +
                        i.numerator * denominator;
    int new_denominator = denominator * i.denominator;
    int gcd = findGCD(new_numerator, new_denominator);
    return new Fraction(new_numerator / gcd,
                        new_denominator/ gcd);
}

public Fraction add(int i) {
    int new_numerator = numerator + i * denominator;
    int new_denominator = denominator;
    int gcd = findGCD(new_numerator, new_denominator);
    return new Fraction(new_numerator / gcd,
                        new_denominator/ gcd);
}
```

```
    public Fraction negate() {
        return new Fraction(-numerator, denominator);
    }

    public Fraction inverse() {
        return new Fraction(denominator, numerator);
    }

    public Fraction multiply(Fraction i) {
        int new_numerator = numerator * i.numerator;
        int new_denominator = denominator * i.denominator;
        int gcd = findGCD(new_numerator, new_denominator);
        return new Fraction(new_numerator / gcd,
                            new_denominator/ gcd);
    }
```

```
    public Fraction multiply(int i) {
        int new_numerator = numerator * i;
        int new_denominator = denominator;
        int gcd = findGCD(new_numerator, new_denominator);
        return new Fraction(new_numerator / gcd,
                            new_denominator/ gcd);
    }
    public String toString() {
        return numerator + "/" + denominator;
    }
    public void setNumerator(int numIn) {
        this.numerator = numIn;
    }
    public void setDenominator(int numIn) {
        this.denominator = numIn;
    }
    public int getNumerator() {return numerator;}
    public int getDenominator() {return denominator;}
}
```