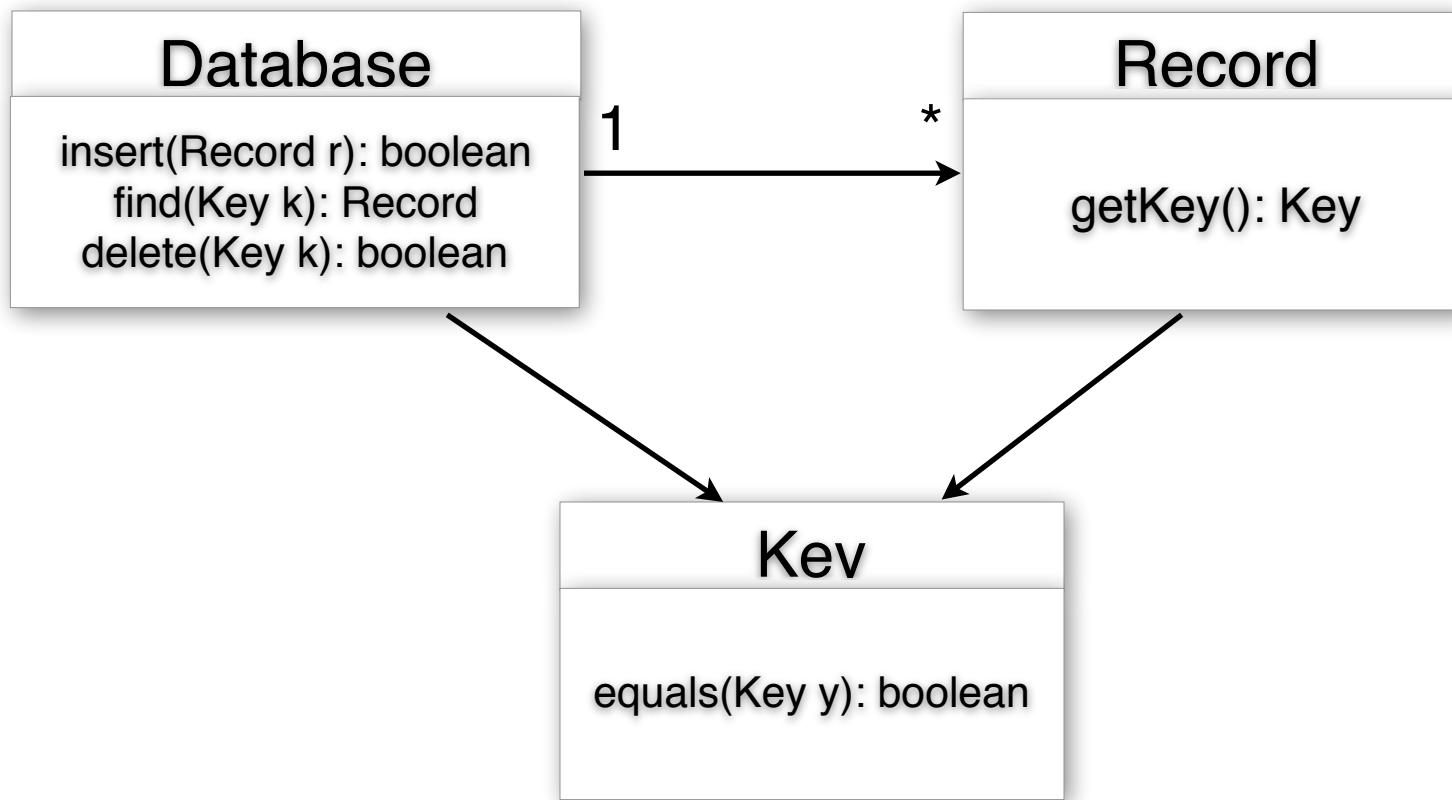


# 8

## 부품구조: 클래스와 메소드2

# 지난시간 데이터베이스 구조



# 한양대 인구 데이터베이스

- 한양대 상주 인원 정보 데이터베이스
- 가능한 상주 인원 신분
  - 학부생 (undergrad student)
  - 대학원생 (grad student)
    - 풀타임 / 파트타임
  - 교직원 (faculty)
    - 풀타임 / 파트타임

# 가능한 솔루션 1 - Person as Record

```
public class Person  
{  
  
    private String name;  
  
    private boolean student;  
  
    private boolean faculty;  
  
    private boolean graduate;  
  
    private boolean fullTime;  
  
    ...  
  
}
```

**Each method becomes:**

```
if (student)  
  
    if (graduate && full-time)  
  
        // some code  
  
    else if (!graduate)  
  
        // more code  
  
    else if (faculty) ...
```

# 가능한 솔루션 1 - 문제점

```
public class Person  
{  
    private String name;  
    private boolean student;  
    private boolean graduate;  
    private boolean fullTime;  
    ...  
}
```

Each method becomes:

```
if (student)  
    if (graduate && full-time)  
        // some code
```

```
else if (!graduate)
```

// more code **Spaghetti Code**

```
else if (faculty) ...
```

# 가능한 솔루션 2

```
public class Student  
{  
    private String name;  
    ...  
}
```

```
public class Faculty  
{  
    private String name;  
    ...  
}
```

# 가능한 솔루션 2

```
public class Student  
{  
    private String firstname;  
    private String lastname;  
    ...  
}
```

```
public class Faculty  
{  
    private String name;  
    ...  
}
```

# 가능한 솔루션 2 - 문제점

```
public class Student  
{  
    private String firstname;  
    private String lastname;  
    ...  
}
```

```
public class Faculty  
{  
    private String name;  
    ...  
}
```

**코드 일관성 유지 어려움!**

# 가능한 솔루션 2 - 문제점

```
public class Student
{
    private String name;
    ...
}
```

```
public class Faculty
{
    private String name;
    ...
}
```

```
public class Database {
    private Person[] base;
    private Student[] base_student;
    private Faculty[] base_faculty;
    private Visitor[] base_visitor;
}
```

모두를 위한 단일 배열을  
정의할 수 없음!

# 우리가 원하는 것

- 공통된 속성 유지
  - 예: 사람의 이름, 생년월일 등은 신분에 상관없이 모두 필요
- 클래스 종류에 따라 각기 고유한 속성은 다른 클래스에 정의
- 모든 객체들을 담을 수 있는 한 종류의 배열 정의

상속 (inheritance)!

```
public class Person  
{  
    private String name;  
    ...  
}
```

```
public class Student  
{  
    private String name;  
    ...  
}
```

```
public class Faculty  
{  
    private String name;  
    ...  
}
```

# “extends”: 물려받는다는 의미

```
public class Person  
{  
    private String name;  
    ...  
}
```

부모 클래스 / super class

```
public class Student extends Person  
{  
    ...  
}
```

자식 클래스 / subclass

```
public class Person  
{  
    private String name;  
    ...  
}
```

## 상속되는 것

- Public 멤버 변수들
- Public 메소드들

## Private 은 상속 안됨

```
public class Student extends Person  
{  
    name 사용 불가!  
    ...  
}
```

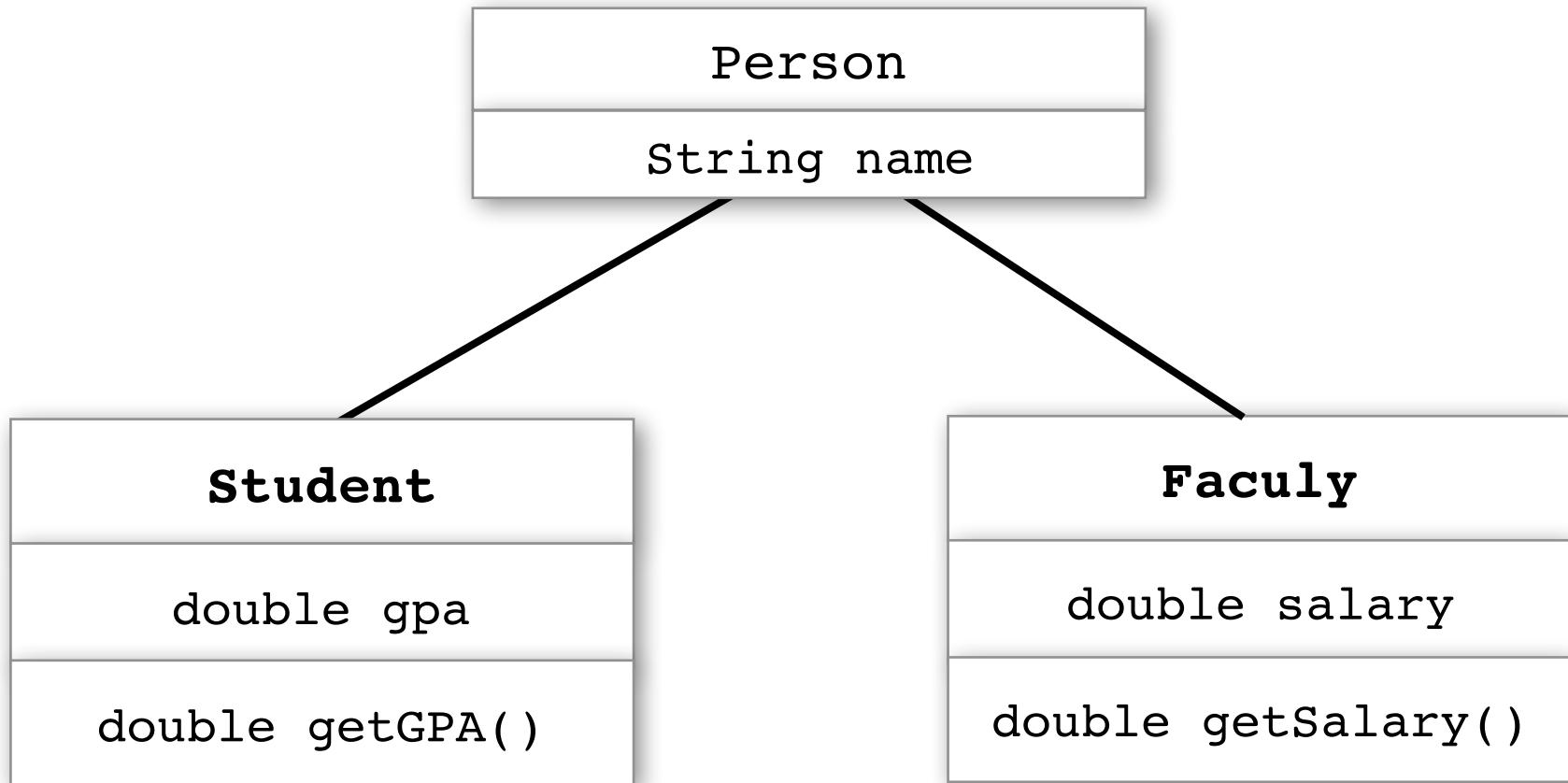
```
public class Person  
{  
    protected String name;  
    ...  
}
```

## Protected 접근 지정자를 사용하면

- 서브 클래스에서 접근 가능
- 그 외 외부 클래스에서는 접근 불가

```
public class Student extends Person  
{  
    name 사용 가능!  
    ...  
}
```

# 클래스 상속 위계 (inheritance hierarchy)



# Is-a 관계

Reference

Person p =

Object

new Person();

Student s =

new Student();

A Person “is-a” Person.

A Student “is-a” Person.

A Student “is-a” Student.

A Person “**is-not-a**” Student.

# Is-a 관계

```
Person[] p = new Person[3];  
p[0] = new Person();  
p[1] = new Student();  
p[2] = new Faculty();
```

**Person** 객체 배열은 **Student**와 **Faculty** 객체도 담을 수 있음.

# Object 클래스

```
public class Person extends Object  
{ ...}
```

- Object 클래스: 모든 클래스의 최상위 클래스
- 어느 클래스도 상속하지 않는 클래스: 자동으로 Object 를 상속.
- 다음 메소드들 포함
  - Boolean equals(Object obj)
  - String toString()
  - Object clone()
  - ...

# 컴파일러의 타입 결정

```
public class Person {
    private String name;
    public String getName() {return name;}
}
```

```
public class Student extends Person {
    private int id;
    public int getID() {return id;}
}
```

```
public class Faculty extends Person {
    private String id;
    public String getID() {return id;}
}
```

○ 다음 중 에러가 나는 지점은?

```
Student s = new Student();
Person s = new Person();
Person s = new Person();
Faculty f = new Faculty();
Object o = new Faculty();
String n = s.getName();
p = s;
int m = p.getID();
f = q;
o = s;
```

# 컴파일러의 타입 결정

```
public class Person {
    private String name;
    public String getName() {return name;}
}
```

```
public class Student extends Person {
    private int id;
    public int getID() {return id;}
}
```

```
public class Faculty extends Person {
    private String id;
    public String getID() {return id;}
}
```

- 다음 중 에러가 나는 지점은?

```
Student s = new Student();
Person s = new Person();
Person s = new Person();
Faculty f = new Faculty();
Object o = new Faculty();
String n = s.getName();
p = s;
int m = p.getID();
f = q;
o = s;
```

# 컴파일러의 타입 결정

```
public class Person {
    private String name;
    public String getName() {return name;}
}
```

```
public class Student extends Person {
    private int id;
    public int getID() {return id;}
}
```

```
public class Faculty extends Person {
    private String id;
    public String getID() {return id;}
}
```

○ 다음 중 에러가 나는 지점은?

```
Student s = new Student();
Person p = new Person();
Person q = new Person();
Faculty f = new Faculty();
Object o = new Faculty();
String n = s.getName();
p = s;
int m = ((Student) p).getID();
f = q;
o = s;
```

# 왜 컴파일러는 p가 학생임을 모를까?

- 일반적으로 객체의 해당 클래스 (참조 타입)을 실행 전에 알기 불가능

```
Person p;
if (사용자 입력 값에 대한 질문) {
    p = new Student();
}
else {
    p = new Faculty();
}
p.getID() // 어느 getID 함수가 호출되지?
```

- 개발자가 타입 강제 변환 (casting)을 통해 컴파일러에게 호출되는 객체의 클래스 정보를 알려줘야 함

# 접근자 (Visibility Modifier)

- 
- Diagram illustrating the hierarchy of Java visibility modifiers:
- Less Restrictive (Top): **Public**
  - Protected**
  - Package**
  - Private**
  - More Restrictive (Bottom): **Less Restricted**
- A vertical arrow points downwards from the less restrictive to the more restrictive levels.
- **public**: 어떠한 클래스에서도 접근 가능
  - **protected**: 자기 자신 / 같은 패키지 안의 클래스 / 자식 클래스
  - **package**: 자기 자신 / 같은 패키지 안의 클래스
  - **private**: 동일 클래스
  - 대개 **public**, **private** 만 사용

# 상속에 관한 컴파일러 규칙

```
public class Person  
{  
    private String name;  
}
```



```
public class Person extends Object  
{  
    private String name;  
}
```

## Rule #1

No superclass? Compiler  
inserts:  
**extends Object**

# 상속에 관한 컴파일러 규칙

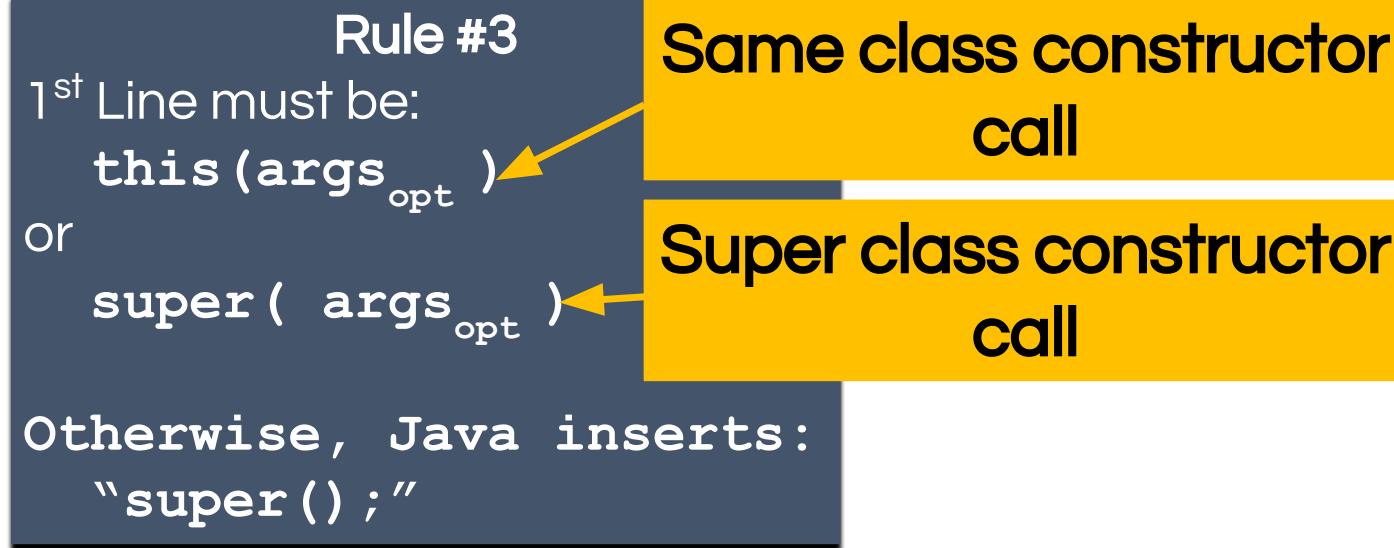
```
public class Person  
{  
    private String name;  
}
```



```
public class Person extends Object  
{  
    private String name;  
  
    public Person() {  
    }  
}
```

**Rule #2**  
No constructor? Java  
gives you one for you.

# 상속에 관한 컴파일러 규칙



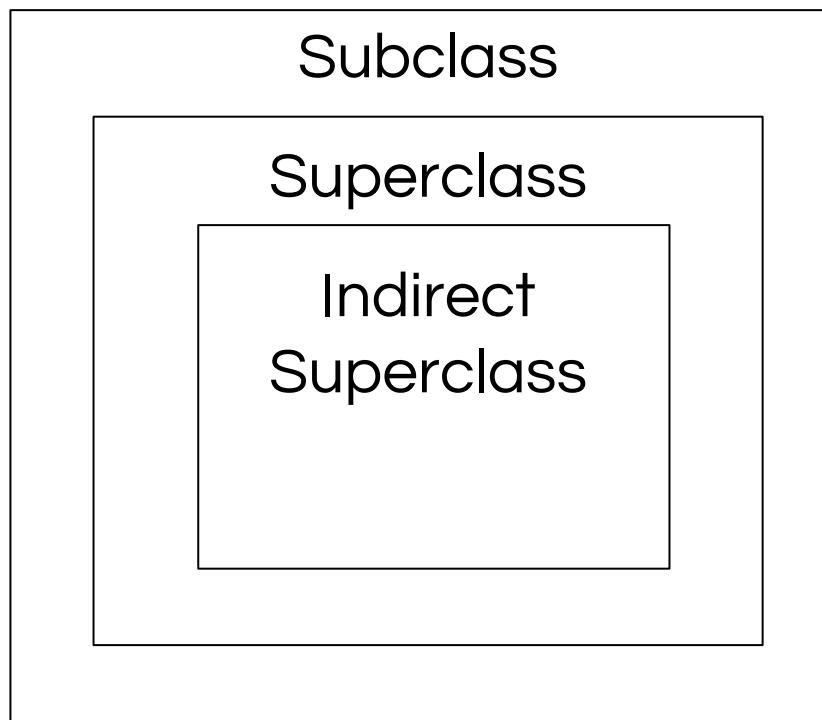
```
public class Person
{
    private String name;
}
```

```
public class Person extends Object
{
    private String name;

    public Person() {
        super();
    }
}
```

# 클래스 생성 시 일어나는 일

```
Student s = new Student();
```



Student()

(3)

Person()

(2)

Object()

(1)

# 속성 정의는 부모 객체 생성 후에

```
public class Person extends Object
{
    private String name;
    public Person( String n ) {
        this.name = n;
        super();
    }
}
```

```
public class Person extends Object
{
    private String name;
    public Person( String n ) {
        super();
        this.name = n;
    }
}
```

**ERROR!**

**super () has to be the  
first line!**

# 다른 대안

```
public class Student extends Person
{
    public Student( String n )
    {
        super(n);
    }

    public Student()
    {
        super( "Student" );
    }
}
```

# 문제 1

```
public class Person {  
    private String name;  
  
    public Person( String n ) {  
        this.name = n;  
        System.out.print("#1 ");  
    }  
}
```

```
public class Student extends Person {  
    public Student () {  
        this("Student");  
        System.out.print("#2 ");  
    }  
    public Student( String n ) {  
        super(n);  
        System.out.print("#3 ");  
    }  
}
```

- Student s = new Student();  
의 실행결과는?  
(1) #1 #2 #3  
(2) #1 #3 #2  
(3) #3 #2 #1  
(4) #3 #1 #2

# 기본생성자 삽입은 아무 생성자도 없을때만

```
public class Person {
    private String name;

    public Person( String n ) {
        super();
        this.name = n;
    }

    public void setName( String n ) {
        this.name = n;
    }
}
```

```
public class Student extends Person {
    // changed for example
    public Student () {
        this.setName("Student");
    }
}
```

- Student()에서 앞서 컴파일러 규칙#3에 의해 자동으로 super() 삽입
- 인자를 받는 다른 생성자 함수가 이미 존재. 그러므로 Person() 삽입 X
- Student()에서 super() 가 호출되었을 때 Person() 찾음 -> 없음. 에러!

# 메소드 여럿정의 (overloading) vs. 재정의 (overriding)

- 여럿정의: 한 클래스에 동일한 이름 (그러나 다른 매개변수)의 메소드가 여러개 정의됨
- 재정의:
  - 자식 클래스가 부모 클래스가 갖고 있는 것과 이름과 매개변수 모두 동일한 메소드를 가짐
  - 상속받으면서 메소드를 정의하면 재정의된다!
  - 이 경우, 상위클래스(super class), 즉, 물려 주는 쪽의 메소드는 사용되지 않고, 하위 클래스(subclass)의 재정의된 메소드가 사용된다.

# 메소드 재정의

```
public class Person {
    private String name;
    public Person(String n) {name=n; }
}
```

```
Person p = new Person("Tim");
System.out.println( p );
```

```
public class Person {
    private String name;
    public Person(String n) { name=n; }
    public toString() { return name; }
}
```

```
Person p = new Person("Tim");
System.out.println( p );
```

○ 출력:

**Person@3343c8b3**

(Object.toString 호출)

○ 출력:

**Tim**

println 이 자동으로 toString() 호출

# 메소드 재정의

```
public class Student extends Person {  
    private int studentID;  
  
    public int getSID() {  
        return studentID;  
    }  
  
    public String toString() {  
        return this.getSID() + ":" +  
            this.getName();  
    }  
}
```

What if Person  
changes?

Goal:  
SID: Person info

# 메소드 재정의

```
public class Student extends Person {  
    private int studentID;  
  
    public int getSID() {  
        return studentID;  
    }  
  
    public String toString() {  
        return this.getSID() + ":" +  
            super.toString();  
    }  
}
```

**“super” refers to  
superclass**

# 상속 예제: 그래픽 사용자 인터페이스 (GUI) 만들기

- 바닥부터 내가 만들면
  - 내 맘대로 할 수 있겠지만
  - 창을 그리기 위해 선도 긋고, 색칠도 하고
  - 마우스 클릭했을 때 어떻게 해야할지도 결정해야 하고
  - 할 일이 너무 많다!
- 라이브러리를 이용하자!
  - javax.swing 패키지
  - 남이 작성한 것인니까, 이 패키지에서 창을 그리는 방법을 이해해야 한다.

# 프레임과 패널

- javax.swing 패키지에서 창은 JFrame 객체이다.
- 그 안에 JPanel 객체를 넣어서 다양한 그림을 그릴 수 있다.

JFrame

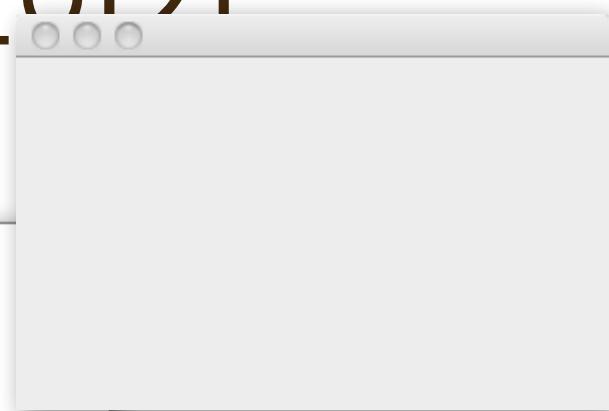
JPanel

# 프레임 만들기

- 프레임은 창의 바탕
- 만들기
  - new JFrame()
- 반드시 보여 주어야 한다.
  - <프레임>.setVisible(true);
- 크기 지정 가능
  - <프레임>.setSize(<가로크기>, <세로크기>);
  - 크기는 점의 수 (pixels)

# 빈 프레임 만들어서 보이기

```
import javax.swing.*;  
/* 빈 프레임 보이기 */  
  
public class FrameTest2  
{ public static void main(String[] args)  
{ JFrame f = new JFrame();  
f.setSize(300, 200);  
    // 크기: 가로, 세로 점 수  
f.setVisible(true);  
    // 보여줘!  
}  
}
```



# 패널 (JPanel)

- 패널은 그림을 그릴 수 있는 객체
- 패널 만들기
  - new JPanel()
- 패널은 프레임에 부착되어야 한다.
  - <프레임>.getContentPane().add(<패널>);

# 패널과 화가

- 패널은 화가가 떨려 있다.

  - JPanel은 메소드 paintComponent가 있다.

- 화가는 때때로 다시 그린다.

  - 필요할 때 창은 자기 패널의 paintComponent를 호출 한다.

- 화가는 그림 도구가 필요하다.

  - 호출시 그림도구인 Graphics 객체를 준다.

JPanel

---

paintComponent(Graphics g)

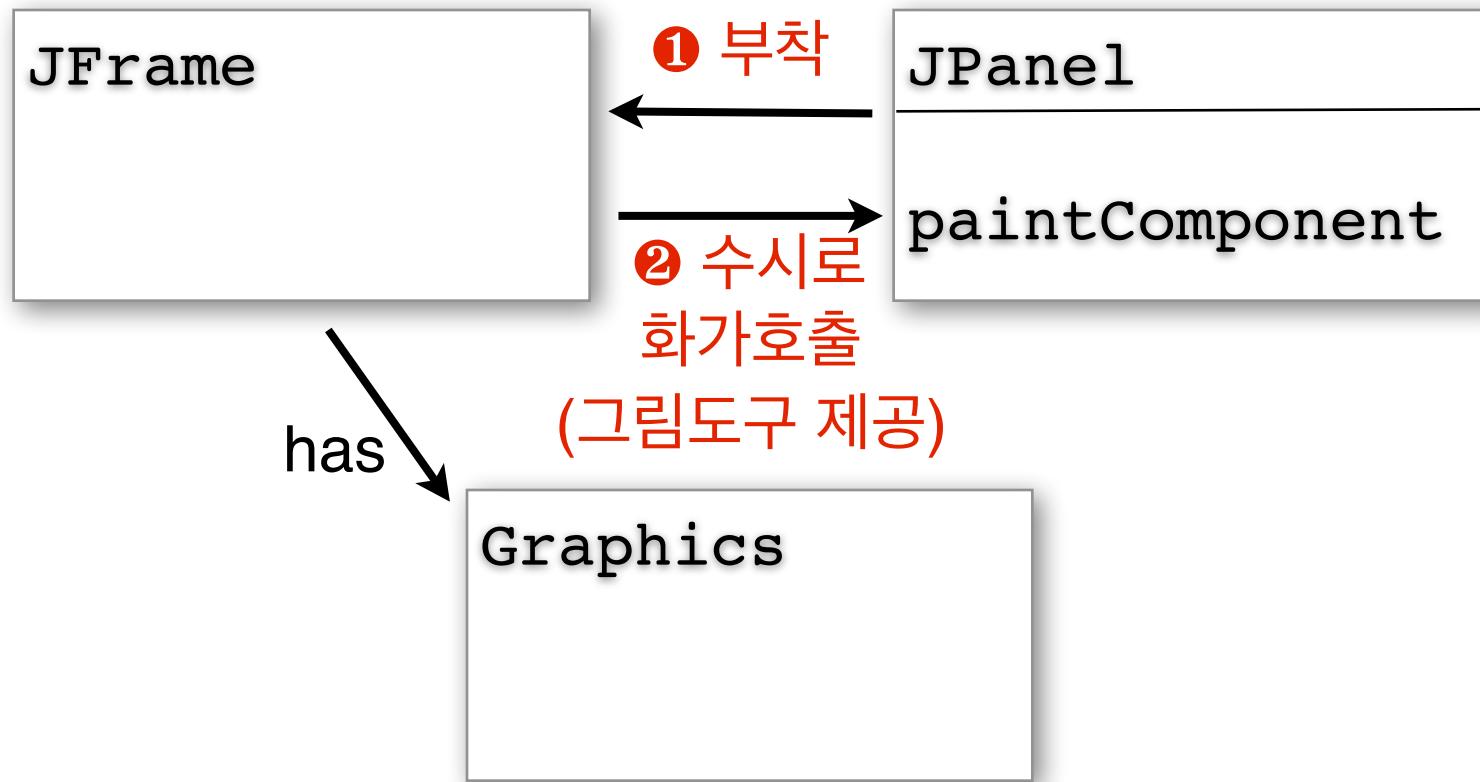
paintComponent

JPanel



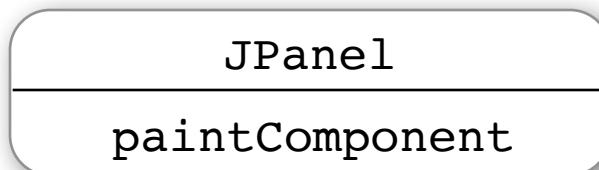
Graphics g

# 프레임, 패널, 화가의 구동 원리



# 내가 원하는 패널을 만드려면

- 화가만 바꾸면 된다.
  - 즉, paintComponent 메소드만 교체해야 한다.
- 방법: 상속 (inheritance)과 메소드 재정의 (overriding)



# 내가 원하는 패널, 상속으로 만들기

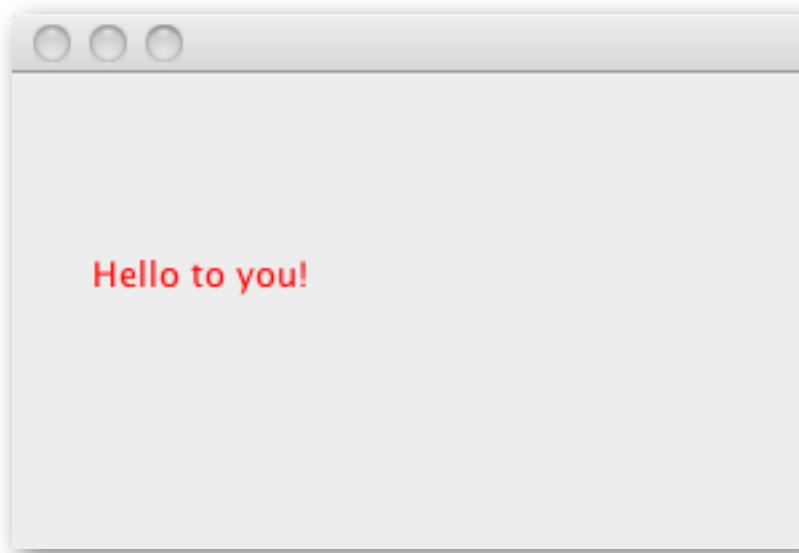
```
import java.awt.*;
import javax.swing.*;

public class MyPanel extends JPanel {
    public void paintComponent(Graphics g) { 그리기 코드 }
}
```

- 화가만 교체된다!
- 나머지는 기존 패널과 다를 게 없으므로 프레임에 그대로 넣어서 사용 가능!

# 예제: Hello to you!

- 다음과 같은 창을 만들어라.



# 패널: Hello to you!

```
import java.awt.*; Graphics  
import javax.swing.*; JPanel
```

```
public class TestPanel extends JPanel {
```

Graphics는 다양한 그리기 도구 제공

```
    public void paintComponent(Graphics g) {
```

```
        g.setColor(Color.red); // 빨간색
```

```
        g.drawString("Hello to you!", 30, 80); // 글씨
```

```
}
```

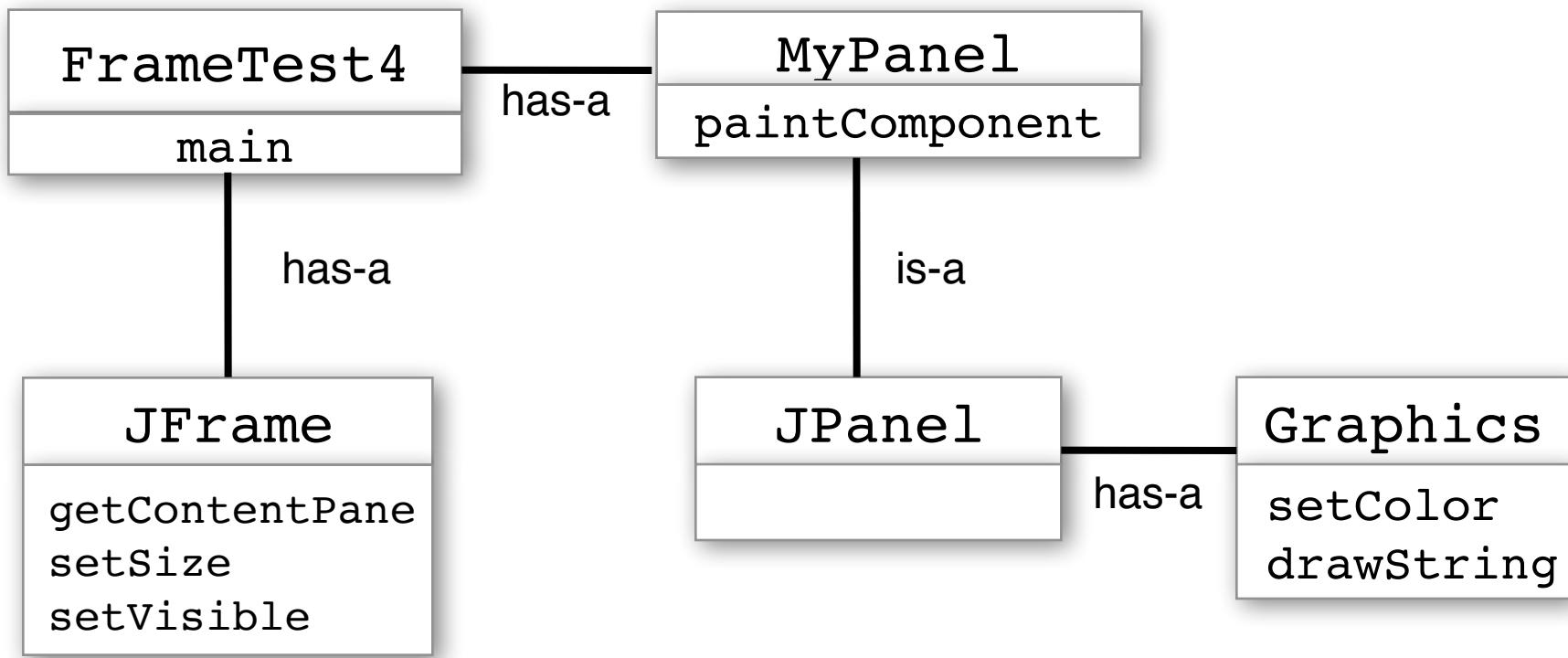
글씨를 쓰는 가로, 세로 위치

```
}
```

# 구동: Hello to you!

```
import javax.swing.*;  
  
public class FrameTest3 {  
  
    public static void main(String[ ] args) {  
        TestPanel p = new TestPanel(); // 내 패널 만들고  
        JFrame f = new JFrame(); // 프레임 만들고  
        f.getContentPane().add(p); // 패널을 프레임에 부착  
        f.setSize(300, 200); // 프레임 크기 지정  
        f.setVisible(true); // 프레임 보이기  
    }  
}
```

# 예제 프로그램의 소프트웨어 구조

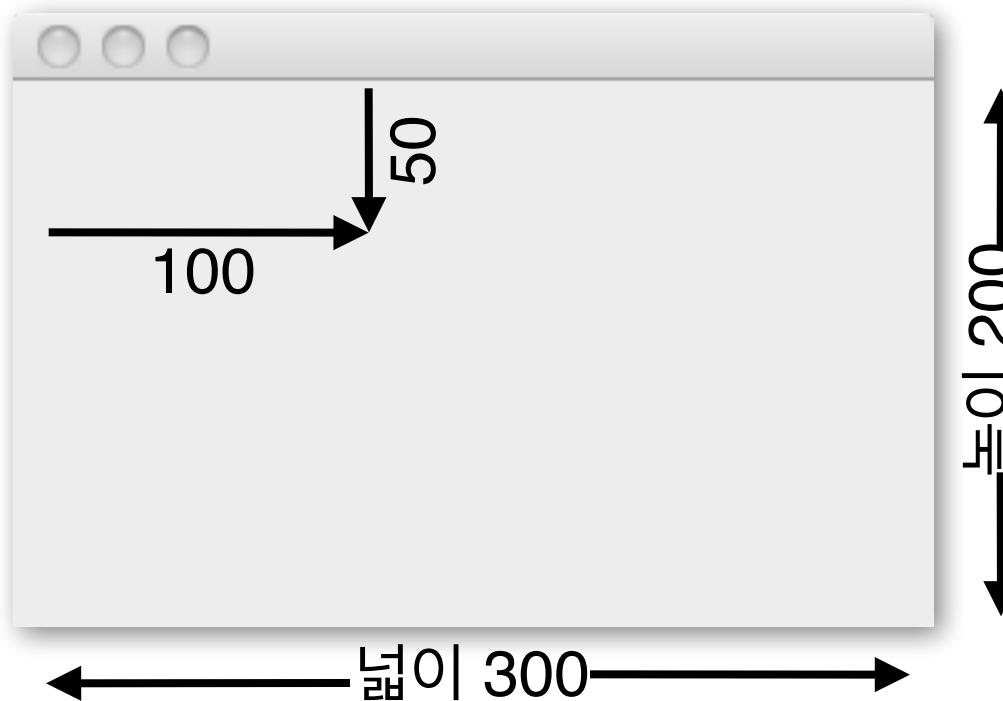


# 화가의 도구 Graphics

- Graphics 는 다양한 도구를 지원한다.
  - `setColor(c)`
  - `drawLine(x1,y1,x2,y2)`
  - `drawString(s,x,y)`
  - `drawRect/fillRect/drawOval/  
fillOval(x,y,dx,dy)`
  - `drawArc/fillArc(x,y,dx,dy,a,da)`
  - `paintImage(i,x,y,ob)`
  - 등등
- 몇 가지만 살펴보고 나머지는 설명서 참조

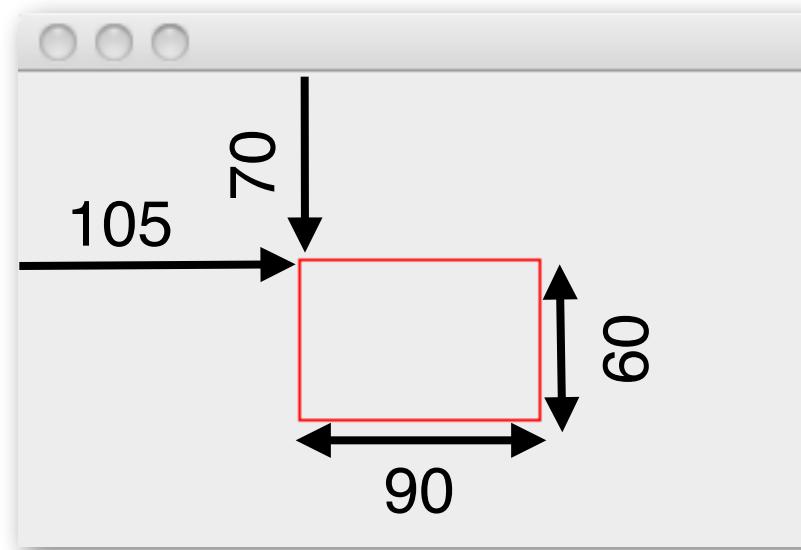
# 위치 및 크기의 단위: 점 수 (Pixels)

- 크기가 (300,200) 일 때 (100,50) 위치는



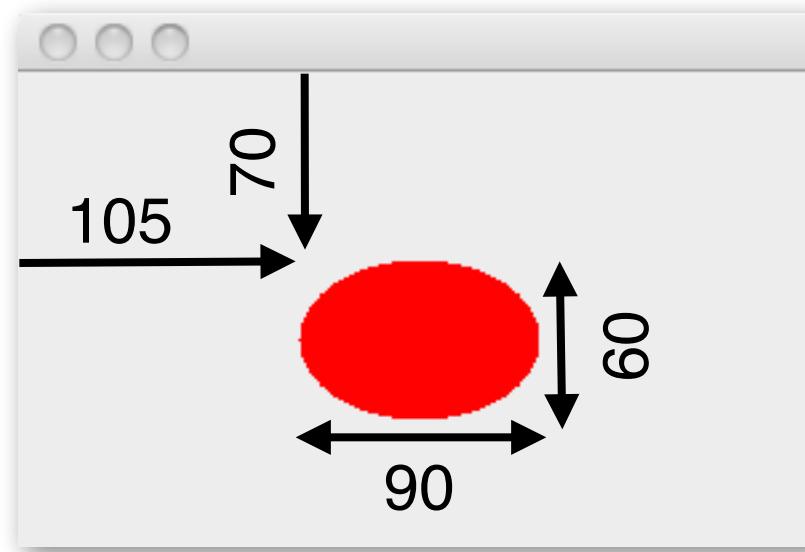
# 사각형 그리기

- `g.drawRect(105, 70, 90, 60)`



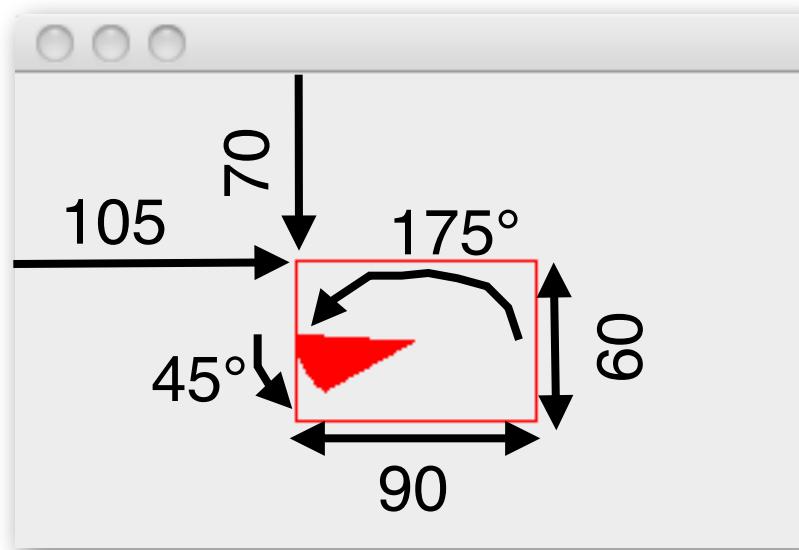
# 타원 그리기

- `g.fillOval(105, 70, 90, 60)`



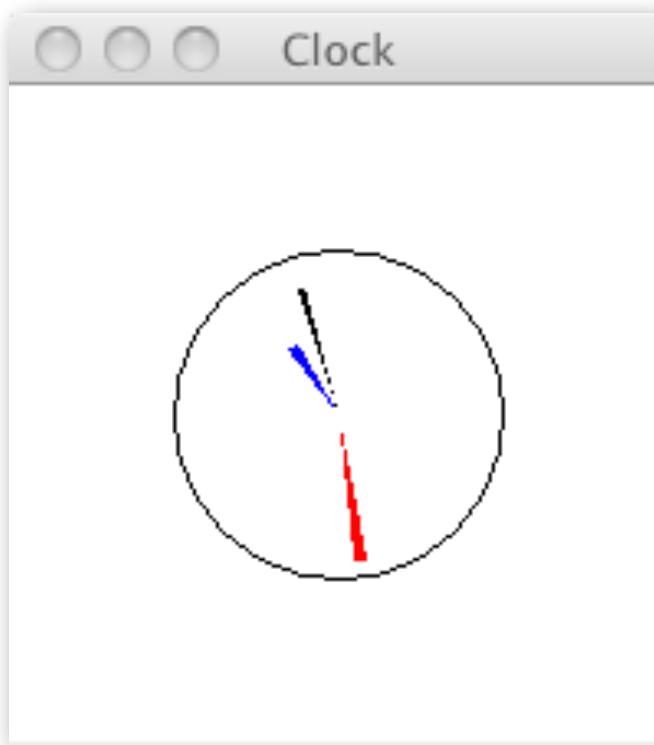
# 채운 호 또는 타원 조각 그리기

- `g.fillArc(105, 70, 90, 60, 175, 45)`



# 예제: 시계

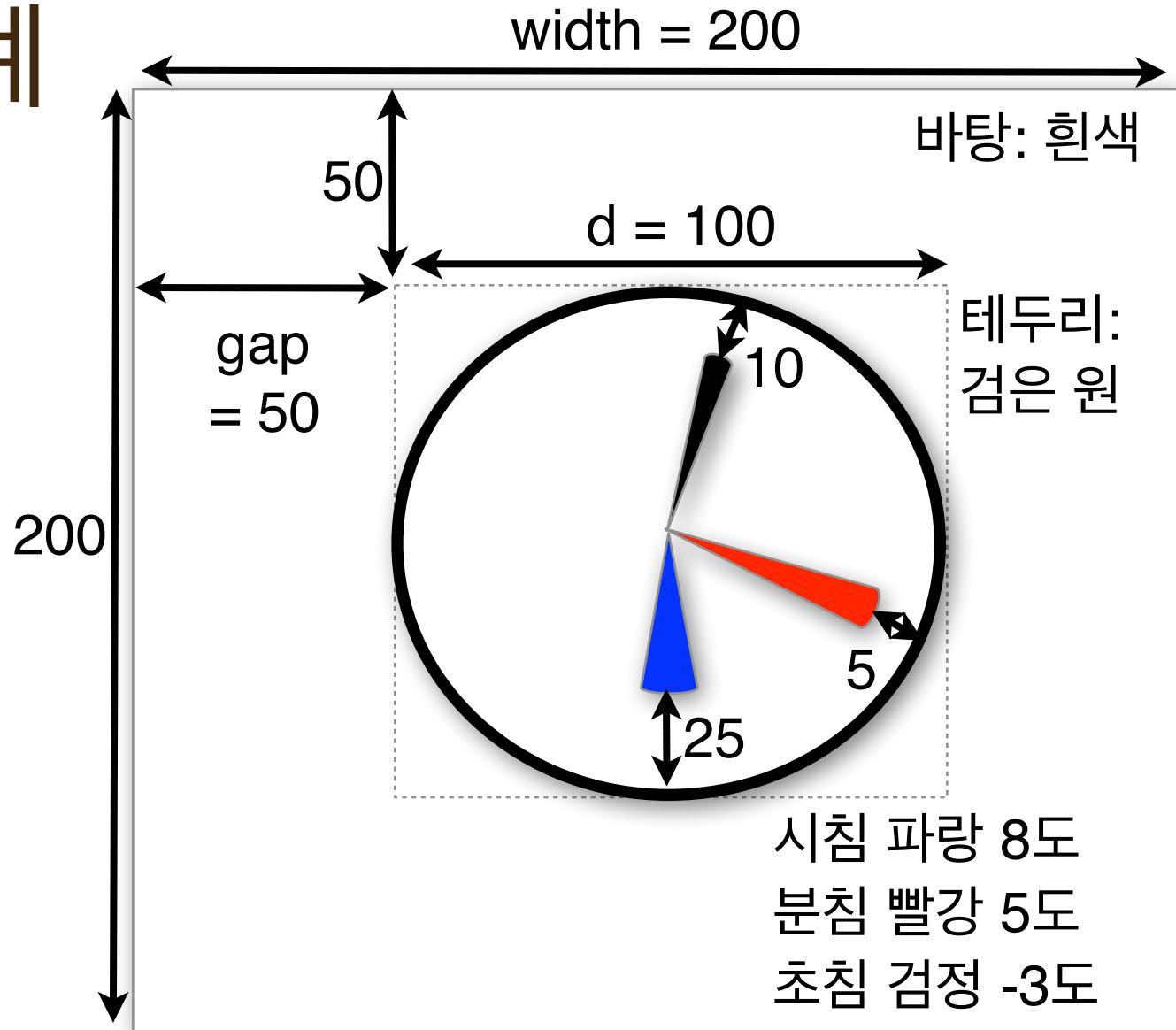
- 아날로그 시계를 보여주는 창을 만들자!



# 예제: 패널 생성자에서 프레임 생성

```
public class ClockWriter extends JPanel {  
    public ClockWriter() {  
        int width = 200;  
        JFrame f = new JFrame();          // 프레임 생성  
        f.getContentPane().add(this);     // 자신을 프레임에 부착  
        f.setTitle("Clock");           // 프레임 제목 설정  
        f.setSize(width, width);        // 프레임 크기 설정  
        f.setVisible(true);            // 프레임 보여줘!  
    }  
    public void paintComponent(Graphics g) { ... }  
    public static void main(String[] args) {  
        new ClockWriter();             // 객체 생성  
    }  
}
```

# 시계 설계



# 시분초침의 각도 계산

```
...
GregorianCalendar time = new GregorianCalendar();
int s_angle = 90 - (time.get(Calendar.SECOND) * 6);
int m_angle = 90 - (time.get(Calendar.MINUTE) * 6);
int h_angle = 90 - (time.get(Calendar.HOUR) * 30);
...
...
```

- GreogianCalender.get 을 사용하면 시분초를 각각 얻을 수 있음
- 각도: 3시 방향이 0도. 양수 - 반시계 방향, 음수 - 시계 방향

# 시계 그리기

```
int width = 200;
int gap = 50;
int d = 100;
g.setColor(Color.white);           // 바닥 그리기
g.fillRect(0, 0, width, width);
g.setColor(Color.black);          // 시계 원 그리기
g.drawOval(gap, gap, d, d);
g.setColor(Color.blue);           // 시침, 분침, 초침 그리기
g.fillArc(gap+25, gap+25, d-50, d-50, h_angle, 8);
g.setColor(Color.red);
g.fillArc(gap+ 5, gap+ 5, d-10, d-10, m_angle, 5);
g.setColor(Color.black);
g.fillArc(gap+10, gap+10, d-20, d-20, s_angle, -3);
```

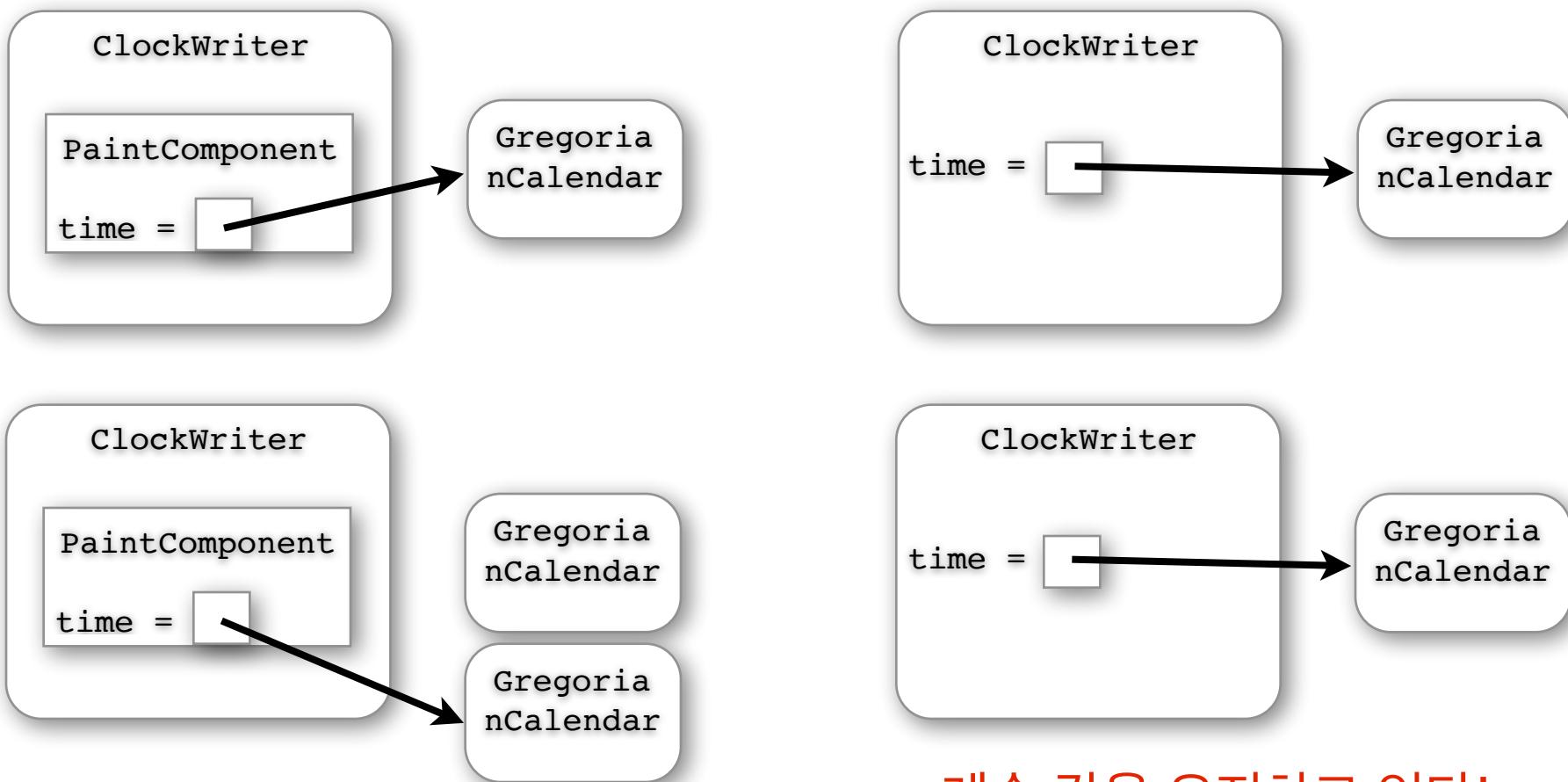
# 메모리에 상주

어떤 일이 발생할까?

```
public class ClockWriter extends JPanel
{
    public ClockWriter() {
        int width = 200;
        ...
    }
    public void paintComponent(Graphics g)
    {
        int width = 200;
        GregorianCalendar time =
            new GregorianCalendar();
        ...
    }
    ...
}
```

```
public class ClockWriter extends JPanel
{
    int width = 200;
    GregorianCalendar time =
        new GregorianCalendar();
    public ClockWriter() {
        ...
    }
    public void paintComponent(Graphics g)
    {
        ...
    }
    ...
}
```

# 원래 코드의 경우 vs 수정된 코드의 경우



계속 값을 유지하고 있다!

# 필드변수 의미구조

- 필드 변수 저장소는 **객체가 생성될 때** 같이 생성된다.
  - 초기값이 있는 경우, 저장소가 생성되면서 초기화된다.
  - 필드 변수는 객체 안에 존재한다.
  - 객체의 생성자에서 조차 접근 가능하다.
- 필드 변수는 객체 안에 (**쓰이지 않을 때에도**) **상주한다**.
  - 메소드가 수행되지 않아도 상주하고 값을 유지한다.
  - 나중에 메소드가 수행될 때 사용될 수 있기 때문이다.

# 필드변수 생성시 초기값이 없는 경우

- 0에 해당하는 값으로 초기화
- int, char: 0
- float, double: 0.0
- boolean: false
- 객체 타입: null

# 조건문 실습1

- 직원의 연봉과 근무평가등급을 받아 새 연봉을 계산하여 반환하는 함수를 작성하세요. 직원의 근무평가등급은 우수 보통과 불량 중 하나입니다.
  - 1(우수) 등급을 받은 직원은 연봉이 6% 인상
  - 2(보통) 등급을 받은 직원은 연봉이 4% 인상
  - 3(불량) 등급을 받은 직원은 연봉이 2% 인상

아래 Employee 클래스 코드에서 applyIncreaseRate 함수의 내용물을. 작성하세요.

예제 모범 출력은 아래 main 함수에 주석으로 있습니다.

```
public class Employee
{
    private Double salary; // 연봉
    private int evalGrade; // 근무평가등급

    public Employee(Double salary, int evalGrade) {
        this.salary = salary;
        this.evalGrade = evalGrade;
    }
    ...

    public Double applyIncreaseRate(){
        if (evalGrade == 1) {
            salary *= (1.06);
        }
        else if (evalGrade == 2) {
            salary *= (1.04);
        }
        else {
            salary *= (1.02);
        }
        return salary;
    }
}
```

# 조건문 실습2

주어진 연도가 윤년인지 아닌지를 출력하는 프로그램을 설계하고 작성하세요.

**윤년:** 연도가 4로 나누어지고 100으로 나누어지지 않는 연도. 단 400으로 나누어지면 윤년

## Example

1796년은 4로 나누어지고 100으로 나누어지지 않으므로 윤년임  
그러나 400으로 나누어지면 그 연도는 윤년임  
2000년은 윤년이나 1800년은 윤년이 아님

아래의 빠대 코드에서 isLeapYear 함수 내용물을 작성하세요.

isLeapYear 함수는 연도를 정수로 받아서 윤년이면 true, 아니면 false를 반환하는 함수입니다.

```
public class LeapYear
{
    public static boolean isLeapYear(int year) {
        boolean cond1 = year%4==0 && year%100 !=0;
        boolean cond2 = year%400 == 0;
        return (cond1 || cond2);
    }
}
```

# 조건문 실습3

아래는 수업시간에 다룬 두 계좌를 관리하는 프로그램입니다. 유일한 차이점은 javax.swing 을 이용한 그래픽 유저인터페이스를 사용하지 않고 콘솔 입출력을 사용하고 있다는 점입니다.

아래 코드에는 다음 클래스들의 정의가 이미 되어있습니다.

- BankAccount
- BankReader
- BankWriter
- AccountController2
- AccountManger2

이 두 계좌 관리 프로그램에 다음 기능을 추가하세요.

- "T 금액" 을 명령어로 입력받아서 현재 활성화 된 계좌에서 비활성 계좌로 금액만큼 이체
- 참고로, 이체는 영어로 transfer 이다.
- "I 이율" 을 명령어로 입력받아서, 현재 활성화 된 계좌의 금액을 이율만큼 증가
- 이율은 0~1 범위의 실수이다.
- 이율은 소수점 두자리까지만 인정한다. 예, 0.055 => 0.05
- 이율만큼 증가한다는 것은 (금액 \* 이율) 만큼 증가시킨다는 뜻이다.
- 참고로, 이율은 영어로 interest 이다.

```
case 'T':  
// 'T' 금액', 활성 계좌에서 비활성 계좌로 금액만큼 이체  
{  
    System.out.println("Transfer");  
    int amount = reader.readAmount();  
    if(account.withdraw(amount)) {  
        writer.setTransaction("transfer $", amount);  
        if(account == primary_account) {  
            secondary_account.deposit(amount);  
        }  
        else primary_account.deposit(amount);  
        break;  
    }  
    else writer.setTransaction("transfer error", amount);  
    break;  
}
```

```
case 'I':  
{  
    System.out.println("Interest");  
    int amount = reader.readAmount(); // amount = 이율 * 100  
    if (0 <= amount && amount <= 100) {  
        writer.setTransaction("Interest % ", amount);  
        account.deposit((account.getBalance() * amount) / 100);  
    }  
    else System.out.println("Interest error");  
    break;  
}
```

혹은... (뒤에 계속)

```
class BankReader {  
    private String input_line = "";  
    public int readAmount(){  
        int answer = 0;  
        String s = input_line.substring(1, input_line.length());  
        if(s.length() > 0) {  
            double dollars_cents = new Double(s).doubleValue();  
            answer = (int)(dollars_cents*100);  
        } else  
            System.out.println("invalid command - input amount: 0");  
        return answer;  
    }  
    public double readAmount_interest() {  
        double answer = 0;  
        String s = input_line.substring(1, input_line.length());  
        if(s.length() > 0) {  
            double dollars_cents = new Double(s).doubleValue();  
            answer = (double)(dollars_cents);  
        } else  
            System.out.println("invalid command - input amount: 0");  
        return answer;  
    }  
}
```

```
...
case 'I':
{
    System.out.println("Interest");
    double amount = reader.readAmount_interest();
    if (0 <= amount && amount <= 1.0) {
        writer.setTransaction("Interest % ", amount);
        account.deposit(account.getBalance() * amount);
    }
    else System.out.println("Interest error");
    break;
}
```

# 반복문 실습1

- 10진수를 입력받아 2진수로 변환하는 프로그램을 작성하세요.

```
class DecimalToBinary
{
    public static String toBinary(int decimal) {
        if (decimal == 0) return "0";
        else {
            int quotient = decimal / 2;
            int remainder = decimal % 2;
            if (quotient == 0) return String.valueOf(remainder);
            else return toBinary(quotient) + remainder;
        }
    }
}
```

# 반복문 실습1

- 원리:

$$\text{decimal} = a \times 2^n + b \times 2^{n-1} + \dots + y \times 2^1 + z \times 2^0$$

$$\text{decimal} / 2 = a \times 2^{n-1} + b \times 2^{n-2} + \dots + y \times 2^0$$

$$\text{decimal \% 2} = z$$

그러므로, ( $\text{decimal} / 2$ )의 2진수화 결과 뒤에 ‘z’를 붙이면  
 $\text{decimal}$  의 2진수화 결과와 동일.

# 반복문 실습2

- 입력: 두 개의 문자열. 빈 문자열도 문자열  
두 개의 문자열을 받아서 각 문자열의 문자들을 차례대로 사이사이에 끼워주는 함수 zipper 를 작성하세요.  
빈 문자열은 무시됩니다.

```
class StringZipper
{
    public static String zipper(String str1, String str2) {
        if (str1.length() == 0) return str2;
        else if (str2.length() == 0) return str1;
        else {
            String zip_first_chars = "" + str1.charAt(0) + str2.charAt(0);
            return zip_first_chars + zipper(str1.substring(1), str2.substring(1));
        }
    }
}
```

# 반복문 실습3

상근이와 선영이가 다른 사람들이 남매간의 대화를 듣는 것을 방지하기 위해서 대화를 서로 암호화 하기로 했다. 그래서 다음과 같은 대화를 했다.

- 상근: 그냥 간단히 암호화 하자. A를 1이라고 하고, B는 2로, 그리고 Z는 26으로 하는거야.
  - 선영: 그럼 안돼. 만약, "BEAN"을 암호화하면 25114가 나오는데, 이걸 다시 글자로 바꾸는 방법은 여러가지가 있어.
  - 상근: 그렇네. 25114를 다시 영어로 바꾸면, "BEAAD", "YAAD", "YAN", "YKD", "BEKD", "BEAN" 총 6가지가 나오는데, BEAN이 맞는 단어라는건 쉽게 알수 있잖아?
  - 선영: 예가 적절하지 않았네 ㅠㅠ 만약 내가 500자리 글자를 암호화 했다고 해봐. 그 때는 나올 수 있는 해석이 정말 많은데, 그걸 언제 다해봐?
  - 상근: 얼마나 많은데?
  - 선영: 구해보자!

어떤 암호가 주어졌을 때, 그 암호의 해석이 몇 가지가 나올 수 있는지 구하는 프로그램을 작성 하시오.

```
public static long getPossible(String code) {  
    long result = 0;  
    if (code.length() == 0 || code.length() == 1) {  
        return 1;  
    }  
    else {  
        int code_num = Integer.parseInt(code.substring(0, 2));  
        long possible_1 = getPossible(code.substring(1));  
        long possible_2 = getPossible(code.substring(2));  
        if (code_num < 27 && code_num % 10 != 0) {  
            result += possible_1 + possible_2;  
        }  
        else {  
            result += possible_1;  
        }  
        return result;  
    }  
}
```