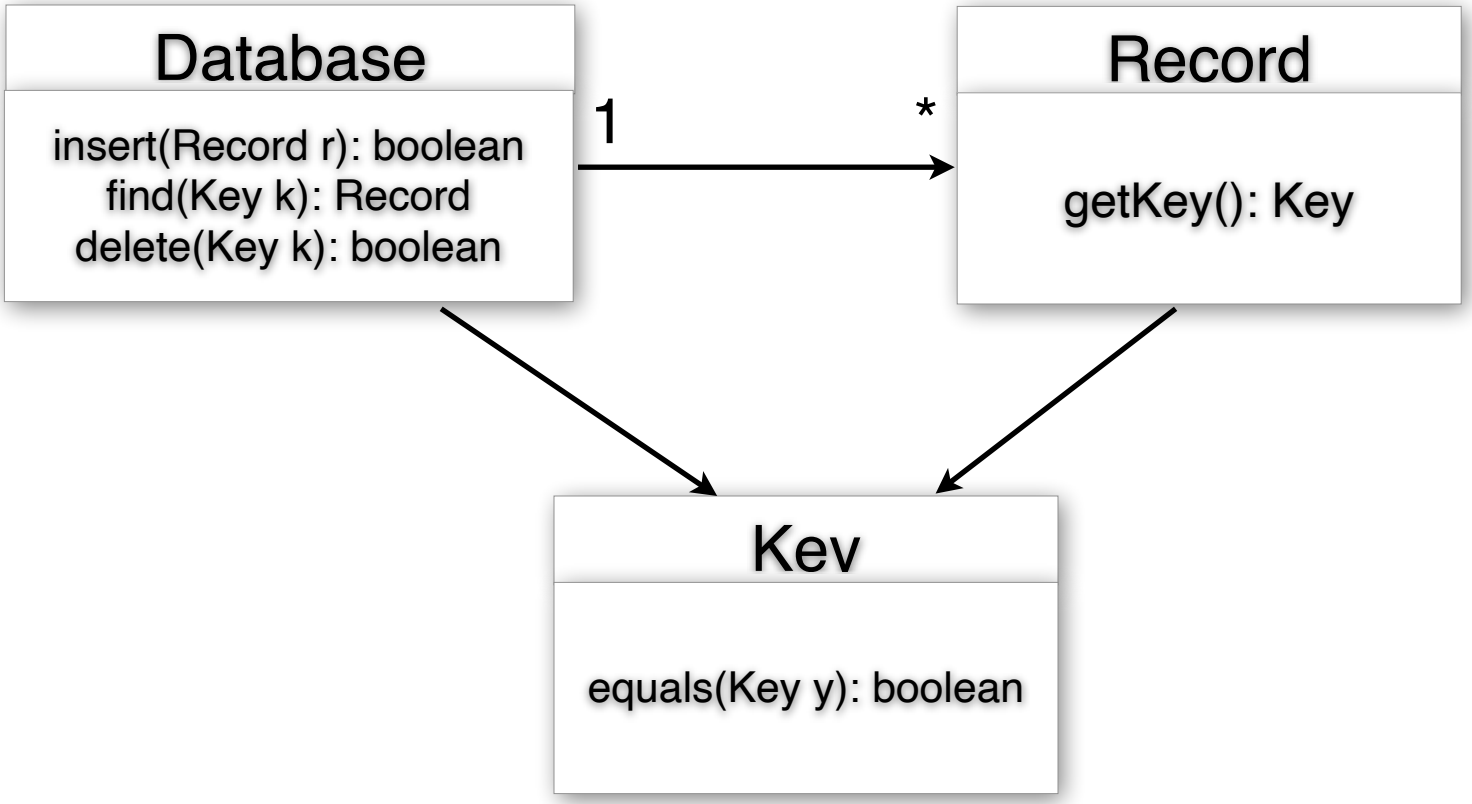
8
부품구조: 클래스와 메소드2

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;
...
}
```

Each method becomes:

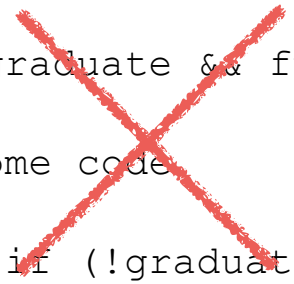
```
if (student)
 // some code
else if (faculty)
 // some code
```

가능한 솔루션 1 - 문제점

```
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) …
```



Spaghetti Code

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;
…

}
```

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

}
```

상속되는 것

- Public 멤버 변수들
- Public 메소드들

Private 은 상속 안됨

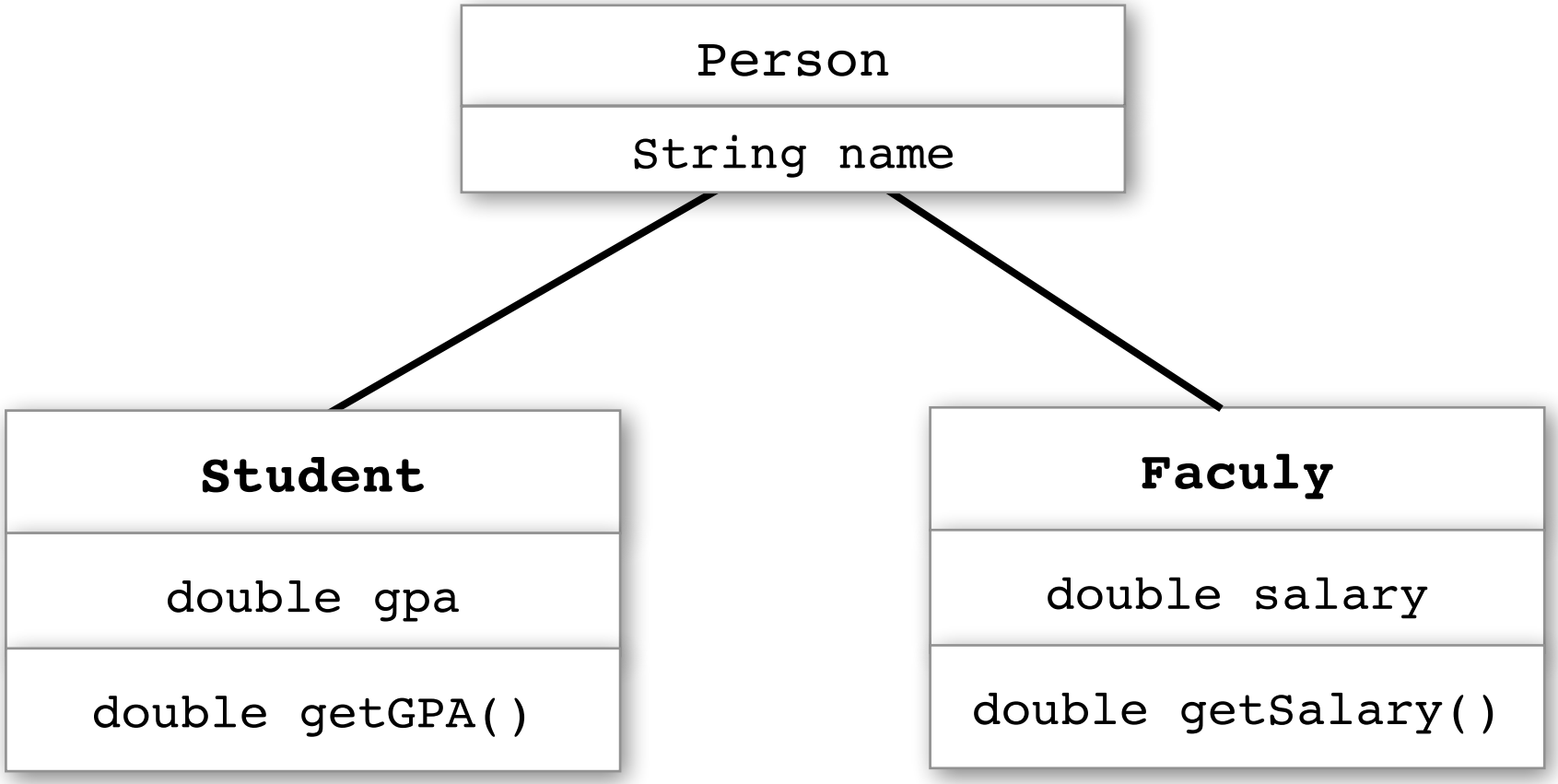
```
public class Person
{
 protected String name;
…

}
```

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

}
```

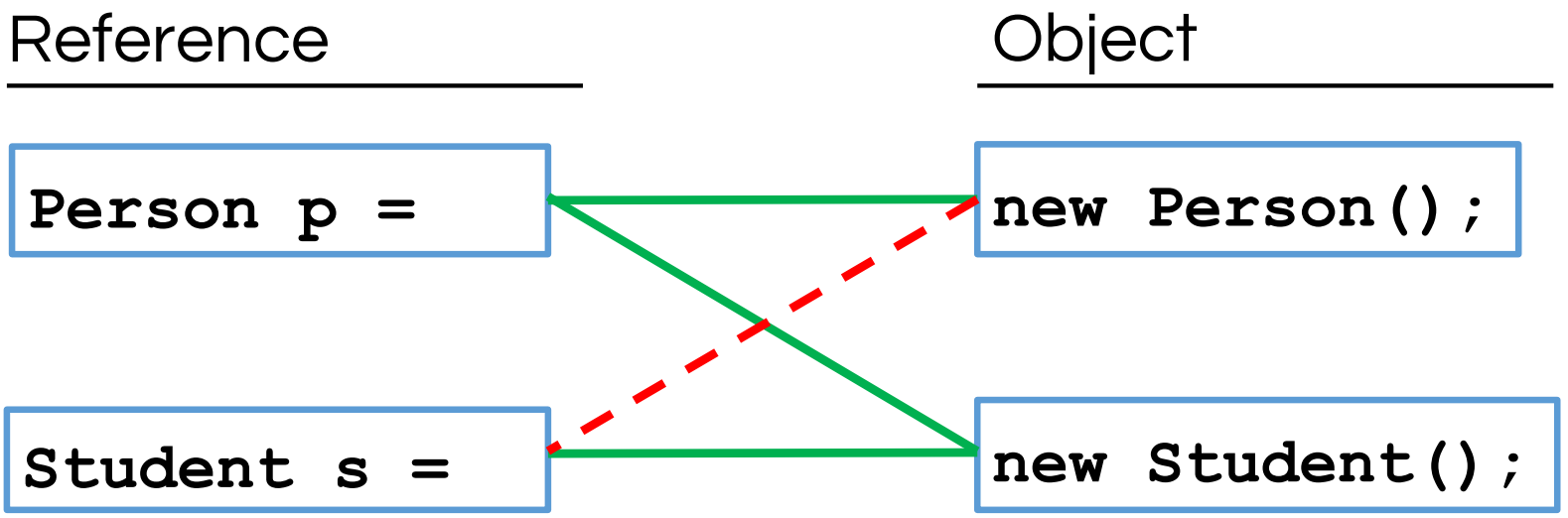
Protected 접근 지정자를 사용하면

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

클래스 상속 위계 (inheritance hierarchy)



Is-a 관계



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 Student와 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 p = new Person();
Person q = 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 = 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;
```

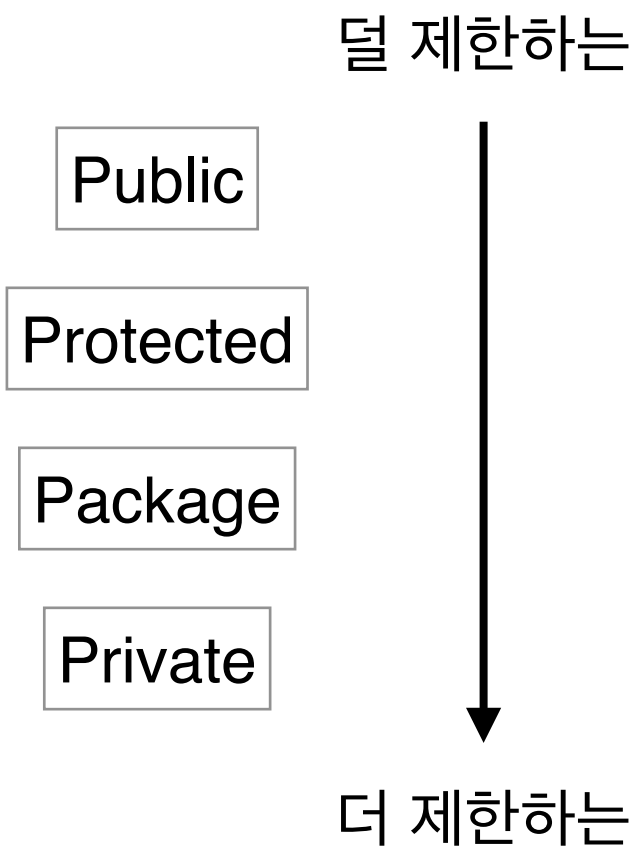
왜 컴파일러는 p가 학생임을 모를까?

- 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)



- 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

생성자 메소드에 관한 규칙들

- 클래스에 생성자 메소드가 없을 경우 기본 생성자 메소드가 자동으로 삽입됨
- 자식 클래스의 생성자 메소드는 부모 클래스 생성자 메소드를 먼저 호출한 후 본인의 일을 수행해야 함.
  - 부모 클래스 생성자 메소드를 명시적으로 호출하지 않으면 부모 클래스의 디폴트 생성자 메소드가 호출됨(super();)

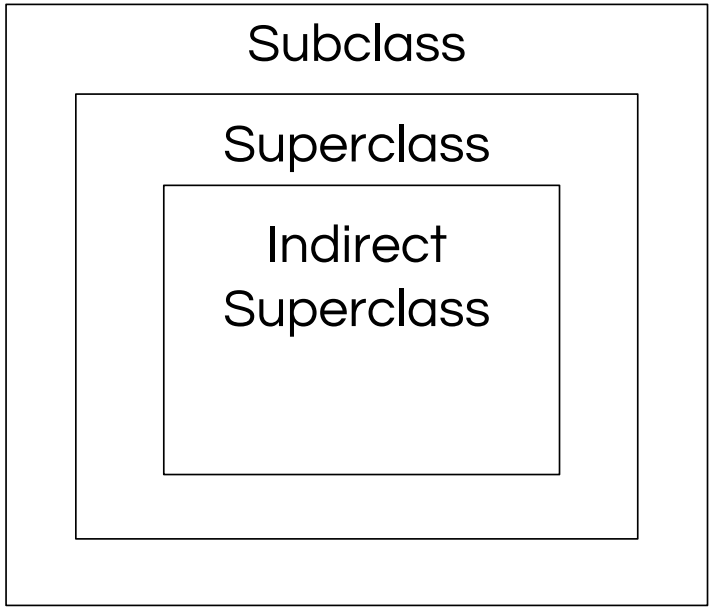
```
public class Person
{
 private String name;

}
```

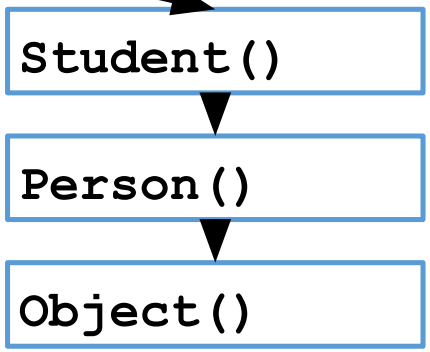
```
public class Person extends Object
{
 private String name;

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

클래스 생성 시 일어나는 일

```
public class Person extends Object {
  public Person() {}
}
public class Student extends Person {
  public Student() {}
}
```

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





문제

```
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

`this` 와 `this(…)`의 차이:
- `this`: 객체 자신을 가리키는 변수
- `this(…)`: 생성자 메소드

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 );
```

- 출력:

  Person@3343c8b3

  (Object.toString 호출)

```
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 );
```

println 이 자동으로 toString() 호출

- 출력:

  Tim

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();
 }
}
```

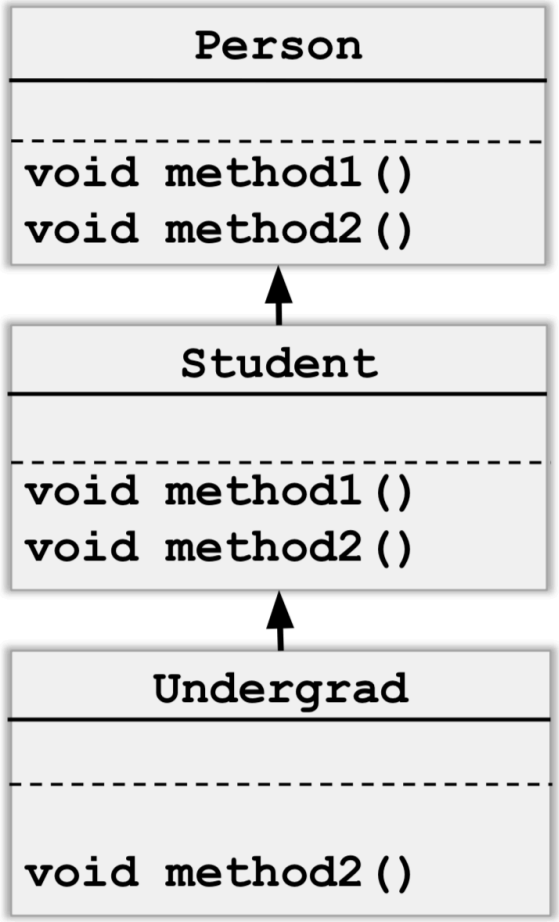
"super" refers to superclass

정적 호출 결정과 동적 호출 결정

- 정적 호출 결정 (static binding): 어떤 메소드가 호출될 지 **실행 전** 결정되는 것
- 동적 호출 결정 (dynamic binding): 어떤 메소드가 호출될지 **실행 중** 결정되는 것

```
public class Person {
 public void method1() {
  System.out.print("Person 1 ");
 }
 public void method2() {
  System.out.print("Person 2 ");
 }
}
public class Student extends Person {
 public void method1() {
  System.out.print("Student 1 ");
  super.method1();
  method2();
 }
 public void method2() {
  System.out.print("Student 2 ");
 }
}
public class Undergrad extends Student {
  public void method2() {
   System.out.print("Undergrad 2 ");
  }
}
```

```
Person u = new Undergrad();
u.method1();
```



```
public class Person {
 public void method1() {
  System.out.print("Person 1 ");
 }
 public void method2() {
  System.out.print("Person 2 ");
 }
}
public class Student extends Person {
 public void method1() {
  System.out.print("Student 1 ");
  super.method1();
  method2();
 }
 public void method2() {
  System.out.print("Student 2 ");
 }
}
public class Undergrad extends Student {
  public void method2() {
   System.out.print("Undergrad 2 ");
  }
}
```

```
Person u = new Undergrad();
u.method1();
```

```
public class Person {
 public void method1() {
  System.out.print("Person 1 ");
 }
 public void method2() {
  System.out.print("Person 2 ");
 }
}
public class Student extends Person {
 public void method1() {
  System.out.print("Student 1 ");
  super.method1();
  method2();
 }
 public void method2() {
  System.out.print("Student 2 ");
 }
}
public class Undergrad extends Student {
  public void method2() {
   System.out.print("Undergrad 2 ");
  }
}
```

```
Person u = new Undergrad();
u.method1();
```

```
public class Person {
 public void method1() {
  System.out.print("Person 1 ");
 }
 public void method2() {
  System.out.print("Person 2 ");
 }
}
public class Student extends Person {
 public void method1() {
  System.out.print("Student 1 ");
  super.method1();
  method2();
 }
 public void method2() {
  System.out.print("Student 2 ");
 }
}
public class Undergrad extends Student {
  public void method2() {
   System.out.print("Undergrad 2 ");
  }
}
```

```
Person u = new Undergrad();
u.method1();
```

```
public class Person {
 public void method1() {
  System.out.print("Person 1 ");
 }
 public void method2() {
  System.out.print("Person 2 ");
 }
}
public class Student extends Person {
 public void method1() {
  System.out.print("Student 1 ");
  super.method1();
  method2();
 }
 public void method2() {
  System.out.print("Student 2 ");
 }
}
public class Undergrad extends Student {
  public void method2() {
   System.out.print("Undergrad 2 ");
  }
}
```

```
Person u = new Undergrad();
u.method1();
```

Output so far:

```
Student 1
```

super of what?

```
public class Person {
 public void method1() {
  System.out.print("Person 1 ");
 }
 public void method2() {
  System.out.print("Person 2 ");
 }
}
public class Student extends Person {
 public void method1() {
  System.out.print("Student 1 ");
  super.method1();
  method2();
 }
 public void method2() {
  System.out.print("Student 2 ");
 }
}
public class Undergrad extends Student {
  public void method2() {
   System.out.print("Undergrad 2 ");
  }
}
```

```
Person u = new Undergrad();
u.method1();
```

Output so far:

```
Student 1
```

Static binding!

super 는 부모 클래스를 가리킴

```
public class Person {
 public void method1() {
  System.out.print("Person 1 ");
 }
 public void method2() {
  System.out.print("Person 2 ");
 }
}
public class Student extends Person {
 public void method1() {
  System.out.print("Student 1 ");
  super.method1();
  method2();
 }
 public void method2() {
  System.out.print("Student 2 ");
 }
}
public class Undergrad extends Student {
  public void method2() {
   System.out.print("Undergrad 2 ");
  }
}
```

```
Person u = new Undergrad();
u.method1();
```

Output so far:

Student 1 Person 1

```
public class Person {
 public void method1() {
  System.out.print("Person 1 ");
 }
 public void method2() {
  System.out.print("Person 2 ");
 }
}
public class Student extends Person {
 public void method1() {
  System.out.print("Student 1 ");
  super.method1();
  method2();
 }
 public void method2() {
  System.out.print("Student 2 ");
 }
}
public class Undergrad extends Student {
  public void method2() {
   System.out.print("Undergrad 2 ");
  }
}
```

```
Person u = new Undergrad();
u.method1();
```

Output so far:

```
Student 1 Person 1
```

which method2()?

```
public class Person {
 public void method1() {
  System.out.print("Person 1 ");
 }
 public void method2() {
  System.out.print("Person 2 ");
 }
}
public class Student extends Person {
 public void method1() {
  System.out.print("Student 1 ");
  super.method1();
  this.method2();
 }
 public void method2() {
  System.out.print("Student 2 ");
 }
}
public class Undergrad extends Student {
  public void method2() {
   System.out.print("Undergrad 2 ");
  }
}
```

```
Person u = new Undergrad();
u.method1();
```

Output so far:

```
Student 1 Person 1
```

type of object at runtime

this 는 현재 실행중인 객체를 가리킴

Dynamic binding!

```
public class Person {
 public void method1() {
  System.out.print("Person 1 ");
 }
 public void method2() {
  System.out.print("Person 2 ");
 }
}
public class Student extends Person {
 public void method1() {
  System.out.print("Student 1 ");
  super.method1();
  this.method2();
 }
 public void method2() {
  System.out.print("Student 2 ");
 }
}
public class Undergrad extends Student {
  public void method2() {
   System.out.print("Undergrad 2 ");
  }
}
```

```
Person u = new Undergrad();
u.method1();
```

Final output:

```
Student 1 Person 1 Undergrad 2
```

```
public class Person {
  private String name;
  public Person(String name)
  {this.name = name;}
  public boolean isAsleep(int hr)
  { return 22 < hr || 7 > hr; }
  public String toString() { return name; }
  public void status( int hr ) {
    if (this.isAsleep(hr))
      System.out.println( "취침중: " + this );
    else
      System.out.println( "깼음: " + this );}
}
public class Student extends Person
{
  public Student(String name) {super(name);}
  public boolean isAsleep( int hr )// override
  { return 2 < hr && 8 > hr; }
}
```

❍ 실행결과?

```
Person p;
p = new Student("지윤");
p.status(1);
```

A. "깼음: 지윤"

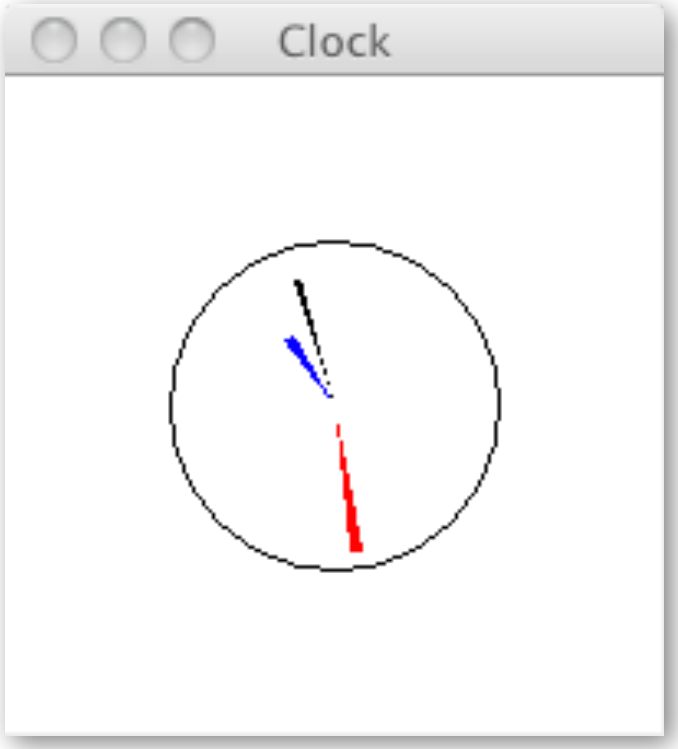
B. "취침중: 지윤"

C. 실행 오류

D. 컴파일 오류

예제: 시계

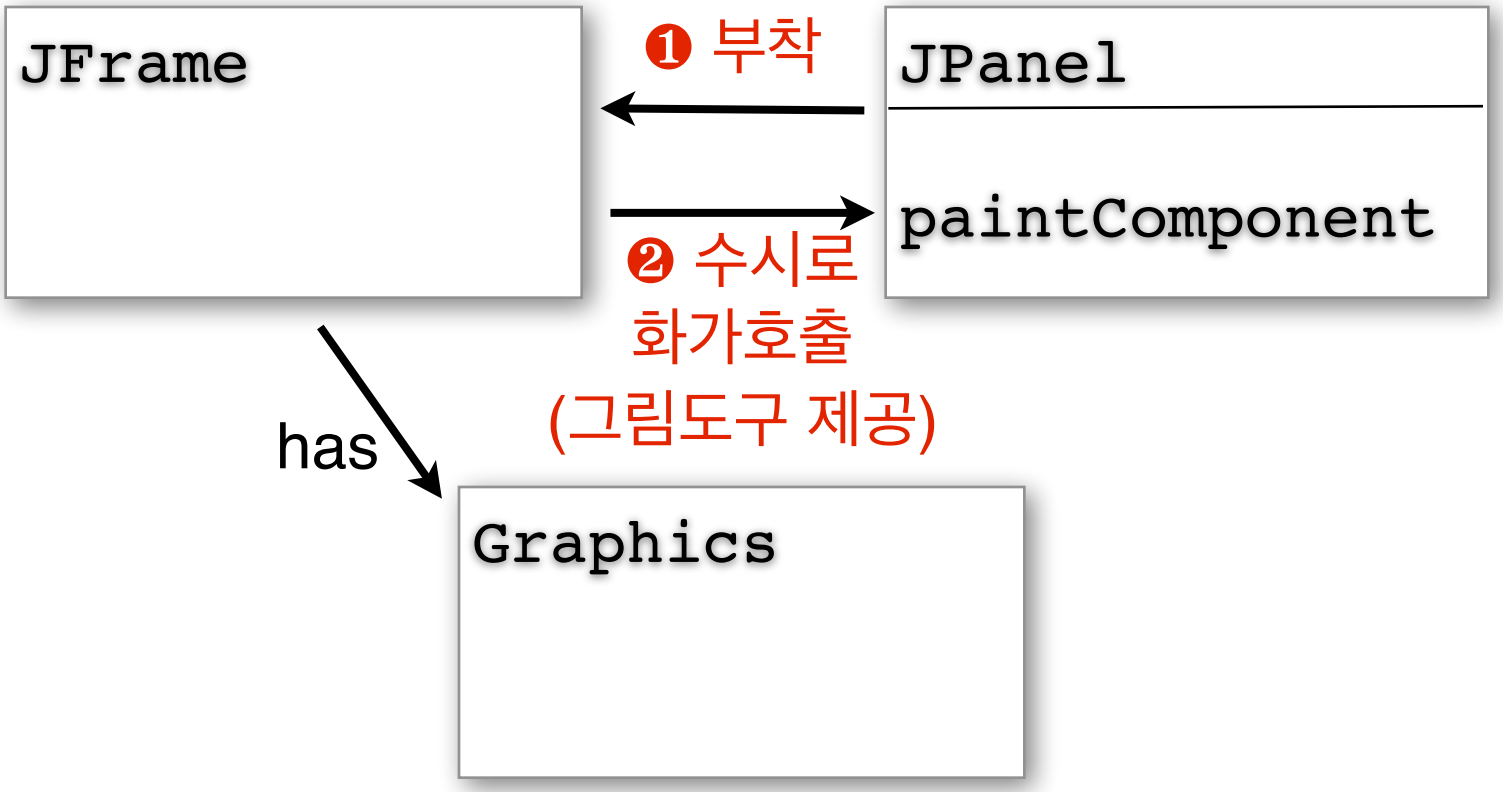
- 아날로그 시계를 보여주는 창을 만들자!



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
   }
}
```

프레임, 패널, 화가의 구동 원리



내가 원하는 패널을 만드려면

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

JPanel
paintComponent

MyPanel
paintComponent

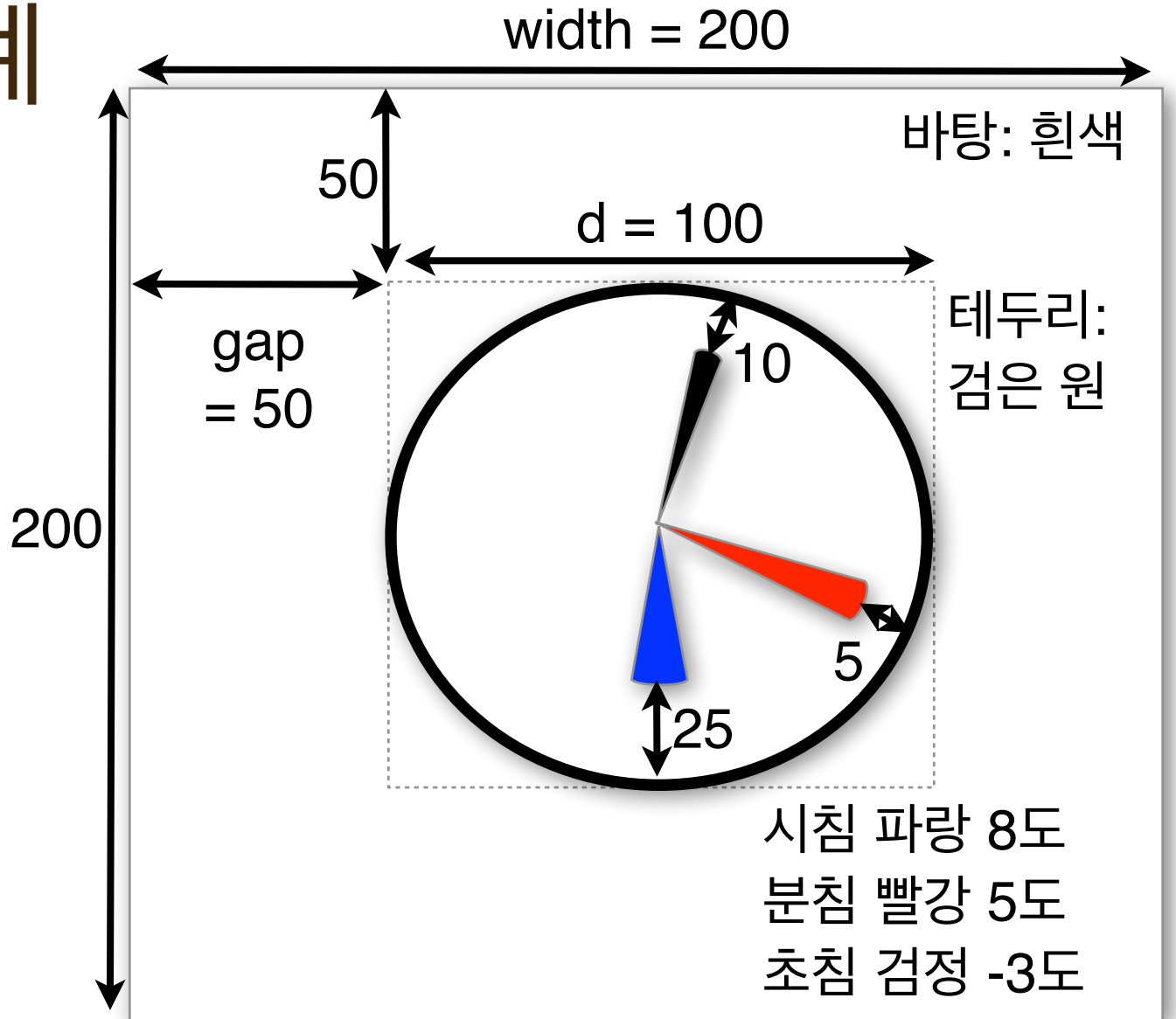
paintComponent

JPanel

paintComponent

MyPanel

시계 설계


width = 200
바탕: 흰색
50
d = 100
gap
= 50
테두리:
검은 원
200
10
5
25
시침 파랑 8도
분침 빨강 5도
초침 검정 -3도

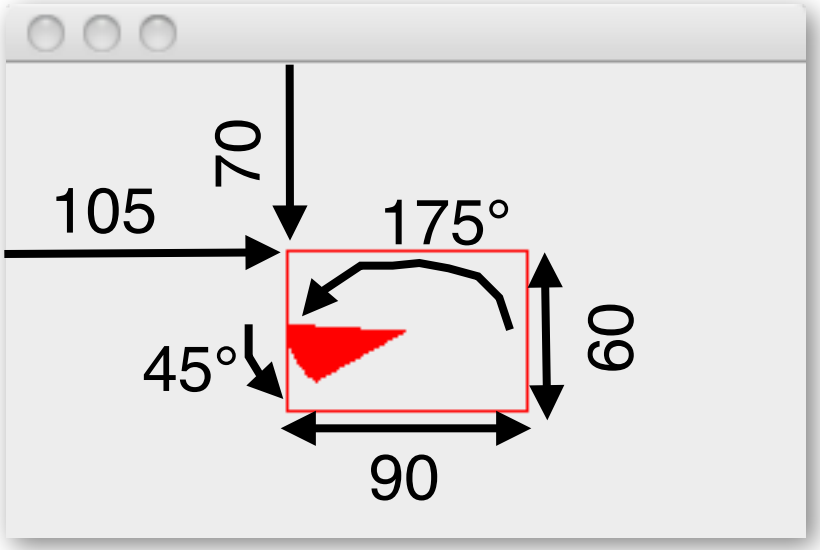
시분초침의 각도 계산

```
...
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

채운 호 또는 타원 조각 그리기

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



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);
```