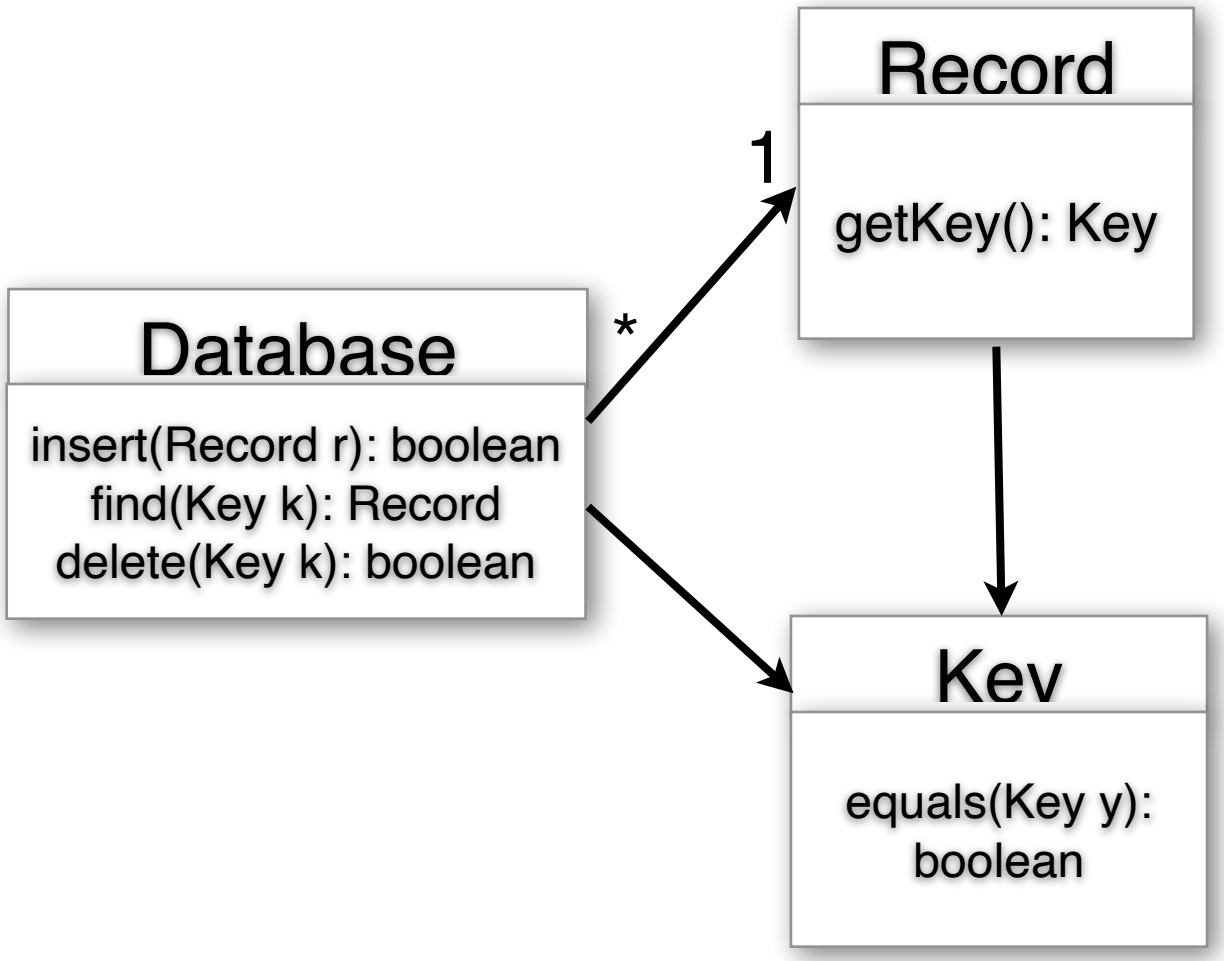
9
부품구조: 인터페이스를 활용한 프로그래밍

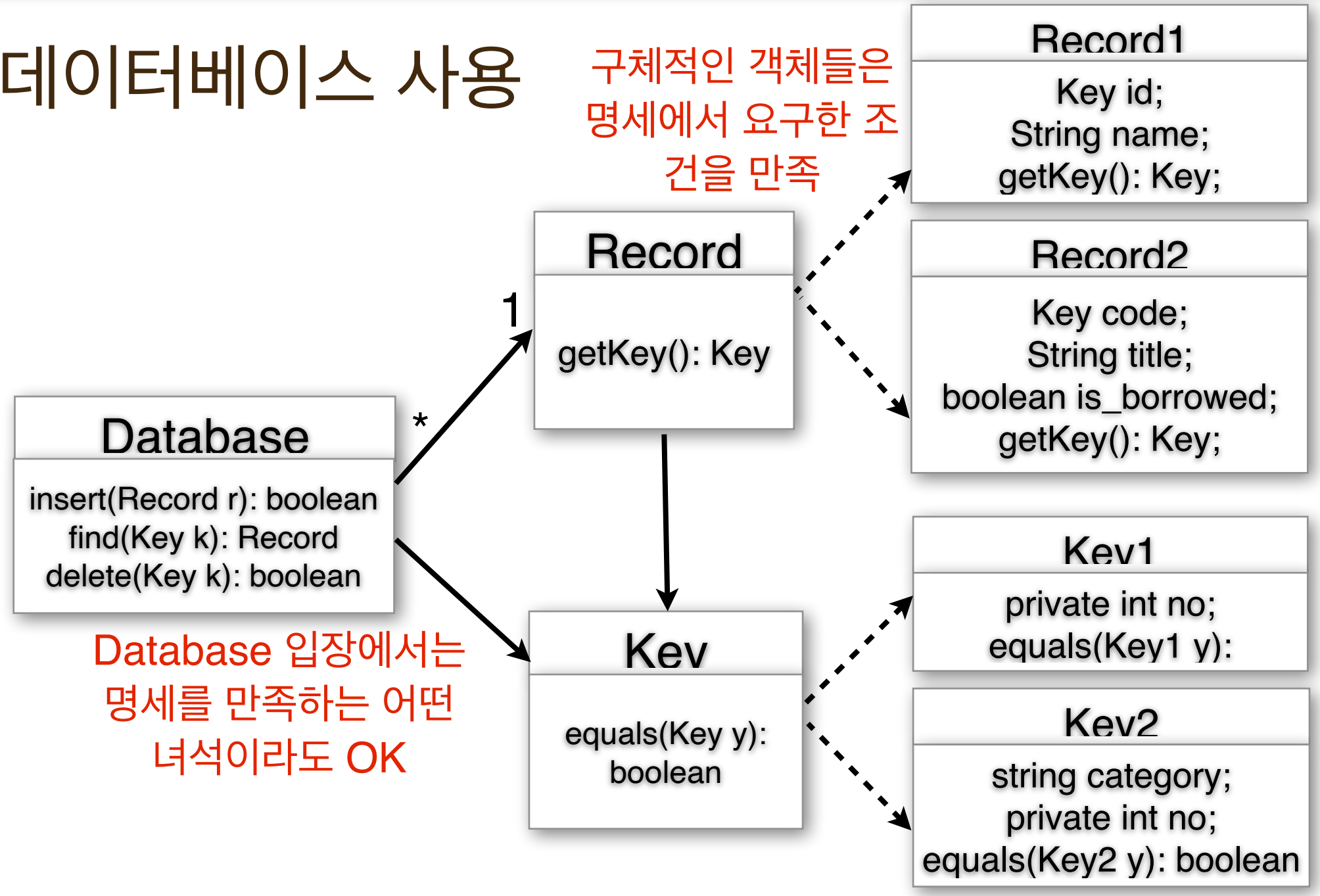
데이터베이스 다시보기



구체적인 Record와 Key에 대한 정보 없이 Database를 구현할 수 있었다. 어떻게?

데이터베이스 사용

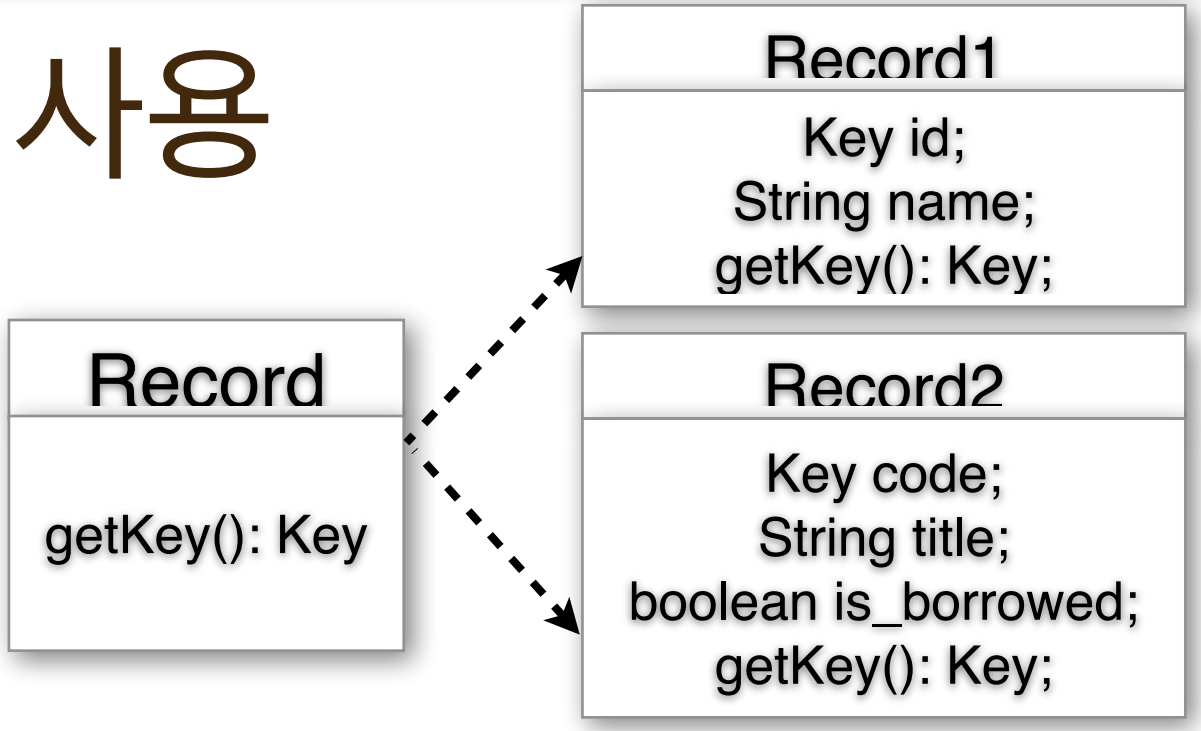
구체적인 객체들은 명세에서 요구한 조건을 만족



Database 입장에서는 명세를 만족하는 어떤 녀석이라도 OK

데이터베이스 사용

Java에서는 이러한 명세를 인터페이스(interface)라고 한다.
인터페이스는 구현이 없다.



Java에서는 이러한 클래스를 Record 인터페이스를 구현했다고 한다.
구현은 인터페이스에 명시된 모든 필드, 메소드를 타입에 맞게 제공해야 한다.

인터페이스 정의

- 인터페이스 정의
  - interface <이름> { <정의나열> }
  - 필드 정의는 기존과 동일, 단, 초기화 금지
  - 메소드 정의는 구현 없이 타입과 이름만 써 준다.

- 예,

인수이름은 상관 없으나 중복되지 않도록

  - `interface Key { boolean equals(Key y); }`
  - `interface Record { Key getKey(); }`

인터페이스 또한 하나의 타입

- 클래스와 동일하게 타입으로 사용할 수 있다.
  - 단, new는 사용할 수 없다. 구현이 없으므로.

- 예,
  - 변수 선언, `Key k;`
  - 인수 전달, `void method (Key k) { ... }`
  - 반환, `Key getKey() { ... }`

인터페이스 구현

- 클래스를 정의할 때 뒤에 implements <인터페이스> 를 붙여 주면 그 인터페이스를 구현한다는 뜻이다.

```
class IntegerKey implements Key {
    private int key;
    public IntegerKey(int i) { key = i; }
    public int getInt() { return key; }

    boolean equals(Key k) {
        return key == ((IntegerKey)k).getInt();
    }
}
```

Key에 명세된 대로 equals 메소드를 작성

인터페이스 구현

- 여러 클래스를 같은 인터페이스로 구현할 수 있다.

```
class CodeKey implements Key {
    private String category;
    private int code;
    public CodeKey(String s, int i) { category=s; code=i; }
    public String getCategory() { return category; }
    public int getCode() { return code; }

    boolean equals(Key k) {
        CodeKey ck = (CodeKey) k;
        return category.equals(ck.getCategory()) &&
               code == ck.getInt();
    }
}
```

Key에 명세된 대로 equals 메소드를 작성

같은 인터페이스를 구현한 객체는 혼용 가능

- IntegerKey 객체, CodeKey 객체 모두 Key 객체이다.

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

클래스, 객체, 인터페이스, 상속, 구현 !?

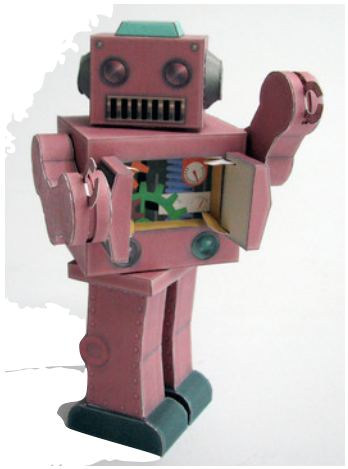
- 뭐가 이리도 복잡해요? 장난감 로봇으로 이해해 보자.

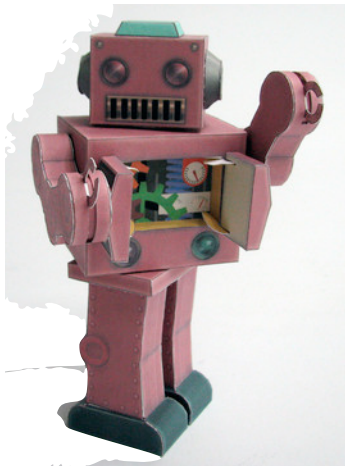
인공지능 로봇:
* 대화
* 장애물을 피하면서 이동

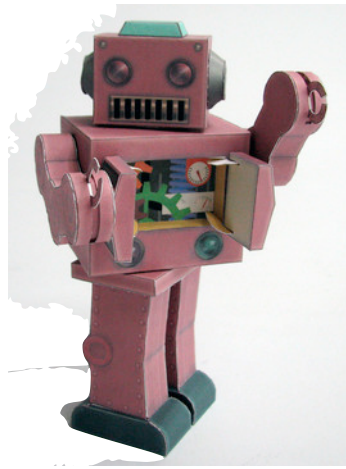
인터페이스

클래스

객체들





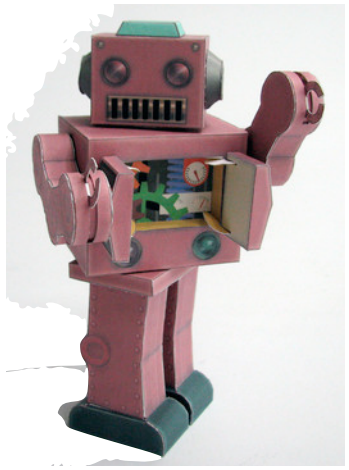


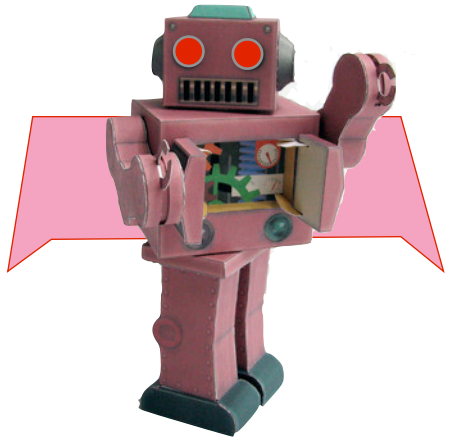
상속: 구현을 재사용

클래스

하위클래스

상속



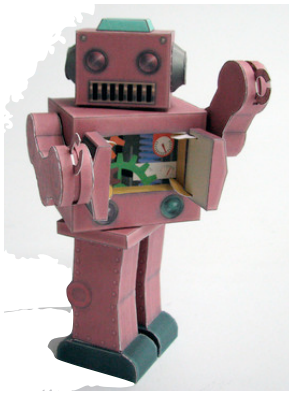


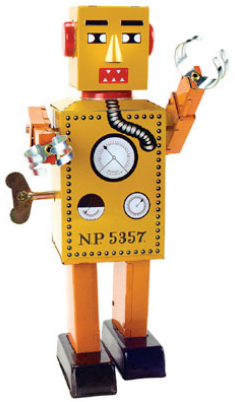
기존기능+
레이저 눈+
날개

인터페이스 구현: 요구조건만 만족

인터페이스

인공지능 로봇:
* 대화
* 장애물을 피하면서 이동





상속 vs 구현

- 상속과 인터페이스 구현은 유사한 속성이 있다.
  - `class B extends A`
  - `class B implements A`
  - `A x = new B();`

- 하지만, 상속과 인터페이스 구현은 엄연히 다른 개념임을 명심하라.
  - 상속은 코드의 재사용
  - 인터페이스 구현은 명세를 만족하는 코드 구현

하위타입 (Subtype)

- 타입을 값의 집합으로 보고
- 타입에 값의 포함관계로 상하위 관계를 줄 수 있다.
- 예,
  - 수학에서 정수 < 실수
  - 한국사람 < 사람 < 포유류 <생물
  - 컴퓨터공학과 학생 < 학생
- 다른 말로, 하위가 할 수 있는 일이 더 많다.

하위타입으로 치환 가능

- 하위타입으로 치환해도 말이 된다.
  - 생물은 죽는다 ===> 사람은 죽는다
  - 사람은 음식을 먹는다 ==> 한국사람은 음식을 먹는다

- 반대가 항상 성립하는 것은 아니다.
  - 사람은 입이 있다 ==> 생물은 입이 있다
  - 한국사람은 한국어를 한다 ==> 사람은 한국어를 한다

Java에서의 하위타입

- 상속 받거나 인터페이스를 구현하면 하위타입
  - `class B extends A`
  - `class B implements A`
  - B<A
- 상위타입을 하위타입으로 치환가능?
  - **상위타입 변수에 하위타입 객체를 넣어도 문제없음**
  - **하지만 하위타입 변수에 상위타입 객체를 넣는 것은 불허**

다시 이야기하면

- `A x = y;`
  - y의 타입이 B라고 할 때 A≥B 이면 OK.

- 인수 전달의 경우
  - `void m(A x)` 에 대해 `m(y)`를 호출하면 y가 x에 저장되는 것과 같다.
  - 즉, y의 타입이 B라고 할 때 A≥B 이면 OK.

다시 레코드로 돌아와서

- IntegerKey 객체, CodeKey 객체 모두 Key 객체이다.

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

이상한 일!

```
interface Key {
    boolean equals(Key y);
}

class IntegerKey implements Key {
    private int key;
    public IntegerKey(int i) { key = i; }
    public int getInt() { return key; }

    boolean equals(Key k) {
        return key == ((IntegerKey)k).getInt();
    }
}
```

키끼리 비교할 수 있어야 한다.

정수 키끼리 비교하고 싶은데...

boolean equals(IntegerKey k)로 하면 안될까요?

⇨ 그러면 interface Key에 위배되므로 안됨

강제 타입 변환을 더 안전하게 하려면

- 강제 타입 변환은 위험하다.
  - `boolean equals(Key k) { ... (IntegerKey) k ... }`
  - `IntegerKey ik; CodeKey ck;`
  - `ik.equals(ck);`

- 안전하게 검사하는 방법은?
  -
```
if(k instanceof IntegerKey) {
        ... (IntegerKey) k ...
}
else { 오류처리 }
```

예, IntegerKey.equals

```
interface Key {
    boolean equals(Key y);
}

class IntegerKey implements Key {
    ...
    boolean equals(Key k) {
        if(k instanceof IntegerKey)
            return key == ((IntegerKey)k).getInt();
        else
            return false;
    }
    ...
}
```

추상 클래스 (Abstract Class)

- 구현이 덜 된 클래스
  - 메소드 중 일부가 정의되지 않은 클래스

- 메소드 구현을 공유하는 경우, 인터페이스보다 유리
  - 하위 클래스들이 메소드 중 일부를 동일하게 사용 ==> 상속을 통한 코드 재사용
  - 하위 클래스들이 메소드 중 일부는 반드시 구현해야 함 ==> 인터페이스 만족

예제, 카드 게임

- class Card와 class CardDeck은 예전 것을 사용
- class Dealer는 게임을 진행
  - 특정 플레이어 차례가 되면 플레이어에게 카드를 원하는가 묻고, 그렇다고 하면 카드통에서 뽑아 카드를 준다.
- interface CardPlayerBehaviour는 게임을 함
  - 다음 interface를 만족

```
public interface CardPlayerBehaviour {
    public boolean wantsACard();
    public void receiveCard(Card c);
}
```

컴퓨터 플레이어, 사람 플레이어

- 컴퓨터 플레이어나 사람 플레이어나 카드를 받을 때는 똑같이 수행된다.
  - 카드를 받아서 자기 손에 쥔다.

- 하지만, 카드를 받을래? 물었을 때
  - 사람 플레이어의 경우 입력창을 통해 답을 구한다.
  - 컴퓨터 플레이어의 경우 내부 알고리즘을 통해 계산해서 답을 구한다.

- 컴퓨터 플레이어와 사람 플레이어의 상위 "추상" 클래스 CardPlayer를 작성해서 상속 받도록 하자.

추상 클래스 카드플레이어

```
public abstract class CardPlayer implements CardPlayerBehaviour
{
    private Card[] my_hand;
    private int card_count;

    public CardPlayer(int max_cards) {
        my_hand = new Card[max_cards];
        card_count = 0;
    }
    public abstract boolean wantsACard();
    public void receiveCard(Card c) {
        my_hand[card_count] = c;
        card_count = card_count + 1;
    }
}
```

사람 플레이어

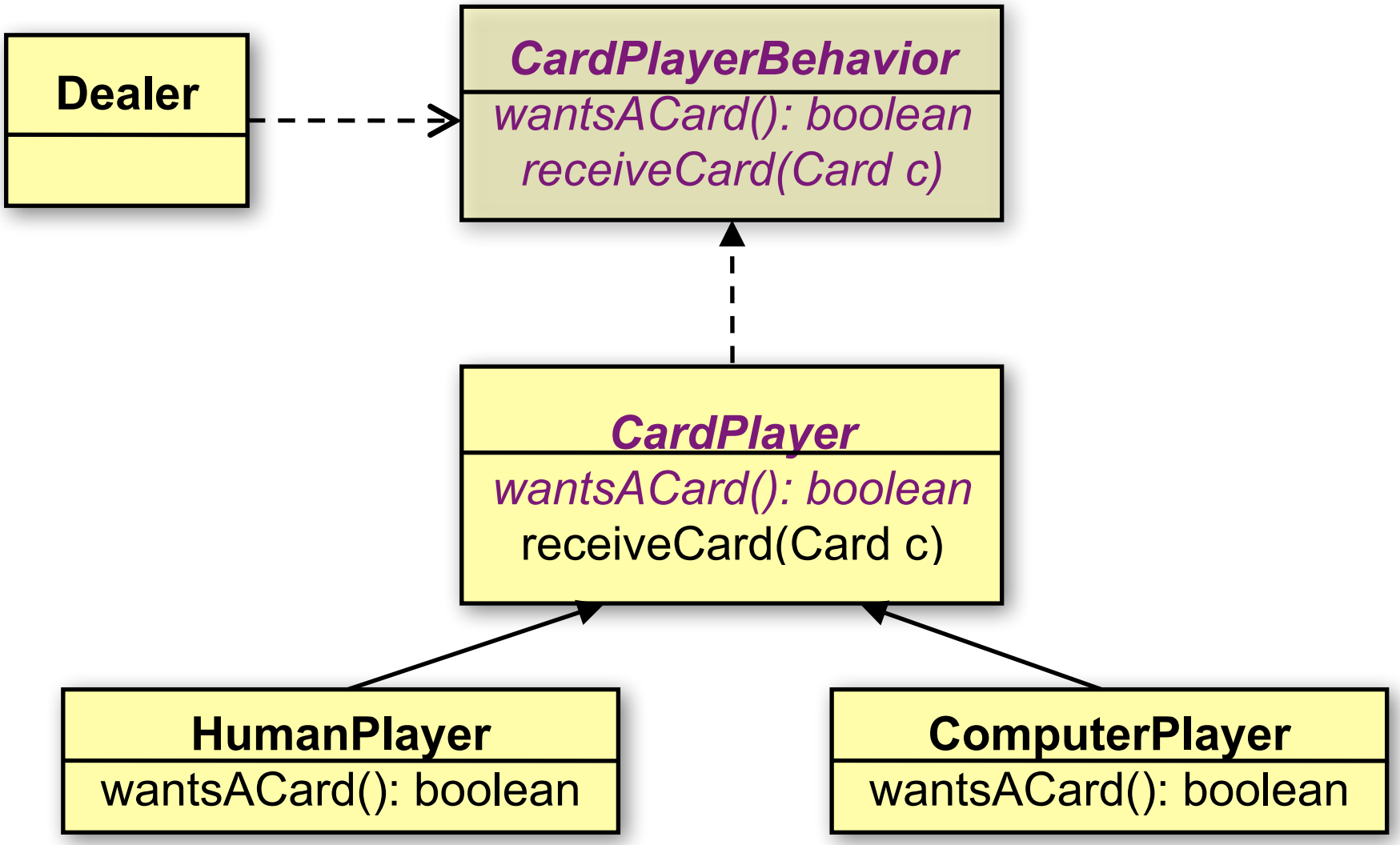
```
import javax.swing.*;

public class HumanPlayer extends CardPlayer {
    public HumanPlayer(int max_cards) {
        super(max_cards);
    }
    public boolean wantsACard() {
        String response = JOptionPane.showInputDialog
            ("Do you want another card (Y or N)?");
        return response.equals("Y");
    }
}
```

컴퓨터 플레이어

```
public class ComputerPlayer extends CardPlayer {
    public ComputerPlayer(int max_cards) {
        super(max_cards);
    }
    public boolean wantsACard() {
        int sum = 0;
        for(int i=0; i<card_count; i++)
            sum += my_hand[i].getCount();
        return sum < 15;
    }
}
```

카드 게임의 소프트웨어 구조



프레임워크 (Framework)

- 프레임워크란 웬만한 것이 다 정의되어 있고 사용자가 일부만 채워주면 실체가 나오는 것을 말한다.

- 예, GraphicsWindow
  - setSize, setBackground, setVisible 등은 다 구현되어 있음
  - paint 메소드만 구현되어 있지 않음
  - paint 메소드만 구현하면 창이 완성

- Java에서는 추상 클래스로 프레임워크를 구현할 수 있다.

인터페이스 vs 추상 클래스

인터페이스	추상클래스
행위에 대한 명세만 정의	일부는 구현, 일부는 구현 없음
인터페이스를 구현한다는 것은, 명세를 만족하는 코드를 작성한다는 것이다.	상속을 받으면 구현된 코드는 재사용하고, 구현 없는 코드만 채워 준다.
클래스가 어떻게 구현되었는지 상관없이 그룹짓거나 연결할 때 사용	클래스를 구현 할 때 일부 코드가 동일하게 사용되는 여러 클래스가 있을 때 사용.

정리

- Java에서는 다양한 부품구조를 제공한다.
  - 클래스 및 상속
  - 인터페이스: 스펙을 정의하고 이에 맞추어 구현
  - 추상클래스: 구현 중 일부를 비워두고 나중에 구현
- 부품들끼리의 관계는 하위타입을 따른다.
  - Java에서는 상속, 구현, 확장을 통해 관계 성립
  - 상위타입 변수에는 하위타입 객체를 저장할 수 있다.