CSE2016

프로그램설계방법론

상속을 활용한 프로그램 부품의 재사용

Component Reuse through Inheritance

도경구

한양대학교 ERICA 소프트웨어학부

Starter

PuzzleStarter

main

종속 관계로 클래스 독립 개별 구현 불가

Controller

Model

PuzzleController

SlidePuzzleBoard board
PuzzleWriter writer

PuzzleController
(SlidePuzzleBoard b, PuzzleWriter w)
void play()

SlidePuzzleBoard

PuzzlePiece[][] board
int empty_row
int empty_col

PuzzlePiece[][] contents()
boolean move(int w)
boolean found(int v, int row, int col)

1

*

Output View

PuzzleWriter

SlidePuzzleBoard board
final int SIZE

PuzzleWriter(SlidePuzzleBoard b)
void displayPuzzle()
void printError(String s)

has
JPanel

uses
javax.swing
java.awt

PuzzlePiece

int face_value

int faceValue()

specification

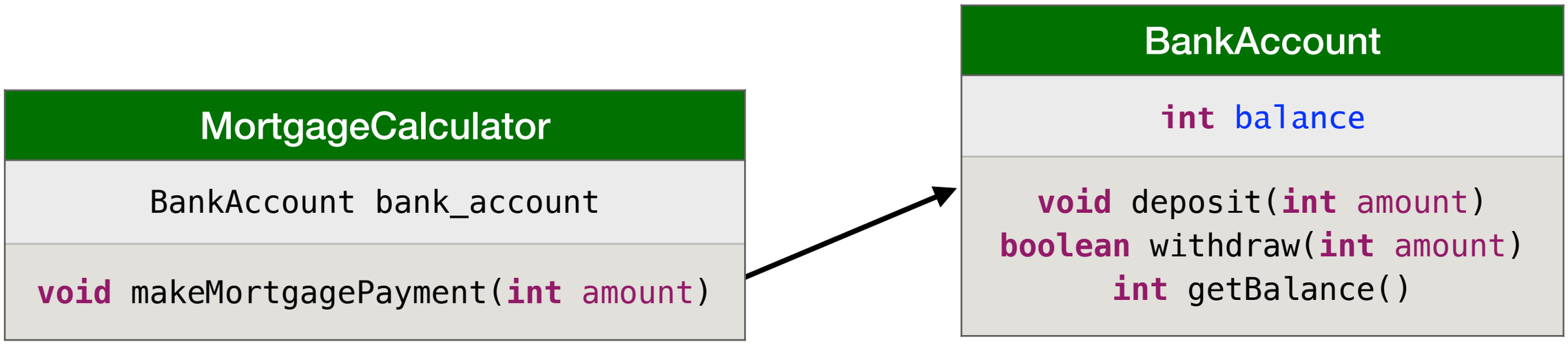
Programming to the interface,

not the implementation

class

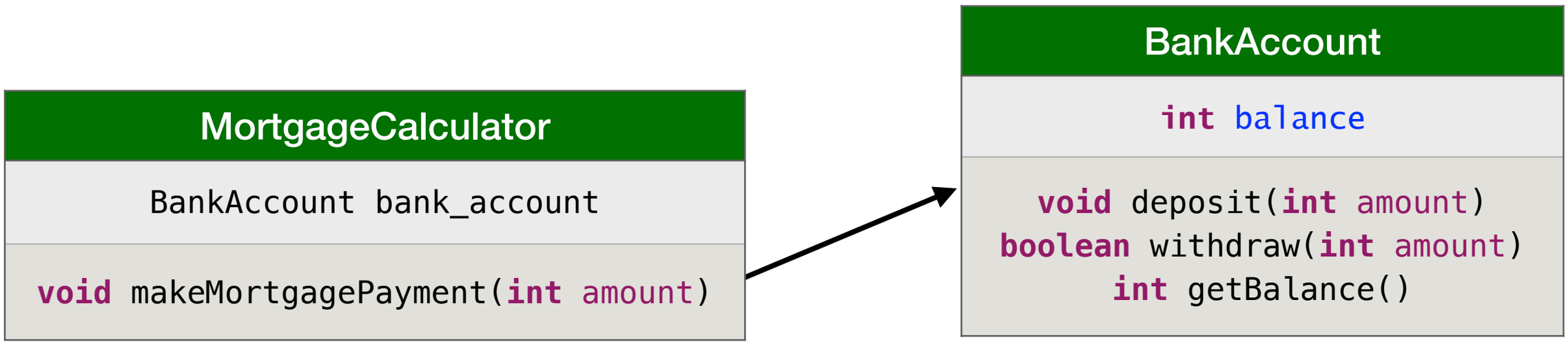
프로그래밍 작업 분업 가능

클래스 다이어그램


MortgageCalculator
BankAccount bank_account
void makeMortgagePayment(int amount)
BankAccount
int balance
void deposit(int amount)
boolean withdraw(int amount)
int getBalance()

클래스 다이어그램

구현 분업?



클래스 다이어그램

구현 분업 가능!

class

MortgageCalculator

BankAccountSpecification bank_account

void makeMortgagePayment(**int** amount)

class

BankAccount

int balance

void deposit(**int** amount)
boolean withdraw(**int** amount)
int getBalance()

implements

BankAccountSpecification

void *deposit(**int** amount)*
boolean *withdraw(**int** amount)*

interface

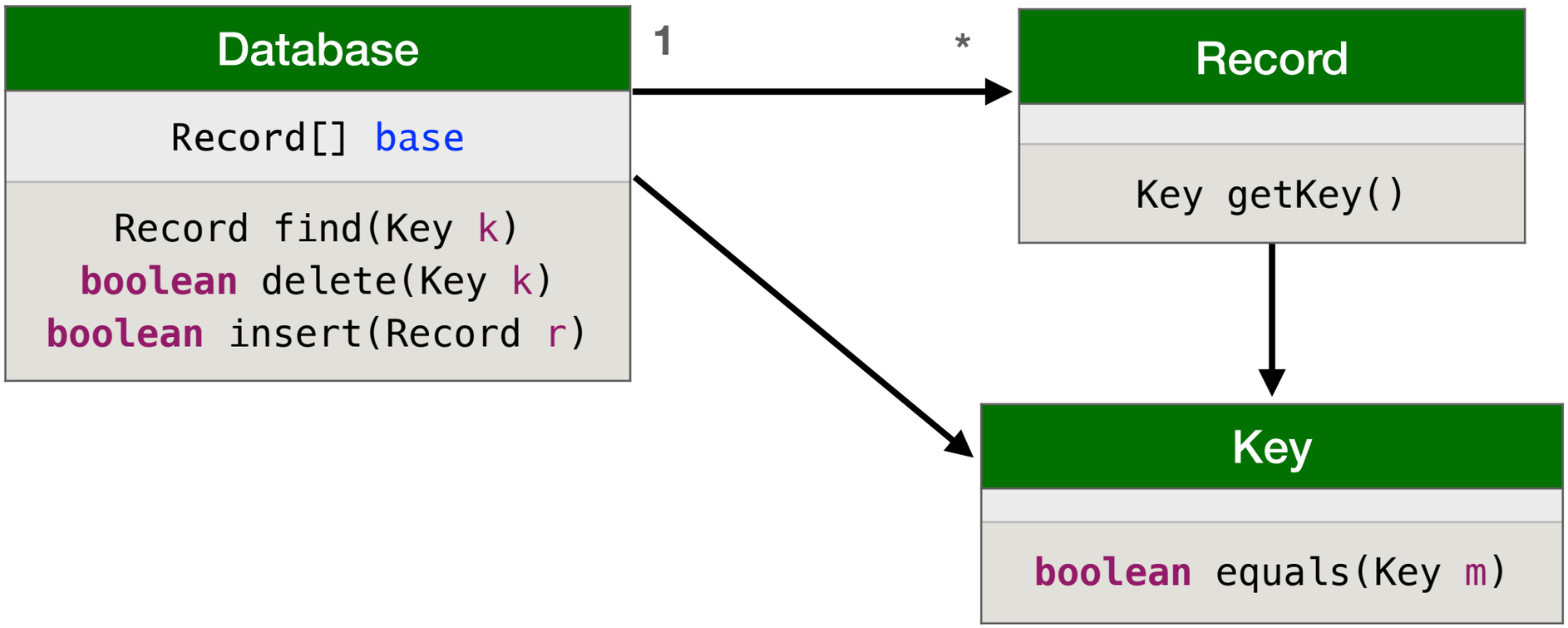
```java
public interface BankAccountSpecification {

    /** deposit - 입금
     * @param amount - 입금액 (0이상의 정수) */
    public void deposit(int amount);

    /** withdraw - 출금 (잔고가 충분하면)
     * @param amount - 출금액 (0이상의 정수)
     * @return true - 출금 성공, false - 출금 실패 */
    public boolean withdraw(int amount);

}
```

```java
public class MortgagePaymentCalculator {

    private BankAccountSpecification bank_account;

    /** Constructor
     * @param account - 사용 계좌 */
    public MortgagePaymentCalculator(BankAccountSpecification account) {
        bank_account = account;
    }

    /** */
    public void makeMortgagePayment(int amount) {
        if (bank_account.withdraw(amount))
            System.out.println(amount + "원 지불");
        else
            System.out.println("잔고 부족");
    }
}
```

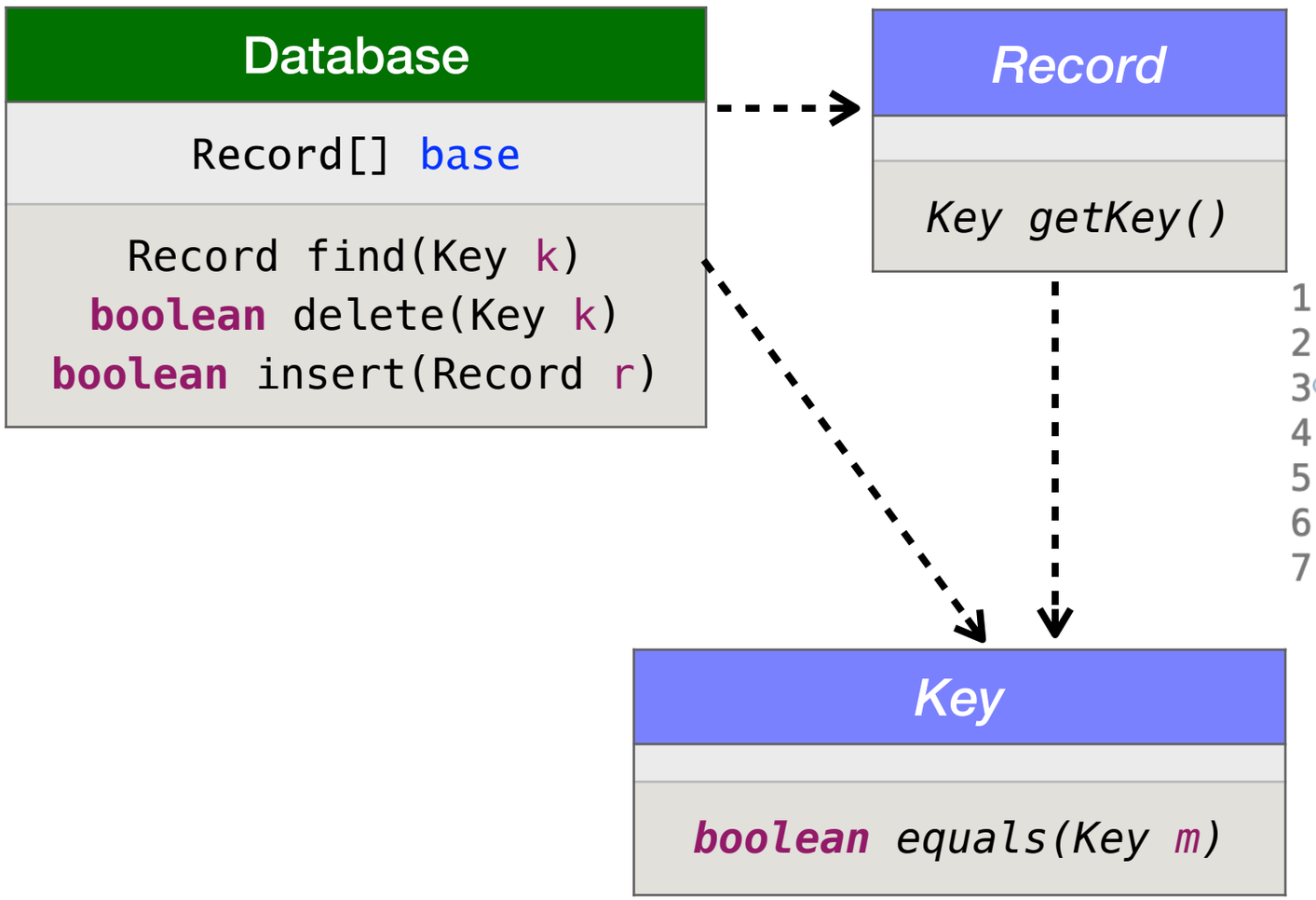
```java
public class BankAccount implements BankAccountSpecification {

    private int balance;

    public BankAccount() {
        balance = 0;
    }

    /** deposit - 입금
     * @param amount - 입금액 (0이상의 정수) */
    public void deposit(int amount) {
        balance = balance + amount;
    }

    /** withdraw - 출금 (잔고가 충분하면)
     * @param amount - 출금액 (0이상의 정수)
     * @return true - 출금 성공, false - 출금 실패 */
    public boolean withdraw(int amount) {
        if (amount <=  balance) {
            balance -= amount;
            return true;
        }
        else
            return false;
    }

    /** getBalance - 잔액 리턴
     * @return - 잔액 */
    public int getBalance() {
        return balance;
    }
}
```

```java
public interface BankAccountSpecification {

    /** deposit - 입금
     * @param amount - 입금액 (0이상의 정수) */
    public void deposit(int amount);

    /** withdraw - 출금 (잔고가 충분하면)
     * @param amount - 출금액 (0이상의 정수)
     * @return true - 출금 성공, false - 출금 실패 */
    public boolean withdraw(int amount);

}
```

Arhitecture for a Database



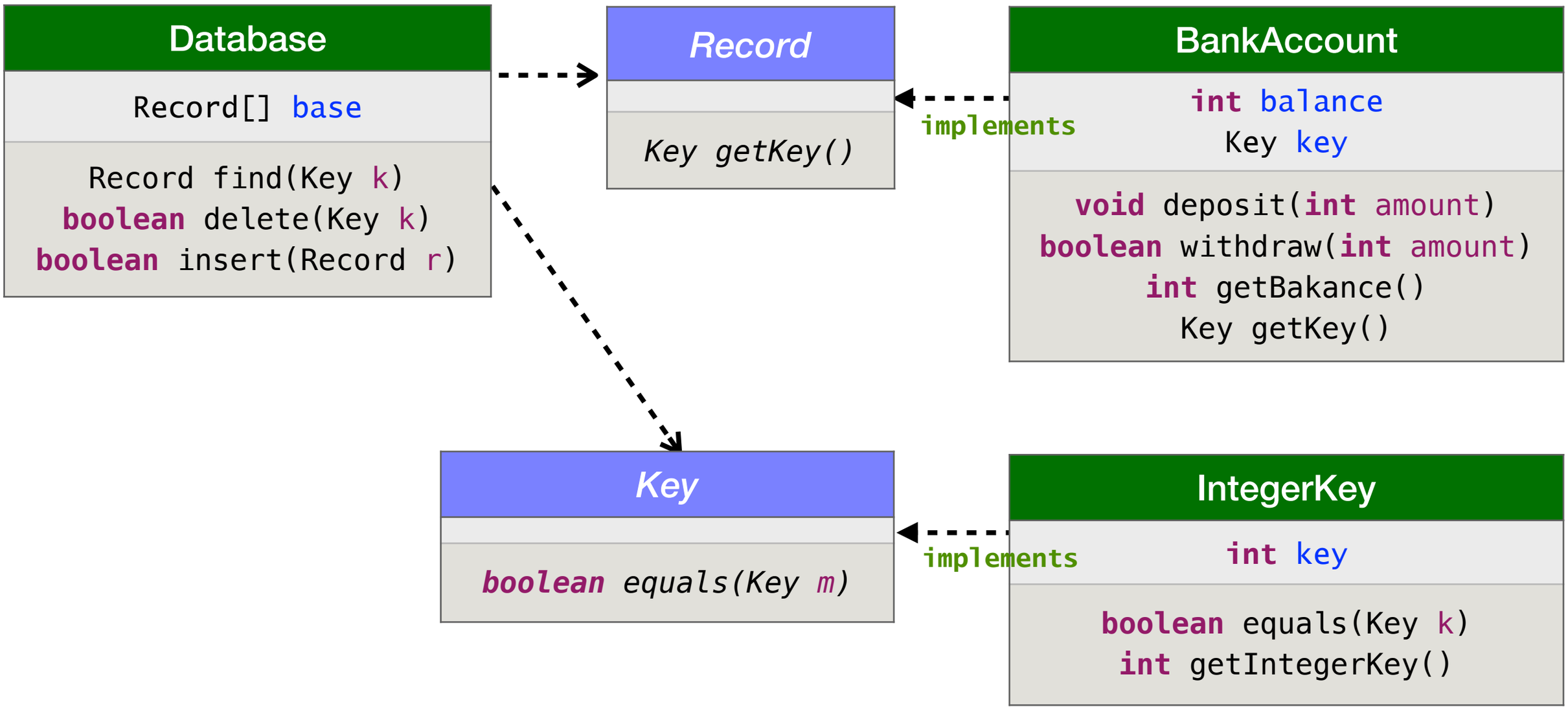
Arhitecture for a Database




```java
public interface Record {

    /** getKey - 레코드를 유일하게 식별하는 키를 리턴
     * @return - 키 */
    public Key getKey();

}
```

```java
public interface Key {

    /** equals - 인수로 제공된 키와 자신과 같은지 비교
     * @param - 비교 대상 키
     * @return - 같으면 true, 다르면 false */
    public boolean equals(Key m);
}
```

Architecture for Bank Database Application



상속을 활용한 프로그램 부품의 재사용

```java
public interface Record {

    /** getKey - 레코드를 유일하게 식별하는 키를 리턴
     * @return - 키 */
    public Key getKey();

}
```

```java
public class BankAccount implements Record {

    private int balance;
    private Key key;

    /** Constructor - 계좌 개설
     * @param initial_amount - 개설 예금액
     * @param id - 계좌의 키 */
    public BankAccount(int initial_amount, Key id) {
        balance = initial_amount;
        key = id;
    }

    /** deposit - 입금
     * @param amount - 입금액 (0이상의 정수) */
    public void deposit(int amount) {
        balance = balance + amount;
    }

    /** withdraw - 출금 (잔고가 충분하면)
     * @param amount - 출금액 (0이상의 정수)
     * @return true - 출금 성공, false - 출금 실패 */
    public boolean withdraw(int amount) {
        if (amount <=  balance) {
            balance -= amount;
            return true;
        }
        else
            return false;
    }

    /** getBalance - 잔액 리턴
     * @return - 잔액 */
    public int getBalance() {
        return balance;
    }

    /** getKey - 계좌 키 리턴
     * @return - 키 */
    public Key getKey() {
        return key;
    }
}
```

```java
public class IntegerKey implements Key {

    private int k;

    public IntegerKey(int i) {
        k = i;
    }

    public boolean equals(Key c) {
        return k == ((IntegerKey)c).getInt();
    }

    public int getInt() {
        return k;
    }

}
```

```java
public interface Key {

    /** equals - 인수로 제공된 키와 자신과 같은지 비교
     * @param - 비교 대상 키
     * @return - 같으면 true, 다르면 false */
    public boolean equals(Key m);
}
```

상속 Inheritance	구현 Implementation
`class Sub extends Super`	`class C implements S`
• 클래스 Super를 상속 받아 클래스 Sub를 구현 • 클래스 Super의 코드를 클래스 Sub에서 재사용 • Sub: 하위 클래스 • Super: 상위 클래스	• 명세(interface) S를 만족하는 클래스 C 구현 • `S x = new C();`

사례 학습 : 상속 Inheritance 이해하기

```
public class Person {
    private String name;

    public Person(String n) {
        name = n;
    }

    public String getName() {
        return name;
    }

    ……
}
```

```
public class PersonFrom extends Person {
    private String city;

    public PersonFrom(String n, String c) {
        super(n);
        city = c;
    }

    public String getCity() {
        return city;
    }

    ……
}
```

```
PersonFrom x = new PersonFrom("마음", "안산");
System.out.println("이름: " + x.getName());
System.out.println("출신: " + x.getCity());
```

상속을 활용한 프로그램 부품의 재사용

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

    public Person(String n) {
        name = n;
    }

    public String getName() {
        return name;
    }

    public boolean sameName(Person other) {
        return getName().equals(other.getName());
    }
}
```

```java
public class PersonFrom extends Person {
    private String city;

    public PersonFrom(String n, String c) {
        super(n);
        city = c;
    }

    public String getCity() {
        return city;
    }

    public boolean same(PersonFrom other) {
        return sameName(other) &&
               city.equals(other.getCity());
    }
}
```

```java
Person p = new Person("마음");
Person q = new PersonFrom("소리","서울");
```

다음 중에서 어떤 문장이 Java 컴파일러를 통과할까?
통과하여 실행하면, 무엇을 프린트할까?

- `System.out.println(p.getCity());`
- `System.out.println(q.getName());`
- `System.out.println(p.sameName(p));`
- `System.out.println(q.sameName(p));`
- `System.out.println(q.same(p));`

기본 타입 Primitive Types

`byte`, `int`, `long`, `float`, `double`, `boolean`

Subtyping

`int <= double`

`int`는 `double`의 서브타입 이다.

`double` 값을 쓸 수 있는 곳에서 `int` 값도 쓸 수 있다.

예:

```
double d = 4.5 / 2;
```

```
public double inverseOf(double d) {
    return 1.0 / d
}
```

```
System.out.println(inverseOf(3));
```

`byte <= int <= long <= float <= double`

객체 타입 Object/Reference Types

Subtyping

상속
inheritance

```
public class MyPanel extends JPanel { . . . }
```

MyPanel <= JPanel

MyPanel는 **JPanel**의 서브타입 이다.

JPanel 객체을 쓸 수 있는 곳에서 **MyPanel** 객체도 쓸 수 있다.

```
JFrame f = new JFrame();
f.getContentPane().add(new MyPanel());
```

JPanel의 서브 타입 객체를 모두 수용

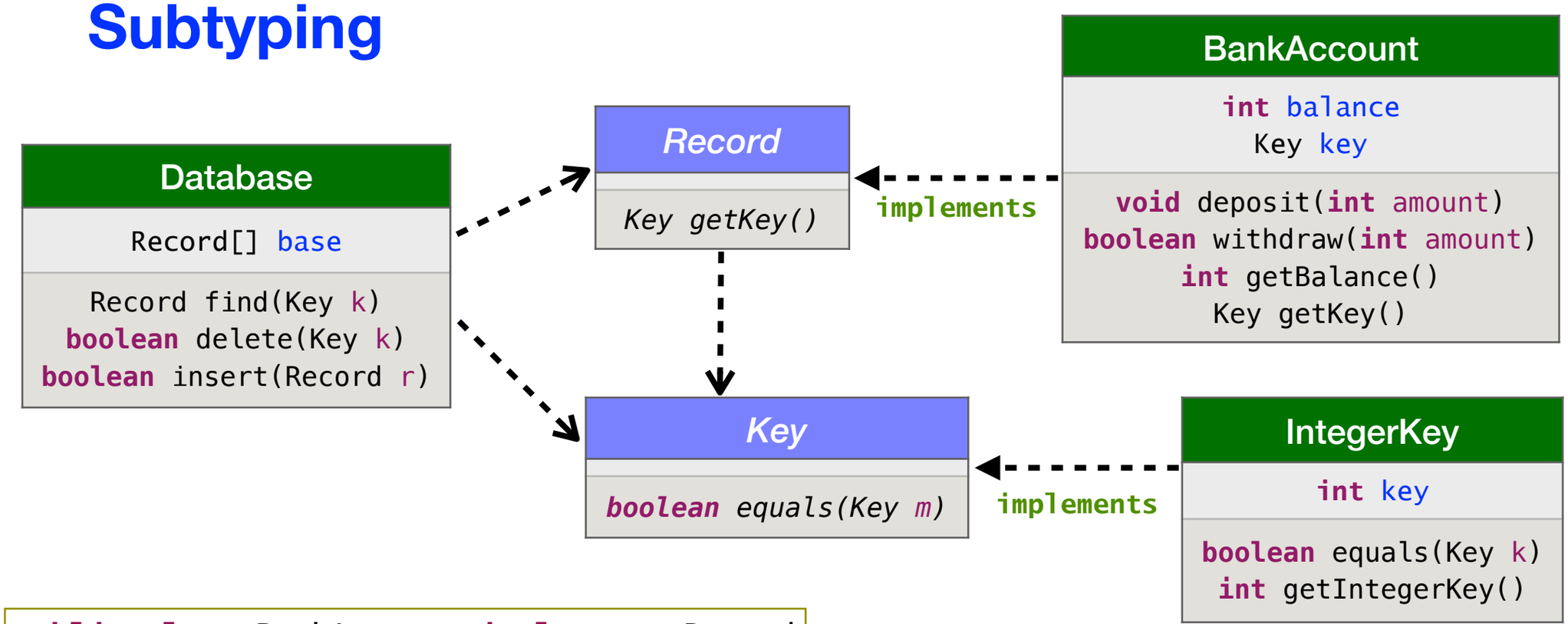
Subtyping

```
public class BankAccount implements Record
```

BankAccount <= Record

```
Database db = new Database(4);
IntegerKey k = new IntegerKey(55);
BankAccount b = new BankAccount(10000, k);
boolean success = db.insert(b)
```

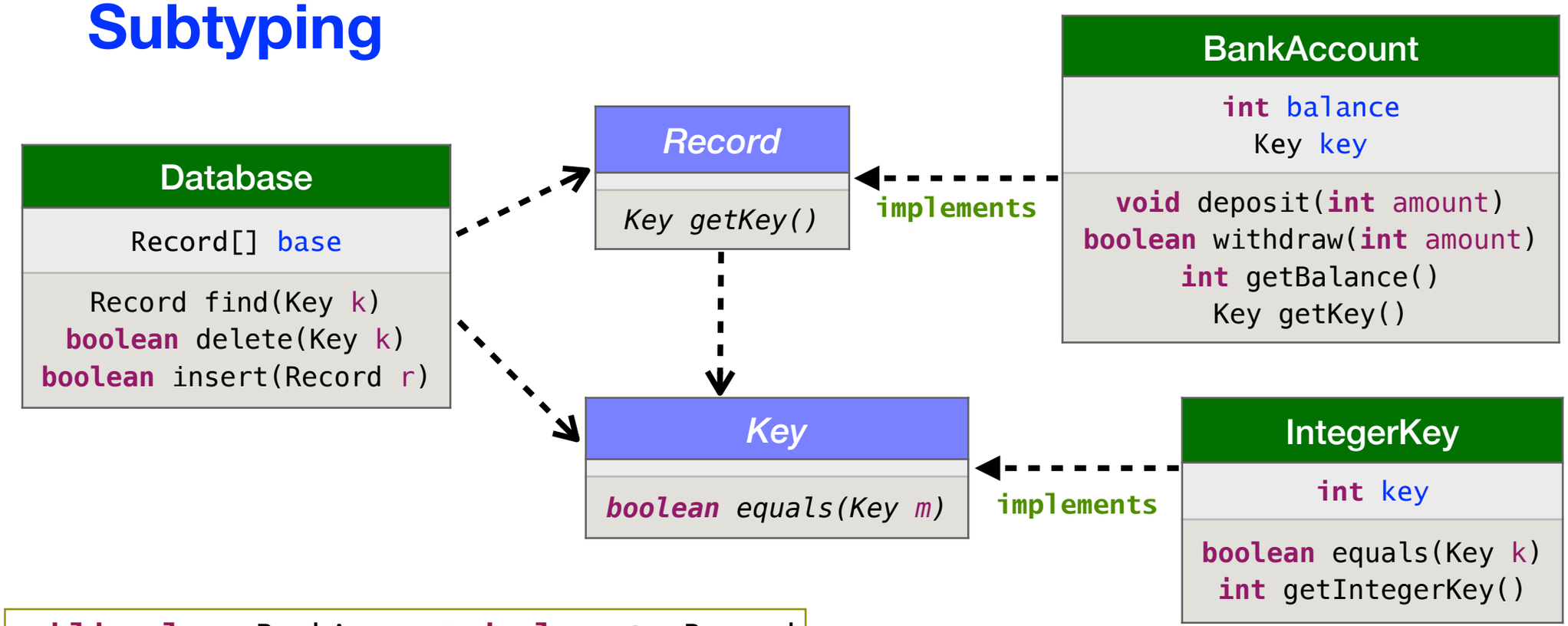
Subtyping




```
public class BankAccount implements Record
```

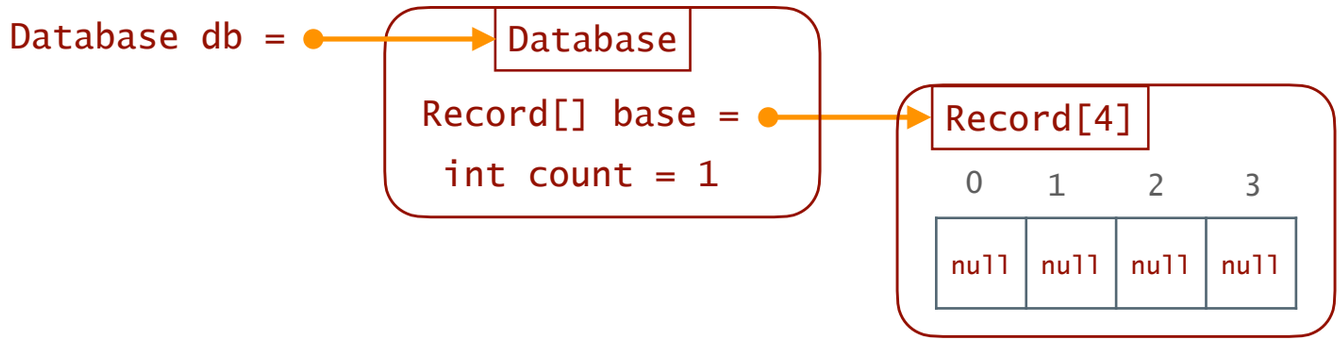
```
Database db = new Database(4);
IntegerKey k = new IntegerKey(55);
BankAccount b = new BankAccount(10000, k);
boolean success = db.insert(b);
```

Subtyping

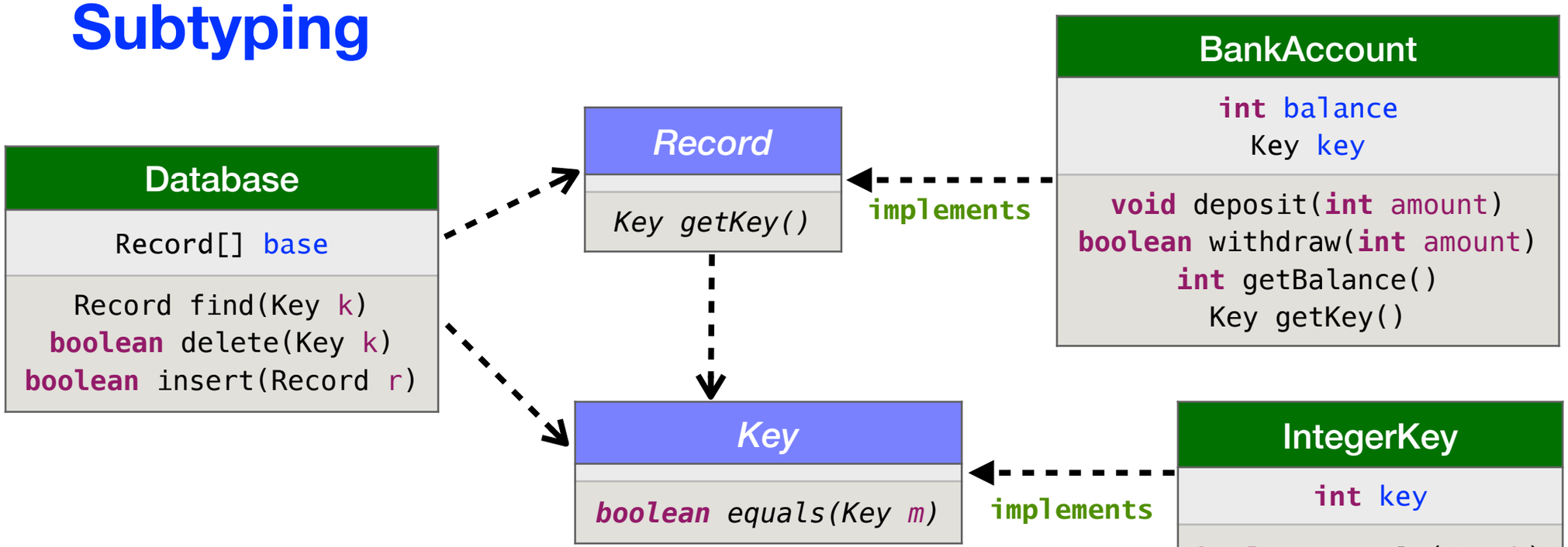



```
public class BankAccount implements Record
```

```
Database db = new Database(4);
IntegerKey k = new IntegerKey(55);
BankAccount b = new BankAccount(10000, k);
boolean success = db.insert(b);
```



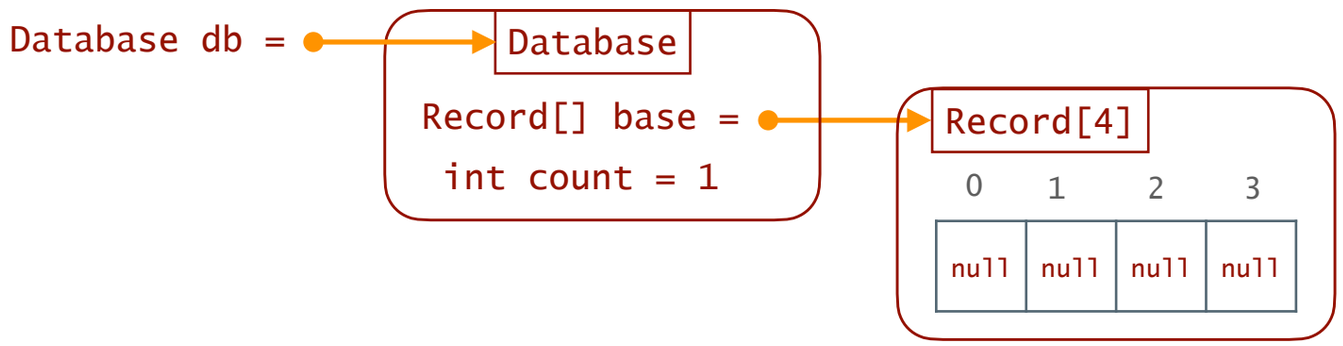
Subtyping



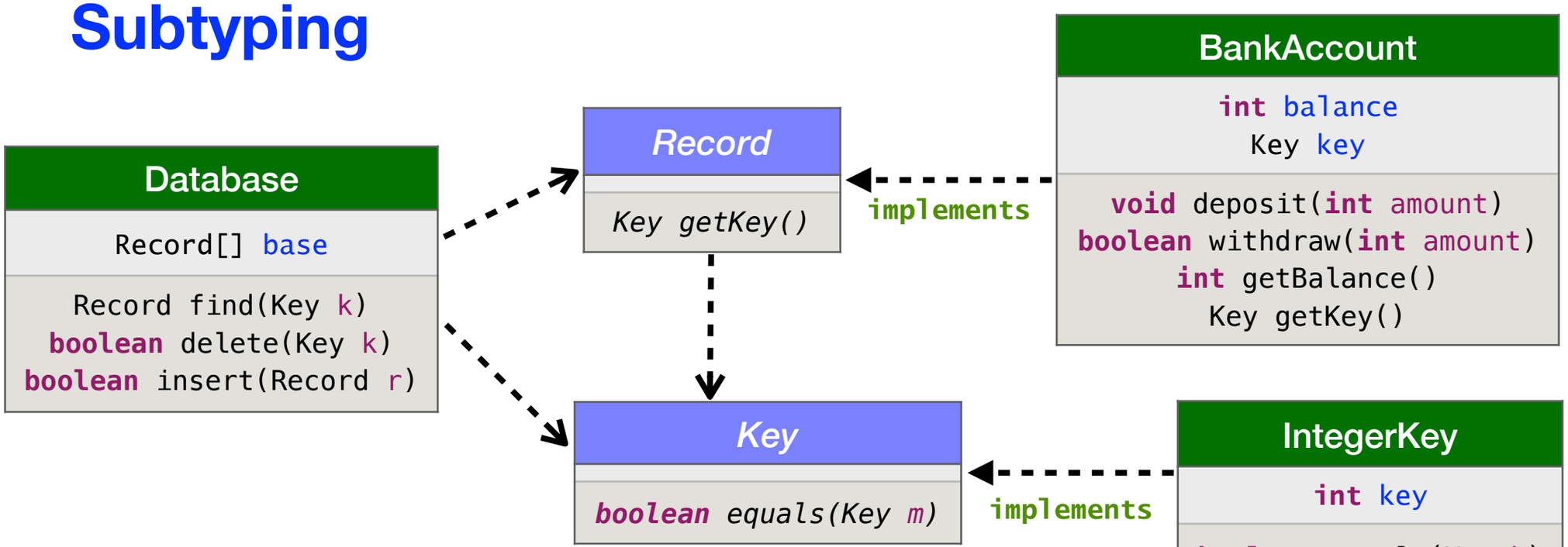

```
public class BankAccount implements Record
```

BankAccount <= Record

```
Database db = new Database(4);
IntegerKey k = new IntegerKey(55);
BankAccount b = new BankAccount(10000, k);
boolean success = db.insert(b);
```



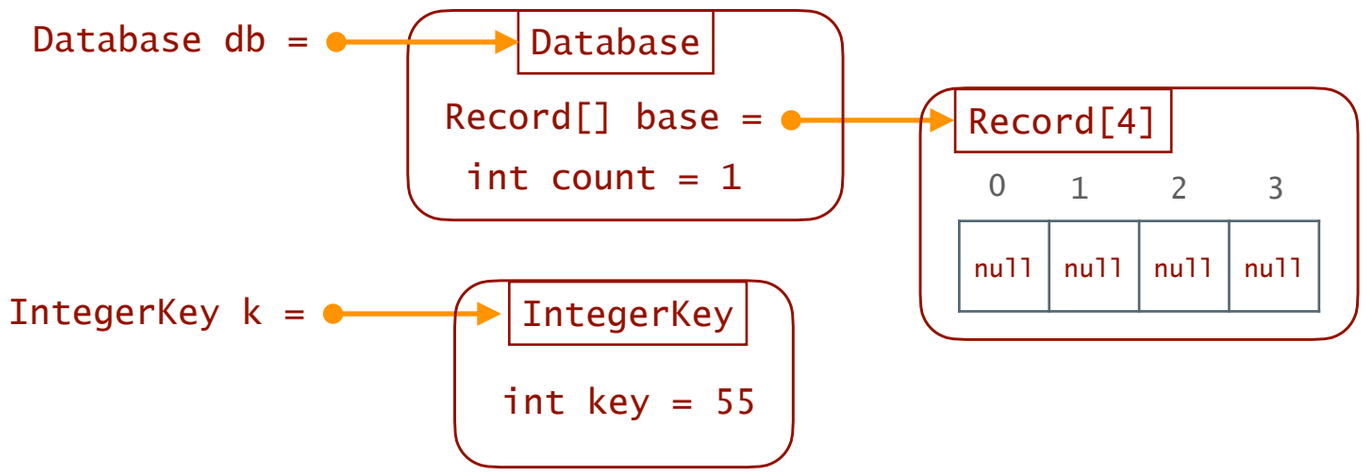
Subtyping



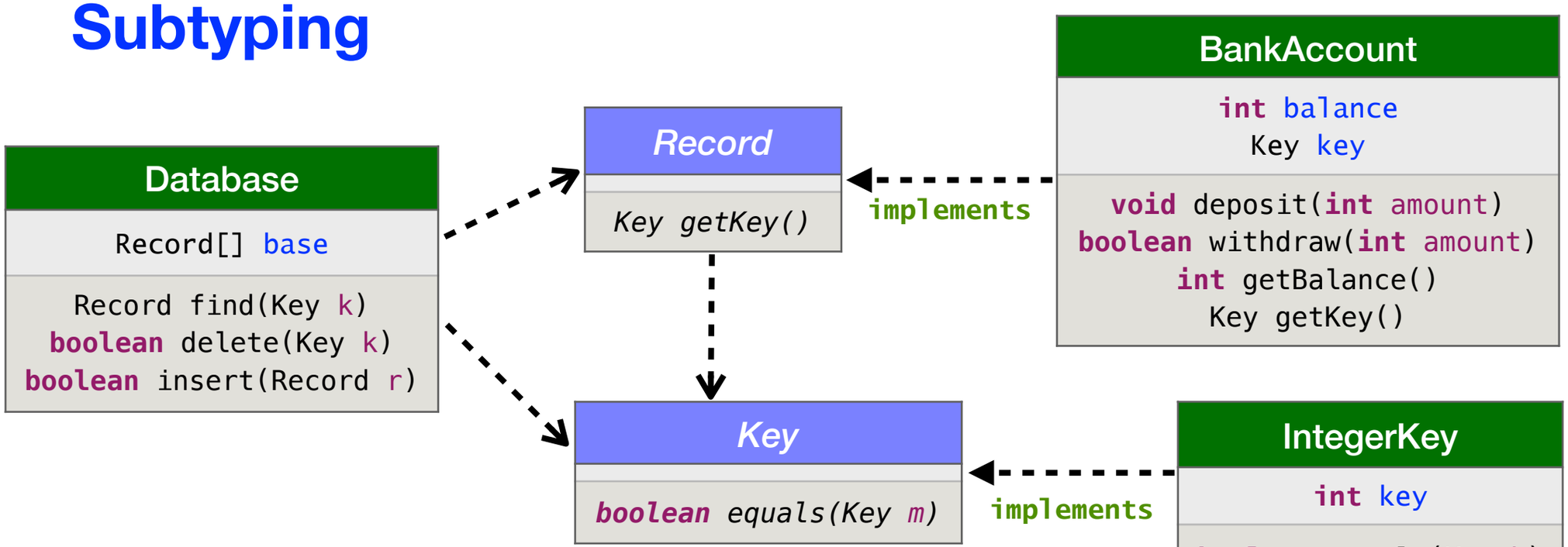

```
public class BankAccount implements Record
```

BankAccount <= Record

```
Database db = new Database(4);
IntegerKey k = new IntegerKey(55);
BankAccount b = new BankAccount(10000, k);
boolean success = db.insert(b);
```



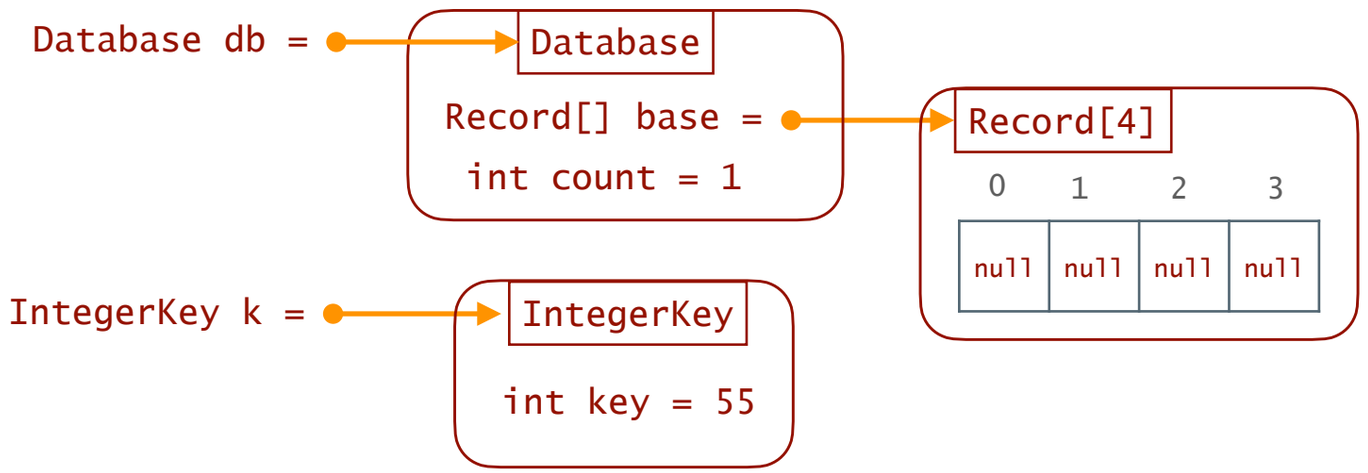
Subtyping



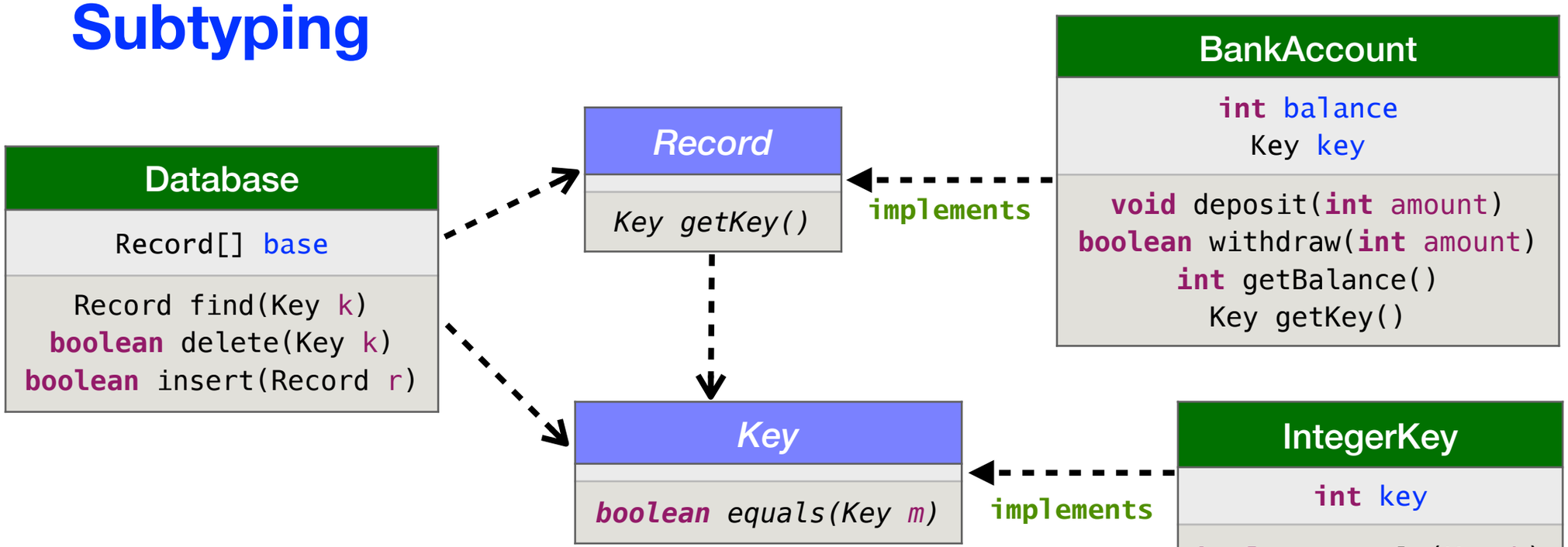

```
public class BankAccount implements Record
```

BankAccount <= Record

```
Database db = new Database(4);
IntegerKey k = new IntegerKey(55);
BankAccount b = new BankAccount(10000, k);
boolean success = db.insert(b);
```



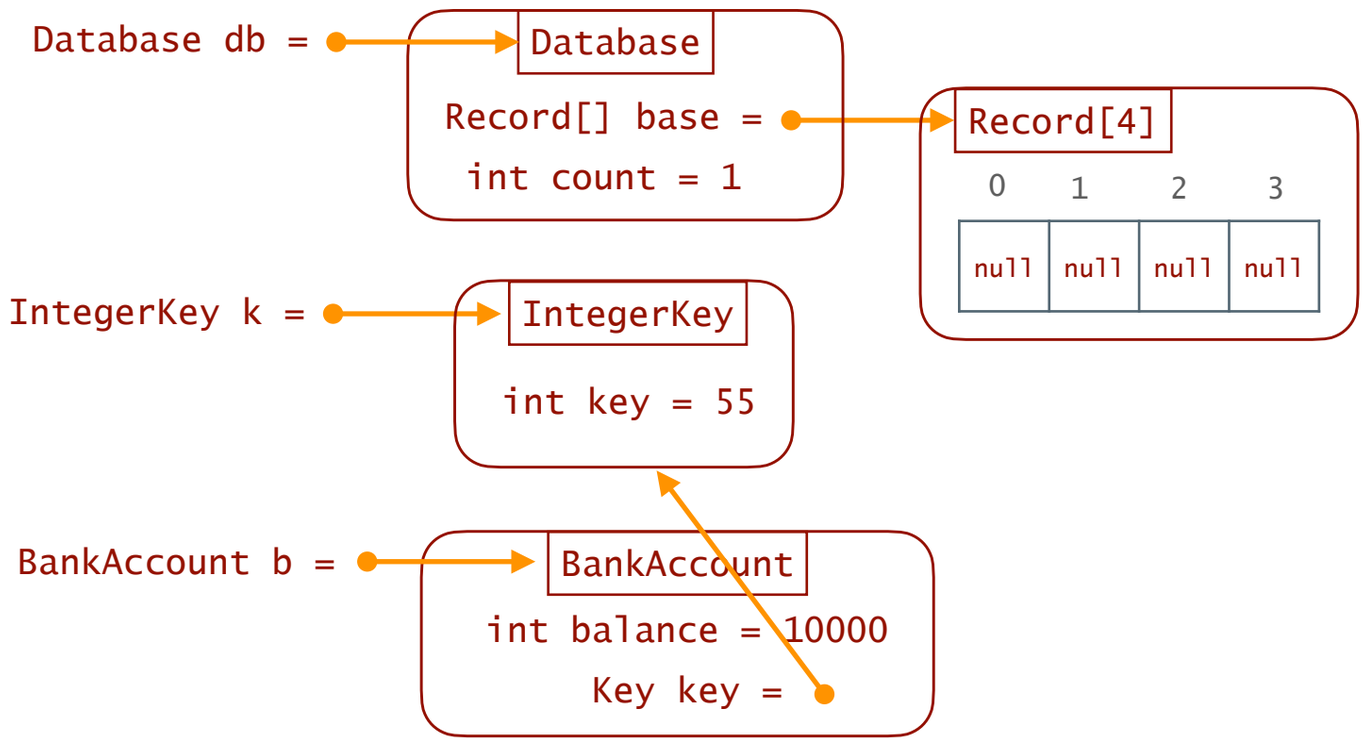
Subtyping



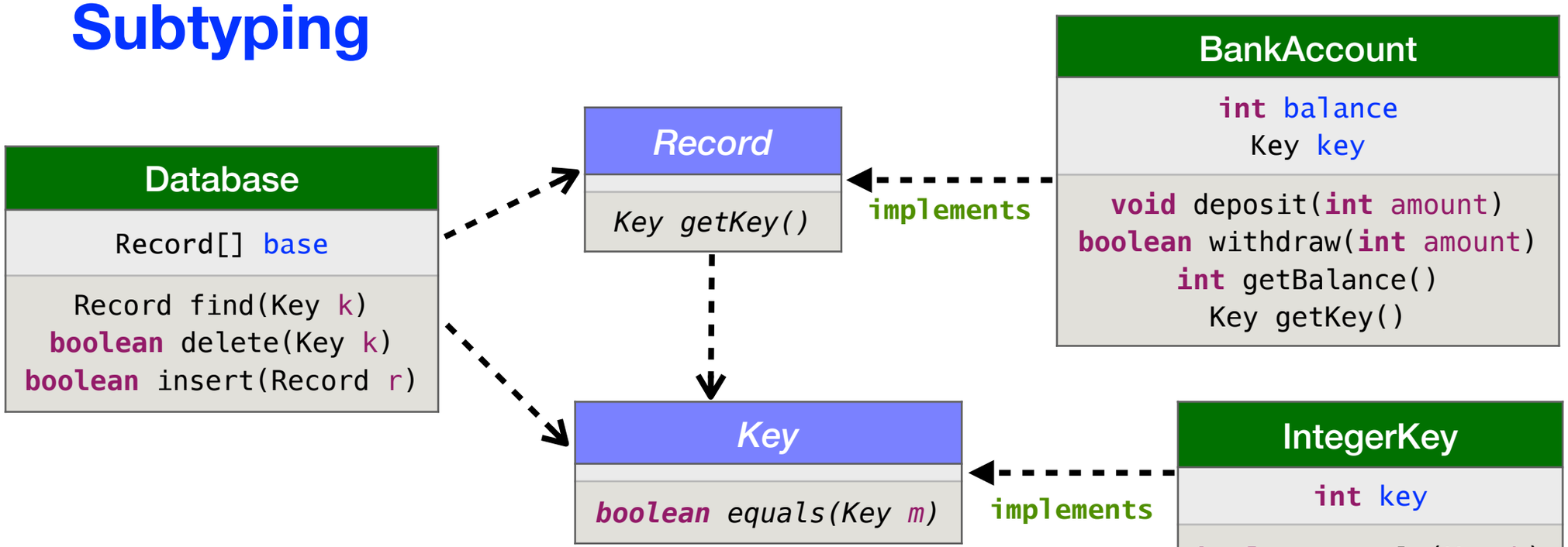

```
public class BankAccount implements Record
```

BankAccount <= Record

```
Database db = new Database(4);
IntegerKey k = new IntegerKey(55);
BankAccount b = new BankAccount(10000, k);
boolean success = db.insert(b);
```



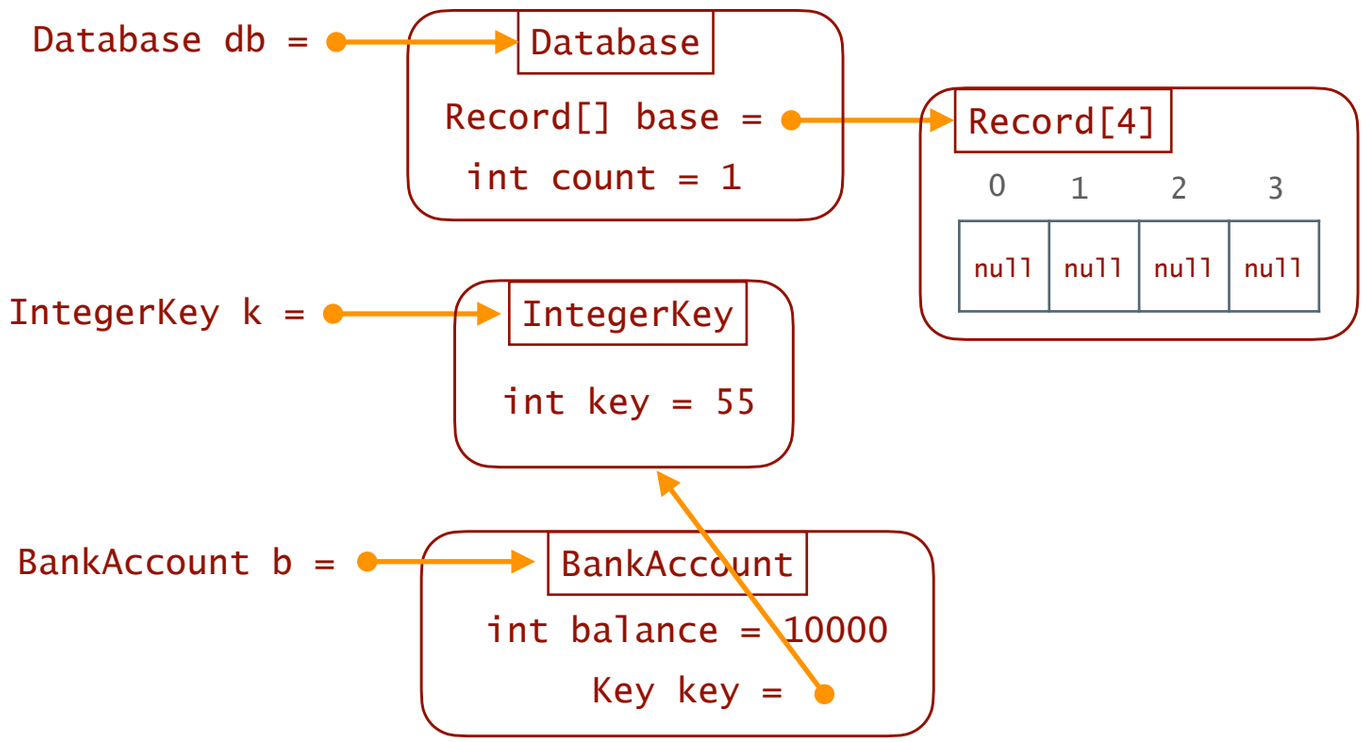
Subtyping



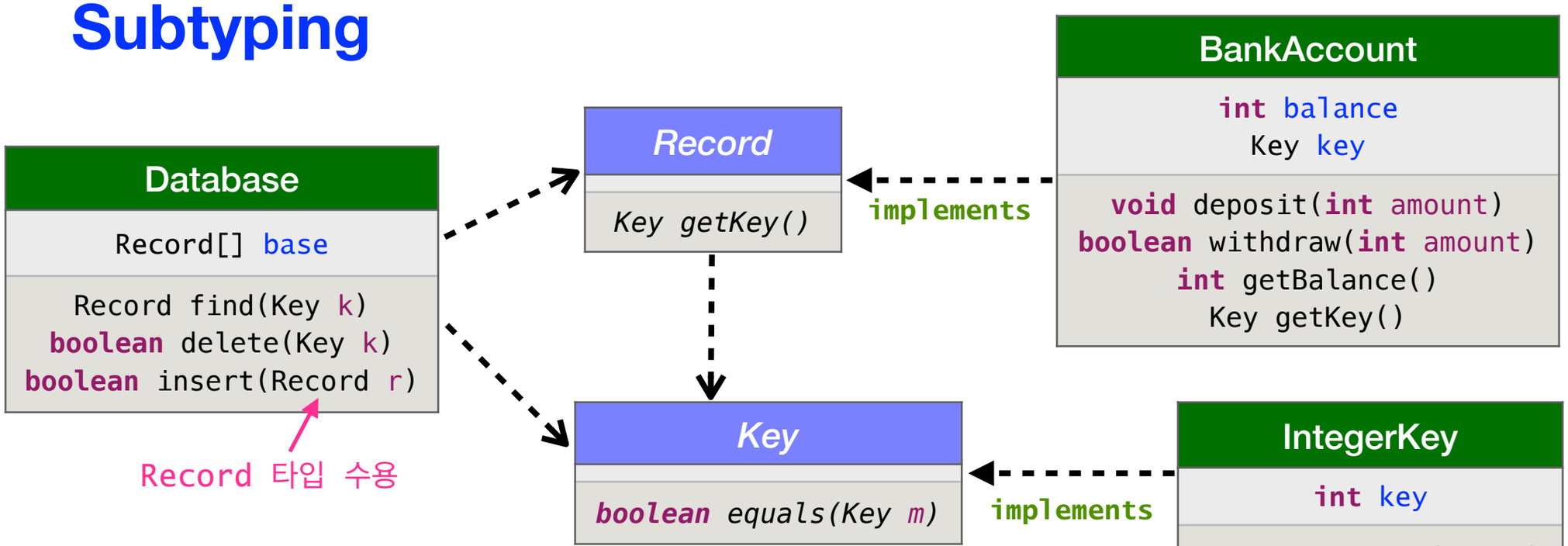

```
public class BankAccount implements Record
```

BankAccount <= Record

```
Database db = new Database(4);
IntegerKey k = new IntegerKey(55);
BankAccount b = new BankAccount(10000, k);
boolean success = db.insert(b)
```



Subtyping

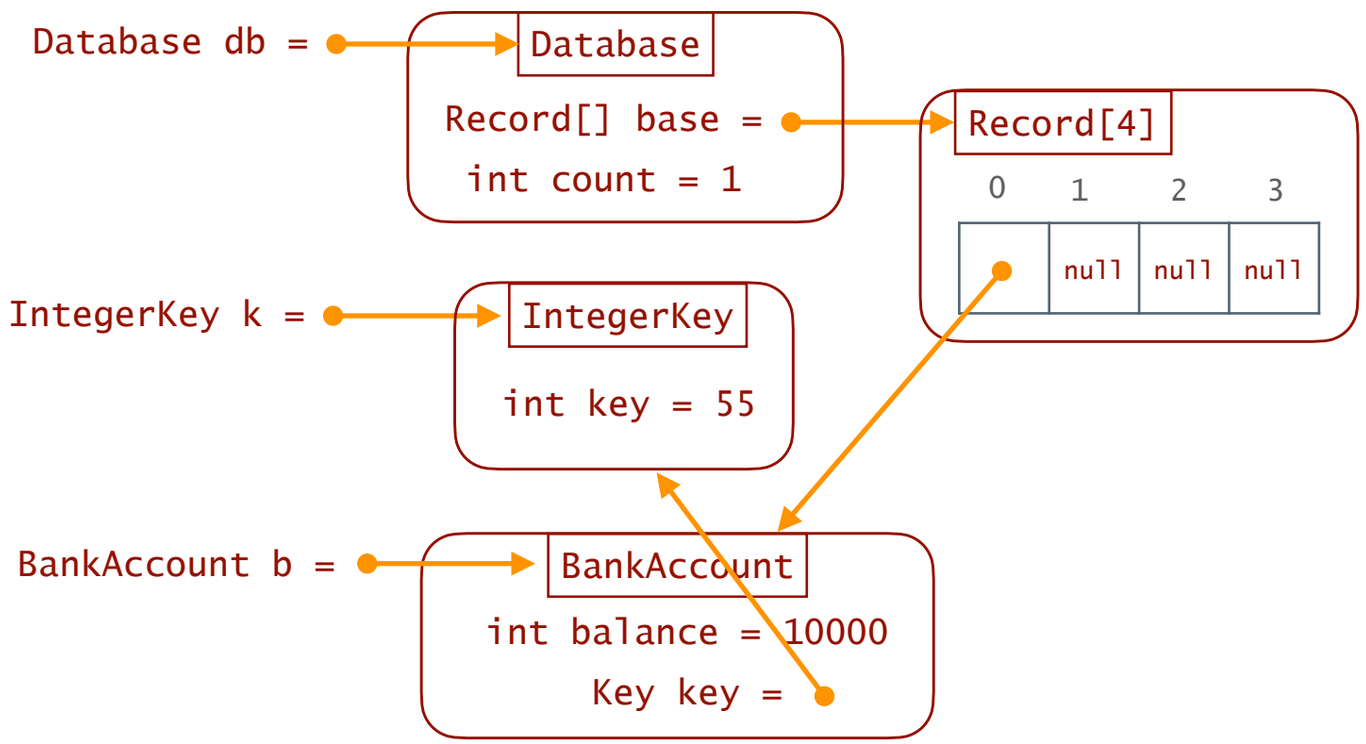



```
public class BankAccount implements Record
```

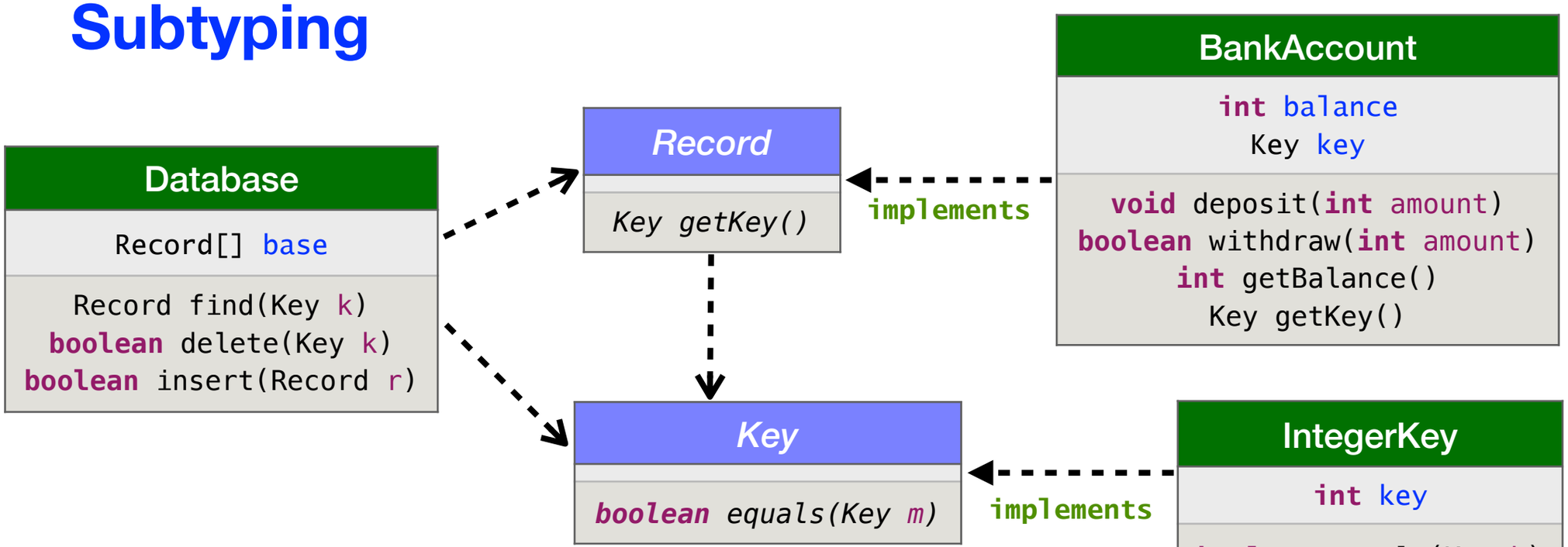
BankAccount <= Record

```
Database db = new Database(4);
IntegerKey k = new IntegerKey(55);
BankAccount b = new BankAccount(10000, k);
boolean success = db.insert(b)
```

BankAccount 타입



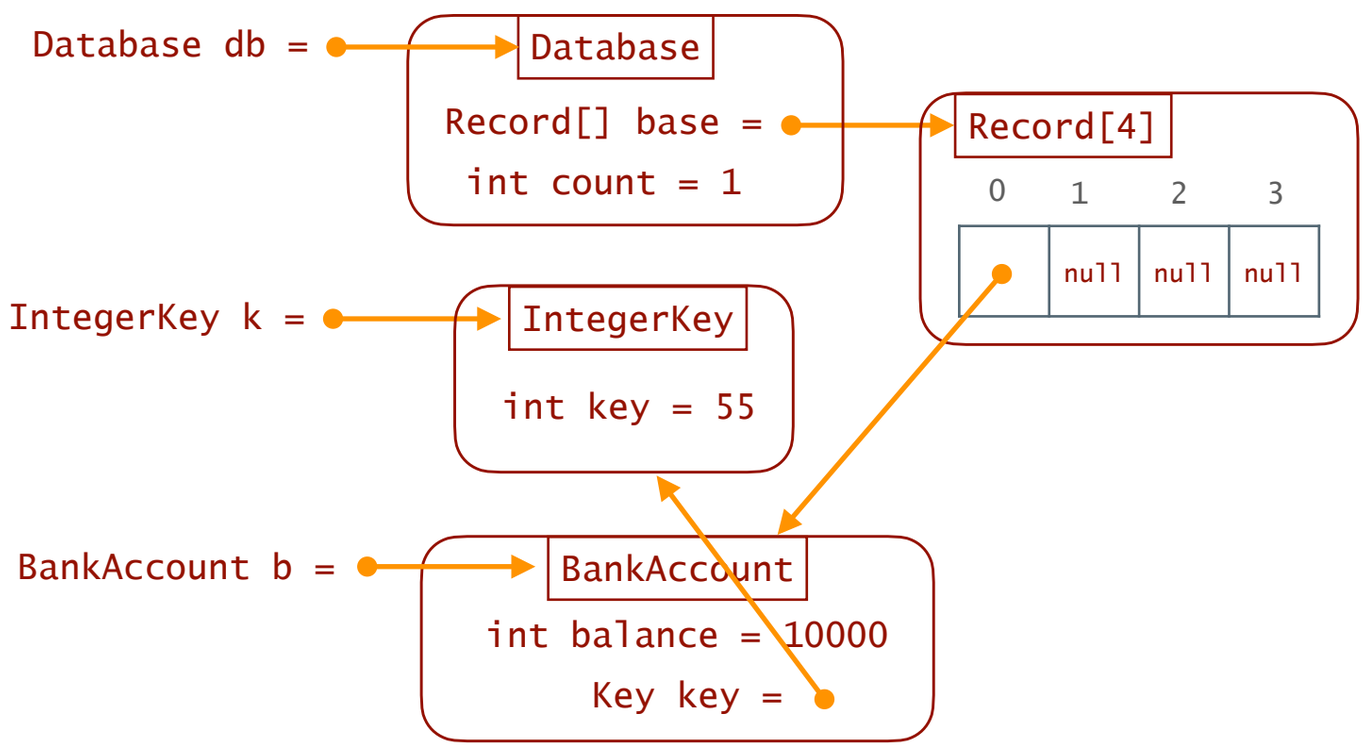
Subtyping



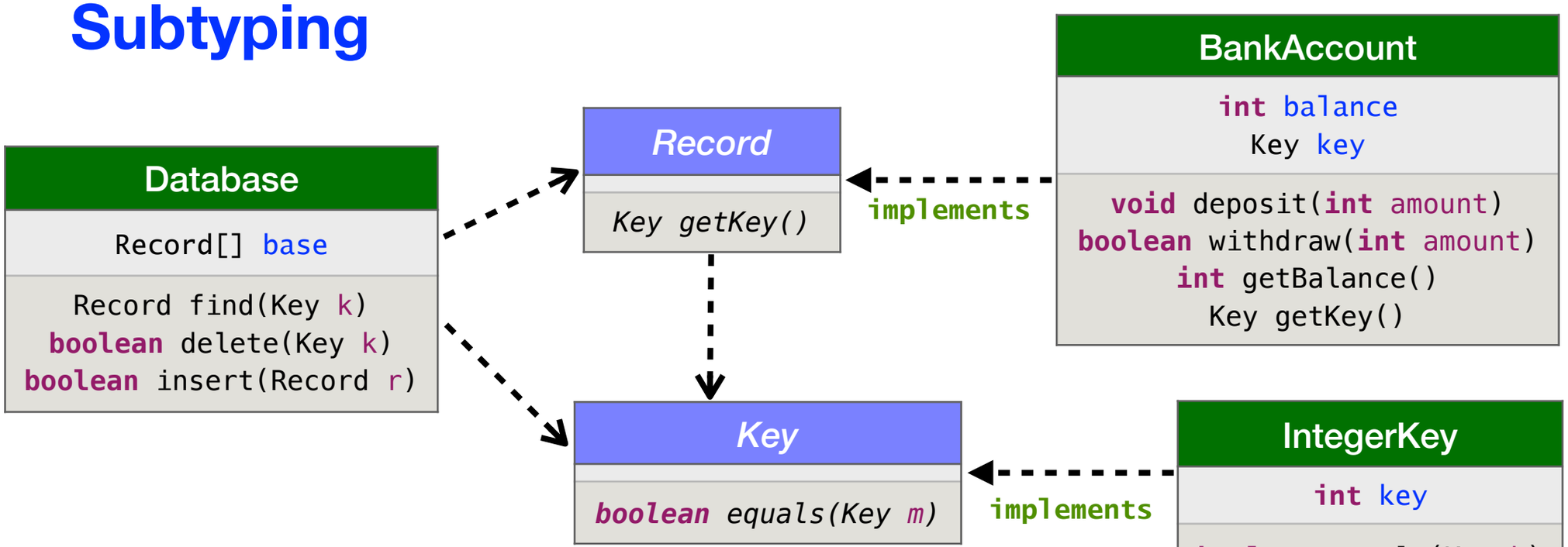

```
public class IntegerKey implements Key
```

IntegerKey <= Key

```
Record r = db.find(k);

… r.getBalance() …

… ((BankAccount)r).getBalance() …;
```



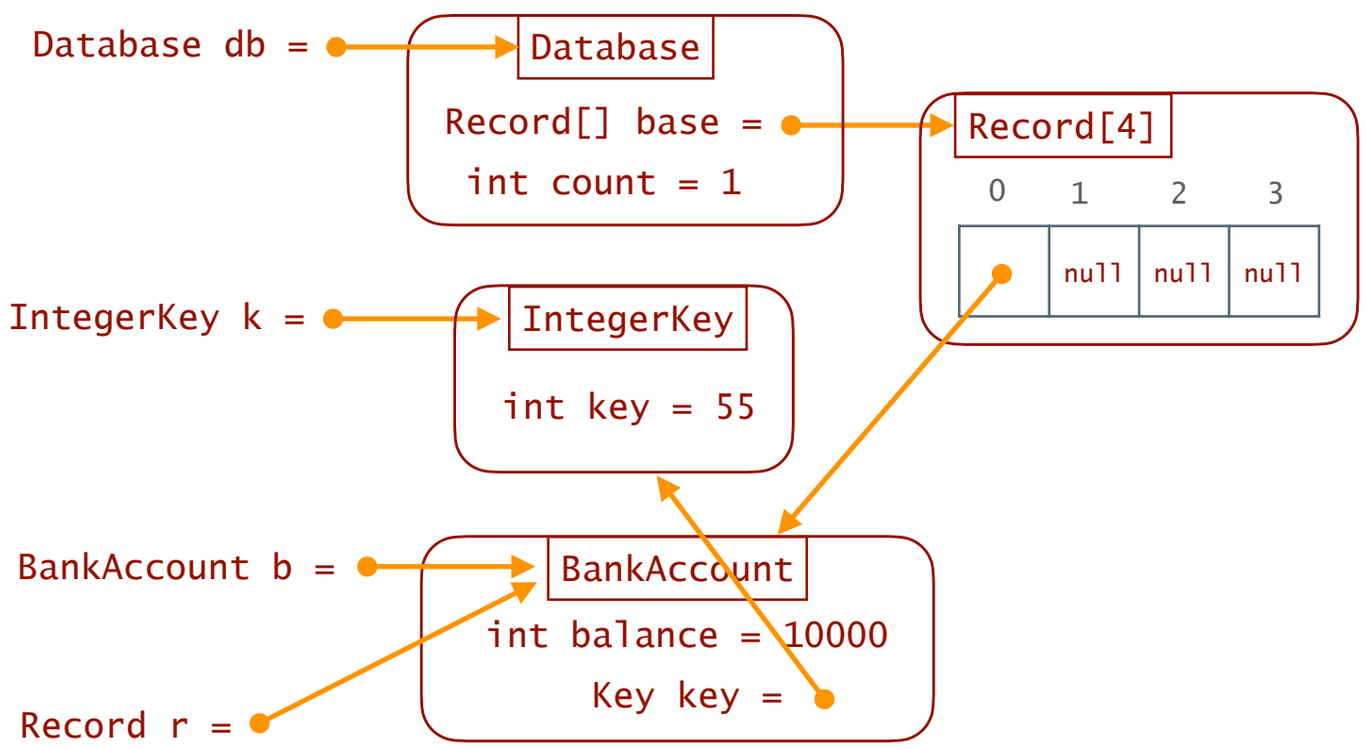
Subtyping



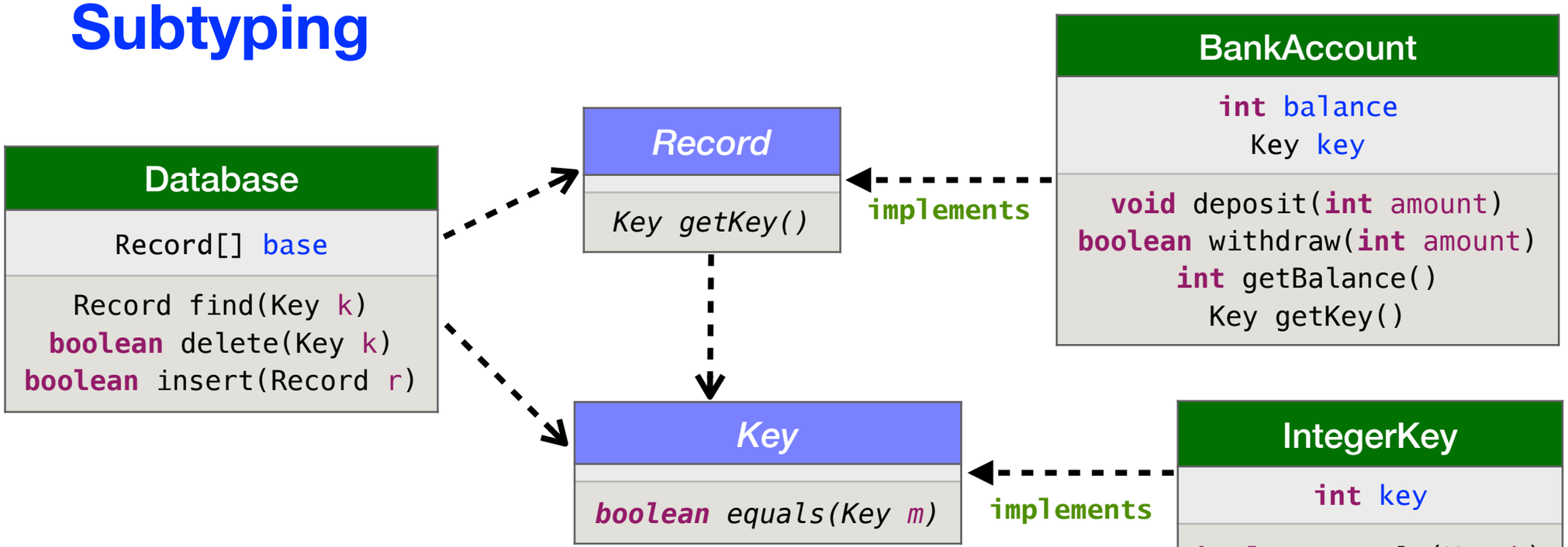

```
public class IntegerKey implements Key
```

IntegerKey <= Key

```
Record r = db.find(k);

… r.getBalance() …

… ((BankAccount)r).getBalance() …;
```



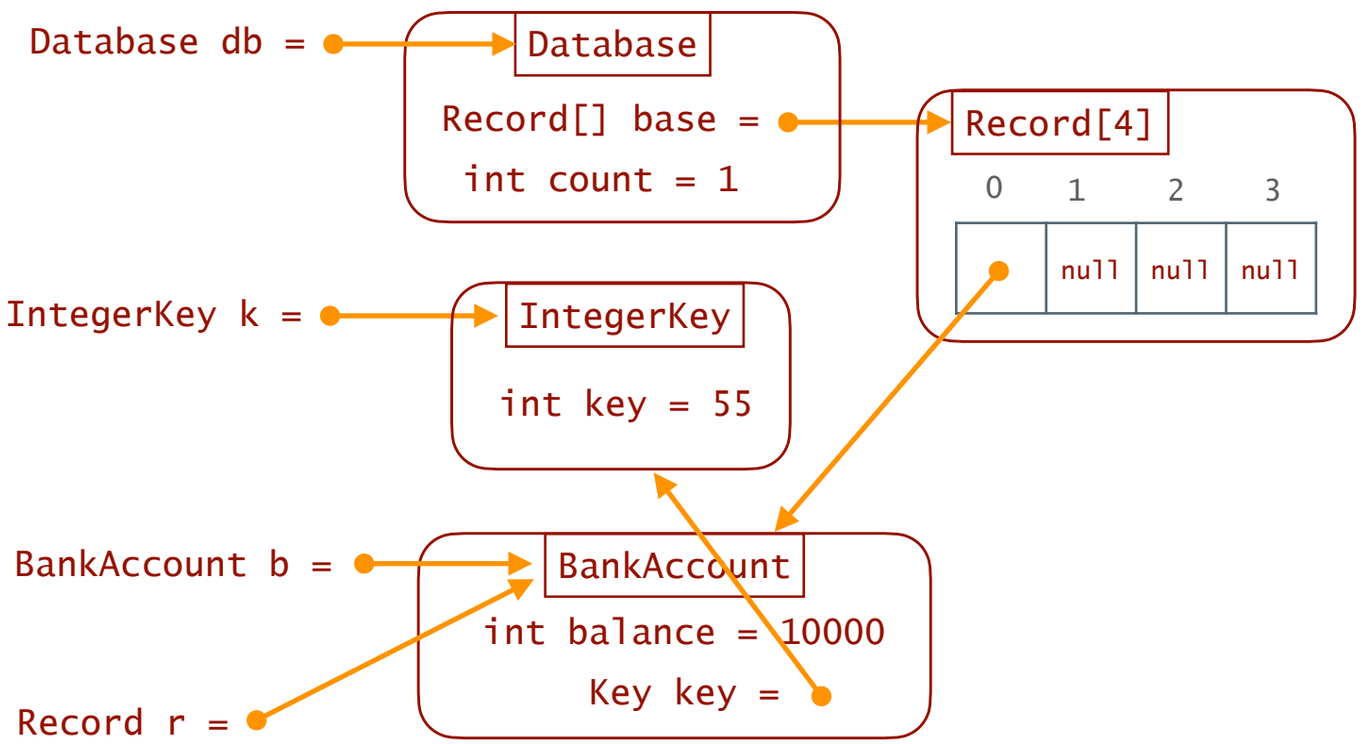
Subtyping



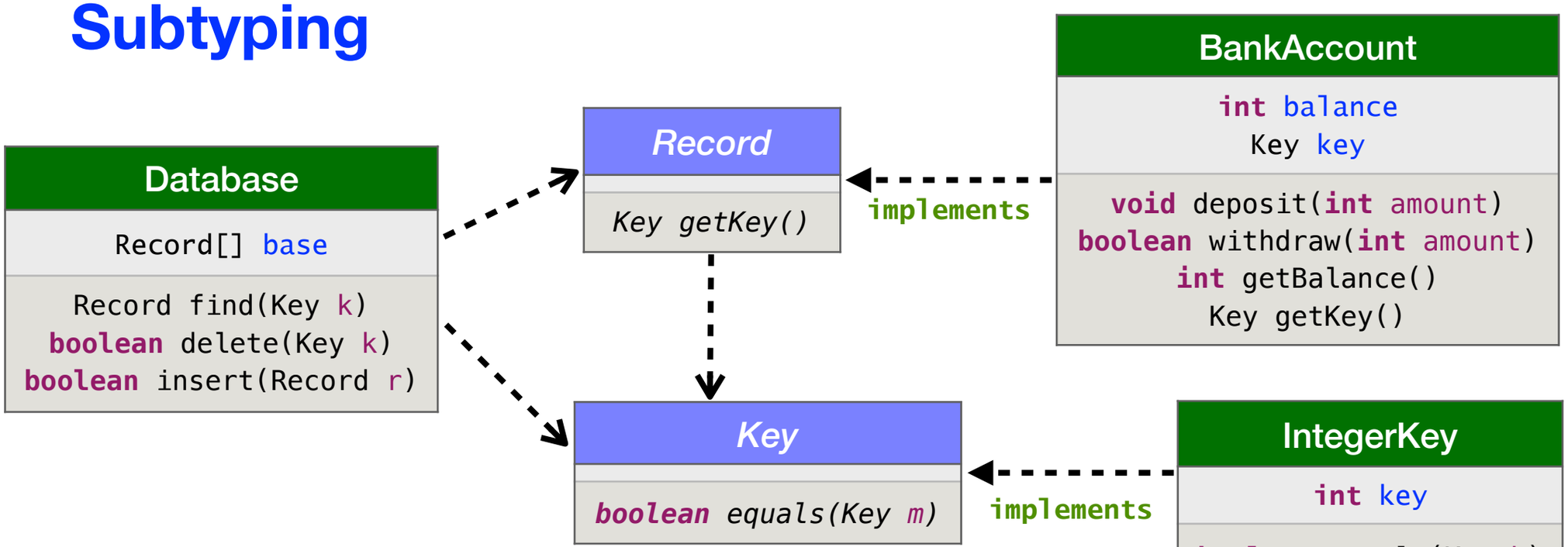

```
public class IntegerKey implements Key
```

IntegerKey <= Key

```
Record r = db.find(k);

… r.getBalance() …

… ((BankAccount)r).getBalance() …;
```



Subtyping

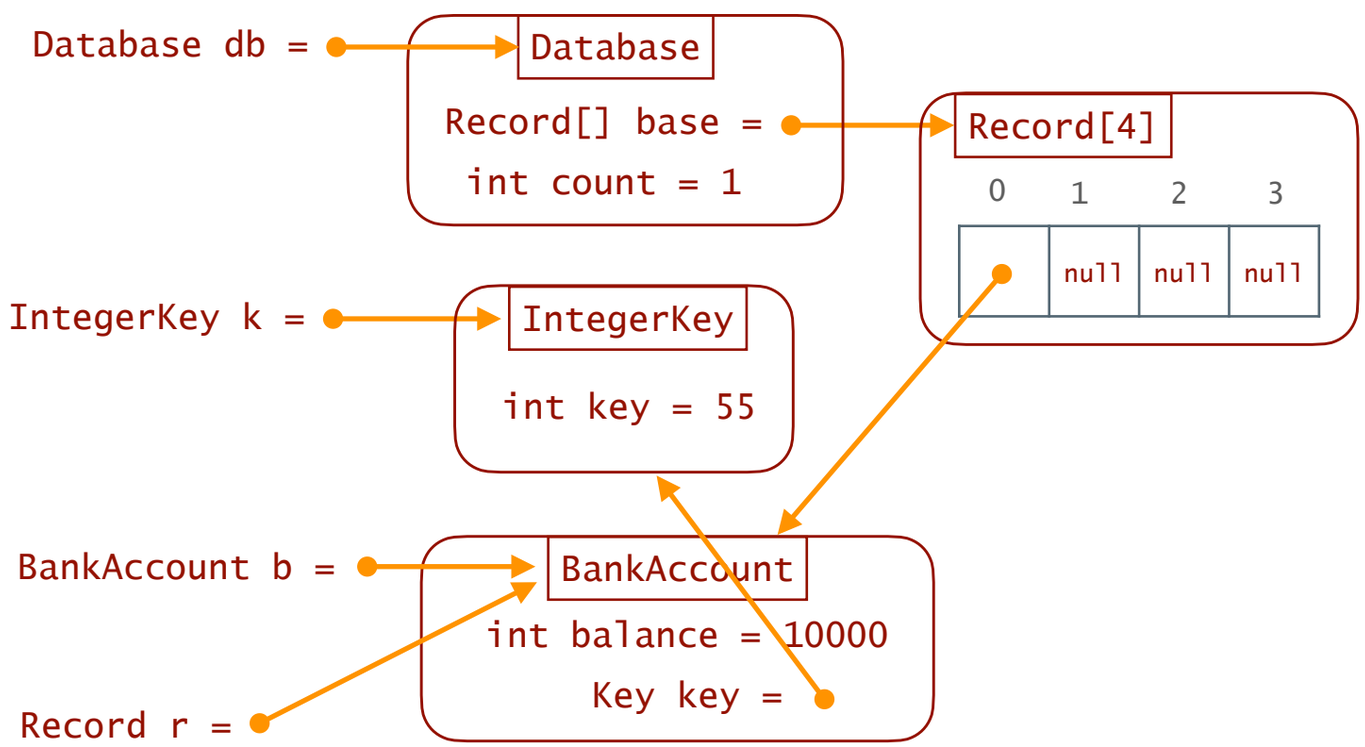



```
public class IntegerKey implements Key
```

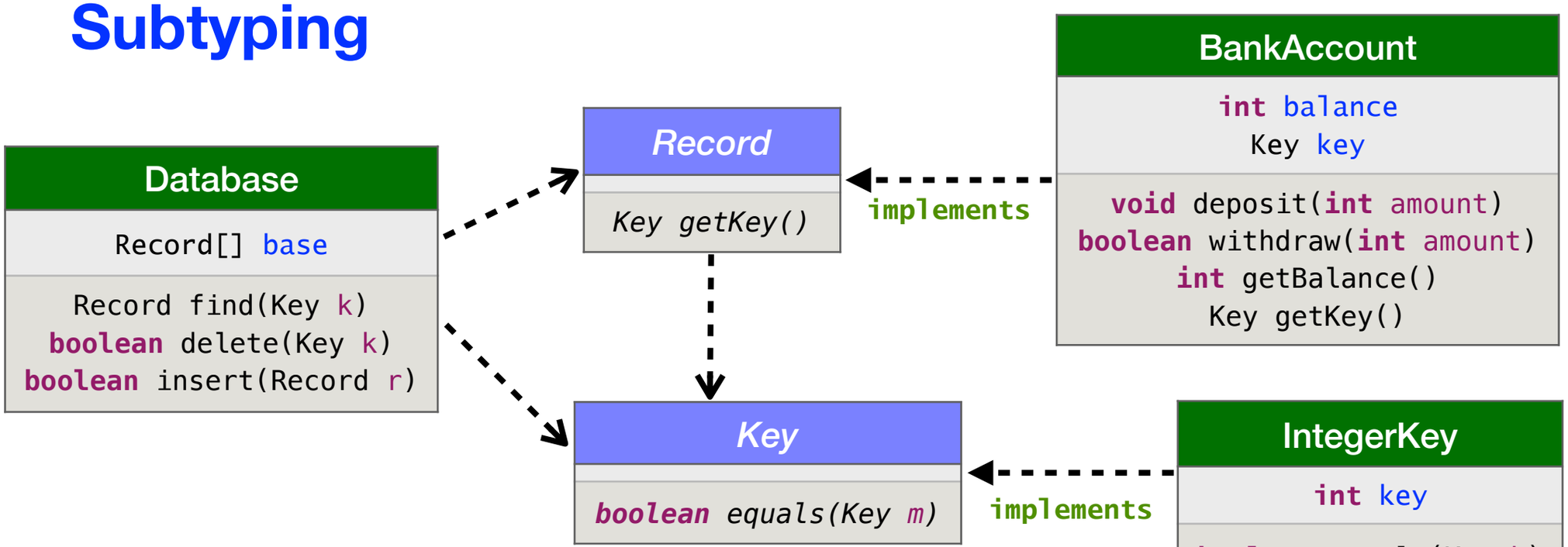
IntegerKey <= Key

```
Record r = db.find(k);

… r.getBalance() …

… ((BankAccount)r).getBalance() …;
```

타입오류! - Record 인터페이스는
getBalance() 메소드를 모른다.



Subtyping

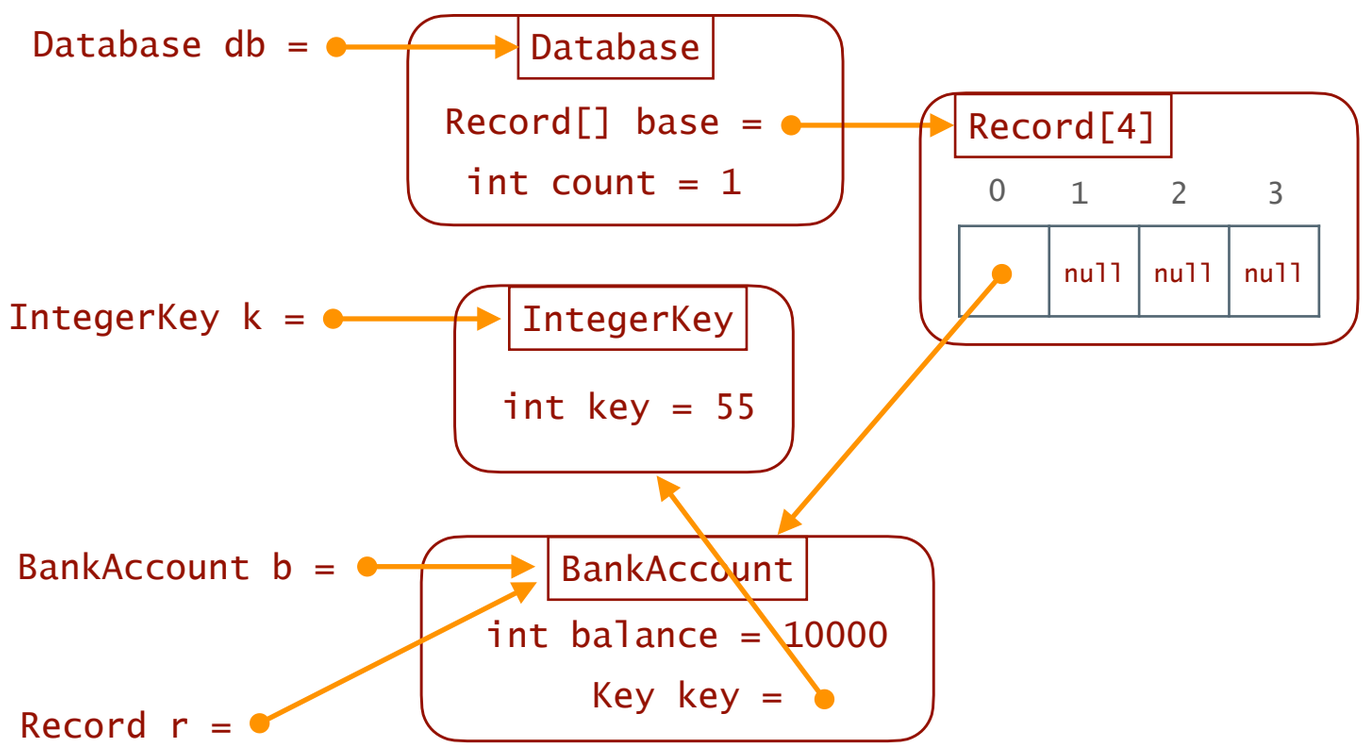



```
public class IntegerKey implements Key
```

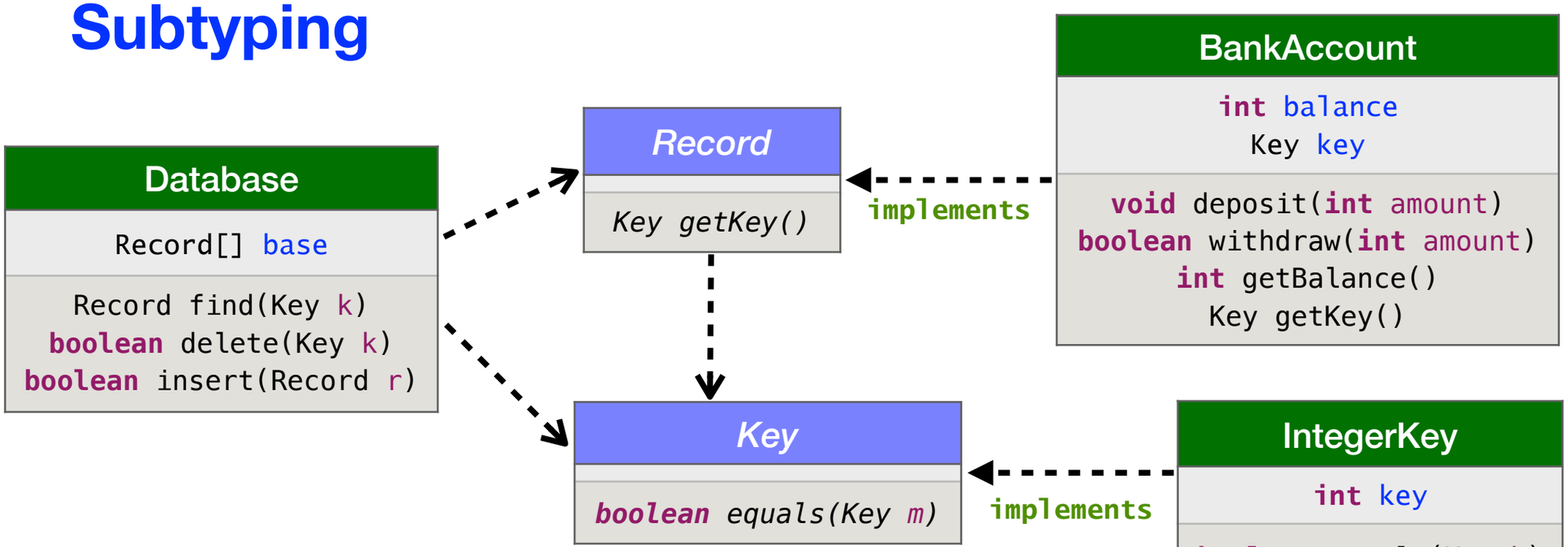
IntegerKey <= Key

```
Record r = db.find(k);

… r.getBalance() …

… ((BankAccount)r).getBalance() …;
```

타입 캐스트하여 컴파일러 통과



Subtyping




```
public class IntegerKey implements Key
```

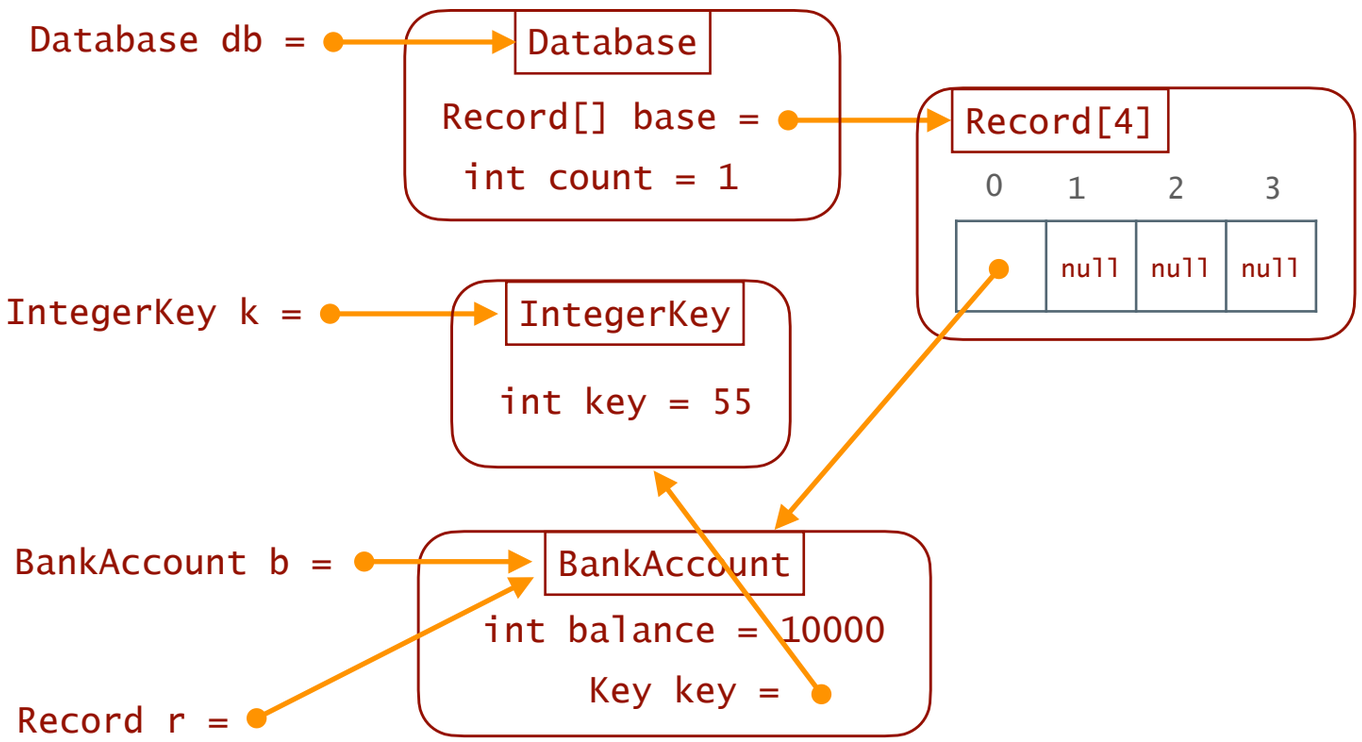
IntegerKey <= Key

```
Record r = db.find(k);

… r.getBalance() …

… ((BankAccount)r).getBalance() …;
```

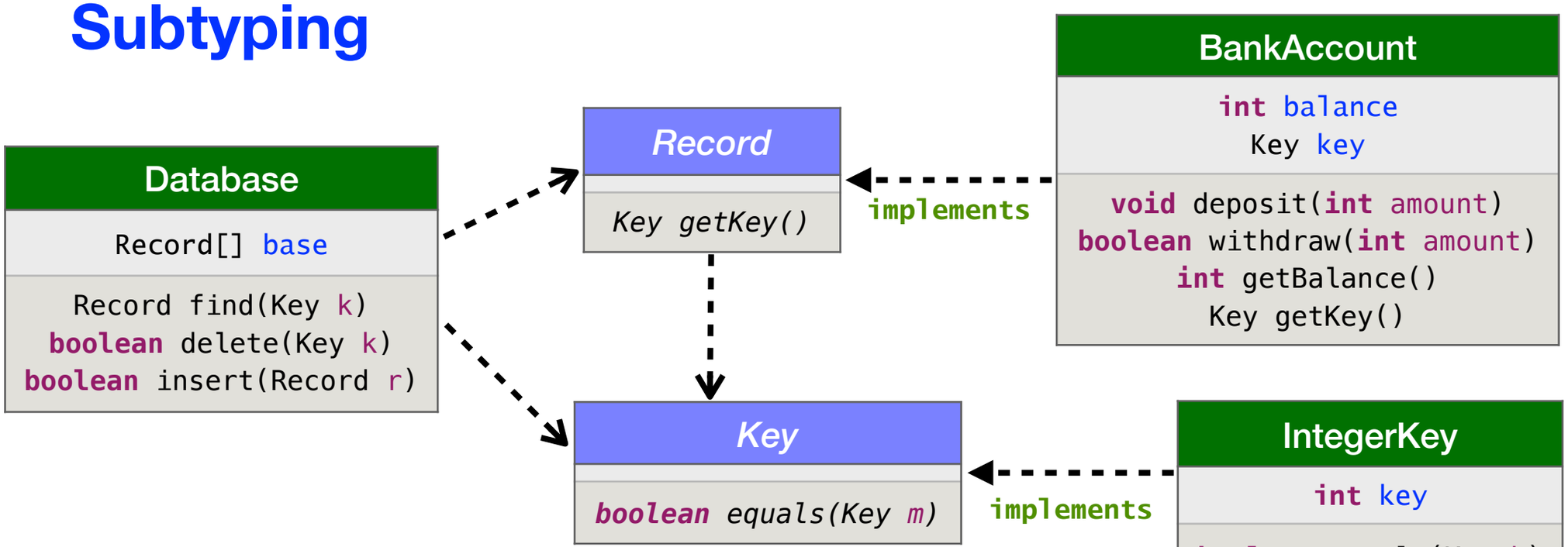
BankAccount 아님 IntegerKey 아님

```
Record r = db.find(mysterious_key);
… ((BankAccount)r).getBalance() … ;
```

컴파일러 통과,
하지만 실행 오류 발생



Subtyping



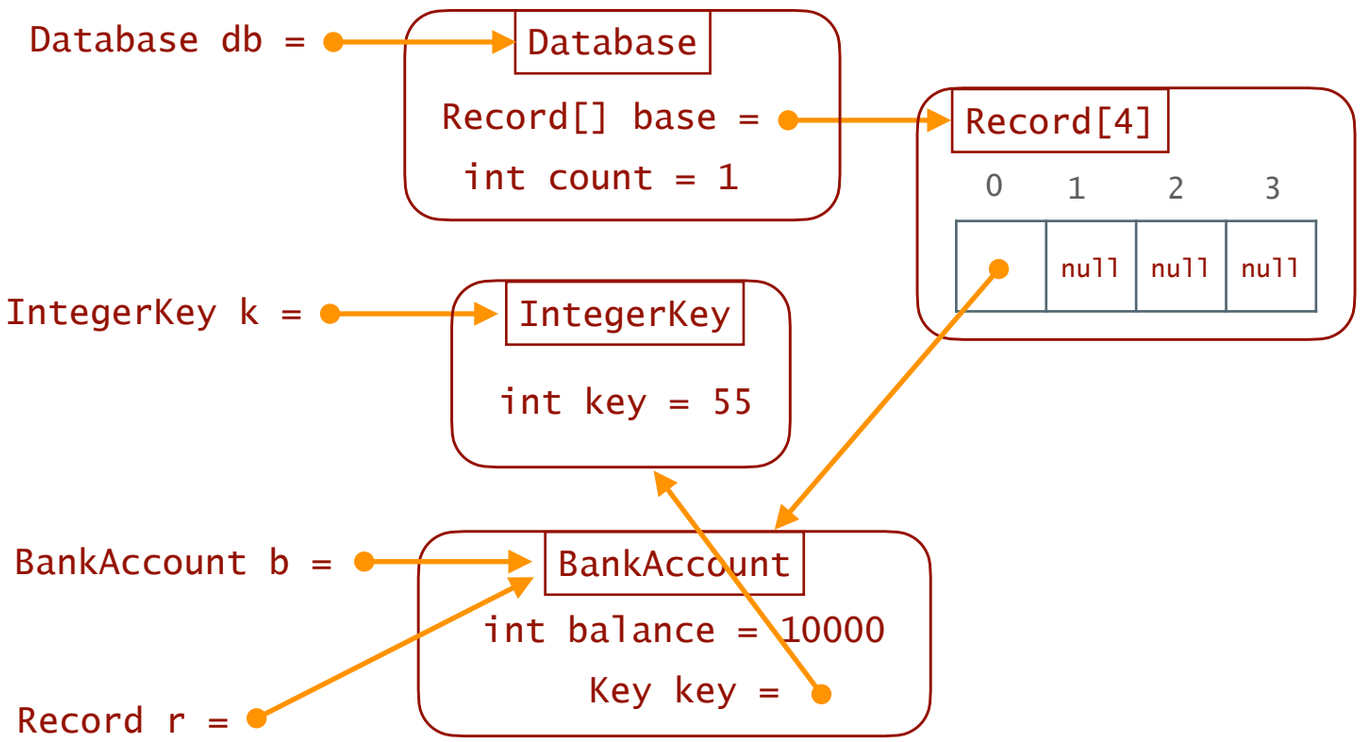

```
public class IntegerKey implements Key
```

IntegerKey <= Key

```
Record r = db.find(k);

… r.getBalance() …

… ((BankAccount)r).getBalance() …;
```

```
Record r = db.find(mysterious_key);
if (r instanceof BankAccount)
    … ((BankAccount)r).getBalance() … ;
else
    System.out.println("취급 불가 Record");
```

r이 BankAccount 인지 사전 확인 하여 실행 오류 방지



컴파일러가 잡아내지 못하는 타입 오류

```
public class StringKey implements Key {
    private String s;

    public StringKey(String j) {
        s = j;
    }

    public boolean equals(Key m) {
        return s == ((StringKey)m).getString();
    }

    public String getString() {
        return s;
    }
}
```

```
IntegerKey k1 = new IntegerKey(3);
StringKey k2 = new StringKey("three");
boolean answer = k2.equals(k1);
```

컴파일 OK,
그러나 실행중 오류 발생

컴파일러가 타입 오류를 찾을 수 없음 => 프로그래머가 챙겨야 할 몫!

수리 방법

```
public class StringKey implements Key {
    private String s;

    public StringKey(String j) {
        s = j;
    }

    public boolean equals(Key m) {
        if (m instanceof StringKey)
            return s.equals(((StringKey)m).getString());
        else
            return false;
    }

    public String getString() {
        return s;
    }
}
```

```
IntegerKey k1 = new IntegerKey(3);
StringKey k2 = new StringKey("three");
boolean answer = k2.equals(k1);
```

컴파일 OK, 실행 OK

Lab#1. 서브 타입 이해하기

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

    public Person(String n) {
        name = n;
    }

    public String getName() {
        return name;
    }

    public boolean sameName(Person other) {
        return getName().equals(other.getName();
    }
}
```

```java
public class PersonFrom extends Person {
    private String city;

    public PersonFrom(String n, String c) {
        super(n);
        city = c;
    }

    public String getCity() {
        return city;
    }

    public boolean same(PersonFrom other) {
        return sameName(other) &&
               city.equals(other.getCity());
    }
}
```

```java
Person p = new Person("마음");
Person q = new PersonFrom("소리","서울");
```

다음 각 문장을 이해하고, Java 컴파일러를 통과하는 문장을 고르고, 그 문장이 무엇을 프린트할지 예측해보자.

- `System.out.println(p.sameName(q));`
- `Person x = q; System.out.println(x.getName());`
- `PersonFrom x = p; System.out.println(x.getCity());`
- `Person x = q; System.out.println(x.getCity());`
- `System.out.println(q.same(p));`

Lab#2. instanceof

1. 앞에서 구현한 IntegerKey와 같은 요령으로 StringKey를 구현하자.
2. 다음 코드를 실행하면 어떤 결과가 실행창에 프린트될까?

```
Database db = new Database(4);

BankAccount a1 = new BankAccount(50000, new IntegerKey(55));
Key k = new StringKey("열려라");
BankAccount a2 = new BankAccount(10000, k);
boolean transaction1 = db.insert(a1);
boolean transaction2 = db.insert(a2);

Record p = db.find(k);
BankAccount q = (BankAccount)p;
System.out.println(q.getBalance());

Key k = q.getKey();
if (k instanceof IntegerKey)
    System.out.println(((IntegerKey)k).getInt());
else if (k instanceof StringKey)
    System.out.println(((StringKey)k).getString());
else
    System.out.println("모르는 Key 출현 오류");
```

Lab#3. Dealer 클래스 구현

```
1  public interface CardPlayerBehavior {
2
3      /** wantsACard - 카드 한 장을 받겠는지 답한다.
4       * @return 카드를 받고 싶으면 true, 아니면 false */
5      public boolean wantsACard();
6
7      /** receiveCard - 카드 한 장을 받아서 손에 넣는다.
8       * @return 카드 수령 성공이면 true, 실패이면 false */
9      public boolean receiveCard(Card c);
10 }
```

class	Dealer	카드 딜러
method	`void dealTo(CardPlayerBehavior p)`	카드를 한 장씩 매번 물어보면서 원하는 만큼 p에게 준다.
	`void dealOneTo(CardPlayerBehavior p)`	카드를 한 장 p에게 준다.
collaborators	CardPlayerBehavior, CardDeck, Card	

Abstract Class

일부 메소드의 몸체가 비어있는 클래스

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

    public Person(String n) {
        name = n;
    }

    public String getName() {
        return name;
    }

    public abstract String getAddress();

    ……
}
```

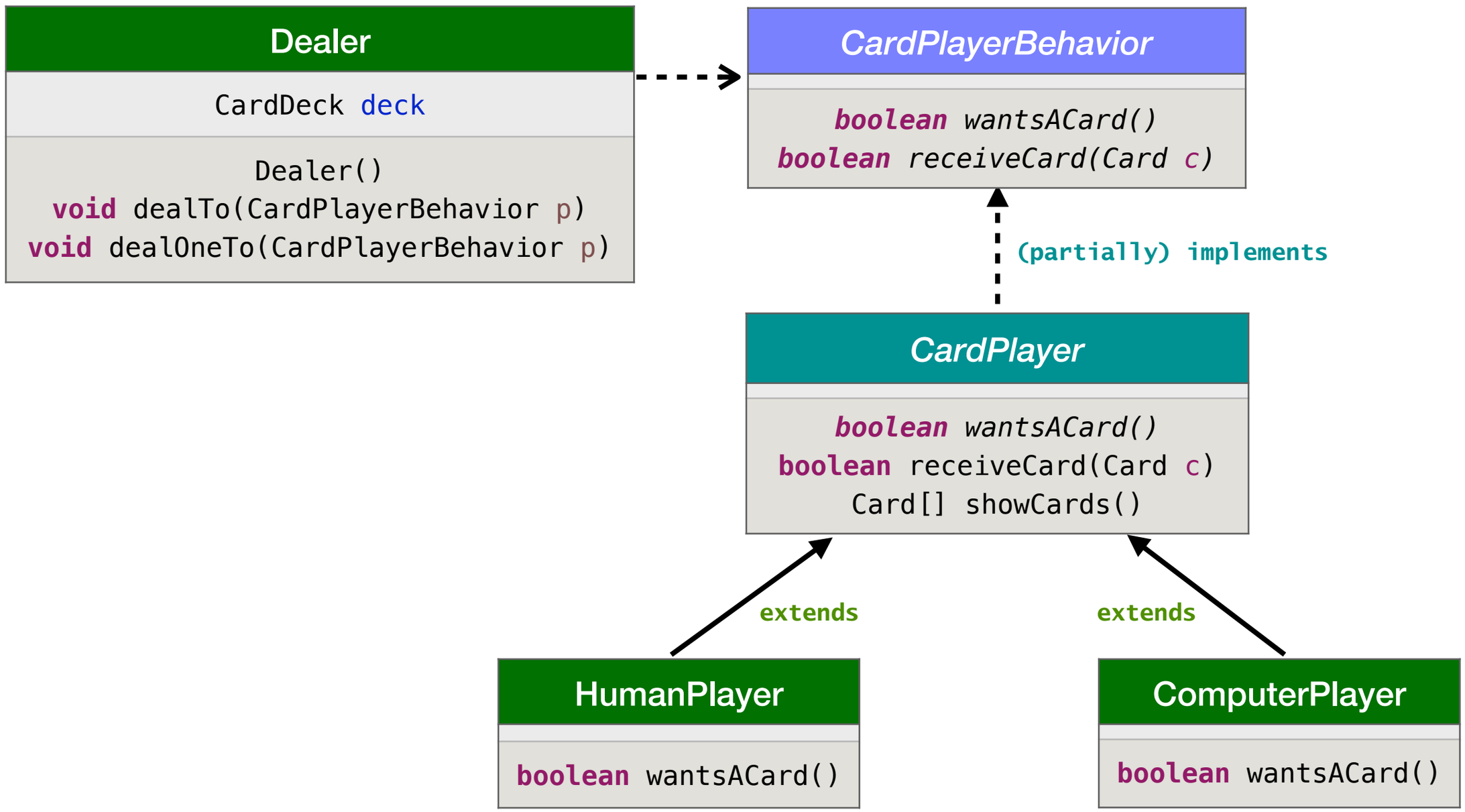
- new Person() 불가!
- extends 가능

```
public class PersonAddress extends Person {
    private String address;

    public PersonAddress(String n, String a) {
        super(n);
        address = a;
    }

    public String getAddress() {
        return address;
    }

    ……
}
```

```
public class PersonAddrInt extends Person {
    private int address;

    public PersonAddrInt(String n, int a) {
        super(n);
        address = a;
    }

    public String getAddress() {
        return "" + address;
    }

    ……
}
```

Lab#4. Card Players 구현

Arhitecture for Dealer and Card Players



```
ComputerPlayer p = new ComputerPlayer(3);
HumanPlayer h = new HumanPlayer(3);
CardPlayer player1 = p;
CardPlayerBehavior player2 = player1;
```

```
if (player2 instanceof CardPlayer) {
    ((CardPlayer)player2).showcards();
}
```

ComputerPlayer p ==

HumanPlayer h ==

CardPlayer player1 ==

CardPlayerBehavior player2 ==

Card[3]

ComputerPlayer

```
public boolean wantsACard { . . . }

// from CardPlayer
Card[] hand ==
int card_count == 0
public boolean receiveCard(Card c) { . . . }
public Card[] showCards() { . . . }
```

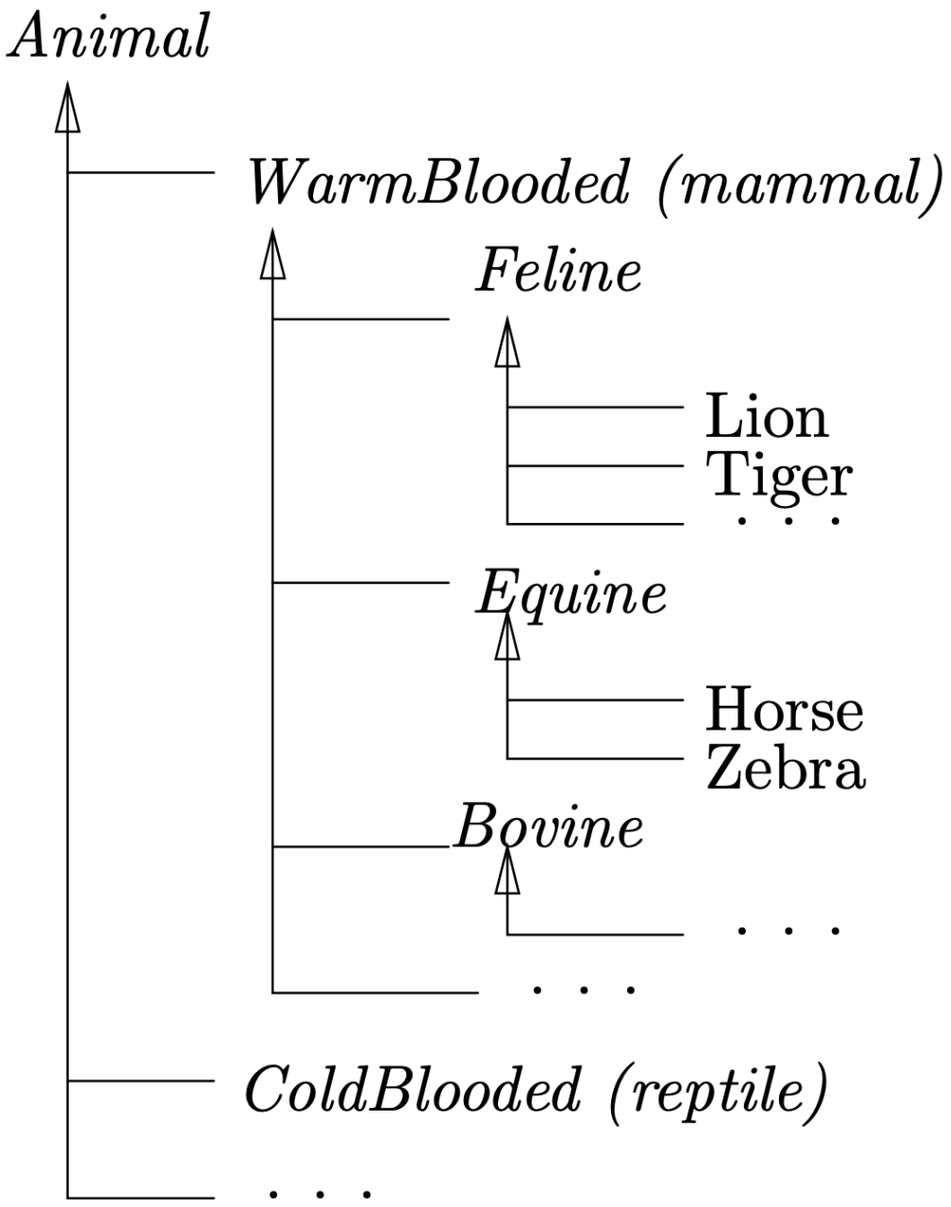
HumanPlayer

```
public boolean wantsACard { . . . }

// from CardPlayer
Card[] hand ==
int card_count == 0
public boolean receiveCard(Card c) { . . . }
public Card[] showCards() { . . . }
```

Card[3]

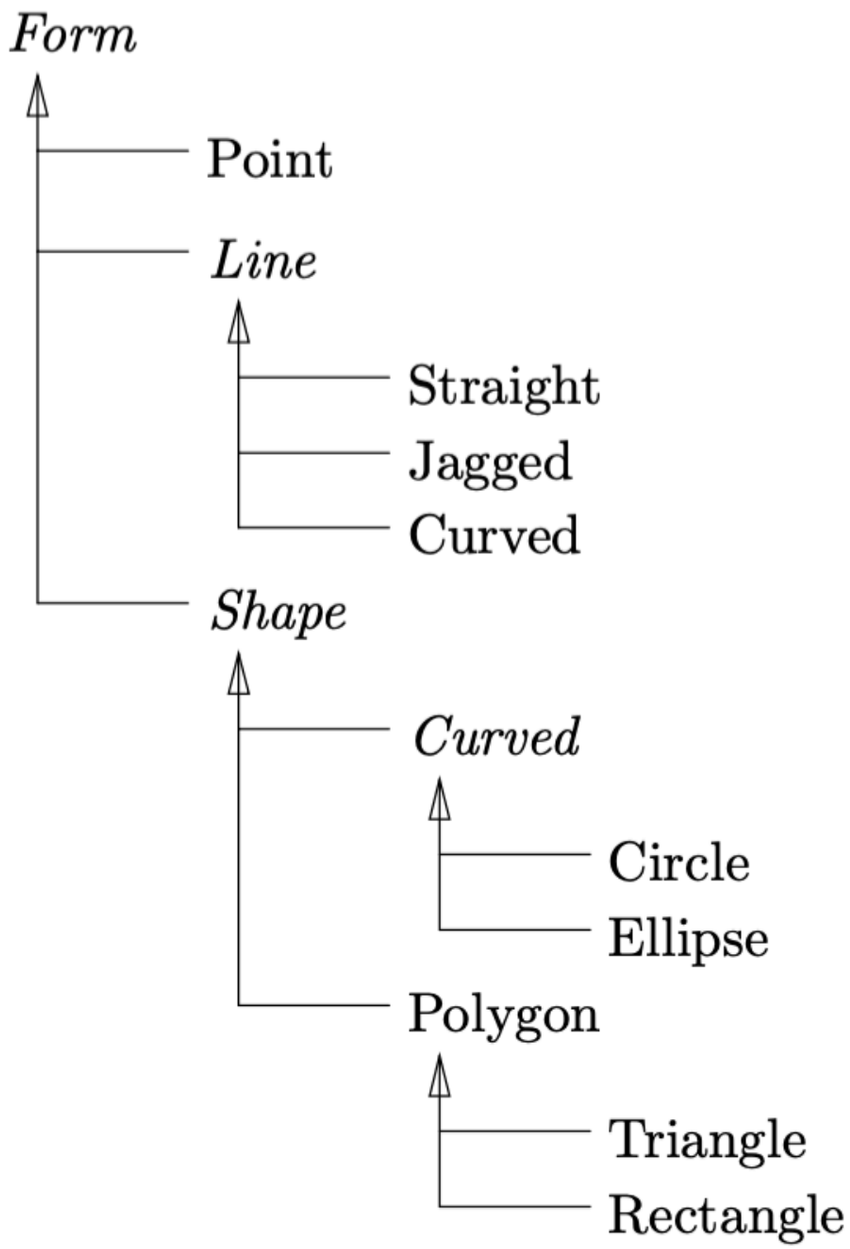
```
ComputerPlayer
<= CardPlayer
<= CardPlayerBehavior

HumanPlayer
<= CardPlayer
<= CardPlayerBehavior
```

Taxonomy of Animals




```
public abstract class Animal
{
  // fields and methods that describe an animal
}

public abstract class WarmBlooded extends Animal
{
  // additional fields and methods specific to warm-blooded animals
}

public abstract class Equine extends WarmBlooded
{
  // fields and methods specific to equines
}

public class Horse extends Equine
{
  // fields and completed methods specific to horses
}
```

Hierarchy of Forms for Drawing



Frameworks

- 프레임워크는 특정 애프리케이션 제작에 특화한 아키텍처 구축용으로 미리 준비하여 모아놓은 클래스와 인터페이스의 집합체
  - 그래픽 윈도우 구축용 프레임워크
  - 애니메이션 제작용 프레임워크
  - 스프레드쉬트 개발용 프레임워크
  - 음악 작곡용 프레임워크
  - 카드게임 개발용 프레임워크
- 프레임워크의 일부는 abstract class 로 비워 둠
- 사례
  - Java’s Abstract Window Toolkit (java.awt) package
  - Java’s Swing (javax.swing) package

Packages

- 폴더 안에 모아놓은 클래스와 인터페이스를 통틀어 패키지라고 한다.
  - java.util
  - java.awt
  - javax.swing
- import <패키지이름> 의 형식으로 불러쓴다.

class Object

소속 패키지: java.lang

모든 클래스 C에 대해서, C <= Object

Object는 존재하는 모든 클래스의 최상위 객체

```
public class Pair {
    Object[] r = new Object[2];

    public Pair(Object ob1, Object ob2) {
        r[0] = ob1;
        r[1] = ob2;
    }

    public Object get1st(){
        return r[0];
    }

    public Object get2nd(){
        return r[1];
    }
}
```

```
Pair p = new Pair("abc", 7);

Object item1 = p.get1st();
System.out.println((String)item1 + (String)item1);

Object item2 = p.get2nd();
System.out.println((int)item2 + 2);
```

Blackjack Class Diagram