제 8 장 상속 (Inheritance) Part-3

정적 메소드 오버라이딩

- 수퍼클래스의 메소드 중에서 정적메소드를 오버라이드하면?
  - 수퍼클래스 객체에서 호출되느냐 아니면 자식 클래스에서 호출되느냐에 따라서 호출되는 메소드가 달라진다(메소드 은폐(hiding))

- 예

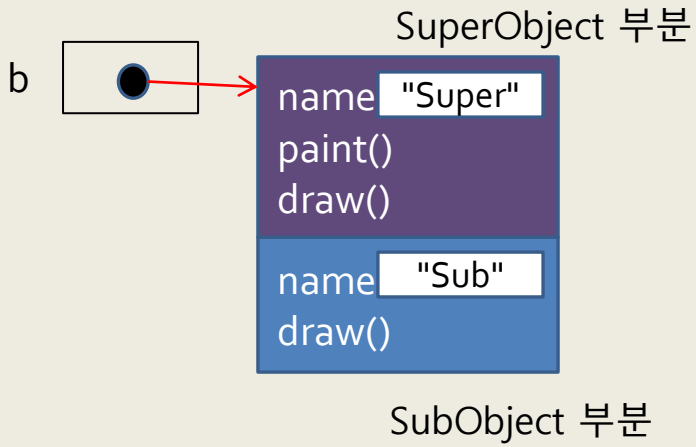
```
public class Animal {
    public static void testStaticMethod() {
        System.out.println("The static method in Animal");
    }
    public void testInstanceMethod() {
        System.out.println("The instance method in Animal");
    }
}
```

```
public class Cat extends Animal {
    public static void testStaticMethod() {            // 은폐(hiding)
        System.out.println("The static method in Cat");
    }
    public void testInstanceMethod() {                 // 오버라이딩
        System.out.println("The instance method in Cat");
    }
    public static void main(String[] args) {
        Cat myCat = new Cat();
        Animal myAnimal = myCat;          //자동 타입 변환
        Animal.testStaticMethod();
        myAnimal.testInstanceMethod();
    }
}
```

The static method in Animal
The instance method in Cat

super 키워드

- super는 서브클래스에서 슈퍼 클래스의 멤버를 접근할 때 사용되는 슈퍼클래스 타입의 레퍼런스.
- 상속관계에 있는 서브 클래스에서만 사용됨
- 오버라이딩된 슈퍼 클래스의 메소드 호출 시 사용




```
class SuperObject {
    protected String name;
    public void paint() {
        draw();
    }
    public void draw() {
        System.out.println(
    }

    blic class SubObject                    { {
    protected String nar
    ublic void draw() {
        name = "Sub";
        super.name = "S          ;
        super.draw();
        System.out.println(name);
    }
    public static void main(String [] args) {
        SuperObject b = new SubObject();
        b.paint();
    }
}
```

Super
Sub

예제 : 메소드 오버라이딩

Person을 상속받는 Professor라는 새로운 클래스를 만들고 Professor 클래스에서 getPhone() 메소드를 재정의하라. 그리고 이 메소드에서 슈퍼 클래스의 메소드를 호출하도록 작성하라.

```
class Person {
    String phone;
    public void setPhone(String phone) {
        this.phone = phone;
    }
    public String getPhone() {
        return phone;
    }
}

class Professor extends Person {
    public String getPhone() {
        return "Professor : " + super.getPhone();
    }
}
```

super.getPhone()은 아래 p.getPhone()과 달리 동적 바인딩이 일어나지 않는다.

```
public class Overriding {
    public static void main(String[] args) {
        Professor a = new Professor();
        a.setPhone("011-123-1234");
        System.out.println(a.getPhone());
        Person p = a;
        System.out.println(p.getPhone());
    }
}
```

```
Professor : 011-123-1234
Professor : 011-123-1234
```

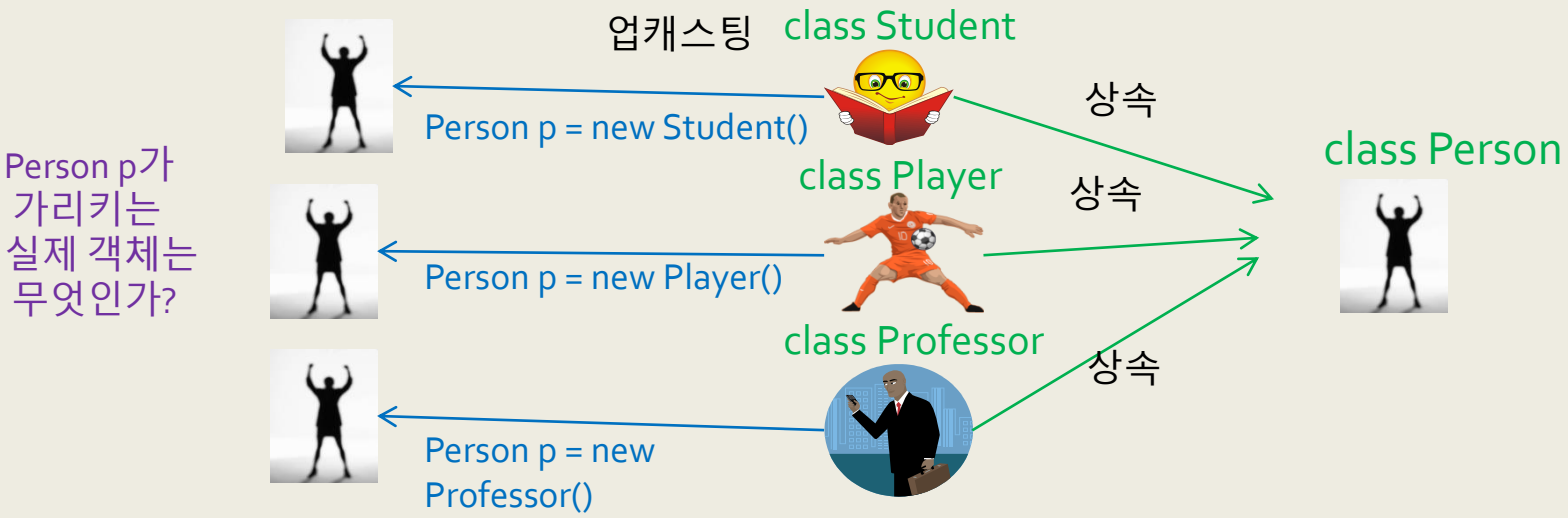
동적 바인딩에 의해 Professor의 getPhone() 호출.

오버라이딩 vs. 오버로딩

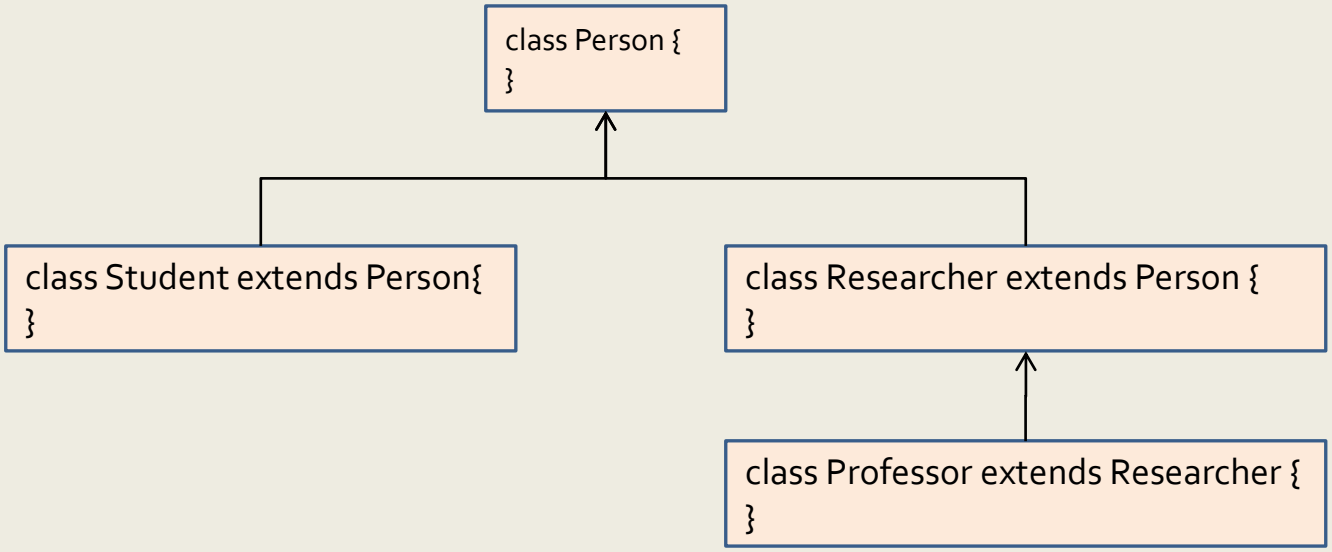
비교 요소	메소드 오버로딩	메소드 오버라이딩
정의	같은 클래스나 상속 관계에서 동일한 이름의 메소드 중복 작성	서브 클래스에서 슈퍼 클래스에 있는 메소드와 동일한 이름의 메소드 재작성
관계	동일한 클래스 내 혹은 상속 관계	상속 관계
목적	이름이 같은 여러 개의 메소드를 중복 정의하여 사용의 편리성 향상	슈퍼 클래스에 구현된 메소드를 무시하고 서브 클래스에서 새로운 기능의 메소드를 재정의하고자 함
조건	메소드 이름은 반드시 동일함. 메소드의 인자의 개수나 인자의 타입이 달라야 성립	메소드의 이름, 인자의 타입, 인자의 개수, 인자의 리턴 타입 등이 모두 동일하여야 성립
바인딩	**정적 바인딩.** 컴파일 시에 중복된 메소드 중 호출되는 메소드 결정	**동적 바인딩.** 실행 시간에 오버라이딩된 메소드 찾아 호출

instanceof 연산자와 객체 구별

- 업캐스팅된 레퍼런스로 객체의 진짜 타입을 구분하기 어려움
  - 하나의 슈퍼 클래스는 여러 서브 클래스에 상속된다.
  - 서브 클래스 객체는 업캐스팅에 의해 슈퍼 클래스 레퍼런스가 가리킬 수 있다.
- instanceof 연산자
  - instanceof를 이용하여 레퍼런스가 가리키는 객체의 정확한 진짜 타입을 식별
  - 사용법
    - 객체 레퍼런스 instanceof 클래스타입 --> true/false의 Boolean 값



instanceof 사용 예




```
Person jee= new Student();
Person kim = new Professor();
Person lee = new Researcher();
if (jee instanceof Person) // jee는 Person 타입이므로 true
if (jee instanceof Student) // jee는 Student 타입이므로 true
if (kim instanceof Student) // kim은 Student 타입이 아니므로 false
if (kim instanceof Professor) // kim은 Professor 타입이므로 true
if (kim instanceof Researcher) // kim은 Researcher 타입이기도 하므로 true
if (lee instanceof Professor) // lee는 Professor 타입이 아니므로 false
if ("java" instanceof String) // "java"는 String 타입의 인스턴스이므로 true
if (3 instanceof int) // 문법적 오류 instanceof는 객체에 대한 레퍼런스에만 사용
```

예제 : instanceof를 이용한 객체 구별

instanceof를 이용하여 객체의 타입을 구별하는 예를 만들어보자.

```
class Person {}
class Student extends Person {}
class Researcher extends Person {}
class Professor extends Researcher {}
```

```
jee는 Student 타입
kim은 Professor 타입
kim은 Researcher 타입
kim은 Person 타입
"java"는 String 타입
```

```
public class InstanceofExample {
    public static void main(String[] args) {
        Person jee= new Student();
        Person kim = new Professor();
        Person lee = new Researcher();
        if (jee instanceof Student) // jee는 Student 타입이므로 true
            System.out.println("jee는 Student 타입");
        if (jee instanceof Researcher) // jee는 Researcher 타입이 아니므로 false
            System.out.println("jee는 Researcher 타입");
        if (kim instanceof Student) // kim은 Student 타입이 아니므로 false
            System.out.println("kim은 Student 타입");
        if (kim instanceof Professor) // kim은 Professor 타입이므로 true
            System.out.println("kim은 Professor 타입");
        if (kim instanceof Researcher) // kim은 Researcher 타입이기도 하므로 true
            System.out.println("kim은 Researcher 타입");
        if (kim instanceof Person) // kim은 Person 타입이기도 하므로 true
            System.out.println("kim은 Person 타입");
        if (lee instanceof Professor) // lee는 Professor 타입이 아니므로 false
            System.out.println("lee는 Professor 타입");
        if ("java" instanceof String) // "java"는 String 타입의 인스턴스이므로 true
            System.out.println("\"java\"는 String 타입");
    }
}
```

추상 메소드와 추상 클래스

- 추상 메소드(abstract method)
  - 선언되어 있으나 구현되어 있지 않은 메소드
  - 추상 메소드 정의
    - 접근 지정자 **abstract** 반환형 메소드이름();
      - *ex) public **abstract** int getValue();*
  - 추상 메소드는 서브 클래스에서 오버라이딩하여 구현
- 추상 클래스(abstract class)
  - 추상 메소드를 하나라도 가지면 추상 클래스임
    - 클래스 앞에 반드시 abstract라고 선언해야 함
  - 추상 메소드가 하나도 없지만 클래스 앞에 abstract로 선언한 경우

```
abstract class DObject {
   public DObject next;

   public DObject() { next = null;}
   abstract public void draw() ;
}
```

추상 클래스 특성

- 추상 클래스의 객체는 생성할 수 없다.
- 추상 클래스 필요성
  - 계층적 상속 관계를 갖는 클래스 구조를 만들 때
  - 설계와 구현 분리
    - 슈퍼 클래스에서는 개념적 특징 정의, 서브 클래스에서 구체적 행위 구현
- 추상 클래스의 상속
  - 추상 클래스를 상속받아, 추상 메소드를 구현하지 않으면 서브 클래스도 추상 클래스 됨.
    - abstract로 정의하여야 한다.
  - 서브 클래스에서 추상 메소드를 구현하면 서브 클래스는 추상 클래스가 되지 않는다.

2 가지 종류의 추상 클래스

```
abstract class Line { // 추상메소드를 포함하므로 반드시 추상 클래스
        int x;
        int y;
        public abstract void setX(int position);
        public abstract void setY(int position);
        public abstract int getLength();
}
public class AbstractError {
        public static void main (String args[]) {
                Line l = new Line(); // 컴파일 오류 발생
                l.setX(0);
                l.setY(10);
        }
}
```

```
abstract class Line { // 개발자가 임의로 추상 클래스 선언
        int x;
        int y;
        public void setX(int position) {
                x = position;
        }
        public void setY(int position) {
                y = position;
        }
        public int getLength() {return 0;}
}
public class AbstractError {
        public static void main (String args[]) {
                Line l = new Line(); // 컴파일 오류 발셍
                l.setX(0);
                l.setY(10);
        }
}
```

```
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
        Cannot instantiate the type Line

        at chap5.AbstractError.main(AbstractError.java:11)
```

동일한 컴파일 오류 발생

추상클래스는 인스턴스를 생성할 수 없음

추상 클래스의 활용 예

```
class DObject {
    public DObject next;

    public DObject() { next = null;}
     public void draw() {
        System.out.println("DObject draw");
    }
}
```

추상 클래스로 수정

```
abstract class DObject {
    public DObject next;

    public DObject() { next = null;}
    abstract public void draw();
}
```

```
class Line extends DObject {
    public void draw() {
        System.out.println("Line");
    }
}
```

```
class Rect extends DObject {
    public void draw() {
        System.out.println("Rect");
    }
}
```

```
class Circle extends DObject {
    public void draw() {
        System.out.println("Circle");
    }
}
```

```
abstract class Shape {
        private int x, y;
        public void move(int x, int y)
                this.x = x;
                this.y = y;
        }
        public abstract void draw();
};

class Rectangle extends Shape {
        private int width, height;
        public void draw() { // 추상 메소드 구현
                System.out .println("사각형 그리기 메소드");
        }
};

class Circle extends Shape {
        private int radius;
        public void draw() {
                System.out .println("원 그리기 메소드");
        }
};
```

예제 : 추상 클래스의 구현

다음의 추상 클래스 Calculator를 상속받는 GoodCalc 클래스를 독자 임의로 작성하라.

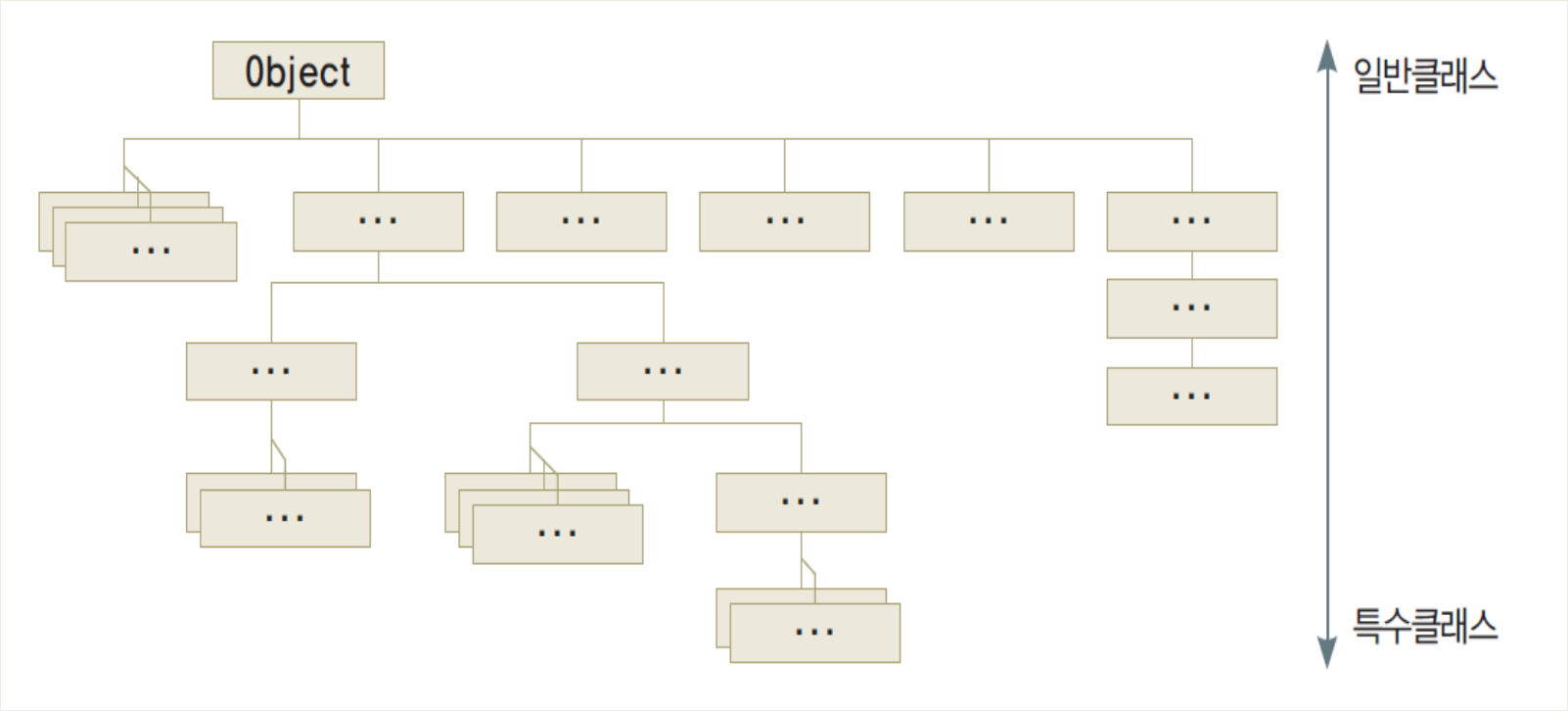
```
abstract class Calculator {
        public abstract int add(int a, int b);
        public abstract int subtract(int a, int b);
        public abstract double average(int[] a);
}
```

```
class GoodCalc extends Calculator {
        public int add(int a, int b) {
                return a+b;
        }
        public int subtract(int a, int b) {
                return a - b;
        }
        public double average(int[] a) {
                double sum = 0;
                for (int i = 0; i < a.length; i++)
                                sum += a[i];
                return sum/a.length;
        }
        public static void main(String [] args) {
                Calculator c = new GoodCalc();
                System.out.println(c.add(2,3));
                System.out.println(c.add(2,3));
                System.out.println(c.add(new int [] {2,3,4}));
        }
}
```

```
5
-1
3.0
```

Object 클래스

- Object 클래스는 java.lang 패키지에 들어 있으며 자바 클래스 계층 구조에서 맨 위에 위치하는 클래스



Object의 메소드

메소드	설명
`Object clone( )`	객체 자신의 복사본을 생성하여 반환한다.
`boolean equals(Object obj)`	obj가 현재 객체와 같은지를 반환한다.
`void finalize()`	사용되지 않는 객체가 제거되기 직전에 호출된다.
`class getclass( )`	실행 시간에 객체의 클래스 정보를 반환한다.
`int hashCode( )`	객체에 대한 해쉬 코드를 반환한다.
`String toString( )`	객체를 기술하는 문자열을 반환한다.

```
class Car {
  ...
}
public class CarTest {
  public static void main(String[] args) {
        Car obj = new Car();
        System.out.println("obj is of type " + obj.getclass().getName());
  }
}
```

obj is of type Car

```
class Point {
   int x, y;
   public Point(int x, int y) {
      this.x = x;
      this.y = y;
   }
}

public class ObjectProperty {
   public static void main(String [] args) {
      Point p = new Point(2,3);
      System.out.println(p.getClass().getName());
      System.out.println(p.hashCode());
      System.out.println(p.toString());
      System.out.println(p);
   }
}
```

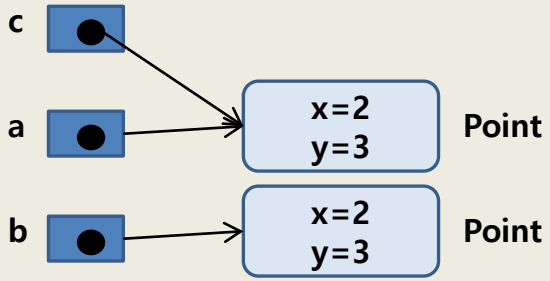
```
Point
12677476
Point@c17164
Point@c17164
```

메소드 equals()

- 메소드 equals()은 두 객체가 동일한 객체인지를 비교하여 그 둘이 동일하면 true를 리턴한다
  - 동일성 판단하기 위해서 identity 연산자(==)를 사용하고 있다
  - 기본 타입에 대해서는 올바른 결과를 리턴하지만 객체를 가리키는 참조형 타입에 대해서는 내용의 동일성을 테스트하지 않는다
  - 두 객체의 참조 값, 즉 주소값이 일치하는 지를 테스트한다.
    - 비교되는 객체가 동일한 지를 알려주는 것이지 객체의 내용이 동일한 지를 알려주는 것이 아니다.
  - 그러므로 두 객체의 콘텐츠에 대한 동일성을 테스트하려면 그에 적합한 동작으로 equals() 메소드를 오버라이드 해야한다
    - 만약에 equals() 메소드를 오버라이드 한다면 반드시 hashCode() 메소드도 오버라이드 해야 한다
  - **String 클래스**는 스트링의 동일성 테스트를 위해서 equals() 메소드를 오버라이드 하여 사용한다

```
class Point {
    int x, y;
    public Point(int x, int y) {
        this.x = x; this.y = y;
    }
}
```

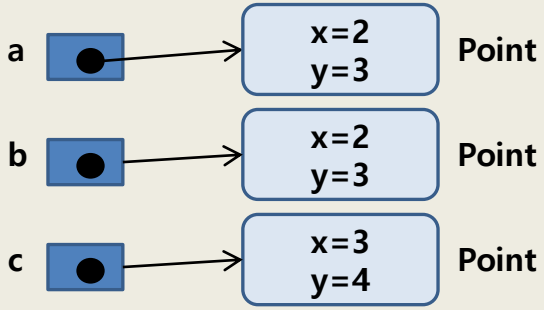
```
Point a = new Point(2,3);
Point b = new Point(2,3);
Point c = a;
if(a == b) // false
    System.out.println("a==b");
if(a == c) // true
    System.out.println("a==c");
```




```
a==c
```

```
class Point {
    int x, y;
    public Point(int x, int y) {
        this.x = x; this.y = y;
    }
    public boolean equals(Point p) {
        if(x == p.x && y == p.y)
            return true;
        else
            return false;
    }
}
```

```
Point a = new Point(2,3);
Point b = new Point(2,3);
Point c =  new Point(3,4);
if(a == b) // false
    System.out.println("a==b");
if(a.equals(b)) // true
    System.out.println("a is equal to b");
if(a.equals(c)) // false
    System.out.println("a is equal to c");
```




```
a is equal to b
```

```
public class Book {
...
    public boolean equals(Object obj) {
        if (obj instanceof Book)
            return ISBN.equals((Book)obj.getISBN());
        else
            return false;
    }
}
```

클래스 Book이 equals 메소드를 오버라이드 하고 있음

```
Book firstBook = new Book("0201914670");
Book secondBook = new Book("0201914670");
if (firstBook.equals(secondBook)) {
    System.out.println("objects are equal");
} else {
    System.out.println("objects are not equal");
}
```

예제:
int 타입의 width, height의 필드를 가지는 Rect 클래스를 작성하고, 두 Rect 객체의 width, height 필드에 의해 구성되는 면적이 같으면 두 객체가 같은 것으로 판별하도록 equals()를 작성하라. Rect 생성자에서 width, height 필드를 인자로 받아 초기화한다.

```
class Rect {
   int width;
   int height;
   public Rect(int width, int height) {
      this.width = width;
      this.height = height;
   }
   public boolean equals(Rect p) {
      if (width*height == p.width*p.height)
         return true;
      else
         return false;
   }
}
```

```
public class EqualsEx {
   public static void main(String[] args) {
      Rect a = new Rect(2,3);
      Rect b = new Rect(3,2);
      Rect c = new Rect(3,4);
      if(a.equals(b)) System.out.println("a is equal to b");
      if(a.equals(c)) System.out.println("a is equal to c");
      if(b.equals(c)) System.out.println("b is equal to c");
   }
}
```

```
a is equal to b
```

메소드 hashcode()

- 메소드 hashCode()에 의해 리턴되는 값은 해당 객체의 해쉬 코드이며, 그 객체의 메모리 영역의 주소를 표현한 16진 주소
- 그러므로 두 객체가 동일한 객체라면 그들 해쉬 코드 또한 동일해야 한다.
- 그러므로 equals() 메소드를 오버라이드 한다면 두 객체의 동일성 체크 방법이 변경되므로 Object가 구현하고 있는 메소드 hashCode()도 같이 변경되어야 하므로, equals() 메소드를 오버라이드 한다면 hasCode() 메소드 또한 오버라이드 해야 한다

메소드 toString()

- Object 클래스의 toString() 메소드는 객체의 String 표현을 리턴하며, 디버깅을 위해서 매우 유용하게 쓰인다.
- 반환되는 문자열 : 클래스 이름@객체의 hash code
- 객체와 문자열이 + 연산이 되는 경우 객체의 toString() 메소드를 호출

```
Point a = new Point(2,3);
String s = a + "점";
System.out.println(s);
```

변환

```
Point a = new Point(2,3);
String s = a.toString()+ "점";
System.out.println(s);
```

Point@c17164점

```
public class Car {
        private String model;
        public Car(String model) {
                this.model = model;
                System.out.println(this);
        }
        public String toString() {
                return "모델: " + model;
        }
}
```

Object의 tostring()를 재정의

```
class Point {
   int x, y;

   public Point(int x, int y) {
      this.x = x; this.y = y;
   }
   public String toString() {
      return "Point(" + x + "," + y+ ")";
   }
}

public class ObjectProperty {
   public static void main(String [] args) {
      Point a = new Point(2,3);
      System.out.println(a.toString());
   }
}
```

Point(2,3)