**기본 자료형과 참조형**

2018-08-27

이승진

**학습목표**

참조 타입(reference type)과 값 타입(value type)의 이해

String 클래스 사용법

equals 메소드 재정의 방법

instanceof 연산자 사용법

@Override 어노테이션 활용

**목차**

[1. Java 소스 파일 규칙 2](#_Toc523148202)

[1) Java 소스 파일과 패키지 규칙 2](#_Toc523148203)

[2) 실습 3](#_Toc523148204)

[2. String 클래스 & Object 클래스 4](#_Toc523148205)

[1) String 클래스 4](#_Toc523148206)

[2) call by value 5](#_Toc523148207)

[3) immutable object 6](#_Toc523148208)

[4) value object 6](#_Toc523148209)

[5) Object 클래스 7](#_Toc523148210)

[6) String01.java 분석 실습 8](#_Toc523148211)

[3. 값 타입과 참조 타입 9](#_Toc523148212)

[1) 값 타입 (value type) 9](#_Toc523148213)

[2) 참조 타입 (reference type) 9](#_Toc523148214)

[3) equals 메소드 10](#_Toc523148215)

[4. 객체 구조 11](#_Toc523148216)

[1) 객체 구조 그리기 #1 11](#_Toc523148217)

[2) 객체 구조 그리기 #2 15](#_Toc523148218)

[3) 객체 구조 그리기 #3 16](#_Toc523148219)

[4) 객체 구조 그리기 실습 #4 17](#_Toc523148220)

[5. equals 메소드 재정의 18](#_Toc523148221)

[1) instanceof 연산자 18](#_Toc523148222)

[2) @Override 어노테이션 18](#_Toc523148223)

[3) equals 메소드 재정의 19](#_Toc523148224)

[4) toString 메소드 재정의 20](#_Toc523148225)

[5) Person.java 수정 21](#_Toc523148226)

[6. String.format 메소드 22](#_Toc523148227)

[1) String.format 메소드 사용법 22](#_Toc523148228)

[2) Format01.java 23](#_Toc523148229)

[3) Printf01.java 24](#_Toc523148230)

# Java 소스 파일 규칙

## Java 소스 파일과 패키지 규칙

Java 소스 코드 파일와 패키지에는 다음과 같은 규칙이 있다.

* 소스 코드 파일 하나에 public 클래스는 한 개만 들어있어야 한다.
* public 아닌 클래스는 소스 코드 파일에 여러 개 들어있어도 된다.
* public 클래스 이름과 소스 코드 파일명이 대소문자까지 정확하게 일치해야 한다.
* 클래스가 속한 패키지 이름과 소스 코드 파일이 위치한 폴더 이름이 대소문자까지 정확하게 일치해야 한다.

### HelloWorld.java

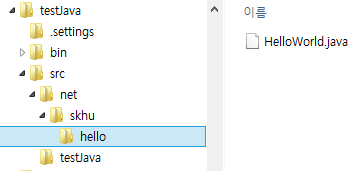
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | package net.skhu.hello;  public class HelloWorld {  public static void main(String[] args) {  GoodMorning g = new GoodMorning();  System.out.println("hello world");  System.out.println(g.hello());  }  }  class GoodMorning {  public String hello() {  return "good morning";  }  } |

(줄3) public 클래스 이름이 HelloWorld 이므로, 소스 파일 이름도 HelloWorld.java 이어야 한다.

(줄11) GoodMorning 클래스는 public 이 아니므로 HelloWorld 클래스와 같은 소스 파일에 들어있어도 된다.

(줄1) HelloWorld 클래스가 net.skhu.hello 패키지에 들어있으므로,

HelloWorld.java 파일도 프로젝트 폴더 아래의 src/net/skhu/hello 폴더에 들어있어야 한다.



Java 웹서버 프로그래밍 (JSP) 프로젝트에서 클래스는 반드시 어떤 패키지 아래 있어야 한다.   
패키지가 정해지지 않은 클래스는 JSP 프로젝트에서 사용할 수 없다.

## 실습

### 실습#1

HelloWorld.java의 줄3을 다음과 같이 수정하면 컴파일 에러가 발생한다.

public class Helloworld {

컴파일 에러 메시지는 무엇인지 확인하라. 에러가 발생하는 원인은 무엇인가?

### 실습#2

줄11을 다음과 같이 수정하면 컴파일 에러가 발생한다.

public class GoodMorning {

컴파일 에러 메시지는 무엇인지 확인하라. 에러가 발생하는 원인은 무엇인가?

### 실습#3

줄1을 다음과 같이 수정하면 컴파일 에러가 발생한다.

package net.skhu.Hello;

컴파일 에러 메시지는 무엇인지 확인하라. 에러가 발생하는 원인은 무엇인가?

# String 클래스 & Object 클래스

## String 클래스

대표적인 Java 문자열은 String 클래스이다.

String 객체는 생성된 뒤 문자열 값이 수정될 수 없다.

String 클래스의 문자열 수정 메소드는, this의 문자열을 수정하는 것이 아니고, 수정된 새 문자열 객체를 리턴한다. this의 문자열은 수정되지 않는다.

|  |
| --- |
| char charAt(int index)  this 문자열에서 index 위치의 문자(char)를 리턴한다. |
| int compareTo(String s)  문자열을 알파벳 순서로 비교한다.  this 문자열의 순서가 빠르면 음수를, s 문자열의 순서가 빠르면 양수를,  두 문자열이 같으면 0을 리턴한다. |
| int compareToIgnoreCase(String s)  대소문자를 구별하지 않고 문자열을 비교한다.  리턴값은 compareTo 메소드와 같다. |
| boolean contains(CharSequence s)  파라미터 문자열 s가 this 문자열의 일부와 일치하면 true를 리턴한다.  즉 문자열 s가 this 문자열에 들어 있으면 true를 리턴한다. |
| boolean endWith(String s)  this 문자열의 끝이 문자열 s와 일치하면 true를 리턴한다. |
| boolean equals(Object s)  this 문자열이 문자열 s와 일치하면 true를 리턴한다. |
| boolean equalsIgnoreCase(Object s)  this 문자열이 문자열 s와 일치하면 true를 리턴한다. 대소문자를 구별하지 않고 비교한다. |
| static String format(String format, Object... args)  이 메소드의 사용 방법은 C 언어의 printf 함수와 유사하다.  첫째 파라미터는 format string 이다. 이 format string과 그 뒤의 파라미터 값으로  문자열을 생성하여 리턴한다. |
| int indexOf(String s)  this 문자열에서 문자열 s를 찾아서 그 위치(index)를 리턴한다.  this 문자열에 문자열 s가 들어있지 않다면 -1을 리턴한다. |
| boolean isEmpty()  this 문자열의 길이가 0이면 true를 리턴한다. |
| int lastIndexOf(String s)  this 문자열에서 문자열 s를 찾아서 그 위치(index)를 리턴한다.  문자열 s가 여러 번 들어있다면 마지막 위치를 리턴한다.  this 문자열에 문자열 s가 들어있지 않다면 -1을 리턴한다. |
| int length()  문자열의 길이를 리턴한다. |
| boolean matches(String regex)  파라미터로 주어진 regex 정규식(regular express)과 this 문자열이 일치하면 true를 리턴한다. |
| String replace(CharSequence s1, CharSequence s2)  this 문자열에서 문자열 s1을 전부 찾아서 s2로 치환한다. (find & replace)  그렇게 치환된 새 문자열을 리턴한다. this 문자열은 수정되지 않는다. |
| String replaceAll(String regex, String s2)  this 문자열에서 regex 정규식과 일치하는 부분을 전부 찾아서 s2로 치환한다.  그렇게 치환된 새 문자열을 리턴한다. this 문자열은 수정되지 않는다. |
| String[] split(String regex)  regex 정규식과 일치하는 부분을 기준으로 this 문자열을 쪼갠다.  그렇게 쪼개진 문자열 배열을 리턴한다. |
| boolean startsWith(String s)  this 문자열의 시작 부분이 문자열 s와 일치하면 true를 리턴한다. |
| String substring(int beginIndex)  this 문자열에서 beginIndex 위치에서 끝까지의 문자열을 리턴한다.  this 문자열은 수정되지 않는다. |
| String substring(int beginIndex, int endIndex)  this 문자열에서 beginIndex 위치에서 endIndex 직전까지의 문자열을 리턴한다.  this 문자열은 수정되지 않는다. |
| String toLowerCase()  this 문자열에서 모든 대문자를 소문자로 변경한 새 문자열을 리턴한다.  this 문자열은 수정되지 않는다. |
| String toUpperCase()  this 문자열에서 모든 소문자를 대문자로 변경한 새 문자열을 리턴한다.  this 문자열은 수정되지 않는다. |
| String trim()  this 문자열의 시작부분과 끝부분의 공백(whitespace)을 제거한 새 문자열을 리턴한다.  this 문자열은 수정되지 않는다. |
| static String valueOf(int i)  i 값을 문자열로 변환하여 리턴한다. |
| static String valueOf(Double d)  d 값을 문자열로 변환하여 리턴한다. |

## call by value

C 언어의 함수 호출은 call by value 이다.

C 함수를 호출하면서, 파라미터 값으로 변수를 전달해도

그 변수의 값은 변경될 수 없다.

예:

|  |
| --- |
| function f1(int a) {  ++a;  printf("%d", a); // 4 출력된다.  }  main() {  int a = 3;  f1(a);  printf("%d", a); // 3 출력된다.  } |

파라미터로 전달될 때, 변수의 값만 복사되어서 전달될 뿐이고

변수 자체가 전달되는 것은 아니기 때문이다.

이런 매카니즘을 call by value 라고 부른다.

## immutable object

String 객체는 생성된 뒤 문자열 값이 수정될 수 없다.

String 클래스의 문자열 수정 메소드는, this의 문자열을 수정하는 것이 아니고, 수정된 새 문자열 객체를 리턴한다. this의 문자열은 수정되지 않는다.

이런 객체를 immutable object 라고 부른다.

내용을 변경할 수 없는 객체를 immutable object 라고 부른다.

immutable = 값을 수정할 수 없는

immutable object = 값을 수정할 수 없는 객체

## value object

(객체지향이란 무엇인가를 고민하게 될 때 필요한 개념이다.

프로그래밍 초보자는 이 개념을 이해하려고 무리하지 않아도 된다.)

기본 자료형과 같은 방식으로 사용할 수 있도록 만들어진 객체를 value object 라고 부른다.

기본 자료형은 변수에 값이 저장된다.

변수에 새 값을 대입하지 않으면, 그 변수 값은 변화가 없다.

예 #1

|  |
| --- |
| static void sub(int a) {  a = a + 1;  System.out.println(a);  }  public static void main(String[] args) {  int i = 3;  sub(i);  System.out.println(i); // 변수 i에 3이 대입된 이후로 새 값이 대입된 것이 없다.  } |

출력

|  |
| --- |
| 4  3 |

예 #2

|  |
| --- |
| static void sub(String a) {  a = a.toUpperCase();  System.out.println(a);  }  public static void main(String[] args) {  String s = "hello";  sub(s);  System.out.println(s); // 변수 s에 "hello" 문자열이 대입된 이후 새 문자열이 대입되지 않았다.  } |

출력

|  |
| --- |
| HELLO  hello |

## Object 클래스

Java 언어에서 모든 클래스의 부모 클래스는 Object 클래스이다.

따로 선언하지 않아도 자동으로 Object 클래스가 부모 클래스가 된다.

|  |
| --- |
| protected Object clone()  자기 자신을 복제해서 새 객체를 만들어 리턴한다. |
| boolean equals(Object obj)  equals 메소드를 제대로 구현했다면,  equals 메소드는 this 객체와 파라미터 obj 객체의 내용이 동일한지 비교해야 한다. (equality 비교)  그런데 Object 클래스에 구현된 equals 메소드는 제대로 구현되어 있지 않다.  Object 클래스에 구현된 equals 메소드는 객체의 내용을 비교하지 않고  두 객체가 동일한 객체이진 비교한다. (identity 비교)  그래서  equality를 제대로 비교하려면 자식 클래스에서 이 메소드를 재정의 해야 한다. |
| protected void finalize()  객체가 더 이상 사용되지 않을 때, 자동으로 객체가 파괴되고 메모리가 회수된다.  이때 객체가 파괴되기 직전에 finalize() 메소드가 호출된다. |
| Class<?> getClass()  객체가 속한 클래스에 대한 정보를 담고 있는 객체를 리턴한다. |
| int hashCode()  객체를 hashtable 테이블 자료구조에 넣을 때 필요한 해쉬 값(hash code value)를 리턴한다. |
| String toString()  객체의 내용을 표현하는 문자열을 리턴한다.  리턴되는 문자열에 객체의 내용이 잘 표현되도록, 자식 클래스에서 이 메소드를 재정의 해야 한다. |

위 표에서 protected로 표시된 clone, finalize 메소드는 protected 메소드이고,

나머지 메소드들은 public 메소드이다.

## String01.java 분석 실습

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32 | package net.skhu.lecture01;  public class String01 {    public static void main(String[] args) {  String s = "hello world\n";  String t = "Hello World\n";    for (int i = 0; i < s.length(); ++i)  System.out.printf("%c", s.charAt(i));    System.out.println(s.compareTo(t) > 0);  System.out.println(s.compareToIgnoreCase(t) == 0);  System.out.println(s.contains("hello"));  System.out.println(s.endsWith("rld\n"));  System.out.println(s.equals(t) == false);  System.out.println(s.equalsIgnoreCase(t));  System.out.println(s.isEmpty() == false);  System.out.println(s.indexOf("o") == 4);  System.out.println(s.lastIndexOf("o") == 7);  System.out.println(s.length() == 12);  System.out.println(s.startsWith("hell"));    String[] a = s.split(" ");  System.out.println(a.length == 2);  System.out.println(a[0].equals("hello"));  System.out.println(s.substring(0, 5).equals("hello"));  System.out.println(t.toLowerCase().equals(s));  }  } |

위 String 클래스 메소드의 설명을 읽고 String01.java의 실행 결과 출력이 무엇일지 계산해 보자.

계산한 것이 맞는지 실행결과 출력과 비교해 보자.

# 값 타입과 참조 타입

## 값 타입 (value type)

|  |  |
| --- | --- |
| 1  2 | int i = 123  double d = 3.14159; |

줄1에서 변수 i에 123 값이 대입된다.

줄2에서 변수 d에 3.14159 값이 대입된다.

이렇게 변수에 값이 대입되는 자료형을 값 타입(value type) 이라고 부른다.

변수가 차지하는 메모리 공간에 값이 저장된다.

Java의 기본 자료형은 모두 값 타입 (value type) 이다.

Java 언어의 기본 자료형은 다음과 같다.

byte, short, int, long, float, double, boolean, char



변수 i 공간에 123 정수값이 들어있다.

## 참조 타입 (reference type)

객체를 가르키는 일종의 주소 같은 것을 객체에 대한 참조(reference)라고 부른다.

변수에 값이 아니라 참조가 대입되는 자료형을 참조 타입(reference type) 이라고 부른다.

변수가 차지하는 메모리 공간에 들어있는 것은 값이 아니고 참조이다.

Java 클래스의 객체는 모두 참조 타입이다.

Java 문자열은 String 클래스 객체이다. 따라서 참조 타입이다.

|  |  |
| --- | --- |
| 1  2  3  4  5 | String s1 = "hello";  String s2 = "world;  s1 = s2;  System.out.println(s1);  System.out.println(s2); |

줄1에서 s1 변수에 대입되는 것은 "hello" String 객체에 대한 참조(reference)이다.

줄2에서 s2 변수에 대입되는 것은 "world" String 객체에 대한 참조(reference)이다.

변수 s1, s2 에는 객체에 대한 참조만 대입되지 객체가 대입되는 것이 아니다.



줄3을 실행하면 s1 변수에 "world" String 객체에 대한 참조가 대입된다.



줄4와 줄5에서 "world" 문자열이 출력된다.

Java 객체는 변수에 대입되거나, 배열이나 자료구조에 저장되거나, 파라미터로 전달되거나, 메소드 리턴 값으로 리턴될 수 없다. 변수에 대입되거나, 배열이나 자료구조에 저장되거나, 파라미터로 전달되거나, 메소드 리턴 값으로 리턴되는 것은, 객체에 대한 참조일뿐 객체 그 자체가 아니다.

## equals 메소드

표준 Java 라이브러리에 Object 클래스가 있다.

Object 클래스는 모든 Java 클래스의 가장 상위 부모 클래스이다.

그래서 Object 클래스의 메소드는 모든 Java 클래스가 상속 받는다.

Object 클래스에 equals 메소드가 정의되어 있다.

객체의 내부 값이 동일한지 비교할 때 equals 메소드를 사용한다.

Object 클래스는 Java의 모든 클래스의 부모 클래스이므로,

Java의 모든 클래스가 equals 메소드를 상속 받는다.

객체의 내부 값이 동일한지 정확하게 검사하기 위해여,

많은 클래스들이 이 equals 메소드를 재정의(override) 한다.

객체를 == 연산자로 비교하는 것은 객체 내부 값이 동일한지 비교하는 것이 아니고,

두 참조가 하나의 객체를 가르키고 있는지 검사하는 것이다.

### 사례#1



참조 변수 s1, s2 변수가 참조하는 객체의 내부 값은 동일하다.

따라서 s1.equals(s2) 리턴 값은 true 이다.

그런데 s1, s2 변수가 하나의 객체를 가르키고 있지는 않다.

따라서 s1 == s2 값은 false 이다.

### 사례#2



참조 변수 s1, s2 변수가 참조하는 객체의 내부 값은 동일하다.

따라서 s1.equals(s2) 리턴 값은 true 이다.

s1, s2 변수가 하나의 객체를 가르키고 있다.

따라서 s1 == s2 값도 true 이다.

### equality & identity

equality 비교: 두 객체의 내부 값이 같은지 비교하는 것 (사례#1)

identity 비교: 하나의 객체인지 비교하는 것 (사례#2)

### eqauals 메소드 재정의

새 클래스를 구현할 때, equals 메소드를 재정의 하는 것을 고려해야 한다.

equals 메소드를 재정의 하지 않으면, Object 클래스의 equals 메소드가 상속된다.

그러데 Object 클래스에서 상속되는 equals 메소드는 identity를 비교하는 방식으로 구현되어 있기 때문에 바람직하지 않다.

모든 멤버 변수 값이 같은지 비교하는 형태의 equals 메소드를 재정의하는 것이 필요하다.

# 객체 구조

## 객체 구조 그리기 #1

**String02.java**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | package net.skhu.lecture01;  public class String02 {    static void swap(String a, String b) {  String temp = a;  a = b;  b = temp;  }    public static void main(String[] args) {  String s0 = "hello world";  String s1 = s0.substring(0, 5);  String s2 = s0.substring(6);  String s3 = s1 + " " + s2;  String s4 = s0.toUpperCase();  System.out.printf("%s %s %s %s\n", s1, s2, s3, s4);    swap(s1, s2);  System.out.printf("%s %s\n", s1, s2);  }  } |



main 메소드가 호출되면, Stack Segment 영역에 main 메소드 메모리 공간이 할당된다.



줄12를 실행하면, "hello world" String 객체가 생성되고, 그 객체에 대한 참조가 지역 변수 s0에 대입된다.

Stack Segment 영역에 위치한 main 메소드의 메모리에는 지역 변수만 생성된다.

객체들은 Heap Segment 영역에 생성된다.



줄13을 실행하면, s0.substring(0, 5) 메소드 호출에 의해서 새 문자열 객체가 생성된다.

그 새 문자열 객체에 대한 참조가 지역 변수 s1에 대입된다.

String 객체는 생성된 후 수정될 수 없음에 주목하자.



줄16까지 실행한 결과 변수와 객체 구조이다.



줄19에서 swap 메소드가 호출되면, Stack Segment 영역에 swap 메소드 메모리 공간이 할당된다.

swap 메소드 메모리 공간에 파라미터 변수가 생성된다.



메소드를 호출하는 코드에서 파라미터 값이, swap 메소드의 파라미터 변수에 대입된다.

swap(s1, s2);

main 메소드의 지역 변수 s1의 값이, swap 메소드의 파라미터 변수 a에 대입된다.

main 메소드의 지역 변수 s2의 값이, swap 메소드의 파라미터 변수 b에 대입된다.



String temp = a;

줄6을 실행하면 swap 메소드 메모리에 지역 변수 temp가 생성되고, 파라미터 변수 a의 값이 temp에 대입된다.



줄8까지 실행한 결과이다.



swap 메소드가 리턴하면, swap 메소드 메모리가 제거된다.

swap 메소드를 호출 전 후의 main 메소드 지역 변수 값은 변화가 없음에 주목하자.

## 객체 구조 그리기 #2

**String03.java**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | package net.skhu.lecture01;  public class String03 {    public static void main(String[] args) {  String s0 = "hello world";  String s1 = "HELLO WORLD";  String s2 = s0;  String s3 = s1.toLowerCase();    boolean b0 = (s0 == s2);  boolean b1 = (s0 == s3);  boolean b2 = s0.equals(s2);  boolean b3 = s0.equals(s3);  System.out.printf("%b %b %b %b\n", b0, b1, b2, b3);    boolean b4 = s0.equals(s1);  boolean b5 = s0.equalsIgnoreCase(s1);  System.out.printf("%b %b\n", b4, b5);  }  } |

줄9까지 실행한 결과이다.



s0, s2 가 동일한 객체를 참조하고 있음에 주목하자.

따라서 줄11의 (s0 == s2) 식의 값은 true 이고, 줄13의 s0.equals(s2) 메소드의 리턴 값도 true 이다.

s0가 참조하는 문자열 객체와 s3가 참조하는 문자열 객체의 값은 동일하다.

줄14의 s0.equals(s3) 메소드는 값을 비교하기 때문에 true를 리턴한다.

줄12의 (s0 == s3) 식은 값을 비교하지 않고 동일한 객체인지 비교하는 것이라서 false 이다.

줄17이 equals 메소드는 대소문자를 구별하고 비교하기 때문에 false 이고,

줄18의 equals 메소드는 대소문자를 구별하지 않고 비교하기 때문에 true 이다.

## 객체 구조 그리기 #3

**Person.java**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | package net.skhu.lecture01;  public class Person {  String name;  int age;    public Person(String name, int age) {  this.name = name;  this.age = age;  }  } |

**Person01.java**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | package net.skhu.lecture01;  public class Person01 {  public static void main(String[] args) {  Person p1 = new Person("홍길동", 18);  Person p2 = new Person("홍길동", 18);    System.out.println(p1);  System.out.println(p2);  System.out.println(p1.equals(p2));  }  } |



위 코드의 실행 결과 출력은 다음과 같다.

|  |
| --- |
| net.skhu.lecture01.Person@15db9742  net.skhu.lecture01.Person@6d06d69c  false |

위 출력에서 "net.skhu.lecture01.Person@15db9742" 부분은 줄19에서 출력되었다.

줄19, 줄20에서 객체의 내용을 화면에 출력해야 하므로, 그 객체의 toString() 메소드가 호출되고,

이 메소드의 리턴 값이 화면에 출력된다.

Person 클래스에 toString() 메소드를 구현하지 않았으므로, Object 클래스의 toString() 메소드가 상속되었다.

상속된 toString() 메소드의 리턴 값이 줄19, 줄20에서 출력된다.

줄21에서 호출되는 eqauls 메소드도, Person 클래스에 equals 메소드를 구현하지 않았으므로, Object 클래스에서 상속된 equals 메소드가 호출된다.

Object 클래스의 equals 메소드는 객체의 equality가 아니라 identity를 비교하기 때문에,

줄21에서 false 값이 출력된다.

## 객체 구조 그리기 실습 #4

**Person02.java**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | package net.skhu.lecture01;  class Family {  Person father, mother;  Person[] children;    public Family() {  father = new Person("아빠", 45);  mother = new Person("엄마", 36);  children = new Person[3];  children[0] = new Person("아들1", 12);  children[1] = new Person("아들2", 9);  children[2] = new Person("딸1", 7);  }    }  public class Person02 {    static Family family;    public static void main(String[] args) {  family = new Family();  }  } |

객체 구조를 그려라.

# equals 메소드 재정의

Person 객체의 euqality를 비교할 수 있도록 equals 메소드를 재정의하자.

## instanceof 연산자

객체가 속한 클래스를 확인할 때, instanceof 연산자를 사용한다.

|  |
| --- |
| String s = "hello";  System.out.println(s instanceof String); |

참조 변수 s가 가르키는 객체는 String 클래스의 객체이므로,

위 코드는 true를 출력한다.

|  |
| --- |
| Object s = "hello";  System.out.println(s instanceof String); |

위 코드에서도 참조 변수 s가 가르키는 객체는 String 클래스 객체이다.

따라서 위 코드는 true를 출력한다.

부모 클래스 타입의 참조 변수가 자식 클래스 객체를 가르킬 수 있다.

예를 들어 사람 클래스는 포유류 클래스의 자식 클래스이다.

포유류 객체를 가르키는 참조 변수가, 포유류의 자식 클래스인 사람 클래스 객체를 가르킬 수 있다.

사람도 포유류이기 때문이다.

|  |
| --- |
| Object s = "hello";  System.out.println(s instanceof Object); |

위 코드도 true를 출력한다.

Object 클래스는 String 클래스의 부모 클래스이기 때문이다.

|  |
| --- |
| Object s = "hello";  System.out.println(s instanceof ArrayList); |

위 코드도 false를 출력한다.

## @Override 어노테이션

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | @Override  public boolean equals(Object obj) {  if ((obj instanceof Person) == false) return false;  Person p = (Person)obj;  return this.name.equals(p.name) && this.age == p.age;  } |

@Override 어노테이션

부모 클래스의 메소드를 자식 클래스에서 재정의(override)할 때, @Override 어노테이션을 메소드 위에 붙여줄 수 있다. 반드시 이 어노테이션을 붙여 주어야 하는 것은 아니고 생략해도 되지만, 붙여 주는 것이 더 좋다.

이 @Override 어노테이션의 장점은 다음과 같다.

(1) 부모 클래스의 메소드를 자식 클래스에서 재정의 하려면, 메소드 이름, 파라미터 수와 타입, 리턴 타입이 부모 클래스의 메소드와 정확히 일치해야 한다. 예를 들어 equals 메소드의 파라미터 타입은 Object 이어야 하고, 리턴 타입은 boolean 이어야 한다. @Override 어노테이션을 붙여주면, 위 사항들이 정확히 일치하는지 자동으로 검사된다.

(2) @Override 어노테이션을 보면, 부모 클래스이 메소드를 재정의한다는 것을 바로 알 수 있어서 소크 코드 읽을 때 편하다.

## equals 메소드 재정의

객체의 equality를 비교할 수 있으려면, equals 메소드를 재정의해야 한다.

### 구현#1

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | @Override  public boolean equals(Object obj) {  if ((obj instanceof Person) == false) return false;  Person p = (Person)obj;  return this.name.equals(p.name) && this.age == p.age;  } |

부모 클래스의 메소드를 재정의하려면, 메소드 이름, 파라미터 타입, 리턴 타입을 부모 클래스에서 정의된 그대로 적어주어야 한다. 그래서 eqauls 메소드의 파라미터는 Object 타입이다.

(줄3) 파라미터 객체가 Person 클래스의 객체가 아니라면, 내용이 동일할 수 없으므로 false를 리턴한다.

(줄5) name, age 멤버 변수 값이 모두 동일하면 true를 리턴한다.

구현#1의 equals 메소드 구현에는 문제가 있다.

만약 this 객체의 name 멤버 변수 값이 null 일때 위 코드가 어떻게 실행될까?

-> NullPointerException 에러가 발생한다. 이 문제를 해결해야 한다.

### 구현#2

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | @Override  public boolean equals(Object obj) {  if ((obj instanceof Person) == false) return false;  Person p = (Person)obj;  return (this.name == null ? p.name == null : this.name.equals(p.name)) && this.age == p.age;  } |

(줄5)

this.name 멤버 변수 값이 null 이면, p.name 값도 null 인지 확인하고,

this.name 멤버 변수 값이 null 이 아니면, this.name 값과 p.name 값이 동일한지 비교한다.

구현#1의 문제를 해결함.

### 구현#3

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | @Override  public boolean equals(Object obj) {  if (obj instanceof Person == false) return false;  Person p = (Person)obj;  return Objects.equals(this.name, p.name) && this.age == p.age;  } |

구현#1의 문제를 해결하는 간단한 방법은 Objects 클래스의 equals 메소드를 활용하여 구현하는 것이다.

import java.util.Objects;

마지만 return 문에서, 모든 멤버 변수를 비교하야하고, 비교식들을 && 연산자로 묶어야 한다.

기본자료형 멤버 변수는 == 연산자로 비교하고,

참조형 멤버 변수는 Objects.equals 메소드를 사용하여 비교한다.

구현#2나 구현#3 중에서 마음에 드는 방식으로 구현하면 된다.

## toString 메소드 재정의

클래스 객체가 문자열로 출력되는 형태를 정하려면, toString 메소드를 재정의해야 한다.

|  |  |
| --- | --- |
| 1  2  3  4 | @Override  public String toString() {  return String.format("Person{name=\"%s\", age=%d}", name, age);  } |

## Person.java 수정

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | package net.skhu.lecture;  import java.util.Objects;  public class Person {  String name;  int age;  public Person(String name, int age) {  this.name = name;  this.age = age;  }  @Override  public boolean equals(Object obj) {  if (obj instanceof Person == false) return false;  Person p = (Person)obj;  return Objects.equals(this.name, p.name) && this.age == p.age;  }  @Override  public String toString() {  return String.format("Person{name=\"%s\", age=%d}", name, age);  }  } |

Person01.java 예제를 다시 실행하고, 수정 전과 후 출력이 어떻게 달라졌는지 확인하자.

# String.format 메소드

## String.format 메소드 사용법

String 클래스의 format 메소드는 C언어의 printf 함수와 사용법이 유사하다.

이 메소드의 첫째 파라미터는 format string 이다.

|  |
| --- |
| "% [argument index] [flag] [width] [.precision] type" |

[argument index] : 출력할 argument의 index를 지정한다. 이 값이 주어지지 않으면, format string 뒤에 오는 argument들 순서대로 출력된다.

[flag] : 출력할 값이 숫자일 경우에, flag 가 + 이면 숫자 앞에 언제나 +- 부호를 붙인다. flag 가 0 이면, 숫자 앞에 0을 채워서 출력한다. 출력할 값이 문자열인 경우에, flag 가 - 이면 왼쪽 정렬로 출력되고, flag 가 없으면 오른쪽 정렬로 출력된다.

[width] : 출력할 최소 폭을 지정한다.

[.precession] : 숫자일 경우에 소숫점 아래 자릿수를 지정한다. 문자열일 경우에, 출력할 최대 문자수를 지정한다.

type : 출력할 값의 타입을 지정한다. d 정수, s 문자열, f 실수, x 16진수.

System.out.printf(...) 메소드는 출력할 내용을 화면에 출력한다.

String.format(...) 메소드는 출력할 내용을 문자열로 만들어서 리턴한다.

## Format01.java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40 | package net.skhu.lecture01;  public class Format01 {  public static void main(String[] args) {  String s;    s = String.format("Integer : %d\n", 15);  System.out.print(s);    s = String.format("Floating point number with 3 decimal digits: %.3f\n", 1.21312939123);  System.out.print(s);    s = String.format("Floating point number with 8 decimal digits: %.8f\n\n", 1.21312939123);  System.out.print(s);    s = String.format("String: %s, integer: %d, float: %.6f\n\n", "Hello World", 89, 9.231435);  System.out.print(s);    s = String.format("%-12s%-12s%s\n", "Column 1", "Column 2", "Column3");  System.out.print(s);    s = String.format("%-12d%-12d%07d\n\n", 15, 12, 5);  System.out.print(s);    s = String.format("%-8s%12s%12s\n", "Column 1", "Column 2", "Column3");  System.out.print(s);  s = String.format("%-8.5s%12d%12.4f\n\n", "Hello World", 1234567, 3.141592);  System.out.print(s);  s = String.format("%-12s%-12s\n", "Column 1", "Column 2");  System.out.print(s);  s = String.format("%-12.5f%.20f", 12.23429837482, 9.10212023134);  System.out.print(s);  }  } |

실행 결과 출력을 확인하자.

## Printf01.java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29 | package net.skhu.lecture01;  public class Printf01 {  public static void main(String[] args) {  System.out.printf("Integer : %d\n", 15);  System.out.printf("Floating point number with 3 decimal digits: %.3f\n", 1.21312939123);  System.out.printf("Floating point number with 8 decimal digits: %.8f\n\n", 1.21312939123);  System.out.printf("String: %s, integer: %d, float: %.6f\n\n", "Hello World", 89, 9.231435);  System.out.printf("%-12s%-12s%s\n", "Column 1", "Column 2", "Column3");  System.out.printf("%-12d%-12d%07d\n\n", 15, 12, 5);  System.out.printf("%-8s%12s%12s\n", "Column 1", "Column 2", "Column3");  System.out.printf("%-8.5s%12d%12.4f\n\n", "Hello World", 1234567, 3.141592);  System.out.printf("%-12s%-12s\n", "Column 1", "Column 2");  System.out.printf("%-12.5f%.20f", 12.23429837482, 9.10212023134);  }  } |

실행 결과 출력을 확인하자.