**enum 타입**

2016-09-04

이승진

**학습목표**

Java의 enum 타입의 장점과 활용 방법을 학습한다.

**목차**

[1. 혈액형 구현 #1 2](#_Toc468445408)

[1) BloodType.java 2](#_Toc468445409)

[2) Person.java 4](#_Toc468445410)

[3) BloodTypeTest.java 5](#_Toc468445411)

[2. 혈액형 구현 #2 6](#_Toc468445412)

[1) BloodType.java 6](#_Toc468445413)

[2) Person.java 7](#_Toc468445414)

[3) BloodTypeTest.java 8](#_Toc468445415)

[4) 결론 8](#_Toc468445416)

[3. 학점 구현 9](#_Toc468445417)

[1) Grade.java 9](#_Toc468445418)

[2) 과제 10](#_Toc468445419)

# 혈액형 구현 #1

먼저 혈액형을 클래스로 구현한 코드를 살펴보자.

## BloodType.java

src/net/skhu/lecture03/bloodType1/BloodType.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  41  42  43  44 | package net.skhu.lecture03.bloodType1;  import java.util.NoSuchElementException;  public class BloodType {    public static final BloodType A = new BloodType("A");  public static final BloodType B = new BloodType("B");  public static final BloodType O = new BloodType("O");  public static final BloodType AB = new BloodType("AB");    private String name;    private BloodType(String name) {  this.name = name;  }    @Override  public String toString() {  return this.name;  }    public static BloodType valueOf(String name) {  switch (name.toUpperCase()) {  case "A": return A;  case "B": return B;  case "AB" : return AB;  case "O": return O;  default: throw new NoSuchElementException(name);  }  }    public boolean canGiveTo(BloodType recipient) {  if (this == recipient) return true;  if (this == O) return true;  if (recipient == AB) return true;  return false;  }    public static BloodType[] values() {  return new BloodType[] { A, B, O, AB };  }  } |

(줄14)

private BloodType(String name) {

생성자가 private 이므로, BloodType 클래스 외부에서는 생성자를 호출할 수 없다.

따라서 BloodType 외부에서는 BloodType 객체를 생성할 수 없다.

(줄7~10)

public static final BloodType A = new BloodType("A");

public static final BloodType B = new BloodType("B");

public static final BloodType O = new BloodType("O");

public static final BloodType AB = new BloodType("AB");

BloodType 클래스 내부에서만 BloodType 객체를 생성할 수 있기 때문에,

BloodType 객체는 줄7~10에서 생성된 4개의 객체 뿐이다.

이 4객의 객체를 constant 처럼 사용하면 된다.

BloodType 클래스 외부에서는 BloodType.A, BloodType.B, BloodType.AB, BloodType.O 객체만 사용할 수 있다.

객체가 4개 뿐이기 때문에, 즉 내부 멤버 변수 값이 같은 객체가 여러개 일 수 없기 때문에,

equality와 identity가 같다.

즉 name 멤버 변수값이 "AB"인 BloodType 객체는 줄10에서 생성된 객체 한 개뿐이므로,

equality 비교와 identity 비교의 결과가 같다.

그래서 equals 메소드를 재정의 할 필요 없다.

(줄18)

@Override

public String toString() {

return this.name;

}

혈액형 객체에 해당하는 문자열을 리턴할 수 있도록 toString 메소드를 재정의하였다.

(줄23)

public static BloodType valueOf(String name) {

파라미터로 전달된 문자열에 해당하는 혈액형 객체를 리턴할 수 있도록 valueOf 메소드를 구현하였다.

이런 종류의 메소드의 이름이 valueOf 인 것이 관례이다.

(줄33)

public boolean canGiveTo(BloodType recipient) {

this 혈액형을 recipient 혈액형에 수혈할 수 있는지 true/false를 리턴하는 메소드이다.

혈액형이 같으면 수혈 할 수 있다.

주는 쪽이 O 형이면 수혈 할 수 있다.

받는 쪽이 AB 형이면 수혈 할 수 있다.

(줄40)

public static BloodType[] values() {

return new BloodType[] { A, B, O, AB };

}

모든 BloodType 객체 인스턴스들이 들어있는 배열을 리턴한다.

## Person.java

src/net/skhu/lecture03/bloodType1/Person.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 | package net.skhu.lecture03.bloodType1;  public class Person {  String name;  int age;  BloodType bloodType;    public Person(String name, int age, BloodType bloodType) {  this.name = name;  this.age = age;  this.bloodType = bloodType;  }    public String getName() {  return name;  }    public int getAge() {  return age;  }  public BloodType getBloodType() {  return bloodType;  }    @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.bloodType == p.bloodType &&  this.age == p.age;  }  @Override  public String toString() {  return String.format("Person{name=\"%s\", age=%d, bloodType=%s}", name, age, bloodType);  }  } |

(줄31) this.bloodType == p.bloodType

bloodType 멤버 변수 값이 참조 변수이지만,

equality 비교와 identity 비교의 결과가 동일하기 때문에,

identity 비교로 구현하였다.

this.bloodType == p.bloodType 에서 bloodType 멤버 변수 값이 null인 경우에도 null exception 에러가 발생하지 않는다. this의 bloodType 값과 p의 bloodType 값이 둘 다 null 이면, == 비교 결과는 true 이다.

(줄37) String.format("Person{name=\"%s\", age=%d, bloodType=%s}", name, age, bloodType);

bloodType 객체의 toString 메소드가 리턴하는 값이 출력된다.

## BloodTypeTest.java

src/net/skhu/lecture03/bloodType1/BloodTypeTest.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 | package net.skhu.lecture03.bloodType1;  public class BloodTypeTest {    public static void main(String[] args) {  Person p1 = new Person("홍길동", 18, BloodType.AB);  Person p2 = new Person("전우치", 24, BloodType.O);    System.out.println(p1);  System.out.println(p2);    BloodType blood1 = BloodType.valueOf("AB");  BloodType blood2 = BloodType.valueOf("A");  System.out.println(blood1.canGiveTo(blood2));  System.out.println(blood2.canGiveTo(blood1));    for (BloodType blood : BloodType.values())  System.out.println(blood);    String s = "";  if (p1.getBloodType() == BloodType.A) s = "A형";  else if (p1.getBloodType() == BloodType.B) s = "B형";  else if (p1.getBloodType() == BloodType.AB) s = "AB형";  else if (p1.getBloodType() == BloodType.O) s = "O형";  System.out.println(s);  }  } |

(줄6,7)

BloodType 클래스의 생성자가 private이기 때문에,

BloodType 클래스 외부에서는 BloodType 클래스 객체를 생성할 수 없고,

BloodType.A, BloodType.B, BloodType.AB, BloodType.O 객체만 사용할 수 있다.

(줄12) BloodType.AB 객체가 리턴된다.

(줄13) BloodType.A 객체가 리턴된다.

(줄14) AB 혈액을 A 혈액에 수혈할 수 없으므로 false가 리턴되어야 한다.

(줄15) A 혈액을 AB 혈액에 수혈할 수 있으므로 true가 리턴되어야 한다.

(줄17~18) 모든 BloodType 객체 인스턴스들의 toString 메소드 리턴 값을 화면에 출력한다.

# 혈액형 구현 #2

enum 타입으로 혈액형을 구현하자.

패키지 내에 이름이 같은 클래스가 두 개면 컴파일 에러가 발생한다.

정확히 얘기하자면, 패키지 내에서 class, interface, enum 들의 이름이 같은 것이 있으면

컴파일 에러가 발생한다.

혈액형 구현 #1 예제와 혈액형 구현 #2 예제의 클래스들 이름이 같기 때문에, 서로 패키지가 달라야 한다.

혈액형 구현 #1 예제의 패키지는 net.skhu.lecture03.bloodType1 이고,

혈액형 구현 #2 예제의 패키지는 net.skhu.lecture03.bloodType2 이다.

## BloodType.java

src/net/skhu/lecture03/bloodType2/BloodType.java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | package net.skhu.lecture03.bloodType2;  public enum BloodType {  A, B, O, AB;    public boolean canGiveTo(BloodType recipient) {  if (this == recipient) return true;  if (this == O) return true;  if (recipient == AB) return true;  return false;  }  } |

생성자를 private로 선언할 필요 없다.

생성자를 private로 선언하지 않아도 외부에서는 BloodType enum 타입의 객체 인스턴스를 생성하지 못한다.

(줄4) A, B, O, AB;

BloodType enum 타입의 객체 인스턴스는 이것 4개 뿐이다.

enum 타입에서 toString 메소드를 재정의할 필요 없다.

enum 타입의 객체 인스턴스 이름("A", "B", "O", "AB")을 문자열로 리턴하는

toString 메소드가 자동으로 생성된다.

enum 타입의 객체 인스턴스는 identity 비교와 equality 비교의 결과가 같다.

값이 같은 객체 인스턴스가 여러개 일 수 없기 때문이다.

그래서 equals 메소드를 재정의할 필요 없다.

public BloodType valueOf(String bloodTypeName);

위 메소드의 파라미터로 "A", "B", "O", "AB" 문자열을 전달하면,

BloodType.A, BloodType.B, BloodType.O, BloodType.AB 를 리턴하는 valueOf 메소드가 자동으로 생성된다.

따라서 enum 을 구현할 때 valueOf 메소드를 구현할 필요 없다.

public static BloodType[] values();

모든 BloodType 객체 인스턴스들이 들어있는 배열을 리턴하는 values 메소드가 자동으로 생성된다.

따라서 enum 을 구현할 때 values 메소드를 구현할 필요 없다.

## Person.java

src/net/skhu/lecture03/bloodType2/Person.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 | package net.skhu.lecture03.bloodType2;  public class Person {  String name;  int age;  BloodType bloodType;    public Person(String name, int age, BloodType bloodType) {  this.name = name;  this.age = age;  this.bloodType = bloodType;  }    public String getName() {  return name;  }    public int getAge() {  return age;  }  public BloodType getBloodType() {  return bloodType;  }    @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.bloodType == p.bloodType &&  this.age == p.age;  }  @Override  public String toString() {  return String.format("Person{name=\"%s\", age=%d, bloodType=%s}", name, age, bloodType);  }  } |

Person 클래스의 구현은, 사례#1과 동일하다.

## BloodTypeTest.java

src/net/skhu/lecture03/bloodType2/BloodTypeTest.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 | package net.skhu.lecture03.bloodType2;  public class BloodTypeTest {    public static void main(String[] args) {  Person p1 = new Person("홍길동", 18, BloodType.AB);  Person p2 = new Person("전우치", 24, BloodType.O);    System.out.println(p1);  System.out.println(p2);    BloodType blood1 = BloodType.valueOf("AB");  BloodType blood2 = BloodType.valueOf("A");  System.out.println(blood1.canGiveTo(blood2));  System.out.println(blood2.canGiveTo(blood1));    for (BloodType blood : BloodType.values())  System.out.println(blood);    String s = "";  switch (p1.getBloodType()) {  case A: s = "A형"; break;  case B: s = "B형"; break;  case AB: s = "AB형"; break;  case O: s = "O형"; break;  }  System.out.println(s);  }  } |

main 메소드 구현도 사례#1과 거의 동일하다.

(줄21~26)

BloodType이 클래스일 경우에, switch 문을 사용할 수 없으므로 if 문으로 구현했었다.

BloodType이 enum 타입일 경우에 switch 문을 사용할 수 있다.

Java의 switch 문에는 int, String, enum 타입을 사용할 수 있다.

## 결론

Java의 enum 타입은 참조형이고, enum 타입의 값들은 객체이다.

Java의 enum 타입은 클래스 타입과 매우 유사하다.

enum 타입이 클래스 타입과 다른 점은, enum 타입 내부에서 생성한 객체 인스턴스만 사용할 수 있고, 객체를 새로 만들 수는 없다는 점이다. 그리고 내부에서 생성한 객체 인스턴스들을 constant 처럼 사용한다.

클래스로 구현하는 편 보다 enum 타입으로 구현하는 편이 훨씬 간결하다.

# 학점 구현

## Grade.java

src/net/skhu/lecture03/Grade.java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | package net.skhu.lecture03;  public enum Grade {  Ap(4.5), A(4), Bp(3.5), B(3), Cp(2.5), C(2), Dp(1.5), D(1), F(0);  private double value;    Grade(double value) {  this.value = value;  }    public double getValue() {  return this.value;  }  } |

(줄4)

A+, A, B+, B, C+, C ...

이렇게 enum 항목 이름을 부여할 수 없다. Java의 identifier에 + 문자를 사용할 수 없기 때문이다.

그래서 대신 Ap, A, Bp, B, Cp, C ... 이름을 부여하였다.

줄4에서 객체 인스턴스 이름 뒤에 괄호는 생성자 파라미터이다.

(줄4)

Ap(4.5), A(4), Bp(3.5)

위 코드는 아래의 코드를 간결하게 구현한 것이라고 보면 된다.

public static final Grade Ap = new Grade(4.5);

public static final Grade A = new Grade(4);

public static final Grade Bp = new Grade(3.5);

enum 타입은 클래스와 유사하므로, 멤버 변수, 메소드, 생성자를 포함할 수 있다.

Java의 enum 타입을, enum 타입 내부에서 생성한 객체 인스턴스만 사용할 수 있는, 클래스 타입이라고 생각하면 된다.

Grade 클래스의 객체 인스턴스의 수는, 줄4에서 정의된 9개 뿐이다. 더 생성할 수 없다.

이 객체 인스턴스들을 constant 처럼 사용한다.

src/net/skhu/lecture03/GradeTest.java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | package net.skhu.lecture03;  public class GradeTest {    public static void main(String[] args) {    Grade[] myGrades = { Grade.Ap, Grade.A, Grade.Bp, Grade.B, Grade.A };    double sum = 0;  for (Grade grade : myGrades)  sum += grade.getValue();  System.out.printf("sum=%.2f average=%.2f\n", sum, sum / myGrades.length);  }  } |

## 과제

혈액형 구현#1을 참고해서, Grade.java를 class 버전으로 구현하라.