

능동적 사고 방식의

java

강사 박주병

Park Ju Byeong

Park Ju Byeong



Part13 기본클래스 및 제네릭

01 기본클래스

02 Util 패키지

03 제네릭

04 실습 문제



01

기본 클래스

java.lang 패키지

```
package joo.thirteen;

public class Main {

    public static void main(String[] args) {
        // TODO Auto-generated method stub

        String str = "스트링타입입니다.";

        System.out.print("a");

        Math.random();

    }
}
```

》》》 기본으로 import 해준다

import 없이 그냥 사용해왔다.

Object 클래스를 자세히 살펴보자

Objec.equals()

객체의 주소가 서로 같은지 비교 한다

```
People p1 = new People();  
People p2 = new People();
```

```
p1.RRN = "951212-1648444";  
p2.RRN = "951212-1648444";
```

```
System.out.println(p1.equals(p2));
```

p1

0x000A

p2

0x000C

메모리주소	값
0x000A	951212-1648444
0x000B	

메모리주소	값
0x000C	951212-1648444
0x000D	

```
<terminated> Mai  
false
```

Park Ju Byeong

```
People p1 = new People();  
People p2 = new People();  
  
p1.RRN = "951212-1648444";  
p2 = p1;  
|  
System.out.println(p1.equals(p2));
```

p1

0x000A

p2

0x000A

메모리주소	값
0x000A	951212-1648444
0x000B	

```
<terminated> Main  
true
```

값을 비교하여 true로 만들순 없을까?


```
public class People {
```

```
    int RRN ;  
    String name;
```

설명의 편의를 위해 int 사용

```
    @Override
```

```
    public boolean equals(Object obj) {  
        // TODO Auto-generated method stub  
  
        if(obj instanceof People)  
            return this.RRN == ((People)obj).RRN;  
        else  
            return false;  
    }
```

```
public static void main(String[] args) {  
    // TODO Auto-generated method stub
```

```
    People p1 = new People();  
    People p2 = new People();
```

```
    p1.RRN = 951212;  
    p2.RRN = 951212;
```

```
    System.out.println(p1.equals(p2));
```

```
PROBLEMS  
<terminated> M  
true
```

```
public static void main(String[] args) {  
    // TODO Auto-generated method stub  
  
    String a = "같을까?";  
  
    String b = new String("같을까?");  
  
    System.out.println(System.identityHashCode(a));  
    System.out.println(System.identityHashCode(b));  
  
    System.out.println(a.equals(b));  
}
```

<terminated> Main [

385337537

789219251

String 클래스는 이미 equals 를 오버라이딩 하여
값을 비교 하고 있다.

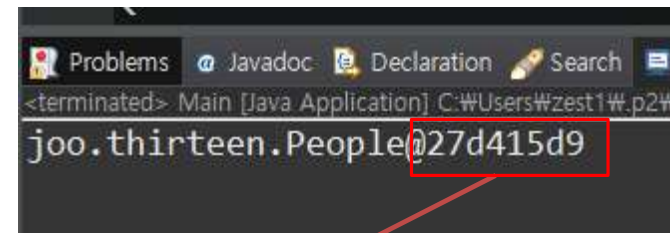
```
public class People {  
  
    String RRN ;  
    String name;  
  
    @Override  
    public boolean equals(Object obj) {  
        // TODO Auto-generated method stub  
  
        if(obj instanceof People)  
            return this.RRN.equals(((People)obj).RRN) ;  
        else  
            return false;  
    }  
}
```

문자열 비교시 String의 equals를 이용하면 된다.

Objec.toString()

객체의 상태를 문자열로 반환 한다.

```
People p1= new People();  
  
p1.name = "사람1";  
p1.RRN = 951225;  
  
System.out.println(p1.toString());
```



The screenshot shows an IDE console window with the following output: `<terminated> Main [Java Application] C:\Users\Wzest1\p2\W...` followed by `joo.thirteen.People@27d415d9`. The hash code `@27d415d9` is highlighted with a red box, and a red arrow points from it to the explanatory text below.

객체의 메모리 주소를 기반으로 생성한 해시코드
(객체마다 다른값을 가진다)

```
People p1= new People();  
People p2= new People();
```

```
p1.name = "사람1";  
p1.RRN = 951225;
```

```
p2.name = "사람1";  
p2.RRN = 951225;
```

```
System.out.println(p1.toString());  
System.out.println(p2.toString());
```

<terminated> Main [Java Application] C:\Users\zest1\p24

```
joo.thirteen.People@27d415d9  
joo.thirteen.People@5c18298f
```

```
People p1= new People();  
People p2= new People();
```

```
p1.name = "사람1";  
p1.RRN = "951225-1234567";
```

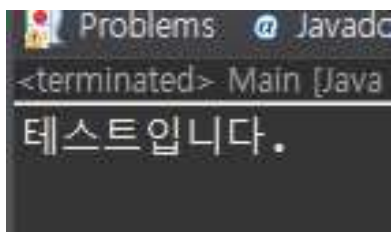
```
p2.name = "사람1";  
p2.RRN = "951225-1234567";
```

```
System.out.println(p1.toString());  
System.out.println(p2.toString());
```

<terminated> Main [Java Application] C:\Users\zest1\p2\pool

```
주민번호: 951225-1234567 이름: 사람1  
주민번호: 951225-1234567 이름: 사람1
```

```
String test = "테스트입니다.";
System.out.println(test.toString());
```



Problems @ Javado
<terminated> Main [Java
테스트입니다.

String 의 toString() 메서드는 이미 오버라이딩 되어 있다.

```
Date date = new Date();  
  
System.out.println(date.toString());
```



The screenshot shows an IDE console window with a toolbar at the top containing icons for Problems, Javadoc, Declaration, and Search. Below the toolbar, the text "<terminated> main [Java Application] C:\Users\zest1\p2\p" is visible. The main output of the program is "Sun Mar 12 08:49:52 KST 2023".

<terminated> main [Java Application] C:\Users\zest1\p2\p
Sun Mar 12 08:49:52 KST 2023

toString() 나는 사용할 일이 없을거 같은데...?
오버라이딩 하면 좋은가??

```
People p1 = new People();  
  
p1.age = 20;  
p1.name = "홍길동";  
  
System.out.println(p1.toString());
```



```
People p1 = new People();  
  
p1.RRN = "030101-1234567";  
p1.age = 20;  
p1.name = "홍길동";  
  
System.out.println(p1);
```

```
terminated> Main (3) [Java Application] C:\Users\USER545\p2\pool\plugins\org.eclipse.justj.c  
주민번호: 030101-1234567 이름: 홍길동 나이: 20
```

```
terminated> Main (3) [Java Application] C:\Users\USER545\p2\pool\plugins\org.eclipse.justj.c  
주민번호: 030101-1234567 이름: 홍길동 나이: 20
```

toString()을 이용하는 다른 라이브러리들을 편하게 쓸수 있다.

Object.clone()

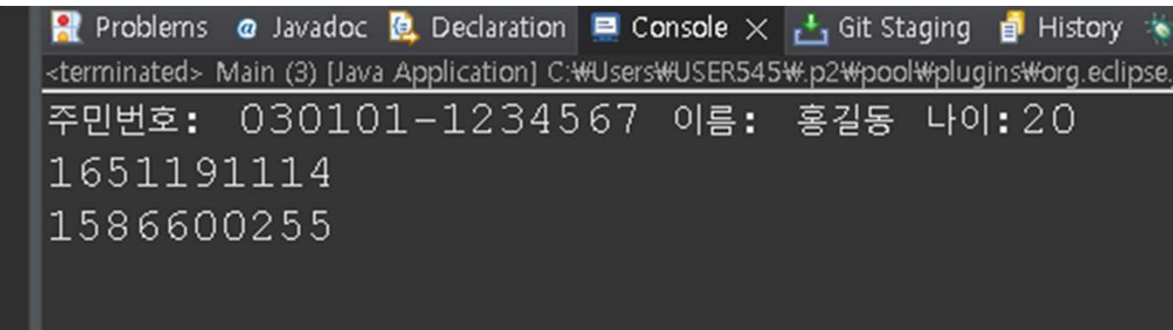
객체를 복사한다.

```
People p1 = new People();

p1.RRN = "030101-1234567";
p1.age = 20;
p1.name = "홍길동";

People p2 = p1.clone();

System.out.println(p2);
System.out.println(System.identityHashCode(p1));
System.out.println(System.identityHashCode(p2));
```

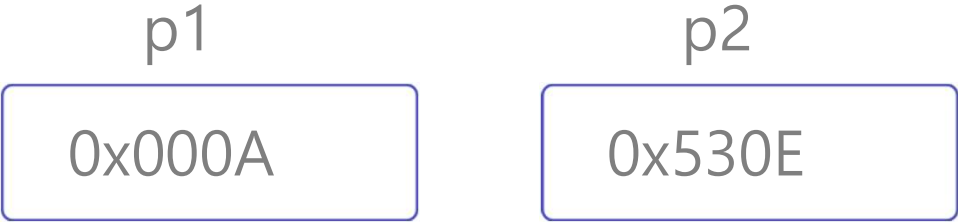


```
<terminated> Main (3) [Java Application] C:\Users\USER545\p2\pool\plugins\org.eclipse.  
주민번호: 030101-1234567 이름: 홍길동 나이: 20  
1651191114  
1586600255
```

```
People p1 = new People();  
  
p1.RRN = "030101-1234567";  
p1.age = 20;  
p1.name = "홍길동";  
  
People p2 = p1;  
  
System.out.println(p2);  
System.out.println(System.identityHashCode(p1));  
System.out.println(System.identityHashCode(p2));
```

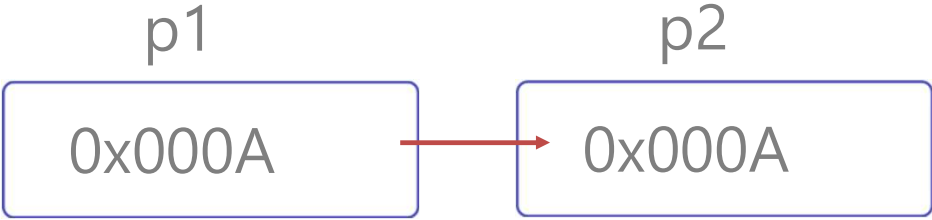
```
<terminated> Main (3) [Java Application] C:\Users\USER545\p2\pool\plugins\org.eclipse  
주민번호: 030101-1234567 이름: 홍길동 나이: 20  
1651191114  
1651191114
```

```
People p2 = p1.clone();
```



변수	값		변수	값
RRN	111111-1111111	→	RRN	111111-1111111
age	20	→	age	20
name	홍길동	→	name	홍길동

```
People p2 = p1;
```



변수	값
RRN	111111-1111111
age	20
name	홍길동

```
public static void main(String[] args) {  
    // TODO Auto-generated method stub
```

```
    People p1 = new People();
```

p1.

- name : String - People
- RRN : String - People
- equals(Object obj) : boolean - People
- getClass() : Class<?> - Object
- hashCode() : int - Object
- notify() : void - Object
- notifyAll() : void - Object
- toString() : String - People
- wait() : void - Object
- wait(long timeoutMillis) : void - Object
- wait(long timeoutMillis, int nanos) : void - Object

Press 'Ctrl+Space' to show Template Proposals

```
public class People
```

```
@Override
```

```
public String toString() {
```

```
    return "주민번호: " + RRN + " 이름: " + name;
```

```
}
```

- People() - Constructor
- People - joo.thirteen
- clone() : Object - Override method in 'Object'
- finalize() : void - Override method in 'Object'
- hashCode() : int - Override method in 'Object'
- private_method - private method
- private_static_method - private static method
- protected_method - protected method
- public_method - public method
- public_static_method - public static method
- abstract
- class

Press 'Ctrl+Space' to show Template Proposals

History Debug
jdk.hotspot.jre.full

Object에 있다면서??!!

People클래스 내부에서 보니 보인다

clone(), finalize() 는 protected로 되어 있다.

```
@Override
protected Object clone() throws CloneNotSupportedException {
    return super.clone();
}
```



오버라이딩시 접근제어를 넓게 확장은 가능하다

```
@Override
public Object clone() throws CloneNotSupportedException {
    return super.clone();
}
```



```
@Override
public People clone() {
```

공변반환타입이 적용되기에 본인타입으로 변경하여도 오버라이딩 가능



Cloneable를 구현하지 않으면 clone() 메서드 사용시 예외를 발생시킨다.

```
public class People implements Cloneable{
```

최종완성된 clone() 오버라이딩

```
public class People implements Cloneable{
```

```
@Override
```

```
public People clone() {
```

```
    Object result = null;
```

```
    try
```

```
    {
```

```
        result = super.clone();
```

```
    } catch (CloneNotSupportedException ex)
```

```
    {
```

```
        System.out.println(ex);
```

```
    }
```

```
    return (People)result;
```

```
    }
```

People이 멤버변수로 객체를 가지고
있으면 clone() 사용시 어떻게 될까?

```
public class People implements Cloneable{  
  
    String RRN ;  
    String name;  
    int age;  
    Phone phone;
```

p1

0x000A

p2

0x530E

변수	값
RRN	111111-11111111
age	20
name	홍길동
Phone	0x001b

0x001b

변수	값
RRN	111111-11111111
age	20
name	홍길동
Phone	

0xA010

변수	값
number	010-3900-7555

변수	값
number	010-3900-7555


```
People p1 = new People();
p1.name = "홍길동";
p1.age = 10;
p1.phone = new Phone("삼성 s20",123456);

People p2 = p1.clone();

System.out.println(p2.name);
System.out.println(p2.age);
System.out.println(p2.phone.Model);
System.out.println(p2.phone.serialNumber);
```

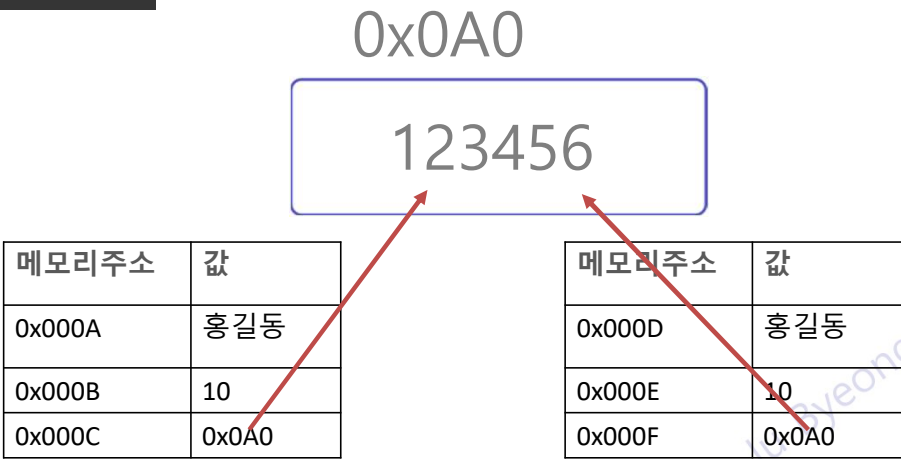
```
홍길동
10
삼성 s20
123456
```

```
People p1 = new People();
p1.name = "홍길동";
p1.age = 10;
p1.phone = new Phone("삼성 s20",123456);

People p2 = p1.clone();

System.out.println(System.identityHashCode(p1.phone));
System.out.println(System.identityHashCode(p2.phone));
```

```
1545087375
1545087375
```

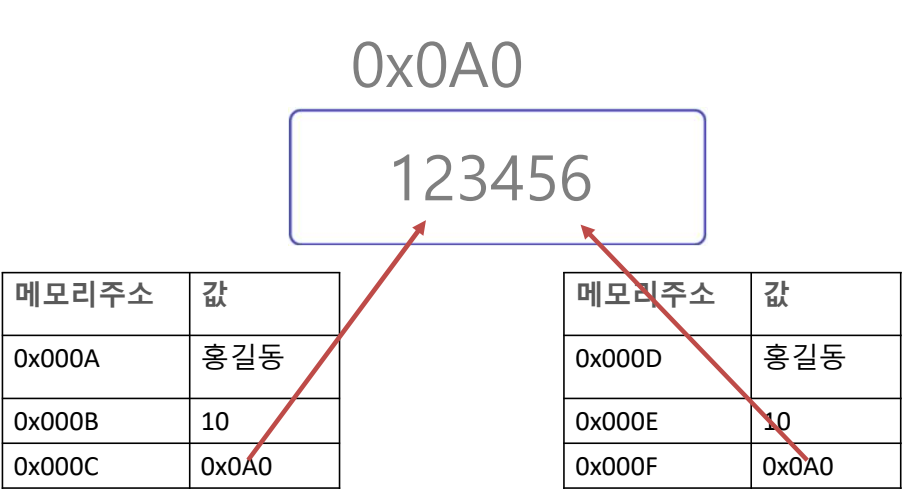


```
People p1 = new People();  
p1.name = "홍길동";  
p1.age = 10;  
p1.phone = new Phone("삼성 s20", 123456);  
  
People p2 = p1.clone();  
|  
p2.phone.serialNumber = 789;  
System.out.println(p1.phone.serialNumber);
```

```
<terminated  
789
```

주소값만 복사할게 아니라 메모리 자체를 새로
할당하여 복사를 해야 한다.

얕은복사 VS 깊은복사



얕은복사



깊은복사

깊은복사 구현

```
@Override
public People clone() {
    Object result = null;
    try
    {
        result = super.clone();
    } catch (CloneNotSupportedException ex)
    {
        System.out.println(ex);
    }
    return (People)result;
}

public People deepCopy()
{
    People result = this.clone();

    result.phone = new Phone(this.phone.Model, this.phone.serialNumber);

    return result;
}
```

```
People p1 = new People();
p1.name = "홍길동";
p1.age = 10;
p1.phone = new Phone("삼성 s20", 123456);

People p2 = p1.deepCopy();

p2.phone.serialNumber = 789;
System.out.println(p1.phone.serialNumber);
```

```
<terminated> main
123456
```

Phone 클래스에서 clone을 오버라이딩하여 만든후 활용하는것이 좀더 객체지향적이다.

```
int[] list = new int[5];  
  
list[0] = 10;  
list[1] = 20;  
list[2] = 30;  
list[3] = 40;  
list[4] = 50;  
int[] newList = list.clone();  
|  
for(int i : newList)  
    System.out.println(i);
```

배열은 기본적으로 clone()을 오버라이딩 하여 구현해놨다.

String 클래스

char[] 대신 문자열을 쉽게 다룰수 있는 클래스

String을 상속할수 없다.

```
9 public final class String
1     implements java.io.Serializable, Comparable<String>, CharSequence {
2         private final char[] value;
3
4         /**
5          * The value is used for character storage.
6          *
7          * @implNote This field is trusted by
8          * constant folding if String instance
9          * field after construction will cause
10          *
11          * Additionally, it is marked with {@link
12          * of the array. No other facility in
13          * {@link Stable} is safe here, because
14          */
15         @Stable
16         private final byte[] value;
```

char 배열 이라고 생각하면 된다.

상수이므로 값을 변경할수 없다.

?????? 변경 되는데???

```
String str = "스트링입니다.";

str = "변경 되는데?";
System.out.println(str);
```

```
<terminated> main [Java Ap
변경 되는데?
```

```
String str = "스트링입니다.";
System.out.println(System.identityHashCode(str));

str = "변경 되는데?";
System.out.println(System.identityHashCode(str));
```

```
<terminated> main [Java
385337537
789219251
```

값 수정이 아니라 새로운 String 객체를 만드는 것이다

문자열의 수정이 잦다면 StringBuffer 클래스를 활용하자

```
String str = "스트링입니다.";
str += "문자열을 뒤에 추가 합니다." ;

StringBuffer str2 = new StringBuffer("스트링입니다.");
System.out.println(System.identityHashCode(str2));
str2.append("문자열을 뒤에 추가 합니다.");
System.out.println(System.identityHashCode(str2));|
```

```
<terminated> main
789219251
789219251
```


String도 클래스니깐 생성자를 통해 객체 생성도 할수 있다.

```
String str =new String("문자열입니다.");
```

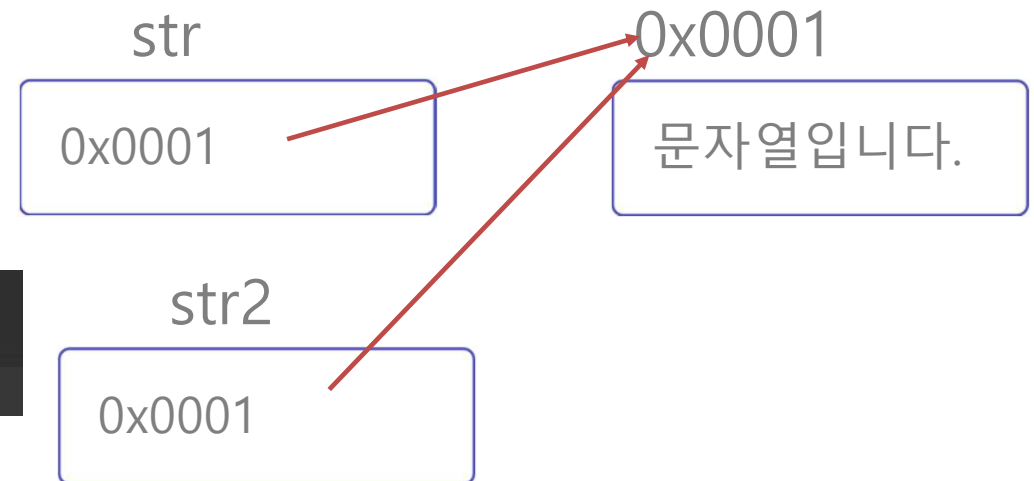
```
String str ="문자열입니다.";
```

뭐가 다른걸까?

```
String str = "문자열입니다.";
```

리터럴 상수를 만들고 그
주소를 String 객체에
넘겨준다.

```
String str2 = "문자열입니다.";
```

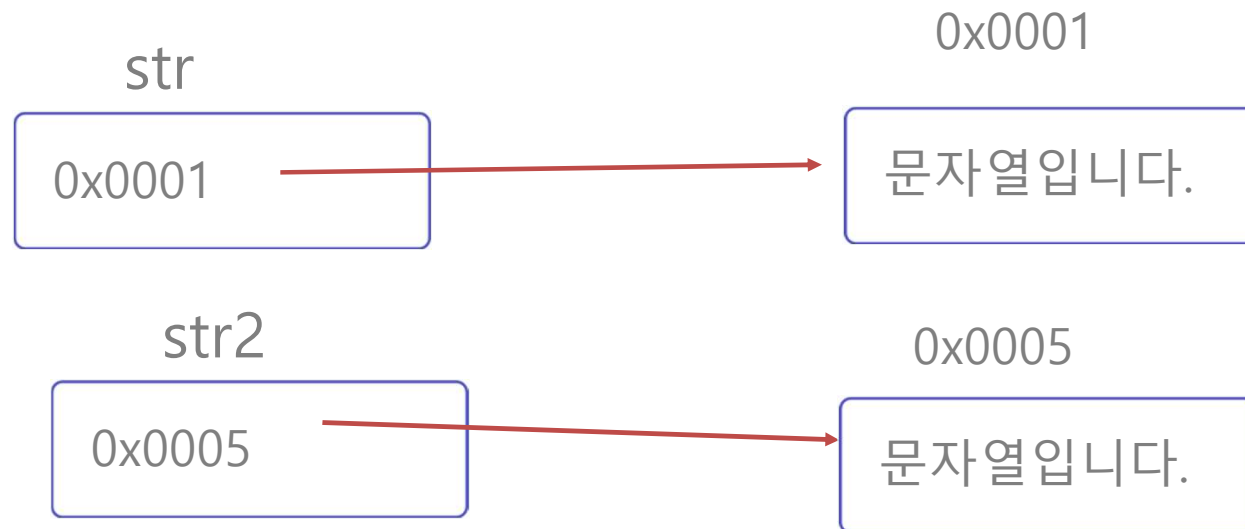


```
System.out.println(System.identityHashCode(str));  
System.out.println(System.identityHashCode(str2));
```

```
<terminated> main [Java A  
385337537  
385337537
```

```
String str =new String("문자열입니다.");  
String str2 =new String("문자열입니다.");  
  
System.out.println(System.identityHashCode(str));  
System.out.println(System.identityHashCode(str2));
```

```
<terminated> main [Jav  
385337537  
789219251
```



비교연산자를 이용한 문자열 비교

```
String str = new String("문자열입니다.");  
String str2 = new String("문자열입니다.");  
System.out.println(str == str2);|
```

```
<terminated> main [  
false
```

String 클래스의 equals

```
String str = new String("문자열입니다.");  
String str2 = new String("문자열입니다.");  
  
System.out.println(str.equals(str2));
```



Problems
<terminated>
true

Object가 물려준 equals 메서드는 단순 주소값을 비교한다
그러나 String클래스를 실질적인 데이터를 비교하도록 오버라이딩 되어 있다.

```
String name = new String("홍길동");  
String name2 = new String("홍길동");  
  
if(name.equals(name2))  
    System.out.println("이름이 같아요!");
```

```
<terminated> main [Java App  
이름이 같아요!
```

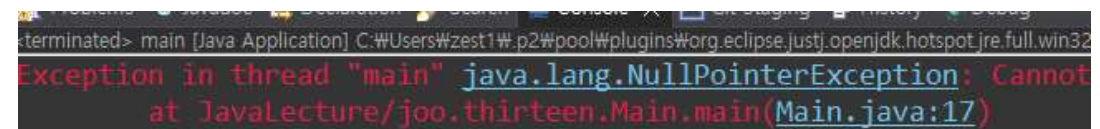
문자열 비교를 한다는것은 내용을 비교할 목적이다.
equals() 를 사용하자!

String 클래스의 초기화

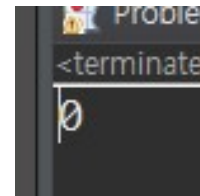
```
String name = null;  
String name2 = "";
```

```
System.out.println(name.length());
```

```
System.out.println(name2.length());
```



terminated> main [Java Application] C:\Users\zest1\p2\pool\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32
Exception in thread "main" java.lang.NullPointerException: Cannot
at JavaLecture/joo.thirteen.Main.main(Main.java:17)



Problem
<terminated>
0

String.join() 과 String.split()

문자열을 구분자 기준으로 합치거나 나눈다.

```
String name = "홍길동,김길동,이길동,박길동";  
String[] split = name.split(",");  
for(String str : split)  
    System.out.println(str);
```

```
<terminated> main  
홍길동  
김길동  
이길동  
박길동
```

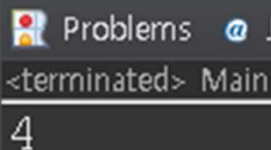
```
String newList = String.join("-", split);  
System.out.println(newList);
```

```
홍길동-김길동-이길동-박길동
```


String.indexOf() 메서드

매개변수로 받은 문자열을 찾아 시작위치를 반환한다.

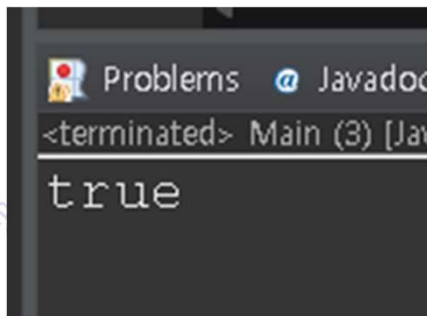
```
String str = "abcdefag";  
  
int index = str.indexOf("d");  
  
System.out.println(index);
```



Problems @ J
<terminated> Main
4

위치까지는 필요없고 포함되어 있는지 여부만 확인하고 싶으면

```
String str = "abcdefag";  
  
int index = str.indexOf("d");  
  
boolean isContaine = str.contains("d");  
  
System.out.println(isContaine);
```



Warpper 클래스

기본형 타입들을 클래스화 한것

기본형	클래스명
boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	double

왜 만든걸까?

클래스화 하여 다양한 메서드들을 제공한다.

```
Integer a = 15;  
String b = a.toString();  
System.out.println(b);  
int c = Integer.parseInt(b);
```

```
<terminated> r  
15
```

제너릭 같이 클래스 타입을 요구할때 사용해야 한다.

```
List<int> list = new ArrayList<int>();
```



```
List<Integer> list = new ArrayList<Integer>();  
  
list.add(15);  
list.add(300);  
list.add(500);
```

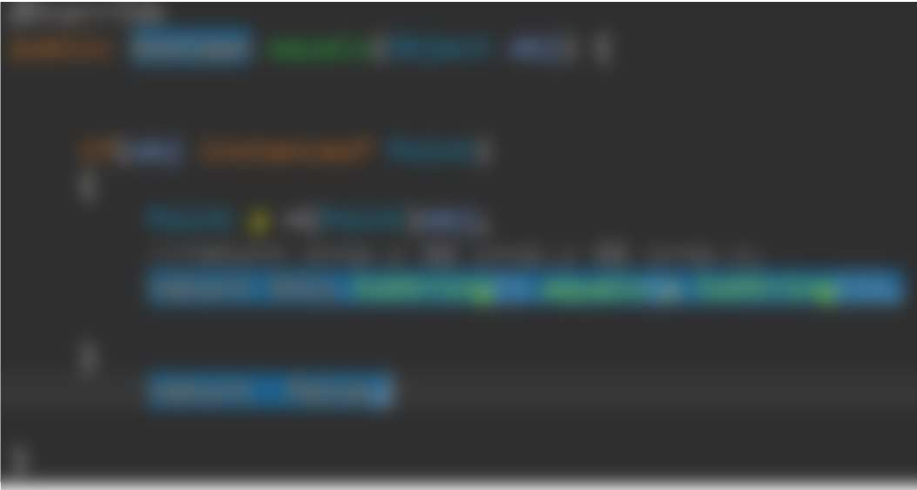
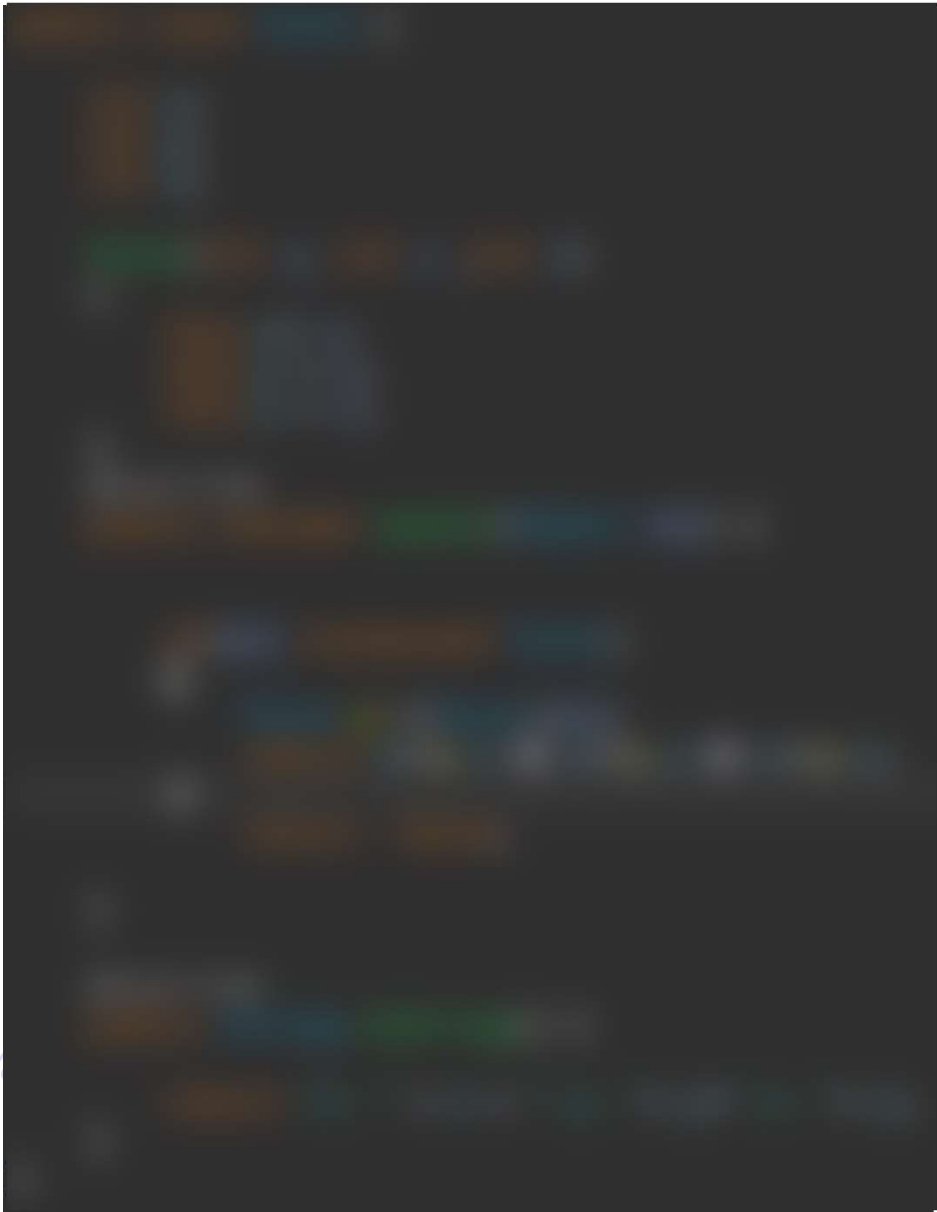
실습문제1

1.Point 클래스를 만드시오

- 멤버변수 int x, int y, int z 를 가진다.
- 생성자, toString() 과 equals() 메서드를 오버라이딩 하자.

```
Point pt1 = new Point(10,20,30);  
Point pt2 = new Point(10,20,30);  
System.out.println(pt1.equals(pt2));  
|  
System.out.println(pt1.toString());
```

```
<terminated> main [Java Application] C:  
true  
x: 10 y: 20 z: 30
```

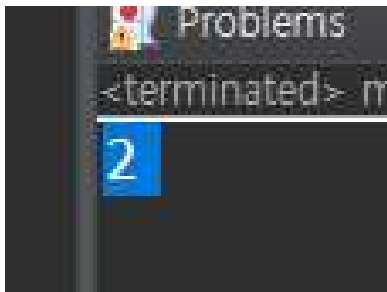


Park Ju Byeong

2.문자열에서 원하는 문자열이 몇 개 포함되어 있는지 찾는 count() 메서드를 만드시오

indexOf() 메서드를 활용하시오

```
String str = "abcdefgab";  
System.out.println(count(str, "ab"));
```

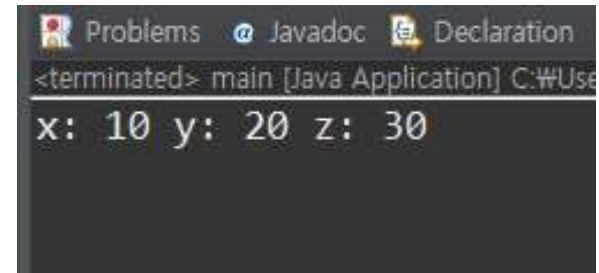


```
public static int count(String str, String target)  
{  
    // 문자열이 비어있을 경우  
    if (str == null || str.isEmpty())  
        return 0;  
    // 문자열이 비어있을 경우  
    if (target == null || target.isEmpty())  
        return 0;  
    // 문자열이 비어있을 경우  
    if (target.length() > str.length())  
        return 0;  
    // 문자열이 비어있을 경우  
    if (target.length() == str.length())  
        return 1;  
    // 문자열이 비어있을 경우  
    if (target.length() < str.length())  
        return 1;  
    // 문자열이 비어있을 경우  
    return count(str, target, 0);  
}  
  
return count;
```

3. Warrior 클래스를 만드시오

- 멤버변수 String id, int hp, 캐릭터의 위치를 저장할 Point pt
- 생성자, clone() 메서드를 오버라이딩 하시오
- clone()를 활용하여 얇은복사, 깊은복사를 구현하시오
- 얇은복사 : copy() 깊은복사: deepCopy()

```
public static void main(String[] args) {  
    // TODO Auto-generated method stub  
  
    Warrior w1 = new Warrior("전사1", 100, new Point(10,20,30));  
  
    Warrior w2= w1.deepCopy();  
    |  
    System.out.println(w2.pt.toString());  
}
```



Problems Javadoc Declaration
<terminated> main [Java Application] C:\#Use
x: 10 y: 20 z: 30

— 02

Util 패키지

Util 패키지의 Random 클래스

`Math.random()`이 아니다.

Random 클래스-> java.Util 패키지의 클래스

Math 클래스 -> java.lang 패키지의 클래스

난수를 생성 하는 방법

1. Math.random() 메서드 활용(java.lang)
2. Random 클래스 활용(java.util)

```
int a =(int)(Math.random()*5)+1;  
int b = new Random().nextInt(6)+1;
```

차이가 뭘까?

```
/**
 * Creates a new random number generator. This constructor sets
 * the seed of the random number generator to a value very likely
 * to be distinct from any other invocation of this constructor.
 */
public Random() {
    this(seedUniquifier() ^ System.nanoTime());
}

private static long seedUniquifier() {
    // L'Ecuyer, "Tables of Linear Congruential Generators of
    // Different Sizes and Good Lattice Structure", 1999
    for (;;) {
        long current = seedUniquifier.get();
        long next = current * 1181783497276652981L;
    }
}
```

```
public Random(long seed) {
    if (getClass() == Random.class)
        this.seed = new AtomicLong(initialScramble(seed));
    else {
        // subclass might have overridden setSeed
        this.seed = new AtomicLong();
        setSeed(seed);
    }
}
```

시드값을 사용한다

```
Random random1= new Random(10);  
Random random2= new Random(10);  
|  
for(int i=0;i<5;i++)  
    System.out.println(random1.nextInt());  
  
System.out.println();  
for(int i=0;i<5;i++)  
    System.out.println(random2.nextInt());
```

```
-1157793070  
1913984760  
1107254586  
1773446580  
254270492  
  
-1157793070  
1913984760  
1107254586  
1773446580  
254270492
```

기본 생성자는 시간값을 시드로 사용하기에 항상 다른값이 나온다.

Math.random() VS Random 클래스

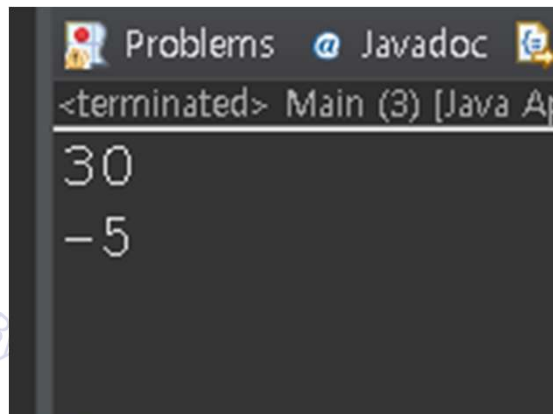
```
private static final class RandomNumberGeneratorHolder {  
    static final Random randomNumberGenerator = new Random();  
}
```

```
/**  
 * Returns a {@code double} value with a positive sign, greater  
 * than or equal to {@code 0.0} and less than {@code 1.0}.
```

Math 클래스는 내부에서 Random 클래스를 이용한다

Math.abs() Math.min()

```
int result = Math.abs(-30);  
  
System.out.println(result);  
  
result = Math.min(15, -5);  
System.out.println(result);
```



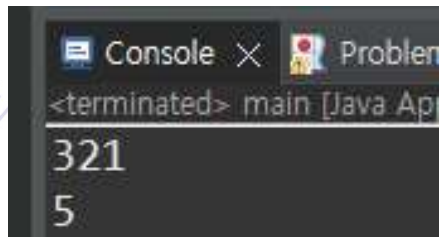
정규식

```
String[] list = {"fdsa", "321", "a23aaa", "5", "ttt"};

//정규식을 객체화 한다.
Pattern p = Pattern.compile("[0-9]*$");

for(String str : list)
{
    //정규식과 비교할 데이터를 넣어준다.
    Matcher m = p.matcher(str);

    //일치여부를 반환한다.
    if(m.matches())
        System.out.println(str);
}
```



```
<terminated> main [Java Ap
321
5
```

정규표현식	표현	설명
x		문자열이 x로 시작합니다.
$x\$$		문자열이 x로 끝납니다.
$.x$		임의의 한 문자를 표현합니다. (x가 마지막으로 끝납니다.)
x^+		x가 1번이상 반복합니다.
$x^?$		x가 존재하거나 존재하지 않습니다.
x^*		x가 0번이상 반복합니다.
$x y$		x 또는 y를 찾습니다. (or연산자를 의미합니다.)
(x)		()안의 내용을 캡처하며, 그룹화 합니다.
$(x)(y)$		그룹화 할 때, 자동으로 앞에서부터 1번부터 그룹 번호를 부여해서 캡처합니다. 결과값에 그룹화한 Data가 배열 형식으로 그룹번호 순서대로 들어갑니다.
$(x)(?:y)$		캡처하지 않는 그룹을 생성할 경우 ?:를 사용합니다. 결과값 배열에 캡처하지 않는 그룹은 들어가지 않습니다.
$x\{n\}$		x를 n번 반복한 문자를 찾습니다.
$x\{n,\}$		x를 n번이상 반복한 문자를 찾습니다.
$x\{n,m\}$		x를 n번이상 m번이하 반복한 문자를 찾습니다.

BigInteger 클래스

Long보다 큰 정수형 타입

```
java.math.BigInteger test ;

//long temp = 41564848482131564189461614894536149841651896;

test = new BigInteger("41564848482131564189461614894536149841651896");
test = new BigInteger("F9F0",16);
String str = test.toString();
int integer = test.intValue();
System.out.println(str);
System.out.println(integer);

java.math.BigInteger result = test.add(new BigInteger("41564848482131564189441651896")); //더하기
result = test.subtract(new BigInteger("415648484821651896")); //빼기
result = test.multiply(new BigInteger("415648484821651896")); //곱셈
result = test.divide(new BigInteger("415648484821651896")); //나눗셈
result = test.remainder(new BigInteger("415648484821651896")); //나머지 연산
```

```

public class BigInteger extends Number implements Comparable<BigInteger> {
    /**
     * The signum of this BigInteger: -1 for negative, 0 for zero, or
     * 1 for positive. Note that the BigInteger zero must have
     * a signum of 0. This is necessary to ensure that there is exactly one
     * representation for each BigInteger value.
     */
    final int signum;
    /**
     * The magnitude of this BigInteger, in big-endian order: the
     * zeroth element of this array is the most-significant int of the
     * magnitude. The magnitude must be "minimal" in that the most-significant
     * int ({@code mag[0]}) must be non-zero. This is necessary to
     * ensure that there is exactly one representation for each BigInteger
     * value. Note that this implies that the BigInteger zero has a
     * zero-length mag array.
     */
    final int[] mag;

```

부호를 의미 (2의보수 형태로 처리된다.)

실질적인 데이터

final 이므로 String처럼 서로 연산을 수행한결과가 새로운 BigInteger 객체를 생성한다.

float은 소수점 7자리가 넘으면 오차가 발생하니
double을 사용하자

double 역시 오차가 발생한다. 이는 부동소수점의 태생적 한계이다.
거기다가 소수점 7자리 이상의 데이터를 다루는 프로그램은 상당한
정밀도를 요하는 것이기에 오류 발생 가능성을 아예 없애야 한다.

```
System.out.println(0.1+0.2);  
/*
```

Problems Javadoc Declaration Console
<terminated> Main (3) [Java Application] C:\Users\USER5
0.30000000000000004

BigDecimal 클래스

int[]을 활용하여 오차가 없는 실수를 구현

```
*/  
public class BigDecimal extends Number implements Comparable<BigDecimal> {  
    /**  
     * The unscaled value of this BigDecimal, as returned by {@link  
     * #unscaledValue}.  
     *  
     * @serial  
     * @see #unscaledValue  
     */  
    private final BigInteger intVal;
```

BigInteger는 int[] 을 사용한다.

final 이므로 값 변경시 새로운 객체를 생성한다.


```
java.math.BigDecimal val = new BigDecimal("123.356");  
System.out.println(val);  
  
val = BigDecimal.valueOf(123.456);  
System.out.println(val);  
  
val = new BigDecimal(123.356);  
System.out.println(val);|
```

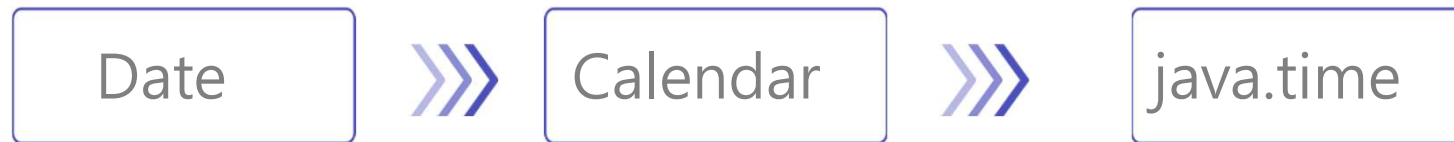
Problems Javadoc Declaration Search Console x Git Staging
<terminated> main [Java Application] C:\Users\zest1\p2\pool\plugins\org.eclipse.justj
123.356
123.456
123.35599999999999994543031789362430572509765625

생성자로 리터럴 실수를
넘기면 오차가 발생한다

```
val = val.add(new BigDecimal("3.141592"));  
val = val.subtract(new BigDecimal("3.141592"));  
val = val.multiply(new BigDecimal("3.141592"));  
val = val.divide(new BigDecimal("3.141592"));  
val = val.remainder(new BigDecimal("3.141592"));
```

Date & Calendar 클래스

날짜와 시간을 다루는 클래스



```
Date d1 = new Date();  
d1.getDate();
```

→ 대부분 deprecated 이다.

```
*/  
@Deprecated  
public int getDate() {  
    return normalize().getDayOfMonth();  
}
```

Calendar

```
* @since 1.1
*/
public abstract class Calendar implements Serializable {
    // Data flow in Calendar
    // -----
}
```

→ 객체 생성 불가!

```
Calendar d = Calendar.getInstance();
```

→ 싱글톤인가?

```
public static Calendar getInstance()
{
    Locale aLocale = Locale.getDefault(Locale.Category.FORMAT);
    return createCalendar(defaultTimeZone(aLocale), aLocale);
}
```

Calendar를 상속받아 구현해놓은 클래스들이 있는데 지역에 따라 다른 클래스를 객체화 한다.

Calendar 사용법

```
Calendar cal = Calendar.getInstance(); // 기본은 현재 시스템의 시간
cal.set(2010, Calendar.OCTOBER, 15);
System.out.print(cal.get(Calendar.YEAR)+"-");
System.out.print(cal.get(Calendar.MONTH)+1+"-"); // 0~11 이므로 1을 더한다
System.out.println(cal.get(Calendar.DATE));
System.out.print(cal.get(Calendar.HOUR)+":");
System.out.print(cal.get(Calendar.MINUTE)+":");
System.out.println(cal.get(Calendar.SECOND));
System.out.println(cal.get(Calendar.MILLISECOND)); // 1000분의 1초
System.out.println(cal.get(Calendar.DAY_OF_WEEK)); // 1~7 일~토
```

<terminated> main [Java Application]

2010-11-15

8:45:52

693

2 → 월요일

Calendar 시간차 계산

```
Calendar cal = Calendar.getInstance(); //기분은 현재 시스템의 시간
cal.set(2010,Calendar.OCTOBER,15);

Calendar cal2 = Calendar.getInstance();

long result = cal2.getTimeInMillis() - cal.getTimeInMillis();
System.out.println(result + "밀리세컨초 차이 납니다.");
System.out.println(result/1000 + "초 차이 납니다.");
System.out.println(result/1000/60 + "분 차이 납니다.");
System.out.println(result/1000/60/60 + "시 차이 납니다.");
System.out.println(result/1000/60/60/24 + "일 차이 납니다.");
System.out.println(result/1000/60/60/24/365 + "년 차이 납니다.");
```

```
<terminated> main [Java Application] C:\Users\zest1\p2\po
391737600000밀리세컨조 차이 납니다.
391737600초 차이 납니다.
6528960분 차이 납니다.
108816시 차이 납니다.
4534일 차이 납니다.
12년 차이 납니다.
```

60*60*24 VS 86400

리터럴은 컴파일단계에서 미리 다 계산되어
진다.

Calendar ⇔ Date 변환

```
Calendar cal = Calendar.getInstance();  
//Calendar -> Date 변환  
Date d = new Date(cal.getTimeInMillis());  
//Date -> Calendar 변환  
cal.setTime(d);
```

```
Calendar cal = Calendar.getInstance(); //기본은 현재 시스템의 시간  
Date date = cal.getTime();
```

SimpleDateFormat 클래스

```
Calendar cal = Calendar.getInstance();//기본은 현재 시스템의 시간
Date date = cal.getTime();

System.out.println(new SimpleDateFormat("yyyy-MM-dd").format(date));
System.out.println(new SimpleDateFormat("yyyy.MM.dd").format(date));
System.out.println(new SimpleDateFormat("yy년 MM월 dd일").format(date));
System.out.println(new SimpleDateFormat("yyyy-MM-dd HH:mm:ss.SSS").format(date));
System.out.println(new SimpleDateFormat("yyyy-MM-dd HH:mm:ss a").format(date));
System.out.println(new SimpleDateFormat("E요일입니다").format(date));
```

<terminated> main [Java Application] C:\Users\zest

```
2023-03-15
2023.03.15
23년 03월 15일
2023-03-15 22:28:12.223
2023-03-15 22:28:12 오후
수요일입니다
```

java.time 패키지



LocalDate : 날짜
LocalTime : 시간
LocalDateTime : 날짜+시간
ZonedDateTime : 날짜+시간+시간대

```
LocalDate date = LocalDate.now();
LocalTime time = LocalTime.now();
LocalDateTime dateTime = LocalDateTime.now();
ZonedDateTime zoneDateTime = ZonedDateTime.now();
```

→ 객체를 직접 생성하지 못한다.

// 값 초기화

```
date = date.of(2014, 10, 15);
dateTime = dateTime.of(date, time);
zoneDateTime = zoneDateTime.of(dateTime, ZoneId.of("Asia/Seoul"));
```

```
System.out.println(date.getYear());
System.out.println(date.getMonthValue());
System.out.println(date.getDayOfMonth());
System.out.println(time.getHour());
System.out.println(time.getMinute());
System.out.println(time.getSecond());
```

<terminated

2014

10

15

23

5

29

Park Ju Byeong

with로 시작하는 메서드를 통해 특정 필드를
변경한다

```
date.withYear(2015);  
date.withMonth(5);  
date.withDayOfMonth(3);
```

```
date = date.withYear(2015);  
date = date.withMonth(5);  
date = date.withDayOfMonth(3);
```

왜 자기 자신한테 다시 대입하는걸까?

String처럼 내부는 final(상수)로 정의되어 값 수정시 새로운 객체를 만든다.

쓰레드 사용할때 상수가 아니면 같은 자원을 동시에 접근시
값수정에 의해 오류가 발생할수 있다.

```
dateTime.format(DateTimeFormatter.ofPattern("yyyy년 MM월 dd일"));
dateTime.format(DateTimeFormatter.ofPattern("yyyy-MM-dd"));
dateTime.format(DateTimeFormatter.ofPattern("yyyy/MM/dd HH:mm:ss"));
System.out.println(dateTime.format(DateTimeFormatter.ofPattern("yyyy/MM/dd HH:mm:ss")));
```

```
<terminated> main [Java Application] C:\User
2014
10
15
23
21
14
2014/10/15 23:21:14
```

java.time 패키지들은 DateTimeFormatter 클래스를 이용하여 형식화 한다.

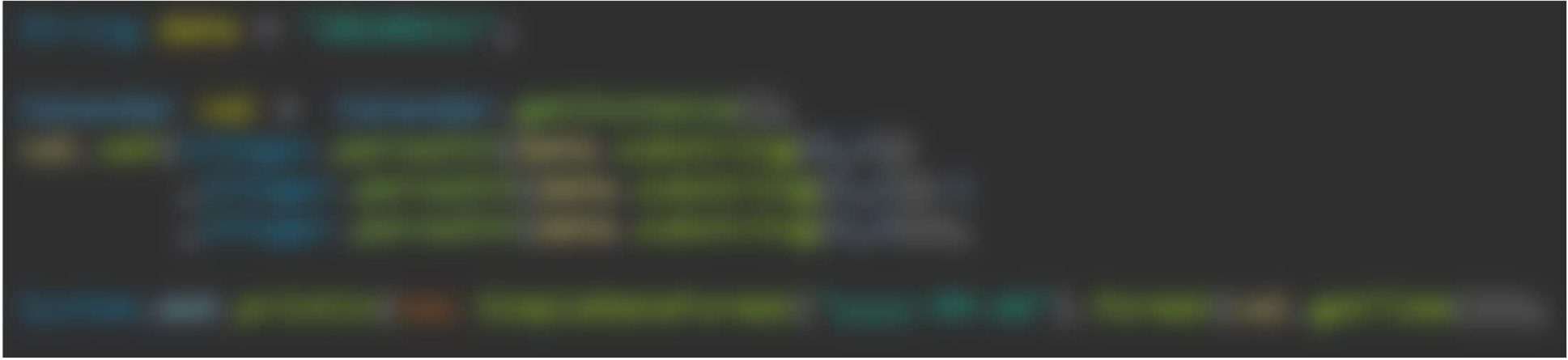
실습문제2

1. Calendar, SimpleDateFormat 클래스를 활용하여 String 으로된 날짜 정보를 Calendar 객체로 만든후 아래와 같이 출력해보자
 - 단순 String을 수정하여 출력 하지말것!

```
String date = "20190311";
```



```
Console X Prot  
<terminated> Main [Java  
2019-03-11
```

2. getRandom() 메서드를 만들어보자

- getRandom(-5,10) 호출하면 -5~10 사이의 랜덤한 값 하나를 반환한다.
- 아래의 메서드 3개를 활용하라
- Math.random() -> 랜덤한 숫자를 반환
 , Math.abs() -> 절대값을 구하는 메서드
 , Math.min() -> 매개변수 2개중 낮은 숫자를 반환
- 매개변수로 받은 숫자까지 포함하자.
- from 의 값이 to보다 클경우도 처리되어야 한다. getRandom(10,-5)

```
for(int i =0 ; i<10;i++)  
    System.out.print(getRandom(-5,10)+",");
```

```
4, -5, -2, 2, 2, 3, -5, 4, 3, 9,
```

```
public static int getRandom(int from, int to) {  
    // Random number generation logic  
    // ...  
}
```



03

제네릭

generic

미국·영국 [dʒəˈnerɪk]  영국식 

형용사

1 포괄적인, 총칭[통칭]의

클래스나 메서드의 코딩시점이 아닌 사용하는 시점으로
타입선언을 미뤄 어떠한 타입이든 다 받을수 있는것

```
List<String> list = new ArrayList<String>();
```

제네릭

```
public class MyArrayList<T> {  
    T[] itemList;  
    public void add(T item)  
    {
```

제네릭 클래스

타입 변수 선언

타입 변수 사용

MyArrayList 클래스를 만드는시점에는 어떤 타입의 객체들을 다룰지 정하지 않는다.

타입 변수명은 T로 정해진건 아니다.

```
public class MyArrayList<E> {  
    E[] itemList;  
    public void add(E item)  
    {  
        /*  
        */  
    }  
}
```

대문자 한글자로 의미있게 짓는게 일반적이다.


```
MyArrayList<String> list = new MyArrayList<String>();  
|  
list.add("테스트입니다.");
```

사용시 정확한 타입을 명시

```
public class MyArrayList {  
    String[] itemList;  
    public void add(String item)  
    {
```

컴파일시점까지도 어떤타입인지 알수 없다. 객체 생성시에 타입이 정해진다.

```

5 public class MyArrayList {
6
7     Object[] itemList;
8
9     public void add(Object item)
10    {
11        /*

```

VS

```

5 public class MyArrayList<E> {
6
7     E[] itemList;
8
9     public void add(E item)
10    {
11        /*

```

그냥 Object 쓰면 안되나?

```
MyArrayList list = new MyArrayList();
```

```
list.add(new Student());
```

```
Student std = (Student)list.get(0);
```

타입 캐스팅을 해줘야 한다.

```
list.add(new Car());
```

→ 의도하지 않은 타입이 사용될수 있다.

```
5 public class MyArrayList<T> {  
6  
7  
8     static T test;  
9  
10    public static void test(T obj)  
11    {  
12  
13    }  
14 }
```

static 멤버에는 타입변수를 사용할수 없다.

타입변수 T는 객체가 생성될때 타입이 정해진다. 그러나 static은 객체생성 이전에 이미 사용되어진다.

```
public class MyArrayList<T> {  
  
    T test = new T();  
}
```

변수의 타입으로 지정은 가능

new 연산자는 컴파일시점에 어떤타입인지
정해져야 한다.
그러나 제네릭은 객체생성 될때 타입이
정해진다.

퀴즈

```
MyArrayList<Integer> list = new MyArrayList<String>();
```

→ 참조변수와 타입이 일치해야한다.

```
MyArrayList<Integer> list = new MyArrayList<>();
```

→ JDK1.7 부터 객체쪽에 타입지정은 생략가능

```
MyArrayList<Car> list = new MyArrayList<>();  
|  
list.add(new OilCar());
```

→ add메서드의 매개변수 타입역시 Car가 되었고 다형성에 의해 가능하다!

제한된 제네릭 타입

```
MyArrayList<Car> list = new MyArrayList<>();  
MyArrayList<People> list2 = new MyArrayList<>();  
list.add(new OilCar());  
list2.add(new People());
```

→ 제네릭으로 만들었기에 사용시 어떠한 객체든 상관없다.

특정 타입으로만 제한을 두고 싶다면?

```
public class MyArrayList<T extends Car> {  
    T[] itemList;  
    public void add(T item)  
    {  
        /*  
        Object[] temp = itemList;  
        */  
    }  
}
```

Car 혹은 자식들만 사용가능

```
MyArrayList<Car> list = new MyArrayList<>();  
MyArrayList<People> list2 = new MyArrayList<>();
```



```
public class MyArrayList<T extends Repairable> {  
  
    T[] itemList;  
  
    public void add(T item)  
    {  
        /*  
        */  
    }  
}
```

해당 인터페이스를 구현한 클래스만 올 수 있다.

```
public class MyArrayList<T extends Car & Repairable> {  
  
    T[] itemList;  
  
    public void add(T item)  
    {  
        /*  
        */  
    }  
}
```

Car를 상속받았고 Repairable 인터페이스를
구현한 클래스만 올 수 있다.

제네릭 메서드

```
public class Test {  
    public void add(List<String> list)  
    {  
    }  
    public void add(List<Integer> list)  
    {  
    }  
}
```



```
public class Test {  
    public void add(List list)  
    {  
    }  
    public void add(List list)  
    {  
    }  
}
```

제네릭의 타입은 오버로딩의 대상이 아니다.
즉 위의 경우는 메서드 중복으로 오류이다.

컴파일시 제네릭 부분은 지워진다.

클래스는 제네릭이 아니다.

```
public class Test {  
    public <T> void add(List<T> list)  
    {  
    }  
}
```

메서드 내에서만 제네릭
타입을 사용하겠다는 것

```
public class Test<T> {  
    public void add(List<T> list)  
    {  
    }  
}
```

물론 클래스 전체에서 제네릭을 쓰게 되면 제네릭클래스로
만들어서 사용해도 된다.

— 04

실습문제

실습문제3

1. 제네릭을 활용하여 어떠한 객체든지 담아 둘 수 있는 Box 클래스를 만들어보자.

getter, setter 를 만들자.

```
Box<String> box = new Box<>();  
Box<Integer> box2 = new Box<>();  
box.setItem("가나다라");  
  
System.out.println(box.getItem());
```

```
<terminated> mai  
가나다라
```

```
public class Box {  
    private int item;  
  
    public void setItem(int item)  
    {  
        this.item = item;  
    }  
  
    public int getItem()  
    {  
        return this.item;  
    }  
}
```



THANK YOU



강사 박주병