

Numbers and Statics: Numbers Matter

Samkeun Kim <skim@hknu.ac.kr>

<http://cyber.hankyong.ac.kr>

MATH 메소드: 거의 전역변수에 가깝다

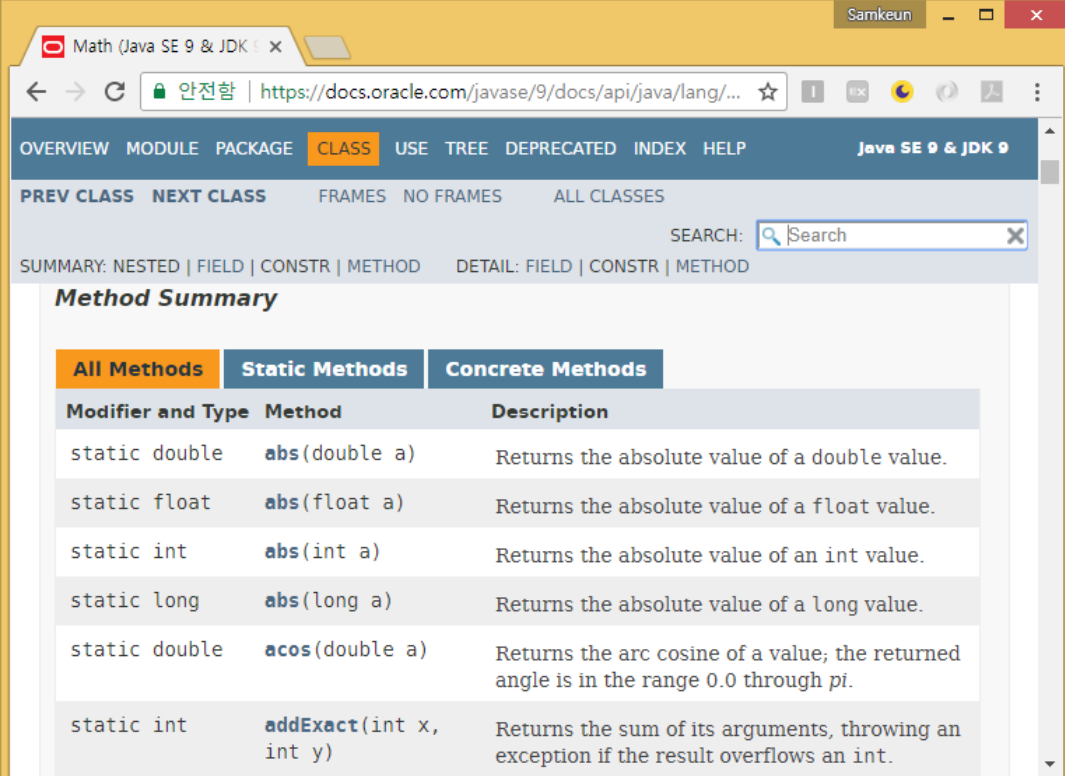
Math 클래스의 메소드는 결코 인스턴스 변수를 사용하지 않는다!

Math 클래스의 메소드들은 '**static**'으로 선언되어 있기 때문에 **Math**의 인스턴스를 가질 필요가 없다.
곧바로 **Math** 클래스를 직접 사용하면 된다.

```
int x = Math.round(42.2);  
int y = Math.min(56,12);  
int z = Math.abs(-343);
```

↑
These methods never use
instance variables, so their
behavior doesn't need to
know about a specific object.

<http://docs.oracle.com/javase/9/docs/api/>



Math (Java SE 9 & JDK 9)

OVERVIEW MODULE PACKAGE **CLASS** USE TREE DEPRECATED INDEX HELP

PREV CLASS NEXT CLASS FRAMES NO FRAMES ALL CLASSES

SEARCH:

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Method Summary

All Methods	Static Methods	Concrete Methods
Modifier and Type	Method	Description
static double	abs (double a)	Returns the absolute value of a double value.
static float	abs (float a)	Returns the absolute value of a float value.
static int	abs (int a)	Returns the absolute value of an int value.
static long	abs (long a)	Returns the absolute value of a long value.
static double	acos (double a)	Returns the arc cosine of a value; the returned angle is in the range 0.0 through pi.
static int	addExact (int x, int y)	Returns the sum of its arguments, throwing an exception if the result overflows an int.

만일 **Math** 클래스의 인스턴스를 만들려고 한다면

Math mathObject = new Math();

아래와 같은 에러가 나올 것이다:

```
File Edit Window Help IwasToldThereWouldBeNoMath
%javac TestMath
TestMath.java:3: Math() has private
access in java.lang.Math
    Math mathObject = new Math();
                        ^
1 error
```

← This error shows that the *Math* constructor is marked private! That means you can NEVER say 'new' on the *Math* class to make a new *Math* object.

regular (non-static) method

```
public class Song {  
    String title;  
    public Song(String t)  
        title = t;  
}  
public void play() {  
    SoundPlayer player = new SoundPlayer();  
    player.playSound(title);  
}
```

인스턴스 변수 값이
play() 메소드의 동작에
영향을 미친다!

Song
title
play()

The current value of the 'title' instance variable is the song that plays when you call play().

two instances of class Song



Song

s2.play();

Calling play() on this reference will cause "Politik" to play.



Song object



Song object



Song

s3.play();

Calling play() on this reference will cause "My Way" to play.

static method

```
public static int min(int a, int b){  
    //returns the lesser of a and b  
}
```

Math
min()
max()
abs()
...

인스턴스 변수가 없다!
메소드의 동작이 인스턴스 변수
상태에 따라 변하지 않는다.

Math.min(42, 36);

Use the Class name, rather than a reference variable name.



NO OBJECTS!!
Absolutely NO OBJECTS
anywhere in this picture!

일반 메소드와 static 메소드 차이

static 메소드는 클래스 이름을 사용하여 호출한다:

Math
min() max() abs() ...

```
Math.min(88, 86);
```

non-static 메소드는 레퍼런스 변수명을 이용하여 호출한다:



```
Song t2 = new Song();
```

```
t2.play();
```

static 메소드에서는 인스턴스 변수를 사용할 수 없다

If you try to compile this code:

```
public class Duck {  
    private int size;  
  
    public static void main (String[] args) {  
        System.out.println("Size of duck is " + size);  
    }  
  
    public void setSize(int s) {  
        size = s;  
    }  
    public int getSize() {  
        return size;  
    }  
}
```

Which Duck?
Whose size?

Static 메소드는 어느
instance 변수를 이용
해야 할지 모른다!!

If there's a Duck on
the heap somewhere, we
don't know about it.

You'll get this error:

```
! Information: java: Errors occurred while compiling module 'JavaClass'  
! Information: javac 1.8.0_161 was used to compile java sources  
! Information: 2018-03-18 오전 11:43 - Compilation completed with 1 error and 0 warnings in 874ms  
v D:\practice\java\JavaClass\src\chap10\Duck.java  
! Error:(7, 49) java: non-static variable size cannot be referenced from a static context
```

Static 메소드에서는 non-static 메소드도 사용할 수 없다

일반적으로 non-static 메소드는 어떻게 수행되는가?

보통 인스턴스 변수 상태를 이용하여 메소드의 동작에 영향을 준다.

This won't compile:

```
public class Duck {  
    private int size;  
  
    public static void main (String[] args) {  
        System.out.println("Size is " + getSize());  
    }  
  
    public void setSize(int s) {  
        size = s;  
    }  
    public int getSize() {  
        return size;  
    }  
}
```

Calling getSize() just postpones the inevitable—getSize() uses the size instance variable.

*Back to the same problem...
whose size?*

```
File Edit Window Help Jack-in  
% javac Duck.java  
Duck.java:6: non-static method  
getSize() cannot be referenced  
from a static context  
  
    System.out.println("Size  
of duck is " + getSize());  
                ^
```

Make it Stick

*Roses are red,
and known to bloom late
Statics can't see
instance variable state*

*Java is
Pass
by value
threads
wait() notify()
Wash
Cat*

Static 변수: 클래스의 모든 인스턴스에 대해 값이 같다

프로그램이 실행되는 동안 **Duck** 인스턴스가 몇 개 만들어지는지 세어보고 싶은 경우를 고려해보자.

다음처럼 인스턴스 변수를 만들고 생성자에서 그 값을 증가시키면 될까?

```
class Duck {  
    int duckCount = 0;  
    public Duck() {  
        duckCount++;  
    }  
}
```

*this would always set
duckCount to 1 each time
a Duck was made*

이렇게 하면 안된다!!

⇒ **duckCount**가 인스턴스 변수이기 때문에 생성자 **Duck**이 호출될 때마다 0으로 초기화되어 버린다.

바로 이런 경우에 **static** 변수를 사용할 수 있다:

static duckCount 변수는 새로운 인스턴스가 만들어질 때마다 초기화되는 것이 아니라 클래스가 맨 처음 로드될 때만 초기화된다.

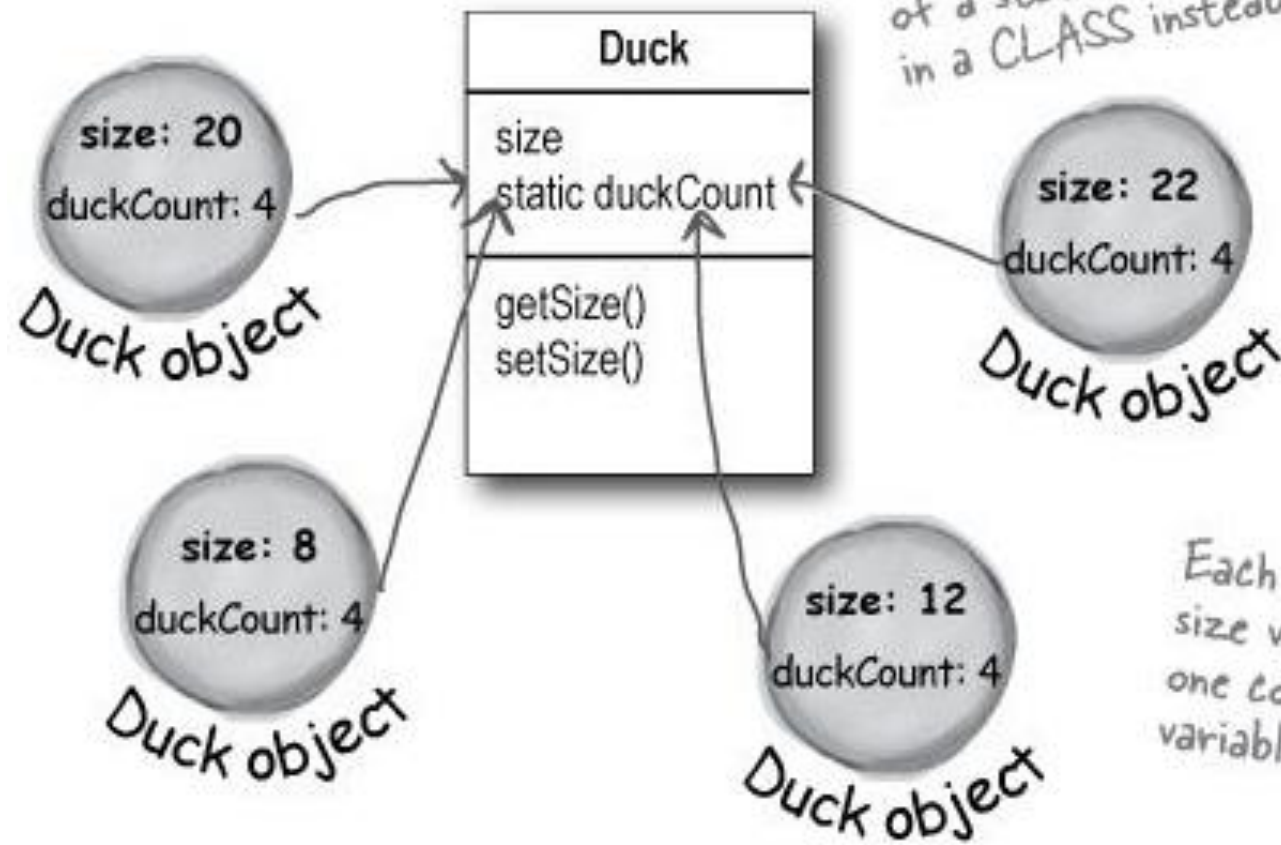
즉, 인스턴스당 하나가 아니라 클래스당 하나의 값을 가진다.

static 변수는 한 클래스의 모든 인스턴스에서 공유한다.

```
public class Duck {  
    private int size;  
    private static int duckCount = 0;  
  
    public Duck() {  
        duckCount++;  
    }  
  
    public void setSize(int s) {  
        size = s;  
    }  
    public int getSize() {  
        return size;  
    }  
}
```

The static duckCount variable is initialized **ONLY** when the class is first loaded, **NOT** each time a new instance is made.

Now it will keep incrementing each time the Duck constructor runs, because duckCount is static and won't be reset to 0.



A Duck object doesn't keep its own copy of `duckCount`.

Because `duckCount` is static, Duck objects all share a single copy of it. You can think of a static variable as a variable that lives in a **CLASS** instead of in an object.

Each Duck object has its own `size` variable, but there's only one copy of the `duckCount` variable—the one in the class.



static 변수 초기화

static 변수는 클래스가 로딩될 때 초기화된다.

누군가 처음으로 **static** 메소드 또는 **static** 변수를 사용하려고 할 때 JVM에서 클래스를 불러온다.

```
class Player {  
    static int playerCount = 0;  
    private String name;  
    public Player(String n) {  
        name = n;  
        playerCount++;  
    }  
}
```



static 변수는 그 클래스에 속하는 객체가 생성되기 전에 초기화된다.
(**static** 변수는 그 클래스에 속하는 **static** 메소드가 실행되기 전에 초기화된다.)

static 변수를 직접 초기화하지 않으면 인스턴스 변수의 경우와 마찬가지로 타입에 해당하는 디폴트 값으로 초기화된다:

Primitive: **0**

Floating-point : **0.0**

Boolean: **false**

Reference: **null**

```
public class PlayerTestDrive {  
    public static void main(String[] args) {  
        System.out.println(Player.playerCount);  
        Player one = new Player("Tiger Woods");  
        System.out.println(Player.playerCount);  
    }  
}
```

Access a static variable just like a static method—with the class name.

```
Run: PlayerTestDrive x  
C:\Program Files\Java\jdk1.8.0_181\bin  
0  
1  
Process finished with exit code 0
```

static final로 선언된 변수는 상수다

final로 선언한 변수는 일단 초기화되면 결코 바뀔 수 없다.

즉, **static final** 변수의 값은 클래스가 로드되어 있는 한 동일한 값으로 유지된다.

```
public static final double PI = 3.141592653589793;
```

- ✓ 변수가 **public**으로 선언되어 있으므로 누구나 접근할 수 있다.
- ✓ 변수가 **static**으로 선언되어 있으므로 인스턴스를 만들 필요가 없다.
- ✓ 변수가 **final**로 선언되어 있으므로 **PI** 값은 결코 바뀌지 않는다.

변수를 상수로 지정하는 방법이 별도로 마련되어 있지 않다.

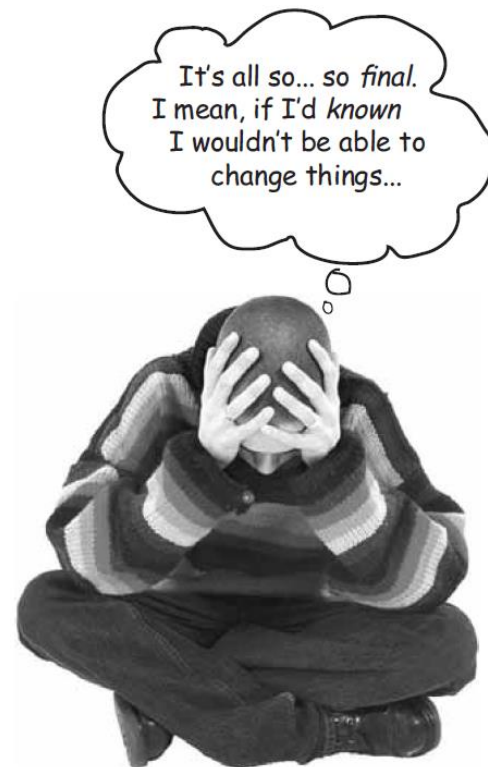
변수를 **static final**로 선언하는 것이 상수로 지정하는 유일한 방법이다.

“상수명을 붙이는 관례는 모두 대문자로 해준다는 것이다!”

final은 static 변수를 위한 것만이 아니다

final 키워드는 인스턴스 변수나 지역 변수, 메소드 인자에까지 적용시킬 수 있다.

- ✓ A **final variable** means you can't change its value.
- ✓ A **final method** means you can't override the method.
- ✓ A **final class** means you can't extend the class (i.e. **you can't make a subclass**).



non-static final variables

```
class Foof {  
    final int size = 3; ← now you can't change size  
    final int whuffie;  
  
    Foof() {  
        whuffie = 42; ← now you can't change whuffie  
    }  
  
    void doStuff(final int x) {  
        // you can't change x  
    }  
  
    void doMore() {  
        final int z = 7;  
        // you can't change z  
    }  
}
```

final method

```
class Poof {  
    final void calcWhuffie() {  
        // important things  
        // that must never be overridden  
    }  
}
```

final class

```
final class MyMostPerfectClass {  
    // cannot be extended  
}
```


실습과제 10-1 What's Legal?

```
① public class Foo {  
    static int x;  
  
    public void go() {  
        System.out.println(x);  
    }  
}
```

```
② public class Foo2 {  
    int x;  
  
    public static void go() {  
        System.out.println(x);  
    }  
}
```

```
③ public class Foo3 {  
    final int x;  
  
    public void go() {  
        System.out.println(x);  
    }  
}
```

```
④ public class Foo4 {  
    static final int x = 12;  
  
    public void go() {  
        System.out.println(x);  
    }  
}
```

```
⑤ public class Foo5 {  
    static final int x = 12;  
  
    public void go(final int x) {  
        System.out.println(x);  
    }  
}
```

```
⑥ public class Foo6 {  
    int x = 12;  
  
    public static void go(final int x) {  
        System.out.println(x);  
    }  
}
```

Math 메소드들

Math.random() returns a double between 0.0 through (but not including) 1.0.

```
double r1 = Math.random();  
int r2 = (int) (Math.random() * 5);
```

Math.abs() returns a double that is the absolute value of the argument. The method is overloaded, so if you pass it an int it returns an int. Pass it a double it returns a double.

```
int x = Math.abs(-240); // returns 240  
double d = Math.abs(240.45); // returns 240.45
```

Math.round() returns an int or a long (depending on whether the argument is a float or a double) rounded to the nearest integer value.

```
int x = Math.round(-24.8f); // returns -25  
int y = Math.round(24.45f); // returns 24
```

Math.min() returns a value that is the minimum of the two arguments. The method is overloaded to take ints, longs, floats, or doubles.

```
int x = Math.min(24,240); // returns 24  
double y = Math.min(90876.5, 90876.49); // returns 90876.49
```

Math.max() returns a value that is the maximum of the two arguments. The method is overloaded to take ints, longs, floats, or doubles.int

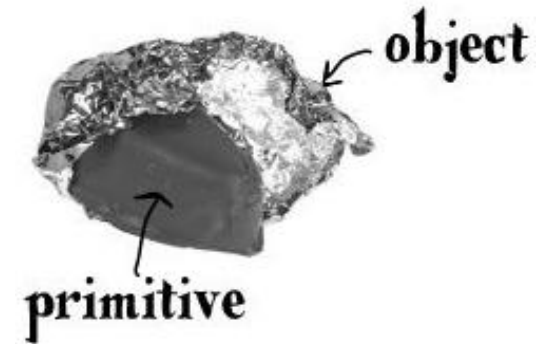
```
int x = Math.max(24,240); // returns 240  
double y = Math.max(90876.5, 90876.49); // returns 90876.5
```

원시 타입 포장하는 방법

종종 원시 타입을 객체처럼 다루어야 할 때가 있다.

Java 5.0 이전 버전에서는 원시 타입을 **ArrayList** 나 **HashMap** 같은 **Collection**에 직접 넣을 수 없었다:

```
int x = 32;  
ArrayList list = new ArrayList();  
list.add(x);
```



모든 원시 타입에 해당하는 **Wrapper** 클래스가 있다.

- ⇒ **Wrapper** 클래스는 **java.lang** 패키지에 들어있기 때문에 구태여 **import** 시킬 필요가 없다.
- ⇒ 각 **Wrapper** 클래스의 이름은 원시타입을 본 따기 때문에 바로 알아 볼 수 있다.

그러나 **API** 설계자가 원시 타입과 클래스 타입의 이름을 완전히 일치시키지는 않았다.

Boolean

Character

Byte

Short

Integer

Long

Float

Double

Watch out! The names aren't mapped exactly to the primitive types. The class names are fully spelled out.

wrapping a value

```
int i = 288;  
Integer iWrap = new Integer(i);
```

Give the primitive to the wrapper constructor. That's it.

unwrapping a value

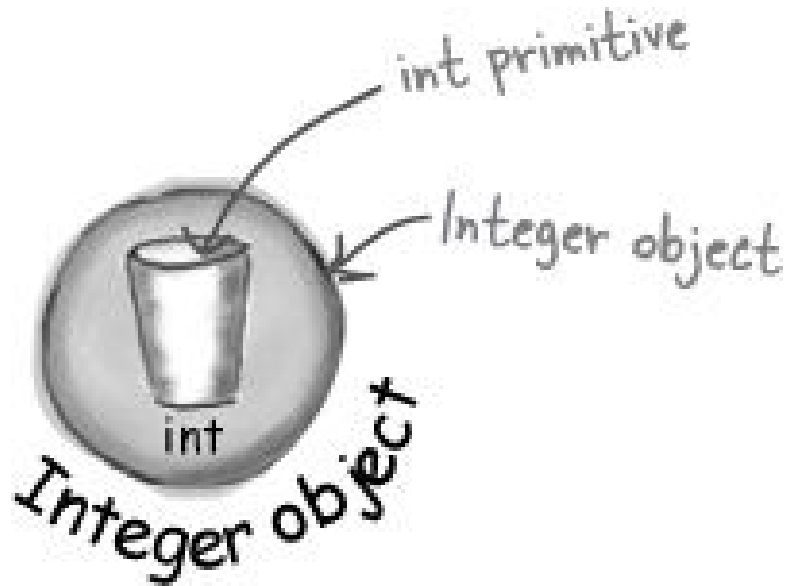
```
int unWrapped = iWrap.intValue();
```

All the wrappers work like this. Boolean has a `booleanValue()`, Character has a `charValue()`, etc.

NOTE

원시 변수를 객체처럼 취급해야 할 때, 그것을 래핑하라!

5.0 이전 버전의 Java를 사용하고 있다면 **ArrayList**나 **HashMap** 처럼 컬렉션에 원시 값을 저장해야 할 때 이 작업을 수행하면 된다.



Before Java 5.0, YOU had to do the work...



오토박싱: 원시타입과 객체 사이의 구분을 흐릿하게

자바 5.0 이전 버전에서는 원시 변수와 객체 레퍼런스가 엄격하게 구분되어 있어서 절대로 서로 맞바꿔서 사용할 수 없었다.

원시 타입 **int**로 구성된 **ArrayList**

Without autoboxing (Java versions before 5.0)

```
public void doNumsOldWay() {
```

```
    ArrayList listOfNumbers = new ArrayList();
```

```
    listOfNumbers.add(new Integer(3));
```

```
    Integer one = (Integer) listOfNumbers.get(0);
```

```
    int intOne = one.intValue();
```

```
}
```

Make an ArrayList. (Remember, before 5.0 you could not specify the TYPE, so all ArrayLists were lists of Objects.)

You can't add the primitive '3' to the list, so you have to wrap it in an Integer first.

It comes out as type Object, but you can cast the Object to an Integer.

Finally you can get the primitive out of the Integer.

자바 5.0부터 도입된 오토박싱

⇒ 원시값과 래퍼 객체 사이의 변환을 자동으로 처리해준다!

원시 타입 `int`로 구성된 `ArrayList`

With autoboxing (Java versions 5.0 or greater)

```
public void doNumsNewWay() {
```

```
    ArrayList<Integer> listOfNumbers = new ArrayList<Integer>();
```

```
    listOfNumbers.add(3); Just add it!
```

```
    int num = listOfNumbers.get(0);
```

```
}
```

And the compiler automatically unwraps (unboxes) the Integer object so you can assign the int value directly to a primitive without having to call the `intValue()` method on the Integer object.

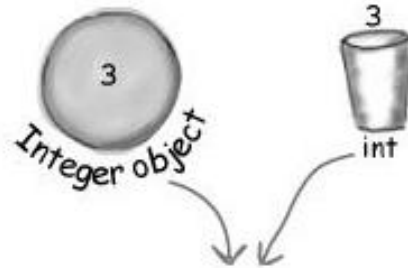
Make an ArrayList of type Integer.



Although there is NOT a method in `ArrayList` for `add(int)`, the compiler does all the wrapping (boxing) for you. In other words, there really IS an Integer object stored in the `ArrayList`, but you get to "pretend" that the `ArrayList` takes ints. (You can add both ints and Integers to an `ArrayList<Integer>`.)

오토박싱은 거의 모든 곳에서 작동한다

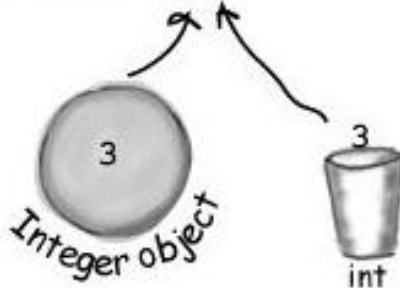
Method arguments



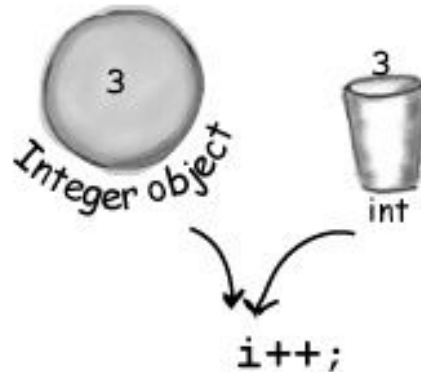
```
void takeNumber(Integer i) { }
```

Return values

```
int giveNumber() {  
    return x;  
}
```



Operations on numbers



```
Integer i = new Integer( 42); i++;  
Integer j = new Integer( 5);  
Integer k = j + 3;
```

Boolean expressions



```
if (bool) {  
    System.out.println("true");  
}
```

Assignments



```
Double d = x;
```

실습과제 10-2

Will this code compile? Will it run? If it runs, what will it do? Take your time and think about this one; it brings up an implication of autoboxing that we didn't talk about. You'll have to go to your compiler to find the answers.

```
public class TestBox {  
  
    Integer i;  
    int j;  
  
    public static void main (String[] args) {  
        TestBox t = new TestBox();  
        t.go();  
    }  
  
    public void go() {  
        j=i;  
        System.out.println(j);  
        System.out.println(i);  
    }  
}
```

잠깐! 래퍼는 static 유틸리티 메소드도 가지고 있다

String을 원시값으로 바꾸는 방법은 간단하다:

```
String s = "2";  
int x = Integer.parseInt(s);  
double d = Double.parseDouble("420.24");  
  
boolean b = Boolean.parseBoolean("True");
```

No problem to parse
"2" into 2.



The (new to 1.5) parseBoolean() method ignores
the cases of the characters in the String
argument.



그러나 아래처럼 하려고 하면:

```
String t = "two";  
int y = Integer.parseInt(t) ;
```

↙ Uh-oh. This compiles just fine, but at runtime it blows up. Anything that can't be parsed as a number will cause a `NumberFormatException`

컴파일은 되겠지만 런타임 때 아래와 같은 예외가 발생할 것이다:

```
File Edit Window Help Clue  
% java Wrappers  
Exception in thread "main"  
java.lang.NumberFormatException: two  
at java.lang.Integer.parseInt(Integer.java:409)  
at java.lang.Integer.parseInt(Integer.java:458)  
at Wrappers.main(Wrappers.java:9)
```

역으로 원시 숫자를 String으로 변환

가장 간단한 방법은 기존 **String**에 숫자를 병합시키는 것이다.


```
double d = 42.5;  
String doubleString = "" + d;
```

Remember the '+' operator is overloaded in Java (the only overloaded operator) as a String concatenator. Anything added to a String becomes Stringified.


```
double d = 42.5;  
String doubleString = Double.toString(d);
```

또 다른 방법

Another way to do it using a static method in class Double.



Yeah,
but how do I make it
look like money? With a dollar
sign and two decimal places
like \$56.87 or what if I want
commas like 45,687,890 or
what if I want it in...



Where's my printf
like I have in C? Is
number formatting part of
the I/O classes?

숫자 포매팅

자바 팀은 자바 5.0에서 **java.util**의 `Formatter` 클래스를 통하여 더 강력하고 유연한 포매팅 기능을 추가시켰다.

`String` 클래스의 `format()` 메소드: **`String.format()`**

`PrintWriter` 클래스와 `PrintStream` 클래스의 `printf()` 메소드: **`System.out.printf()`**

콤마를 사용한 숫자 포매팅

```
public class TestFormats {  
    public static void main (String[] args) {  
        String s = String.format("%, d", 1000000000);  
        System.out.println(s);  
    }  
}
```

The number to format (we want it to have commas).

The formatting instructions for how to format the second argument (which in this case is an int value). Remember, there are only two arguments to this method here—the first comma is *INSIDE* the String literal, so it isn't separating arguments to the format method.

1,000,000,000

Now we get commas inserted into the number.

포매팅 분해

- ① **포매팅 지시자** (formatting instructions)
인자가 어떻게 포맷되어야 하는지를 기술해주는 특수한 포맷 지시자
- ② **포맷될 인자** (argument to be formatted)
포맷 지시자를 사용하여 포매팅 할 수 있는 것이어야 된다.

Do this... to this.

① ②

`format("%, d", 10000000000);`

Use these instructions... on this argument.

Percent (%) says, "insert argument here"

Characters to include in the final String returned from format().
Format specifiers for the second argument to the method (the number).
More characters to include in the String after the second argument is formatted and inserted.
Argument to be formatted.

```
format("I have %.2f bugs to fix.", 476578.09876);
```

Output

```
I have 476578.10 bugs to fix.
```

Adding a comma

```
format("I have %, .2f bugs to fix.", 476578.09876);
```

```
I have 476,578.10 bugs to fix.
```

By changing the format instructions from "%.2f" to "%, .2f", we got a comma in the formatted number.

포맷 String은 자신만의 간단한 구문을 사용한다

%, d "콤마를 삽입하고 숫자를 10진 정수로 포맷하라.

%.2f "소수점 이하 두 자리의 정밀도를 갖는 부동 소수점으로 숫자의 포맷을 지정하라."

%,.2f "콤마를 삽입하고 숫자를 소수점 두 자리의 정밀도의 부동 소수점 포맷으로 지정하라."

포맷 지시자

포맷 지시자는 ("% 기호를 제외하고) 5개
까지 서로 다른 부분으로 구성할 수 있다.

[] 안의 순서는 정확하게 지켜져야 한다.

`%[argument number][flags][width][.precision]type`

We'll get to this later...
it lets you say WHICH
argument if there's more
than one. (Don't worry
about it just yet.)

These are for
special formatting
options like inserting
commas, or putting
negative numbers in
parentheses, or to
make the numbers
left justified.

This defines the
MINIMUM number
of characters that
will be used. That's
minimum not
TOTAL. If the number
is longer than the
width, it'll still be used
in full, but if it's less
than the width, it'll be
padded with zeroes.

You already know
this one...it defines
the precision. In
other words, it
sets the number
of decimal places.
Don't forget to
include the "." in
there.

Type is mandatory
(see the next page)
and will usually be
"d" for a decimal
integer or "f" for
a floating point
number.

`%[argument number][flags][width][.precision]type`

`format("%,6.1f", 42.000);`

There's no "argument number"
specified in this format String,
but all the other pieces are there.

The only required specifier is for TYPE

Type은 필수, 나머지는 선택!

%d

decimal

```
format("%d", 42);
```

42

A 42.25 would not work! It would be the same as trying to directly assign a double to an int variable.

%f

floating point

```
format("%.3f", 42.000000);
```

42.000

Here we combined the "f" with a precision indicator ".3" so we ended up with three zeroes.

%x

hexadecimal

```
format("%x", 42);
```

2a

%c

character

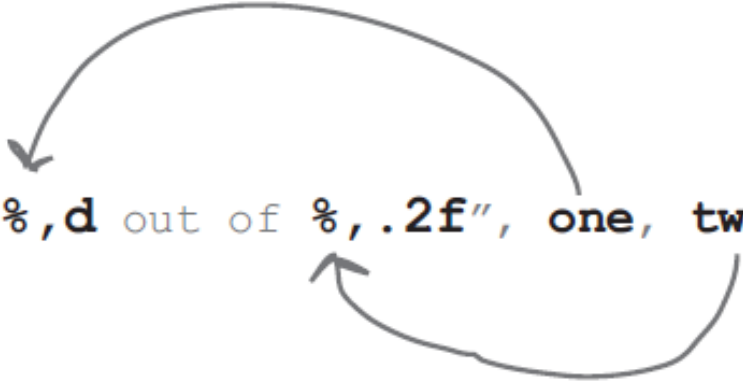
```
format("%c", 42);
```

*

The number 42 represents the char "*".

What happens if I have more than one argument?

```
int one = 20456654;  
double two = 100567890.248907;  
String s = String.format("The rank is %,d out of %, .2f", one, two);
```

A diagram with two curved arrows. The first arrow starts at the variable 'one' in the format string and points to the first format specifier '%,d'. The second arrow starts at the variable 'two' and points to the second format specifier '%, .2f'.

The rank is 20,456,654 out of 100,567,890.25

We added commas to both variables, and restricted the floating point number (the second variable) to two decimal places.

When you have more than one argument, they're inserted using the order in which you pass them to the `format()` method.

So much for numbers, what about dates?

날짜 포맷은 **t**로 시작하는 두 글자 타입을 사용한다.

The complete date and time **%tc**

```
String.format("%tc", new Date());
```

Sun Nov 28 14:52:41 MST 2004

Just the time **%tr**

```
String.format("%tr", new Date());
```

03:01:47 PM

Day of the week, month and day

%tA %tB %td

```
Date today = new Date();  
String.format("%tA, %tB %td", today, today, today)
```

↗
The comma is not part of the formatting... it's just the character we want printed after the first inserted formatted argument.

Sunday, November 28

딱히 한번에 이걸 표현할 수 있는 포맷 지시자는 없으므로 요일(%tA), 월(%tB), 일(%td)을 결합해야 한다.

Same as above, but without duplicating the arguments

%tA %tB %td

```
Date today = new Date();  
String.format("%tA, %<tB %<td", today);
```

You can think of this as kind of like calling three different getter methods on the Date object, to get three different pieces of data from it.

“<“은 단지 Formatter에게 “이전 인자를 다시 사용하라.”라고 말해주는 지정자일 뿐이다.

Working with Dates

시간을 앞 뒤로 옮겨 다니기

`java.util.Date`는 좀 구식이다...

날짜 조작을 위해서는 `java.util.Calendar`를 사용하자!

Calendar를 확장하는 객체를 얻어오는 방법

다음은 동작하지 않을 것이다:

```
Calendar cal = new Calendar();
```

The compiler won't allow this!

대신에 **static** 메소드인 "**getInstance()**" 를 사용해야 한다:

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

This syntax should look familiar at this point - we're invoking a static method.

Calendar 객체 사용법

An example of working with a Calendar object:

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

```
c.set(2004, 0, 7, 15, 40);
```

Set time to Jan. 7, 2004 at 15:40.
(Notice the month is zero-based.)

```
long day1 = c.getTimeInMillis();
```

Convert this to a big ol'
amount of milliseconds.

```
day1 += 1000 * 60 * 60;
```

```
c.setTimeInMillis(day1);
```

Add an hour's worth of millis, then update the time.
(Notice the "+=", it's like day1 = day1 + ...).

```
System.out.println("new hour " + c.get(c.HOUR_OF_DAY));
```

```
c.add(c.DATE, 35);
```

Add 35 days to the date, which
should move us into February.

```
System.out.println("add 35 days " + c.getTime());
```

```
c.roll(c.DATE, 35);
```

"Roll" 35 days onto this date. This
"rolls" the date ahead 35 days, but
DOES NOT change the month!

```
System.out.println("roll 35 days " + c.getTime());
```

```
c.set(c.DATE, 1);
```

We're not incrementing here, just
doing a "set" of the date.

```
System.out.println("set to 1 " + c.getTime());
```

```
File Edit Window Help Time-Files
new hour 16
add 35 days Wed Feb 11 16:40:41 MST 2004
roll 35 days Tue Feb 17 16:40:41 MST 2004
set to 1 Sun Feb 01 16:40:41 MST 2004
```

This output confirms how millis,
add, roll, and set work.

Highlights of the Calendar API

Key Calendar Methods

add(int field, int amount)

Adds or subtracts time from the calendar's field.

get(int field)

Returns the value of the given calendar field.

getInstance()

Returns a Calendar, you can specify a locale.

getTimeInMillis()

Returns this Calendar's time in millis, as a long.

roll(int field, boolean up)

Adds or subtracts time without changing larger fields.

set(int field, int value)

Sets the value of a given Calendar field.

set(year, month, day, hour, minute) (all ints)

A common variety of set to set a complete time.

setTimeInMillis(long millis)

Sets a Calendar's time based on a long milli-time.

// more...

Key Calendar Fields

DATE / DAY_OF_MONTH

Get / set the day of month

HOUR / HOUR_OF_DAY

Get / set the 12 hour or 24 hour value.

MILLISECOND

Get / set the milliseconds.

MINUTE

Get / set the minute.

MONTH

Get / set the month.

YEAR

Get / set the year.

ZONE_OFFSET

Get / set raw offset of GMT in millis.

// more...

Even more Statics!... static imports

Some old-fashioned code:

```
import java.lang.Math;

class NoStatic {

    public static void main(String [] args) {

        System.out.println("sqrt " + Math.sqrt(2.0));
        System.out.println("tan " + Math.tan(60));

    }

}
```

Same code, with static imports:

```
import static java.lang.System.out;
```

```
import static java.lang.Math.*;
```

```
class WithStatic {  
    public static void main(String [] args) {  
        out.println("sqrt " + sqrt(2.0));  
        out.println("tan " + tan(60));  
    }  
}
```

Static imports in action.



실습과제 10-3 BE the compiler

```
class StaticSuper{

    static {
        System.out.println("super static block");
    }

    StaticSuper{
        System.out.println(
            "super constructor");
    }
}

public class StaticTests extends StaticSuper {
    static int rand;

    static {
        rand = (int) (Math.random() * 6);
        System.out.println("static block " + rand);
    }

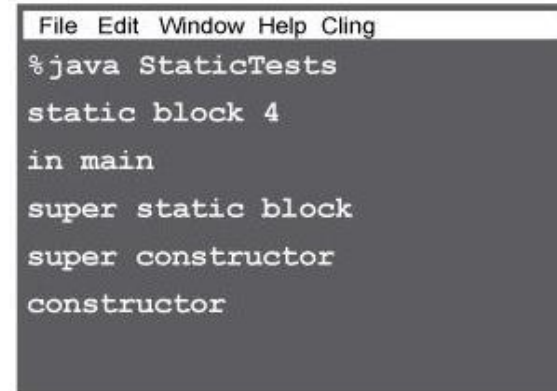
    StaticTests() {
        System.out.println("constructor");
    }

    public static void main(String [] args) {
        System.out.println("in main");
        StaticTests st = new StaticTests();
    }
}
```

Your job is to play compiler and determine whether this file will compile. If it won't compile, how would you fix it, and if it does compile, what would be its output?

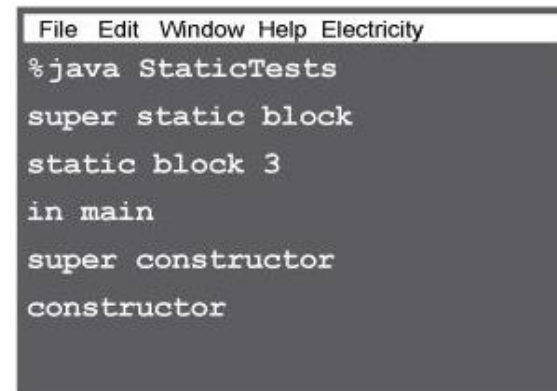
가능한 후보들:

A



```
File Edit Window Help Cling
%java StaticTests
static block 4
in main
super static block
super constructor
constructor
```

B



```
File Edit Window Help Electricity
%java StaticTests
super static block
static block 3
in main
super constructor
constructor
```

실습과제 10-4 Lunar Code Magnets

This one might actually be useful! In addition to what you've learned in the last few pages about manipulating dates, you'll need a little more information... First, full moons happen every 29.52 days or so. Second, there was a full moon on Jan. 7th, 2004. Your job is to reconstruct the code snippets to make a working Java program that produces the output listed below (plus more full moon dates). (You might not need all of the magnets, and add all the curly braces you need.) Oh, by the way, your output will be different if you don't live in the mountain time zone.

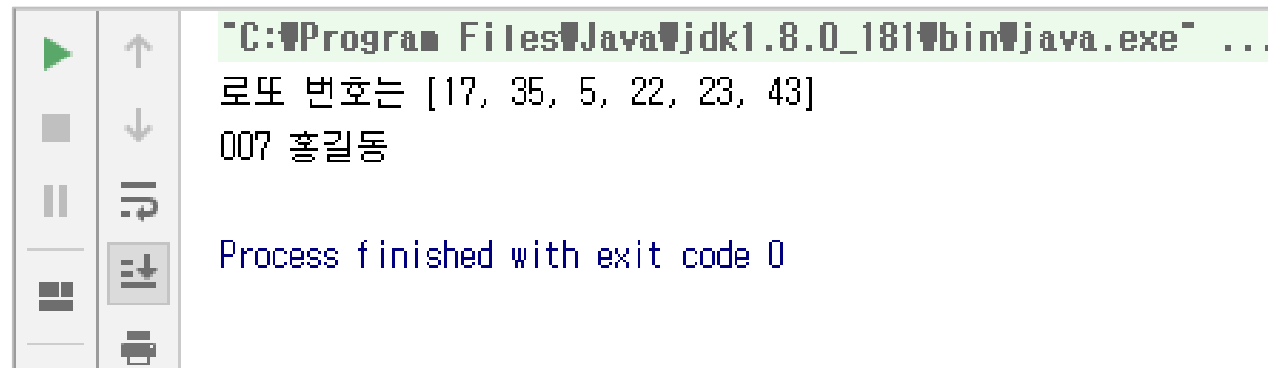
```
File Edit Window Help Howl
$ java FullMoons
full moon on Fri Feb 06 04:09:35 MST 2004
full moon on Sat Mar 06 16:38:23 MST 2004
full moon on Mon Apr 05 06:07:11 MDT 2004
```

```
long day1 = c.getTimeInMillis();
c.set(2004,1,7,15,40);
import static java.lang.System.out;
static int DAY_IM = 60 * 60 * 24;
("full moon on %tc", c));
(c.format
Calendar c = new Calendar();
class FullMoons {
public static void main(String [] args) {
day1 += (DAY_IM * 29.52);
for (int x = 0; x < 60; x++) {
static int DAY_IM = 1000 * 60 * 60 * 24;
println
import java.io.*;
import java.util.*;
static import java.lang.System.out;
c.set(2004,0,7,15,40);
out.println
c.setTimeInMillis(day1);
(String.format
Calendar c = Calendar.getInstance();
```

실습과제 10-5 로또 번호 생성기 구현

로또 번호를 생성하는 프로그램을 작성하여 보자. 로또는 1부터 45까지의 숫자 중에서 6개를 선택한다. 로또 번호는 중복되면 안된다. 따라서 집합을 나타내는 **HashSet**을 사용하여 중복검사를 해야 한다. **Math.random()**을 사용하면 0부터 1사이의 난수를 생성할 수 있다. 0부터 1사이의 난수가 생성되면 여기에 44를 곱하고 1을 더하면 1부터 45사이의 정수를 생성할 수 있다. 생성된 정수는 **HashSet**의 **contains()** 메소드를 이용해서 이미 선택된 정수인지를 검사한다.

참고: <http://docs.oracle.com/javase/8/docs/api/>



```
"C:\Program Files\Java\jdk1.8.0_181\bin\java.exe" ...  
로또 번호는 [17, 35, 5, 22, 23, 43]  
007 홍길동  
  
Process finished with exit code 0
```

