10. 숫자와 정적 변수, 정적 메서드

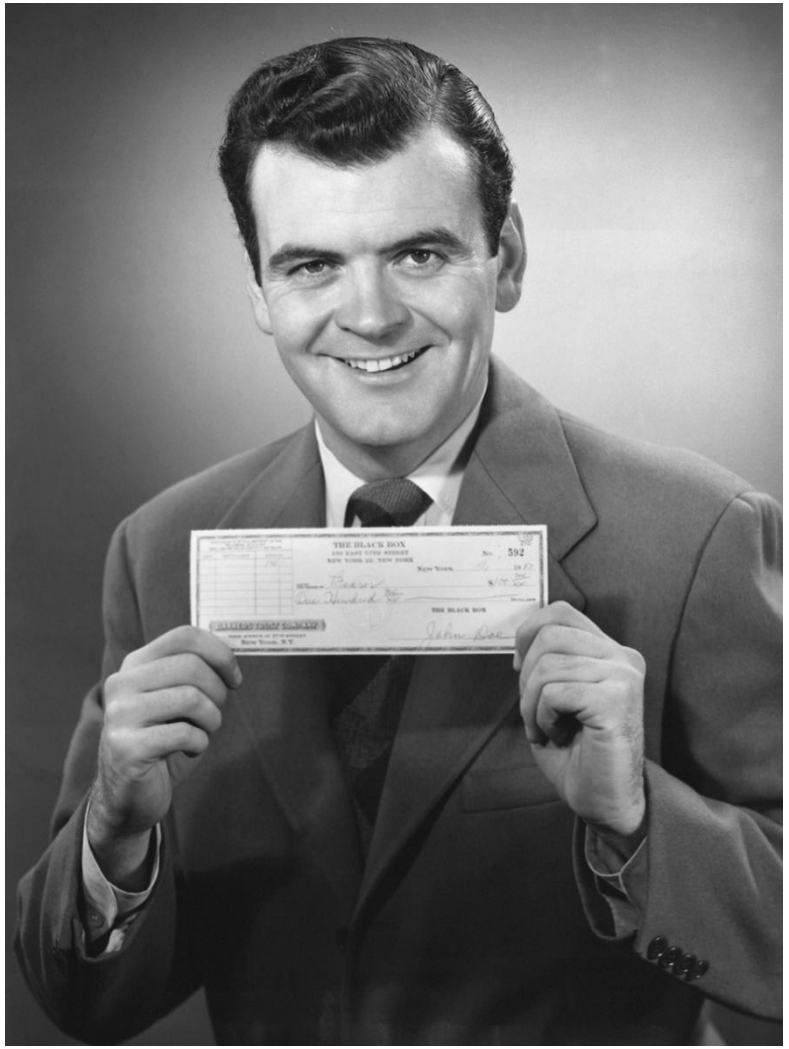
정적 메서드와 정적 변수

상수

래퍼 클래스

포매팅

숫자는 정말 중요합니다



- Math 클래스
- 출력 방식 조절
- String을 수로 변환하는 방법
- 수를 String으로 변환하는 방법
- 정적 메서드
- 정적 변수
- 상수 (static final)

p.317

java.lang.Math 클래스

- '거의' 정적 메서드(static method)로 구성
- 인스턴스 변수를 전혀 사용하지 않음
- 인스턴스를 만들 수 없음

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

Math mathObject = new Math();

```
$ javac TestMath.java
TestMath.java:3:Math() has private access in java.lang.Math
  Math mathObject = new Math();
                    ^
1 error
```

All Methods | Static Methods | Concrete Methods

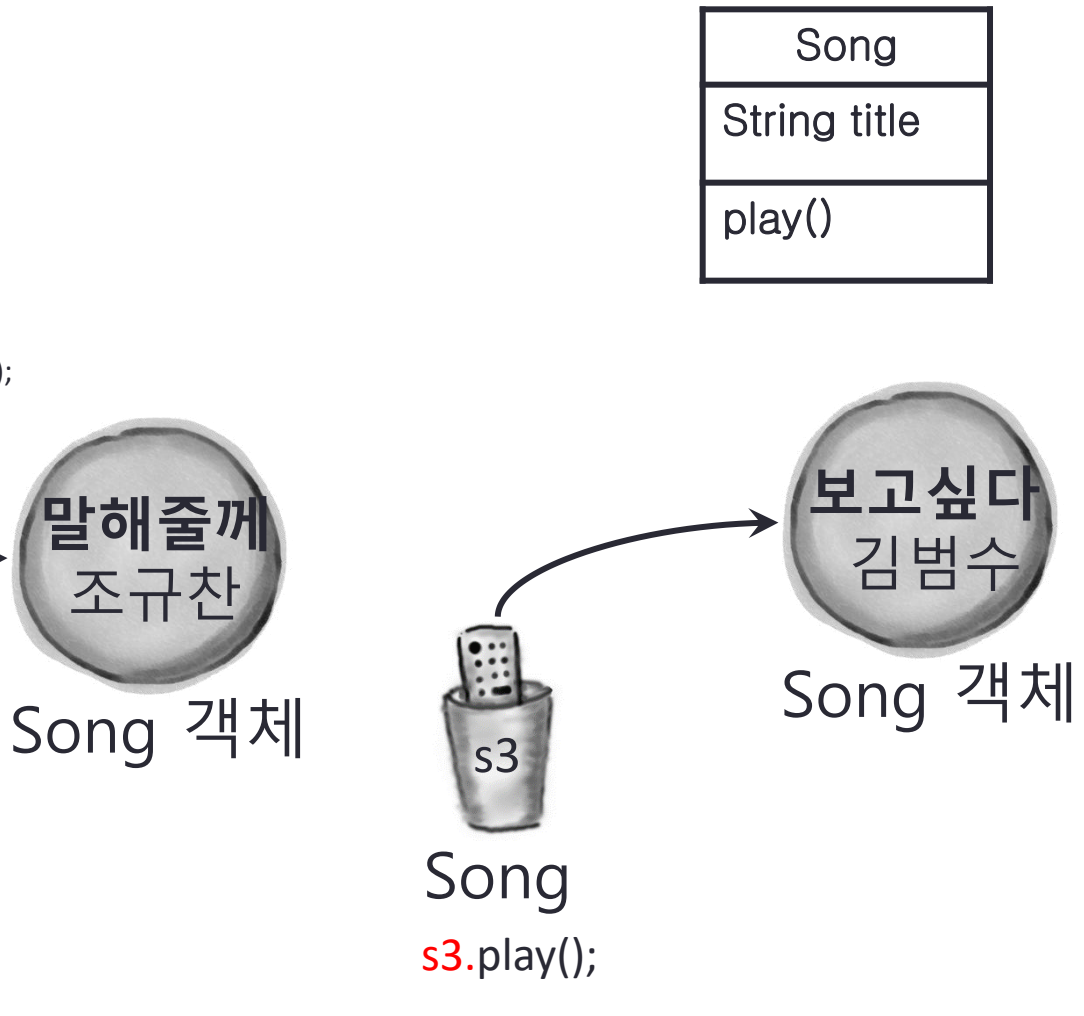
Modifier and Type	Method and Description
static double	**abs**(double a) Returns the absolute value of a double value.
static float	**abs**(float a) Returns the absolute value of a float value.
static int	**abs**(int a) Returns the absolute value of an int value.
static long	**abs**(long a) Returns the absolute value of a long value.
static double	**acos**(double a) Returns the arc cosine of a value; the returne

p.318

일반 메서드와 정적 메서드

일반 메서드

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



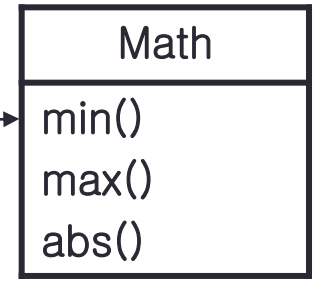
p.319

일반 메서드와 정적 메서드

* 클래스의 인스턴스 없이 메서드 실행 가능
* 인스턴스 변수에 따라 행동이 의존하지 않으므로 인스턴스/객체가 필요없음.

정적 메서드 (static method)

```
public static int min(int a, int b) {
  // a와 b 중에서 더 작은 것을 리턴
}
```




```
Math.min(42, 36);
```

정적 메서드를 호출할 때는 **클래스명**을,

정적 메서드가 아닌 메서드를 호출할 때는 **레퍼런스 변수명**을 사용

객체가 없다!!

인스턴스를 만들 수 없는 메서드

- **추상 클래스**
- **생성자를 private으로 지정한 클래스**

- 정적 메서드만 들어있는 클래스는 생성자를 private으로 지정하여 인스턴스를 만들 수 없도록 하는 것이 좋습니다.
- 정적 메서드가 있다고 해서 무조건 인스턴스를 만들 수 없도록 하진 않습니다.

p.320

정적 메서드와 인스턴스 변수***

- **정적 메서드에서는** 인스턴스 변수 사용 불가, **정적 변수만 사용 가능**

```
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;
  }
}
```

```
$ javac Duck.java
Duck.java:4: non-static variable size cannot be referenced from
a static context
    System.out.println("Size of duck is " + size);
                                          ^
```

p.321

정적 메서드에서의 메서드 호출***

- **정적 메서드에서는 정적 메서드만 호출** 가능

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

```
$ javac Duck.java
Duck.java:4: non-static method getSize() cannot be referenced
from a static context
    System.out.println("Size of duck is " + getSize());
  ^$
```

p.322

바보 같은 질문은 없습니다.

- 클래스명이 아니라 레퍼런스 변수명을 써서 정적 메서드를 호출하는 건 어떻게 된 건가요?
  - 그렇게 하는 것도 가능합니다. 하지만 그런다고 해서 그 메서드가 인스턴스 메서드인 것은 아닙니다.

```
Duck d = new Duck();
String[] s = {};
d.main(s);  // 실행 결과는?
```

정적 메서드에서는 인스턴스 변수 상태를 볼 수 없습니다.

p.322

(참고) SillyQuestion.java

```
6   package cse.oop2.ch10;
7
8   /**...4 lines */
12  public class SillyQuestion {
13
14      /**...3 lines */
17      public static void main(String[] args) {
18          test();
19
20          SillyQuestion.test();
21
22          SillyQuestion q = new SillyQuestion();
            q.test();
24
25      }
26
27      public static void test() {
            Duck d = new Duck();
29      }
30
31  }
32
33  class Duck {
34      public  Duck() {
35          System.out.println("Duck 객체 생성!");
36      }
37  }
```

```
--- exec-maven-plu
Duck 객체 생성!
Duck 객체 생성!
Duck 객체 생성!
```

정적 변수

- 정적 변수 (static variable)
  - 어떤 인스턴스에서도 값이 똑같은 변수

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

```
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;
  }
}
```

p.323

실습: Duck.java

```java
package cse.oop2.ch10.practice;

/**
 * Head First Java 10장. p.311
 * Author: Prof. Jong Min Lee
 */
public class Duck {
    private int size;
    private static int duckCount = 0;
    private int nth;

    private static Duck[] ducks; // 정적 변수

    public Duck() {
        this(10);
    }

    public Duck(int size) {
        this.size = size;
        nth = duckCount++;
    }
```

```
23      @Override
24      public String toString() {
25          StringBuilder sb = new StringBuilder();
26          sb.append("나는 ");
27          sb.append(nth);
28          sb.append("번째 오리이고, ");
29          sb.append("크기는 ").append(size).append("입니다.");
30
31          return sb.toString();
32      }
33
34      public static void initialize() {  // 정적 메소드
35          ducks = new Duck[10];
36
37          ducks[0] = new Duck();
38
39          for (int i=1; i<ducks.length; i++) {
40              int size = (int)(Math.random()*20) + 10;  // [10,29] 구간의 정수
41              ducks[i] = new Duck(size);
42          }
43      }
```

```
45      public static void main(String[] args) {
46          initialize();
47
48          for (Duck d : ducks) {
49              System.out.println(d);
50          }
51
52          // System.out.println(size);
53          // this.toString();
54      }
55
56  }
```

```
--- exec-maven-plugin:1.5.0:exec (defa
나는 0번째 오리이고, 크기는 10입니다.
나는 1번째 오리이고, 크기는 13입니다.
나는 2번째 오리이고, 크기는 28입니다.
나는 3번째 오리이고, 크기는 22입니다.
나는 4번째 오리이고, 크기는 11입니다.
나는 5번째 오리이고, 크기는 20입니다.
나는 6번째 오리이고, 크기는 19입니다.
나는 7번째 오리이고, 크기는 18입니다.
나는 8번째 오리이고, 크기는 10입니다.
나는 9번째 오리이고, 크기는 27입니다.
```

정적 변수

- 정적 변수는 공유됨
- 같은 클래스에 속하는 모든 인스턴스에서 정적 변수의 하나뿐인 복사본을 공유함

- **인스턴스 변수**
  - 인스턴스마다 하나씩
- **정적 변수**
  - 클래스마다 하나씩
  - 클래스의 인스턴스에서 모두 볼 수 있음

p.324

정적 변수 초기화

- 초기화 시기
  - 클래스가 로딩될 때
- 두 가지 규칙
  - 객체가 생성되기 전에 초기화됨
  - 정적 메서드가 실행되기 전에 초기화됨

```
$ java PlayerTestDrive
0
1
```

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

```
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);
  }
}
```

p.325

상수

- static final 변수 == 상수

public static final double PI = 3.141592653589793;

Math.PI

- 상수 변수명은 모두 대문자와 밑줄(underscore)을 사용하여 표현

- **정적 초기화** 부분(static initializer)

```
class Foo {
  final static int x;
  static {
    x = 42;
  }
}
```

p.326

static final 변수 초기화 방법

- **선언**할 때 초기화

```
public class Foo {
  public static final int FOO_X = 25;
}
```

- **정적 초기화** 부분에서 초기화

```
public class Bar {
  public static final double BAR_SIGN;
  static {
    BAR_SIGN = (double) Math.random();
  }
}
```

```
$ javac Bar.java
Bar.java:1: variable BAR_SIGN might not have
been initialized
1 error$
```

이 부분이 없다면?

p.326

final 키워드

- **변수를 final로 지정하면** 그 값을 바꿀 수 없음
- **메서드를 final로 지정하면** 오버라이드할 수 없음
- **클래스를 final로 지정하면** 확장할 수 없음

```
class Foof {
  final int size = 3;
  final int whuffie;
  Foof() {
    whuffie = 42;
  }
  void doStuff(final int x) {
  }
  void doMore() {
    final int z = 7;
  }
}
```

```
class Poof {
  final void calcWhuffie() {
  }
}

final class MyMostPerfectClass {

}
```

p.327

바보 같은 질문은 없습니다

- 클래스를 final로 지정하는 경우는?
  - 보안 문제 때문에 이렇게 하는 경우가 종종 있습니다. String 같은 클래스를 누군가가 확장해서 String 객체가 들어갈 자리에 다형성을 이용해서 하위클래스를 집어넣으면 보안에 심각한 구멍이 생길 수도 있습니다. 어떤 클래스의 특정 메서드를 반드시 있는 그대로 써야 한다면 **클래스를 final로 지정**해서 확장할 수 없도록 하면 됩니다.

```
class InsecureString extends String {
  public InsecureString(String original) {
    super(original);
    // original 문자열을 내 서버에 보고
    ……
  }
```

```
// 코드 일부
String secureThing = new String("…");

→

String secureThing = new InsecureString("…");
```

p.328

java.lang.Math클래스의 메서드

- Math.random()
  - 0.0 이상 1.0 미만의 double 값 리턴

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

- Math.abs()
  - 절대값을 리턴

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

- Math.round()
  - 반올림하여 int 또는 long을 리턴

```
int x = Math.round(-24.8f);
int y = Math.round(24.45f);
```

- Math.min()
  - 두 인자 중 더 작은 값 리턴

```
int x = Math.min(24, 490);
double y = Math.min(90876.5, 90876.49);
```

- Math.max()

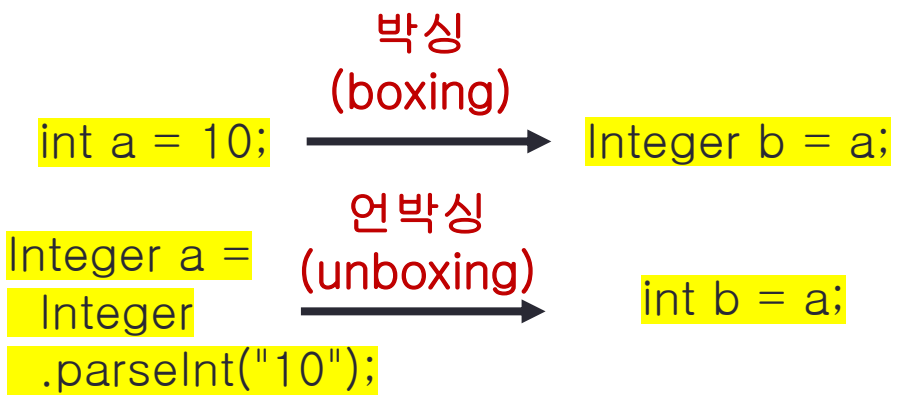
https://cr.openjdk.org/~iris/se/21/latestSpec/api/java.base/java/lang/Math.html

API에서 다른 Math 메서드도 찾아봅시다.

p.330

래퍼(wrapper) 클래스***

- 래퍼 클래스
  - **원시 유형에 해당하는 객체**
  - Boolean
  - Character
  - Byte
  - Short
  - Integer
  - Long
  - Float
  - Double




```
int x = 32;
ArrayList list = new ArrayList();
list.add(x);  // 오토박싱! ArrayList는 원래 Object만 사용 가능


int i = 288;
Integer iWrap = ~~new Integer(i)~~;  // @Deprecated(since="9")
Integer iWrap = Integer.valueOf(i);  // generally a better choice

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

p.332

Boxing Conversion

- From type **boolean** to type **Boolean**
- From type **byte** to type **Byte**
- From type **short** to type **Short**
- From type **char** to type **Character**
- From type **int** to type **Integer**
- From type **long** to type **Long**
- From type **float** to type **Float**
- From type **double** to type **Double**

Unboxing Conversion

- From type **Boolean** to type **boolean**
- From type **Byte** to type **byte**
- From type **Short** to type **short**
- From type **Character** to type **char**
- From type **Integer** to type **int**
- From type **Long** to type **long**
- From type **Float** to type **float**
- From type **Double** to type **double**

오토박싱과 원시/객체유형

- 자바 5.0 이전

```
public void doNumsOldWay() {
  ArrayList listOfNumbers = new ArrayList();
  listOfNumbers.add(new Integer(3));
  Integer one = (Integer) listOfNumbers.get(0);
  int intOne = one.intValue();
}
```

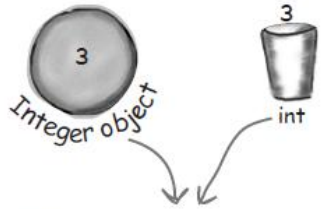
- 자바 5.0 이후 (오토박싱을 쓰는 경우)
  - 원시유형 ↔ 래퍼 객체 변환을 자동으로 처리

```
public void doNumsNewWay() {
  ArrayList<Integer> listOfNumbers
        = new ArrayList<Integer>();  // or = new ArrayList<>();  (since JAVA 1.7)
  listOfNumbers.add(3);  // 3은 int 형으로 원시형임. 컴파일러가 new Integer(3)으로 자동 변경
  int num = listOfNumbers.get(0);  // Integer 형으로 반환 → obj.intValue() 적용하여 int 형 반환
}
```

p.333

오토박싱이 작동하는 경우

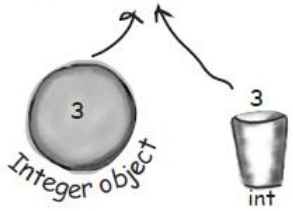
- 메서드 인자 매개변수



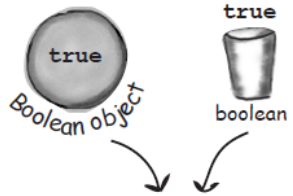

```
void takeNumber(Integer i) { }
```

- 리턴값

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



- 부울 표현식

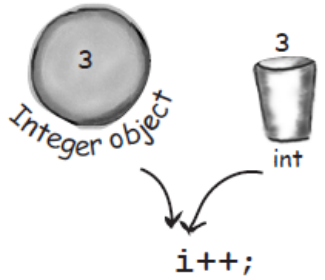



```
if (bool) {
   System.out.println("true");
}
```

- 수에 대한 연산

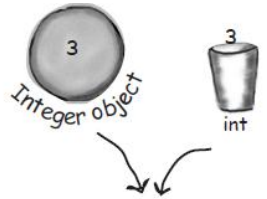
Integer j = new Integer(5);

Integer k = j + 3;



- 대입

Double d = x;




```
Integer intVal = x;
```

p.334

래퍼 클래스의 정적 유틸리티 메서드

- String을 원시 값으로 바꾸는 방법

```
String s = “2”;
int x = Integer.parseInt(s);  //  Float.parseFloat(), Short.parseShort() 등
double d = Double.parseDouble(“420.24”);

boolean b = new Boolean(“true”).booleanValue();
// (Integer 객체).intValue(), (Float 객체).floatValue() 등
// new Boolean(“true”)로 해도 됨 → 오토박싱!

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

p.336

원시 숫자를 String으로 바꾸는 방법

- String의 + 연산자를 활용하는 방법

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

- 유틸리티 메서드를 사용하는 방법

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

p.337

WrapperClassTest.java

```
6   package cse.oop2.ch10.practice;
7
8   import java.util.LinkedList;
9   import java.util.List;
10  import java.util.Optional;
11  import java.util.OptionalDouble;
12
13  /**
14   * Head First Java 10장. 연습 문제
15   * @author Prof. Jong Min Lee &lt;동의대학교 컴퓨터소프트웨어공학과&gt;
16   */
17  public class WrapperClassTest {
18
19      public static void doSomething(List<Integer> numbers) {
20          System.out.print("첫 번째 출력: ");
21          numbers.forEach(e -> System.out.print(e.toString() + "  "));
22
23          System.out.print("₩n두 번째 출력: ");
25          for (int e : numbers) {
26              System.out.print(e + "  ");
            }
```

(참고) Stream.mapToInt(ToIntFunction<? super T> mapper)
→ ToIntFunction 인터페이스의 메서드: int applyAsInt(T value)

```
28          System.out.print("₩n합계1: ");
29          Optional<Integer> sum1 = numbers.stream().reduce((x, y) -> x + y);
30          if (sum1.isPresent()) {
31              System.out.println(sum1.get());
32          }
33
34          System.out.print("합계2: ");          // Stream<Integer> 자료형
35          Integer sum2 = numbers.stream().reduce(0, Integer::sum);
36          System.out.println(sum2);
37
38          System.out.print("합계3: ");          // IntStream 자료형
39          int sum3 = numbers.stream().mapToInt(e -> e).sum();
40          System.out.println(sum3);
41
42          System.out.print("평균: ");
43          OptionalDouble avg = numbers.stream().mapToInt(e -> e).average();
44          if (avg.isPresent()) {
45              System.out.println(avg.getAsDouble());
46          }
47      }
```

e -> e 대신 Integer::intValue 사용 가능

참고.
Integer Integer.valueOf(String s)
int Integer.parseInt(String s)

```
49    public static void doConversion() {
50       Integer i1 = new Integer("123");  // not recommended
51       Integer i2 = Integer.parseInt("123");  // preferred
52       String s1 = i1.toString();
         int number = i2.intValue();  // shortValue(), byteValue()
54
55       System.out.printf("%d %d %s %d%n", i1, i2, s1, number);
56
57       try {
58          Integer i3 = Integer.parseInt("123.5");  // "hello"
59       } catch (NumberFormatException ex) {
60          System.err.println("예외 발생: " + ex);
61       }
62    }
63
64    public static void main(String[] args) {
65       List<Integer> numbers = new LinkedList<>();
66
67       for (int i=0; i<20; i++) {
68          int value = (int)(Math.random() * 100);
69          numbers.add(value);
70       }
71       doSomething(numbers);
72       doConversion();
73    }
74 }
```

Integer.valueOf("123")

```
--- exec-maven-plugin:1.5.0:exec (default-cli) @ java_maven ---
첫 번째 출력: 27 31 41 8 97 75 27 56 85 73 43 66 97 13 77 51 34 44 96 31
두 번째 출력: 27 31 41 8 97 75 27 56 85 73 43 66 97 13 77 51 34 44 96 31
합계1: 1072
합계2: 1072
합계3: 1072
평균: 53.6
123 123 123 123
예외 발생: java.lang.NumberFormatException: For input string: "123.5"
```

숫자 포매팅

- 자바에서는 숫자 포매팅 기능과 입출력 기능이 분리

- 포매팅 방법
  1. 포매터를 만듭니다.
  2. 포매팅시킵니다.

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

```
$ java TestFormats
1,000,000,000
```

p.338

숫자 포매팅

https://cr.openjdk.java.net/~iris/se/11/latestSpec/api/java.base/java/util/Formatter.html#syntax

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

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

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

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

- **%[argument_index$][flags][width][.precision]conversion**

  String.format("%,10.1f", 42000.000);

```
jshell> String.format("%,10.1f", 42000.000)
$2 ==> "  42,000.0"
```

- 참고: argument_index$
  - 첫번째 인자 1$, 두번째 인자 2$, …

유형 지시자

- %d 십진수
- %f 부동소수점수
- %x 16진수
- %c 문자

Conversion

Conversion	Argument Category	Description
'b', 'B'	general	If the argument *arg* is `null`, then the result is "`false`". If *arg* is a `boolean` or `Boolean`, then the result is the string returned by `String.valueOf(arg)`. Otherwise, the result is "true".
'h', 'H'	general	The result is obtained by invoking `Integer.toHexString(arg.hashCode())`.
's', 'S'	general	If *arg* implements `Formattable`, then `arg.formatTo` is invoked. Otherwise, the result is obtained by invoking `arg.toString()`.
'c', 'C'	character	The result is a Unicode character
'd'	integral	The result is formatted as a decimal integer
'o'	integral	The result is formatted as an octal integer
'x', 'X'	integral	The result is formatted as a hexadecimal integer
'e', 'E'	floating point	The result is formatted as a decimal number in computerized scientific notation
'f'	floating point	The result is formatted as a decimal number
'g', 'G'	floating point	The result is formatted using computerized scientific notation or decimal format, depending on the precision and the value after rounding.
'a', 'A'	floating point	The result is formatted as a hexadecimal floating-point number with a significand and an exponent. This conversion is **not** supported for the `BigDecimal` type despite the latter's being in the *floating point* argument category.
't', 'T'	date/time	Prefix for date and time conversion characters. See Date/Time Conversions.
'%'	percent	The result is a literal '%' ('\u0025')
'n'	line separator	The result is the platform-specific line separator

Date/Time Conversion

- 시간 포맷팅에 사용

'H'	Hour of the day for the 24-hour clock, formatted as two digits with a leading zero as necessary i.e. 00 - 23.
'I'	Hour for the 12-hour clock, formatted as two digits with a leading zero as necessary, i.e. 01 - 12.
'k'	Hour of the day for the 24-hour clock, i.e. 0 - 23.
'l'	Hour for the 12-hour clock, i.e. 1 - 12.
'M'	Minute within the hour formatted as two digits with a leading zero as necessary, i.e. 00 - 59.
'S'	Seconds within the minute, formatted as two digits with a leading zero as necessary, i.e. 00 - 60 ("60" is a special value required to support leap seconds).
'L'	Millisecond within the second formatted as three digits with leading zeros as necessary, i.e. 000 - 999.
'N'	Nanosecond within the second, formatted as nine digits with leading zeros as necessary, i.e. 000000000 - 999999999.
'p'	Locale-specific morning or afternoon marker in lower case, e.g."am" or "pm". Use of the conversion prefix 'T' forces this output to upper case.
'z'	RFC 822 style numeric time zone offset from GMT, e.g. -0800. This value will be adjusted as necessary for Daylight Saving Time. For long, Long, and Date the time zone used is the default time zone for this instance of the Java virtual machine.
'Z'	A string representing the abbreviation for the time zone. This value will be adjusted as necessary for Daylight Saving Time. For long, Long, and Date the time zone used is the default time zone for this instance of the Java virtual machine. The Formatter's locale will supersede the locale of the argument (if any).
's'	Seconds since the beginning of the epoch starting at 1 January 1970 00:00:00 UTC, i.e. Long.MIN_VALUE/1000 to Long.MAX_VALUE/1000.
'Q'	Milliseconds since the beginning of the epoch starting at 1 January 1970 00:00:00 UTC, i.e. Long.MIN_VALUE to Long.MAX_VALUE.

Data 포맷팅에 사용하는 Conversion

'B'	Locale-specific full month name, e.g. "January", "February".
'b'	Locale-specific abbreviated month name, e.g. "Jan", "Feb".
'h'	Same as 'b'.
'A'	Locale-specific full name of the day of the week, e.g. "Sunday", "Monday"
'a'	Locale-specific short name of the day of the week, e.g. "Sun", "Mon"
'C'	Four-digit year divided by 100, formatted as two digits with leading zero as necessary, i.e. 00 - 99
'Y'	Year, formatted as at least four digits with leading zeros as necessary, e.g. 0092 equals 92 CE for the Gregorian calendar.
'y'	Last two digits of the year, formatted with leading zeros as necessary, i.e. 00 - 99.
'j'	Day of year, formatted as three digits with leading zeros as necessary, e.g. 001 - 366 for the Gregorian calendar.
'm'	Month, formatted as two digits with leading zeros as necessary, i.e. 01 - 13.
'd'	Day of month, formatted as two digits with leading zeros as necessary, i.e. 01 - 31
'e'	Day of month, formatted as two digits, i.e. 1 - 31.

'R'	Time formatted for the 24-hour clock as "%tH:%tM"
'T'	Time formatted for the 24-hour clock as "%tH:%tM:%tS".
'r'	Time formatted for the 12-hour clock as "%tI:%tM:%tS %Tp". The location of the morning or afternoon marker ('%Tp') may be locale-dependent.
'D'	Date formatted as "%tm/%td/%ty".
'F'	ISO 8601 complete date formatted as "%tY-%tm-%td".
'c'	Date and time formatted as "%ta %tb %td %tT %tZ %tY", e.g. "Sun Jul 20 16:17:00 EDT 1969".

Flags

Flag	General	Character	Integral	Floating Point	Date/Time	Description
'-'	y	y	y	y	y	The result will be left-justified.
'#'	y[1]	-	y[3]	y	-	The result should use a conversion-dependent alternate form
'+'	-	-	y[4]	y	-	The result will always include a sign
' '	-	-	y[4]	y	-	The result will include a leading space for positive values
'0'	-	-	y	y	-	The result will be zero-padded
','	-	-	y[2]	y[5]	-	The result will include locale-specific grouping separators
'('	-	-	y[4]	y[5]	-	The result will enclose negative numbers in parentheses

[1] Depends on the definition of `Formattable`.

[2] For `'d'` conversion only.

[3] For `'o'`, `'x'`, and `'X'` conversions only.

[4] For `'d'`, `'o'`, `'x'`, and `'X'` conversions applied to `BigInteger` or `'d'` applied to `byte`, `Byte`, `short`, `Short`, `int` and `Integer`, `long`, and `Long`.

[5] For `'e'`, `'E'`, `'f'`, `'g'`, and `'G'` conversions only.

날짜 포매팅

String.format("%tc", new Date()); // t: data.time coversion, c: 날짜, 시각 모두

```
Sun Nov 28 14:52:41 MST 2004
```

String.format("%tr", new Date()); // r: %tI:%tM:%tS %Tp

```
03:01:47 PM
```

Date today = new Date();

String.format("%tA, %tB %td", today, today, today); // A: 요일, B: 월, d: 일

```
Sunday, November 28
```

String.format(%tA, %<tB %<td", today); // today를 한 번만 인자로 전달하면 됨

(3판에서는 없음)

java.util.Calendar https://cr.openjdk.java.net/~iris/se/11/latestSpec/api/java.base/java/util/Calendar.html

- java.util.Date보다는 java.util.Calendar를 써야 함
  - 표준 API에서는 대부분 java.util.GregorianCalendar를 사용
  - 현재 시각에 대한 타임스탬프용으로는 계속 java.util.Date를 써도 됨
  - java.time 패키지 사용할 것(v1.8~): LocalDateTime, ZonedDateTime, Instant 등 (cf. LocalDateTime.now())
    - LocalDateTime: A date-time without a time-zone in the ISO-8601 calendar system
- Calendar 객체 생성 방법
  - Calendar cal = new Calendar();
  - Calendar cal = Calendar.getInstance();

```
jshell> Calendar cal = Calendar.getInstance();
cal ==> java.util.GregorianCalendar[time=1601669172999,ar ... SET=32400000,DST_OFFSET=0]

jshell> cal.toString()
$4 ==> "java.util.GregorianCalendar[time=1601669172999,areFieldsSet=true,areAllFieldsSet=true,lenient=true,zone=sun.util
.calendar.ZoneInfo[id=₩"Asia/Seoul₩",offset=32400000,dstSavings=0,useDaylight=false,transitions=22,lastRule=null],firstD
ayOfWeek=1,minimalDaysInFirstWeek=1,ERA=1,YEAR=2020,MONTH=9,WEEK_OF_YEAR=40,WEEK_OF_MONTH=1,DAY_OF_MONTH=3,DAY_OF_YEAR=2
77,DAY_OF_WEEK=7,DAY_OF_WEEK_IN_MONTH=1,AM_PM=0,HOUR=5,HOUR_OF_DAY=5,MINUTE=6,SECOND=12,MILLISECOND=999,ZONE_OFFSET=3240
0000,DST_OFFSET=0]"
```

import static 구문

- 자바 5.0에서 새로 추가된 문법
- 정적 클래스나 정적 변수, 열거형(enum)을 쓸 때 import static 구문을 사용하여 소스 코드를 간단하게 할 수 있음.
- 가독성(readability) 나빠질 수 있으므로 주의!!

```
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));
   }
}
```

p.345

숙제

- 본문을 꼼꼼하게 읽어봅시다.
- 연필을 깎으며, 두뇌 운동 및 10장 끝에 있는 연습문제를 모두 각자의 힘으로 해결해봅시다.
- 포매팅 관련 코드들을 실행시켜보면서 어떤 결과가 나오는지 확인해보고, 조금씩 고쳐가면서 포매팅 연습을 해 봅시다. API 문서를 살펴보면서 또 어떤 다른 메서드들이 있는지 확인해 봅시다.