05_2 Static

Object-Oriented Programming

05-2 Static 에 대해 강의하겠습니다.

Instance Member vs Static Member

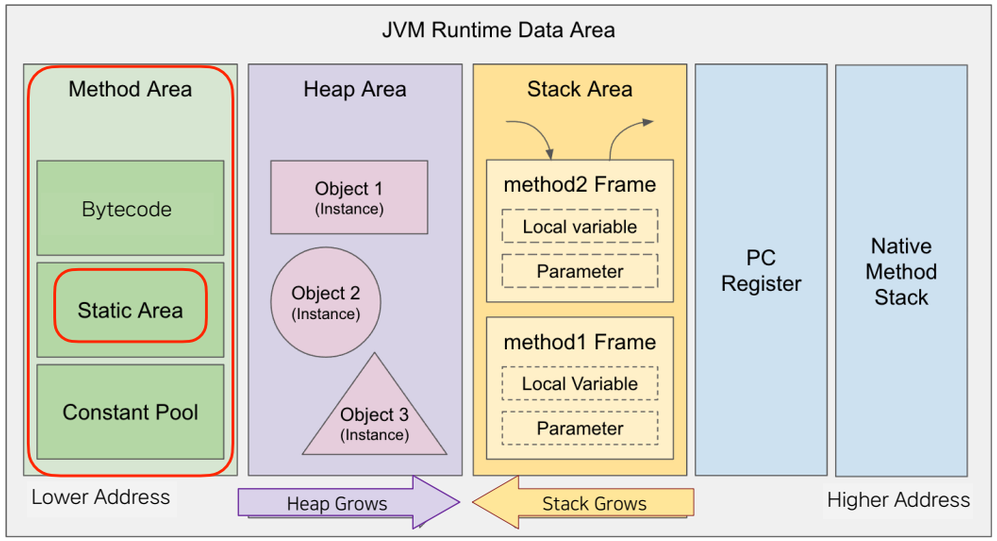
- Instance Member
  - Members that have separate values, one for each object
  - ex) Student class in a specific middle school
    - instance member: student's name
    - different students have different names
- Static Member
  - Exist only once in a class and are shared by all objects
  - ex) Student class in a specific middle school
    - static member: the name of the school
    - the same school's name for all students

페이지 2

Instance member와 Static member와의 차이에 대해 알아보겠습니다.
Instance member는 각각의 object마다
다른 value를 가지고 있는 member입니다.
예를 들어 특정한 중학교의 Student class를 고려해보면
student의 name 이라는 member는
instance member라고 할 수 있겠습니다.
왜냐하면 각각의 student마다 서로 다른 이름을 가질 수 있기 때문입니다.

static member는
하나의 class에서 모든 object가 공유하는 member를 말합니다.
예를 들어 위의 경우와 같이 특정한 중학교의 Student class를 고려해보면
school의 name은 static member라 할 수 있습니다.
왜냐하면 모든 student는 이 특정한 같은 중학교의 학생이라서
그들의 학교 이름은 모두 같기 때문입니다.

Memory Structure of JVM (Revisited)



페이지 3

JVM의 memory 구조를 다시 한번 상기해 봅시다.
여기서 static member들은 Method area에 저장됩니다.
즉, class에서 유일하게 존재하며
그 class의 object들이 모두 공유하는 것은
Method area의 static area에 저장되기 때문입니다.

Instance and Static Member

```
public class Student {
    private int id;
    private String name;
    public static schoolName;
    public final static numStudents = 120;
}
```

Method Area
static schoolName static numStudents

Heap Area
student1 id = 240427 name = "John" student2 id = 240432 name = "Tom"

페이지 4

Student class를 다시 고려해 봅니다.
private instance variable들로 id와 name이 있는데
Student class로 부터 생성된 object들, 즉 학생들은
서로다른 이름과 학번을 가질 수 있기 때문에
id와 name은 instance member가 되어야 합니다.
생성된 각각의 object들에 대한 정보
특히 instance variable들의 값이 따로 따로
Heap area에 저장되게 됩니다.
반면에 Student class의 schoolName과
numStudent는 static member 들입니다.
특정한 학교에 함께 다니는 학생들을 위한 class라고 가정했을때
schoolName은 모든 학생들에 대해 값이 같고
그 학교의 학생수도 모든 학생들에 대해 같기 때문입니다.
다만 이 example에서는 schoolName은 바뀌는 것이 가능한
public static 으로 정의되었고
numSudents는 절대 바뀌지 못하는 named constant로 되어 있습니다.
이 static member들은 이전 slide에서 보았듯이
Method area에 저장되게 됩니다.

Static Variable – Without Object Creation

- Static variable can be accessed using only class name

- ex)

```
Student st1 = new Student();
Student st2 = new Student();
st1.setName(“John”);
st2.setname(“Tom”);
System.out.println(“st1’s name: “ + st1.getName());
Student.schoolName = “Saint Jone’s Middle School”; // static variable
```


페이지 5

static variable에 접근할 때에
object를 만들지 않고 class이름으로도 접근이 가능합니다.
이 예에서 우리는 Student class의 두 object들인 st1, st2를 만들었습니다.
st1과 st2의 instance variable인 name의 값을
각각 "John" 과 "Tom" 으로 assign 했습니다.
name이 private variable이기 때문에
mutator인 setName을 사용했습니다.
그리고 st1의 name을 읽어오기 위해
st1.getName() 이라는 accessor method를 사용하여
값을 읽어오게 하였습니다.
이렇게 instance member들에 접근하기 위해서는
object를 만들고 object dot operation을 통해
접근하는 방식이 일반적입니다.
반복해서 말하지만, instance member들은
각각의 object마다 다 다른 것들이기 때문입니다.
그러나 static variable에 access할 경우에는
object를 생성할 필요가 없으며
class 이름 dot 와 같이 사용할 수 있습니다.
이 example에서도 Student dot schoolName 처럼
class이름인 Student를 사용해서 access했는데요.
static member는 class에 오로지 단 하나이기 때문에
이런 방식의 접근이 가능합니다.

The `Math` Class

- Provides a number of standard mathematical methods
  - In **`java.lang`** package, **no `import` needed**
  - All of its methods and data are **static**
  - Two predefined constants,
    - **`E`** (e: the base of the natural logarithm system)
    - **`PI`** (π = 3.141592 . . .)

ex)

```
area = Math.PI * radius * radius;  // πr²
double r = Math.random();
```


페이지 6

이제 Java의 대표적인 built-in class들의
static member들에 대해 살펴보도록 합니다.
Math class는 여러가지 표준 수학 method들을 제공합니다.
Math class는 java.lang package에 속해 있으며
따라서 import를 하지 않고 그냥 사용할 수 있습니다.
Math class의 모든 method와 data는 모두 static 입니다.
두 개의 predefined named constants가 있는데
Math.E는 자연로그의 base e 값이며
Math.PI는 pi 값을 가지고 있습니다.
이 example에서는 원의 면적을 구하는 것을 보여주는데요.
원의 면적은 pi r^2 이므로 Math.PI * radius * radius 와 같이
구할 수 있습니다.
그 아래에서 Math.random() method는 0보다 크고 1보다 작은
random 실수를 하나 return하게 됩니다.

Math.random() Method

- Get 10 random integers in [1, 6]

```
System.out.println(Math.random()); // [0.0,1.0) ex)0.366755

for (int i = 0; i < 10; i++) {
    System.out.println((int)(Math.random() * 6) + 1);
}
```

```
2
6
6
5
5
2
3
3
3
2
```

$$0 \le \text{Math.random}() < 1$$
$$0 \le \text{Math.random}() * 6 < 6$$
$$1 \le \text{Math.random}() * 6 + 1 < 7$$
$$(\text{int})(\text{Math.random}() * 6 + 1) \in \{1,2,3,4,5,6\}$$

페이지 7

Math.random() method는
비교적 간단하게 random number를
generate할 수 있는 method입니다.
이 예에서는 1에서 6사이의 폐구간에 있는
random 정수를 10개 생성하고 있습니다.
기본적으로 Math.random() 은
0보다 같거나 크고 1보다 작은 랜덤한 실수를 하나 return합니다.
예를 들면 0.366755 와 같은 랜덤한 실수가 나오게 됩니다.
이제 10개의 정수를 generate하기 위해
for 문 안에서 Math.random() 을 사용합니다.
그런데 0에서 1사이의 실수를 1에서 6사이의 정수로
어떻게 바꿀 수 있을까요?
먼저 Math.random() 은 0보다 크고 1보다 작은 실수 입니다.
여기에 6을 곱하면 0보다 크고 6보다 작은 실수가 됩니다.
여기에 1을 더하면 1보다 크고 7보다 작은 실수가 됩니다.
마지막으로 이 실수를 (int) 를 사용하여 정수로 바꾸어 주면
랜덤 넘버는 1, 2, 3, 4, 5, 6 중의 하나의 정수가 됩니다.
이와 같이 특정한 구간의 랜덤 실수 또는 정수를 generate하기 위해
차근차근 따져 보고 구하는 식을 확정하는 것이 필요합니다.
이렇게 구한 10개의 랜덤 정수는
이 output과 같습니다.
물론 output은 프로그램 실행 때마다 달라지게 될 것입니다.

Example: Math.Random() – CoinFlipDemo

```java
public class CoinFlipDemo {

    public static void main(String[] args) {
        int counter = 1;
        while (counter <= 5){
            System.out.print("Flip number " + counter + ": ");
            int coinFlip = (int)(Math.random() * 2.0); // ∈ {0,1}
            if (coinFlip == 0)
                System.out.println("Heads");
            else // coinFlip == 1
                System.out.println("Tails");
            counter++;
        }
    }
}
```

```
Flip number 1: Heads
Flip number 2: Tails
Flip number 3: Tails
Flip number 4: Heads
Flip number 5: Tails
```


페이지 8

이번에는 동전을 던져서 앞면 (Heads) 또는 뒷면 (Tails) 을
랜덤하게 나오도록 하는 프로그램을 보겠습니다.
class CoinFlipDemo 의 main method에서
우리는 while 문을 사용하기 위해
counter를 1로 초기화 했습니다.
물론 while문 말고 for문을 사용할 수도 있습니다.
“Flip number ” + counter 와 콜론을 프린트 한 후
(int)(Math.random() * 2.0) 을 실행하면
0보다 크거나 같고 2보다 작은 실수가 나오게 되고
이것을 (int) 를 이용하여 type 변환하면
정수 0 또는 1이 나오게 됩니다.
만약 0 이라면 “Heads” 라고 프린트하고
1이라면 “Tails” 라고 프린트 합니다.
그리고 counter를 하나 증가시킨 후
다시 while문의 조건으로 올라가서
counter가 5보다 작거나 같은 동안
계속 반복하게 됩니다.
이 프로그램의 output 중의 한 경우를
여기에서 보여주고 있습니다.
물론 이 output도 실행때마다 다르게 나올 수 있습니다.

Random Object

```java
import java.util.Random;

long seed = 365428;

Random rand = new Random();  // constructor
Random rand = new Random(seed);  // constructor with seed
// NOTE: when we give the same seed,
// the same random number series will be generated

int r = rand.nextInt(); // random integer [minimum int, maximum int]
r = rand.nextInt(n); // random integer in {0, 1, ..., n-1}
r = rand.nextInt(3) + 4;  // random integer in {4, 5, 6}
double rd = rand.nextDouble(); // random double 0.0 <= r < 1.0

nextBoolean() // random true or false
nextBytes()   // random byte integer
nextFloat()   // random float in [0.0, 1.0)
nextLong()    // random long integer
setSeed(long) // change the seed
```


페이지 9

이번에는 random number를 generate하기 위해
java.util.Random class를 사용하도록 하겠습니다.
이를 위해 가장 먼저 "import java.util.Random" 을 하여
Random class를 import 하였습니다.
Random class의 장점은 long integer type의 seed를
사용할 수 있다는 것입니다.
seed를 사용하면, 같은 seed를 사용하는 한
random number가 같은 순서로 generate되기 때문에
특히 똑같은 순서의 random number들을 이용하여
프로그램을 debugging할 때 편리합니다.
여기서는 우선 seed를 365428 로 주었습니다.
Default constructor로 Random object를 생성하면
seed를 사용할 수 없습니다.
이 경우에는 random number가 앞으로
어떤 순서로 어떤 수가 나오게 될지를 모르게 됩니다.
그 아래 것은 seed를 parameter로 주는 constructor 사용하여
random seed를 주는 경우입니다.
Random class의 method들로
nextInt() 는 가능한 최소 integer와 최대 integer 사이의
random integer를 발생시킵니다.
nextInt(int n) 은 parameter로 integer n을 준 경우인데
이 때에는 0부터 n-1 사이의 random integer를 발생시킵니다.
따라서 nextInt(3) + 4는 4부터 6 사이의
random integer를 발생시키게 됩니다.
nextDouble() 은 0보다 크거나 같고 1보다 작은
random double number를 발생시킵니다.
이 외에도 true나 false 중의 하나를
random 하게 발생시키는 nextBoolean()
또 nextByte(), nextFloat(), nextLong() 등의 method가 있습니다.
setSeed(long) method는 Random object의
seed를 정해주는 method인데
Random(seed) 라는 constructor를 사용하는 것과
Random() 이라고 default constructor를 사용한 후
바로 setSeed(seed) 를 call해 주는 것은
정확히 같은 동작을 하게 됩니다.

Other Methods in Class Math (1/2)

- **public static double** pow(b, e)
  - ex) Math.pow(2.0, 3.0) returns $2^3 = 8.0$
- **public static int** abs(int), float abs(float), double abs(double), long abs(long)
  - ex) Math.abs(-6) returns 6, Math.abs(-5.5) returns 5.5
- **public static int** min(int,int), long min(long,long), float min(float,float), double min(double,double)
  - ex) Math.min(3, 2) returns 2
- **public static int** max(int,int), long max(int,int), float max(float,float), double max(double,double)
  - ex) Math.max(3,5) returns 5
- **public static int** round(float), long round(double)
  - ex) Math.round(3.4523) returns 3

페이지 10

Math class의 다른 method들로 유용한 것들을 보면
double pow(b, e) 는 b의 e승을 계산합니다.
abs() 는 절대값을 return하는 method로,
int, float, double, long type을 모두 사용할 수 있도록
overloading 되어 있습니다.
min method는 두 개의 parameter 중 작은 쪽을 return하며
int, long, float, double type에 대해 overloading 되어 있습니다.
max method는 두 개 중 큰쪽을 return합니다.
round는 float와 double type에 대해 overloading되어 있는데
소수점 이하를 반올림한 정수를 return합니다.

Other Methods in Class Math (2/2)

- **public static double** ceil(double)
  - ex) Math.ceil(3.2), Math.ceil(3.8) both return 4.0
- **public static double** floor(double)
  - ex) Math.floor(3.2), Math.floor(3.8) both return 3.0
- **public static double** sqrt(double)
  - ex) Math.sqrt(4.0) returns 2.0

페이지 11

double ceil(double d) method는 d보다 크면서 가장 작은 정수를
double type으로 return합니다.
즉, "올림" operation 입니다.
double floor(double d) method는 d보다 작으면서 가장 큰 정수를
double type으로 return 합니다.
즉, "버림" operation 입니다.
double sqrt(double d)는 d의 square root를 return 합니다.

Wrapper Classes

- Class type corresponding to each of the primitive types
  - Helping class types to behave like primitive types
  - **Byte** for byte
  - **Short** for short
  - **Integer** for int
  - **Long** for long
  - **Float** for float
  - **Double** for double
  - **Character** for char
- Useful predefined constants and static methods

페이지 12

Wrapper class도 java.lang package에 속하는
대표적인 class들 입니다.
Wrapper class는 각 primitive type들에 대해
하나씩 대응하여 존재합니다.
Byte, Short, Integer, Long, Float, Double, Character 들이며
이것들은 각각 byte, short, int, long, float, double, char type의
wrapper class들 입니다.
이 wrapper class들에는 유용한 predefined constants와
static method들을 가지고 있습니다.

Boxing and Unboxing

- Boxing
  - Primitive type →wrapper class
  - auto boxing by assignment
- Unboxing
  - Wrapper class →primitive type
  - Using dedicated conversion method: …Value()

```
public class ATest5 {
    public static void main(String[] args)
{
        Byte bObj = 5;
        Short sObj = 15;
        Integer iObj = 256;
        Long lObj = 897584L;
        Float fObj = 243.563f;
        Double dObj = -98603.2543;
        Character cObj = 'y';

        byte b = bObj.byteValue();
        short s = sObj.shortValue();
        int i = iObj.intValue();
        long l = lObj.longValue();
        float f = fObj.floatValue();
        double d = dObj.doubleValue();
        char c = cObj.charValue();
    }
}
```


페이지 13

Boxing이란 primitive type의 variable이나 literal을
대응하는 wrapper class object에 바로 assign할 수 있는
기능을 말합니다.
primitive type의 value를 assign했을 때
이 value가 자동으로 boxing되어
wrapper class로 assign 되는 것처럼 보여서
boxing operation이라고 하는 것입니다.
예제 코드를 보면
Byte wrapper object인 bObj에 값 5가 바로 assign되고 있고
Short wrapper object인 sObj에는 15가
Integer wrapper object인 iObj에는 256이
바로 assign될 수 있는 것을 볼 수 있습니다.
한편, 반대로 wrapper class object를
대응하는 primitive type의 value로 바꾸려면
각 wrapper class의 …Value() 라는 method를 사용해야 합니다.
즉, Byte wrapper object인 bObj가 나타내는
primitive byte 값은 bObj.bytevalue() 로 return 됩니다.
마찬가지로 Short wrapper object일 경우
sObj.shortValue(),
비슷한 원리로 intValue(), longValue(), floatValue(),
doubleValue(), charValue() 와 같은 method를
사용해야 합니다.

Automatic Unboxing

```
public class ATest5 {
    public static void main(String[] args) {
        Byte bObj = 5;
        Short sObj = 15;
        Integer iObj = 256;
        Long lObj = 897584L;
        Float fObj = 243.563f;
        Double dObj = -98603.2543;
        Character cObj = 'y';

        byte b = bObj;
        short s = sObj;
        int i = iObj;
        long l = lObj;
        float f = fObj;
        double d = dObj;
        char c = cObj;
    }
}
```


페이지 14

그러나 ...Value() method를 쓰지 않아도
Wrapper class object를 직접 primitive type variable에
assign하게 되면
automatic unboxing 이 일어나면서
primitive variable에 적절한 값들이
assign됩니다.

Constants in Wrapper Classes

- Min, Max values
  - `Integer.MAX_VALUE`, `Integer.MIN_VALUE`
  - `Double.MAX_VALUE`, `Double.MIN_VALUE`

- true and false
  - `Boolean.TRUE` and `Boolean.FALSE`

페이지 15

Wrapper class들에 있는 유용한 constants 들 중에
먼저, 각 primitive type이 나타낼 수 있는
가장 작은 수 MIN_VALUE와
가장 큰 수 MAX_VALUE 들이 있습니다.
예를 들면 Integer.MAX_VALUE, Double.MIN_VALUE
같은 것들입니다.

Some Static Methods in Wrapper Classes

- Conversion from String to other primitive types

```
int i = Integer.parseInt("365");
double d1 = Double.parseDouble("199.98");

String theString = "  673.23  “;
String trimedString = theString.trim();              // “673.23”
double d2 = Double.parseDouble(trimedString);        // 673.23

String str = Double.toString(123.99);                // "123.99"
```


페이지 16

Wrapper class들은 String을 각 primitive type value로
conversion해 주는 method들도 가지고 있습니다.
예를 들어 Integer.parseInt(“365”) 는 int 365를 return하고
Double.parseDouble(“199.98”) 은 double 199.98를 return합니다.

theString이라는 String variable에 ” 673.23 ” 이라는
String을 assign 해 놓았는데
앞뒤로 space들이 있습니다.
이런 경우는 특히 web에서 사용자로부터 입력을 받을 때
불필요한 space들이 함께 입력되는 경우에 많이 발생합니다.
앞뒤 space들을 없애기 위해 우선 String method인 trim() 을 써서
불필요한 space들을 제거 합니다.
그럼 이제 trimedString의 값은 “673.23” 만 남아 있겠죠.
이것을 double type으로 바꾸기 위해서
wrapper class인 Double.parseDouble(trimedString) 을 call하여
673.23 이라는 double 값을 얻을 수 있습니다.

이것과는 반대로 숫자를 String으로 바꾸는 것은
wrapper class의 toString method를 사용하면 됩니다.
예를 들어 Double.toString(123.99) 를 하면
“123.99” 라는 String을 return하게 됩니다.

Some Methods in Class Character (1/2)

- public static char toUpperCase(char arg)
  - ex) char c = Character.toUpperCase('a'); // 'A"
- public static toLowerCase(char arg)
  - ex) char c = Character.toLowerCase('A'); // 'a'
- public static boolean isLowerCase(char arg);
  - ex) boolean answer = Character.isLowerCase('c'); // true
- public static boolean isWhiteSpace(char arg);
  - ex) boolean answer = Character.isWhiteSpace('\t'); // true
  - // white space characters: space (blank), tab (\t), line break (\n)
- public static boolean isLetter(char arg);
  - ex) // letter: alphabet character: a ~ z, A ~ Z
  - // non-letter character: %, &, ^, *, ...

페이지 17

char type의 wrapper class인 Character class에는
character들을 다루기 위한 유용한 static method들이 존재합니다.
우선 Character.toUpperCase('a') 를 실행하면
대문자 character 'A'를 return 합니다.
Character.toLowerCase는 반대로 대문자를 소문자로 바꾸어 return합니다.
Character.isLowerCase(c) 는 c가 소문자일때 true를, 아니면 false를 return합니다.
Character.isWhiteSpace(c) 는 c가 white space 문자일 경우 true를 return하는데
여기서 white space 문자는 space, tab, line break 등을 말합니다.
Character.isLetter(c)는 c가 alphabet 문자일 경우 true를 return하게 됩니다.

Some Methods in Class Character (2/2)

- public static boolean isDigit(char arg)
  - ex) // digit: number character such as '0' ... '9'
- public static boolean isLetterOrDigit(char arg)
  - ex) // 'a'...'z', 'A'...'Z', '0'...'9'

페이지 18

Character.isDigit(c)는 c가 '0' 부터 '9' 사이의 숫자 문자일 경우 true를 return 합니다.
Character.isLetterDigit(c)는 isLetter(c)나 isDigit(c)가 true인경우,
즉, c가 alphabet 문자이거나 숫자 문자인 경우 true를 return합니다.

Invocation Counter (1/2)

```java
// Counting how many invocation made for all methods
// using static variable

public class InvocationCounter
{
    private static int numberOfInvocations = 0;

    public void demoMethod( ) {
        numberOfInvocations++;
    }

    public void outPutCount( ) {
        numberOfInvocations++;
        System.out.println("Number of invocations so far = "
        + numberOfInvocations);
    }
```


페이지 19

이번에는 static variable의 특징을 사용하여
어떤 class에 속하는 모든 method들이
얼마나 많이 call되었는지를 count하는 프로그램을
작성해 보겠습니다.

class InvocationCounter 에
우선 static int type의 numberOfInvocation을 0으로 초기화 하였습니다.
이 variable은 모든 method들이 call되는 횟수를 저장하게 됩니다.
demoMethod() 가 call되면 static인 numberOfInvocation을
1 증가시켜 줍니다.
outPutCount() method 안에서는
numberOfInvocation을 1 증가시켜 주고
현재 numberOfInvocation 의 값을 print합니다.

Invocation Counter (2/2)

```
    public static int numberSoFar( ) {
        numberOfInvocations++;
        return numberOfInvocations;
    }

    public static void main(String[] args) {
        InvocationCounter object1 = new InvocationCounter( );
        object1.outPutCount( );  // 1
        for (int i = 1; i <= 5 ; i++)
            object1.demoMethod( );  // +1
        object1.outPutCount( ); // 7

        InvocationCounter object2 = new InvocationCounter( );
        for (int i = 1; i <= 5 ; i++) {
            object2.demoMethod( ); // +1
            object2.outPutCount( ); // +1
        }

        System.out.println("Total number of invocations = " + numberSoFar( )); // 18
    }
}
```


페이지 20

static method인 numberSoFar() 에서는
역시 numberOfInvocation을 1 증가시키고
그 값을 return합니다.
numberSoFar() 안에서는 static variable 이외에
다른 instance variable이 사용되지 않았기 때문에
이 method를 static으로 할 수 있다는 것을 눈여겨 보시기 바랍니다.

main method에서는
InvocationCounter object1을 하나 생성합니다.
이때 InvocationCounter의 값은 0으로 초기화 되었습니다.
object1.outPutCount() 를 call합니다.
이 때 invocationCounter는 1 증가 하므로 이제 값이 1이 되었습니다.
outPutCount() 에서는 이 값을 print해서 보여주게 됩니다.
이제 for문으로 object1.demoMethod()를
다섯번 연속으로 call하게 되는데
각 call마다 invocationCounter의 값이 1씩 증가되어 값은 6이 됩니다.
object1.outPutCount() 를 call하는 순간
다시 invocationCounter가 1 증가하여 7이되고
7이 print됩니다.

이제 InvocationCounter object2를 하나 더 생성했습니다.
이 때는 static variable 값이 0으로 초기화 되지 않습니다.
왜냐하면 static variable은 최초로 그 class object가 생성될 때
즉, 이 프로그램에서는 object1이 생성될 때 딱 한번 초기화가 되고
그 이후로 다시 초기화 되지는 않습니다.
for 문으로 다시 object2.demoMethod()와 object2.outPutCount()를
5번씩 연속으로 call하게 되니까
이제 invocationCounter의 값은 17이 되었습니다.
마지막 line에서 numberSoFar() 를 call 하면
invocationCounter를 다시 1 증가시키고 print하므로
마지막으로 18이 print됩니다.

이와같이 static variable은 그 class의 모든 object들이 공유합니다.
따라서 여러 object들이 공유하는 데이터를 표현하는데
유용한 면이 있습니다.