Lecture1: Java Basics

김강희
khkim@ssu.ac.kr

목 차

- 이론: 자바 문법 기초
  - 변수, 객체, 클래스, 생성자
  - 객체 생성
  - final 선언과 static 선언
  - 접근 제한자

- 실습:
  - 문자열 출력하기
  - 문자열과 숫자 값 사이의 변환
  - 2차원 배열
  - Static 선언과 내부 클래스
  - 클래스 상속

변수

- 자바 언어에는 두 가지 변수 타입이 존재함
  - 기본 (primitive) 타입
    - boolean: true or false
    - byte: 8-bit 2's complement integer
    - short: 16-bit 2's complement integer
    - int: 32-bit 2's complement integer
    - long: 64-bit 2's complement integer
    - char: 16-bit unsigned integer (UTF-16 code unit)
    - float: 32-bit IEEE 754 floating-point number
    - double: 64-bit IEEE 754 floating-point number
  - 객체 (object) 타입 : heap 할당만 가능, stack 할당 불가 (C++ 언어는 stack 할당도 가능)
    - Predefined object type : 자바 라이브러리에서 제공하는 타입
    - User-defined object type : 프로그래머가 새로 정의한 타입

객체와 클래스

- 객체 (object) : 인스턴스(instance)라는 단어와 같은 의미
  - 정의 1: 클래스(class) 타입으로 선언된 변수
    - 참고 : '변수'는 컴퓨터 주메모리에 할당된 기억 공간으로서 크기와 타입을 가짐 (예: "int a"는 변수 a가 정수형 4바이트 공간임을 선언함)
    - 예 : Matrix A = new Matrix(3, 3);
  - 정의 2: (할당된 메모리 공간을 분해하면) 기본 타입을 가진 변수들의 집합체
- 클래스 (class)
  - 객체를 구성하는 변수(data field)들과 이 변수들에 작용하는 절차(method)들을 프로그래밍 언어로 "추상적으로" 정의한 것
    - 각 변수는 또 다른 클래스 타입이거나 기본 타입(primitive type)임
    - 예 :

```
class Matrix {
    int dy; int dx;
    int array[][];
    Matrix(int cy, int cx);
    Matrix add(Matrix obj);
}
```

객체와 클래스

- 클래스 상속 (inheritance)
  - 새로운 클래스 C를 정의할 때 다른 클래스 P의 정의의 전부 또는 일부를 재사용하는 기법 (클래스 C에서는 새 변수 및 메쏘드를 추가, 또는 P의 메쏘드를 교체할 수 있음)
  - 클래스 C를 child, derivate, subclass, subtype 등으로 지칭하고 클래스 P를 parent, base, superclass, supertype 등으로 지칭
    - 예 : class TriMatrix extends Matrix {

```
int dy; int dx; // 상속됨
int array[][]; // 상속됨
int upper_or_lower;
TriMatrix(int cy, int cx);
TriMatrix add(Matrix obj);
}
```

객체 생성

- 객체 생성 방법
  - Matrix m = new Matrix(3, 3);
  - 좌변은 객체의 이름(reference)을 정의함
  - 우변은 객체를 생성함 :
    - 메모리를 할당하고,'생성자'라는 메쏘드를 호출함
      - 호출 인자가 없는 생성자를 디폴트(default) 생성자라고 부름
    - 그 결과 소속 변수들이 초기화된 값을 가지게 됨
      - dy = 3, dx = 3, array = ...
  - 질문 1: 다음 두 객체 선언은 서로 다른 객체를 생성하는가?
    - Matrix m1 = new Matrix();
    - Matrix m2 = m1;

    → 동일한 객체를 가리킨다. 즉 동일한 메모리 공간을 두 개의 이름 m1과 m2로 가리킨다.
  - 질문 2: m2 객체가 m1 객체의 내용과 동일하되 독립된 객체로서 선언하고자 하면, 어떻게 생성해야 하는가?

    → 다음과 같이 복사 생성자를 사용해야 함
    - Matrix m2 = new Matrix(m1);

생성자 작성법

- 생성자 작성시 주의할 점
  - 주의사항1: 리턴형이 없어야 함
  - 주의사항2: 반드시 public으로 선언해야 함

```
class Matrix {
  int dy ;  int dx ;
  public Matrix() { dy = 1; dx = 1; }  // 디폴트 생성자
  public Matrix(int cy, int cx) { dy = cy; dx = cx; }  // 일반 생성자
  public Matrix(Matrix obj) { dy = obj.dy; dx = obj.dx; } //복사생성자
}
```

  - 주의사항3: 네트워크 초기화 또는 데이터베이스 초기화와 같은 복잡한 동작을 하지 말아야 함
  - 주의사항4: 서브클래스의 생성자는 수퍼클래스의 생성자를 반드시 호출함

```
class TriMatrix extends Matrix {
  int upper_or_lower;
  public TriMatrix() { super(); upper_or_lower=0; }
  public TriMatrix(int cy, int cx) { super(cy, cx); upper_or_lower=0;}
  public TriMatrix(TriMatrix obj) { super(obj); … }
}
```

실습에서 알아야 할 사항들

- Matrix라는 이름의 method들은 왜 필요한가?
- 2차원 배열의 사용법은 어떠한가?
- 키워드 public, private, protected는 어느 경우에 사용하는가?
- 키워드 final은 어느 경우에 사용하는가?
- 키워드 static은 어느 경우에 사용하는가?
- 클래스 상속은 어느 경우에 필요한가?

실습: Main1()

- 하나의 자바 프로그램 안에서 main method를 갖는 class는 오직 하나
- System.out.print() or println() : arg 인자를 화면에 출력함
- String class의 format(), length(), equals(), '+' 사용하기

```
8       public static void main1(String[] args) {
9           String s1 = "Hello, Java!";
10          System.out.print("s1="); System.out.println(s1);
11          String s2 = String.format("%s, %s!", "Hello", "Java");
12          System.out.print("s2="); System.out.println(s2);
13
14          int len1 = s1.length();
15          int len2 = s2.length();
16          System.out.println("len1=" + len1);
17          System.out.println("len2=" + len2);
18
19          boolean b1 = s1.equals(s1); // compare s1 and s2 (in content)
20          boolean b2 = s1.equals(s2); // compare s1 and s2 (in content)
21          boolean b3 = (s1 == s2); // compare s1 and s2 (in objects)
22
23          System.out.println("s1.equals(s1)=" + b1);
24          System.out.println("s1.equals(s2)=" + b2);
25          System.out.println("(s1==s2)=" +  b3);
26      }
```

```
s1=Hello, Java!
s2=Hello, Java!
len1=12
len2=12
s1.equals(s1)=true
s1.equals(s2)=true
(s1==s2)=false
```

실습: Main2()

- Integer class의 parseInt(), Double class의 parseDouble()
- String class의 valueOf()

```
before : 1234, 12.34
after  : 2345, 23.45
```

```
28      public static void main2(String[] args) {
29          String istr = "1234";
30          String dstr = "12.34";
31
32          int ival = Integer.parseInt(istr); // string -> integer value
33          double dval = Double.parseDouble(dstr); // string -> double value
34          System.out.println("before : " + ival + ", " + dval);
35
36          ival = ival + 1111;
37          dval = dval + 11.11;
38          String istr2 = String.valueOf(ival); // integer value -> string
39          String dstr2 = String.valueOf(dval); // double value -> string
40
41          System.out.println("after  : " + istr2 + ", " + dstr2);
42      }
```

실습: Main3()

- 자바에서 배열 이름은 객체의 이름임 (C언어 포인터와 유사한 개념)

```
44      public static void main3(String[] args) {
45          int A1[] = null;
46          int A2[] = {1, 2, 3, 4, 5};
47          int[] A3 = new int[5];
48          int A4[] = new int[]{1, 2, 3, 4, 5};
49
50          System.out.print("A1:");
51          printArray(A1);
52          System.out.print("A2:");
53          printArray(A2);
54          System.out.print("A3:");
55          printArray(A3);
56          System.out.print("A4:");
57          printArray(A4);
58          System.out.println("A2.equals(A3)=" + A2.equals(A3));
59          System.out.println("A2.equals(A4)=" + A2.equals(A4));
60          System.out.println("Arrays.equals(A2, A3)=" + Arrays.equals(A2, A3));
61          System.out.println("Arrays.equals(A2, A4)=" + Arrays.equals(A2, A4));
62      }
63
64      public static void printArray(int a[]) {
65          if (a != null) {
66              for (int i = 0; i < a.length; i++)
67                  System.out.print(a[i] + " ");
68          }
69          System.out.println();
70      }
```

```
A1:
A2:1 2 3 4 5
A3:0 0 0 0 0
A4:1 2 3 4 5
A2.equals(A3)=false
A2.equals(A4)=false
Arrays.equals(A2, A3)=false
Arrays.equals(A2, A4)=true
```

Static 선언

- Static 선언이 없으면, (자동으로) dynamic으로 간주함
- 변수 앞에 static 선언이 붙으면,
  - 그 변수가 속한 클래스의 인스턴스들 사이에서 공유되는 변수 (개별 객체 수준에서 static 변수를 위한 공간 할당이 없음)
  - 그 변수는 클래스에 소속된다고 말하며, 객체에 소속되지 않음
- 메쏘드 정의 앞에 static 선언이 붙으면,
  - Static 메쏘드는 서브클래스에서 재정의(override)할 수 없음
  - 그 메쏘드는 클래스에 소속된다고 말하며, (객체의) dynamic 변수들을 접근할 수 없고, 오직 그 클래스의 static 변수들만 접근할 수 있음
- 클래스와 static 선언의 관계
  - { } 으로 감싼 블록 안에서 선언된 클래스는 (static 선언 없으면) 무조건 dynamic 클래스임
    - 예: Matrix class 안에서 선언된 dynamic class의 객체 생성은 Matrix 객체를 먼저 생성한 후 그것에 의존적인 형태로만 가능함 (해당 Matrix 객체의 모든 변수들을 접근할 수 있음)
  - { } 블록 밖에서 선언된 클래스는 무조건 static 클래스임
    - 예: Matrix class 안에서 선언된 static class의 객체 생성은 Matrix 객체 생성이 없어도 가능함 (Matrix 클래스의 static 변수들만 접근할 수 있음)

실습: Main4()

```
64      public static void main4(String[] args) {
65          Nested m1 = new Nested( cy: 1,  cx: 2);
66          Nested m2 = new Nested( cy: 3,  cx: 4);
67          System.out.println("m1.get_dy()=" + m1.get_dy() + ", m1.get_dx()=" + m1.get_dx());
68          System.out.println("m2.get_dy()=" + m2.get_dy() + ", m2.get_dx()=" + m2.get_dx());
69          Nested.InnerD d1 = m1.new InnerD();
70          Nested.InnerS s1 = new Nested.InnerS();
71          Nested.InnerD d2  = m2.new InnerD();
72          Nested.InnerS s2 = new Nested.InnerS();
73          System.out.println("d1.get_dy()=" + d1.get_dy() + ", s1.get_dx()=" + s1.get_dx() + ", d1.sum()=" + d1.sum());
74          System.out.println("d2.get_dy()=" + d2.get_dy() + ", s2.get_dx()=" + s2.get_dx() + ", d2.sum()=" + d2.sum());
75      }
```

```
m1.get_dy()=1, m1.get_dx()=4
m2.get_dy()=3, m2.get_dx()=4
d1.get_dy()=1, s1.get_dx()=4, d1.sum()=5
d2.get_dy()=3, s2.get_dx()=4, d2.sum()=7
```

```
101 class Nested {
102     private int dy; // dynamic variable
103     private static int dx; // static variable
104     public int get_dy() { return dy; }
105 @   public static int get_dx() { return dx; } // can be declared 'dynamic'
106     public Nested(int cy, int cx) { dy = cy; dx = cx; }
107     public class InnerD {
108         public int get_dy() { return dy; }
109         public int sum() { return dy+dx; }
110     }
111     public static class InnerS {  public int get_dx() { return dx; }  }
112 }
```

접근 제한자

- **private** modifier:
  - 필드 또는 메쏘드 앞에 붙어서, 소속 클래스 외부에서 해당 필드 또는 메쏘드를 참조할 수 없게 한다.
- default (or package) access:
  - 필드 또는 메쏘드 앞에 접근 제한자를 사용하지 않는 경우로서, 한 패키지 내부 어디에서든지 접근할 수 있다. **클래스 간에 공유하는 상태 변수를 정의할 때** 유용하다.
- **protected** modifier:
  - 필드 또는 메쏘드 앞에 붙어서, 서브클래스가 해당 필드 또는 메쏘드를 참조하는 것을 허용한다.
  - 또한, default access를 허용한다.
- **public** modifier:
  - 필드 또는 메쏘드 앞에 붙어서, 소속 클래스 외부 어디에서든지 해당 필드 또는 메쏘드를 참조하는 것을 허용한다.

실습: Main5()

- Matrix class를 상속받은 MyMatrix class는 부모 클래스의 print()를 교체함
- Matrix형 변수는 MyMatrix형 객체를 가리킬 수 있음

```
87      public static void main5(String[] args) {
88          Matrix m1 = new Matrix( cy: 3, cx: 3);
89          m1.print(); System.out.println();
90          int A[][] = { { 0, 1, 0, }, { 1, 1, 1, }, { 0, 0, 0, } };
91          Matrix m2 = new Matrix(A);
92          m2.print(); System.out.println();
93          MyMatrix m3 = new MyMatrix( cy: 3, cx: 3);
94          m3.print(); System.out.println();
95          MyMatrix m4 = new MyMatrix(A);
96          m4.print(); System.out.println();
97          m2 = m4; // polymorphism: Matrix covers MyMatrix!!
98          m2.print(); System.out.println();
99      }
```

```
Matrix(3,3)
0 0 0
0 0 0
0 0 0

Matrix(3,3)
0 1 0
1 1 1
0 0 0

□ □ □
□ □ □
□ □ □

□ ■ □
■ ■ ■
□ □ □

□ ■ □
■ ■ ■
□ □ □
```

실습: Main5()

- 자식 클래스 MyMatrix의 생성자는 부모 클래스의 생성자를 호출함으로써 작성함 → super()
- Matrix class 안에서 dy, dx를 protected로 정의하면 어떤 이점이 생기나?

```
114 class MyMatrix extends Matrix {
115     public MyMatrix() { super(); }
116     public MyMatrix(int cy, int cx) { super(cy, cx); }
117     public MyMatrix(int[][] a) { super(a); }
118     public void print() {
119         int dy = get_dy();
120         int dx = get_dx();
121         int array[][] = get_array();
122         for (int y=0; y < dy; y++) {
123             for (int x=0; x < dx; x++) {
124                 if (array[y][x] == 0) System.out.print("□ ");
125                 else if (get_array()[y][x] == 1) System.out.print("■ ");
126                 else System.out.print("X ");
127             }
128             System.out.println();
129         }
130     }
131 }
```

Final 선언

- 클래스 P 앞에 final 선언되면,
  - P의 서브클래스를 정의할 수 없음
- 클래스 P에 소속된 메쏘드 M 앞에 final 선언되면,
  - P의 서브클래스에서 메쏘드 M을 재정의(override)할 수 없음
- 변수(필드,형식인자,지역변수) 앞에 final 선언되면,
  - 그 변수는 (선언과 동시에 또는 생성자 안에서) 한번 값이 정해지면 변경될 수 없음
  - 그 변수는 그 변수를 감싸는 블록 안에서 사용 전에 오직 1회만 초기화 가능함
- 형식인자(parameter) 앞에 final 선언되면,
  - 실제인자의 값은 해당 메쏘드 안에서 변경될 수 없음
  - 다만, 실제인자가 객체이면 객체 내부의 필드들은 변경될 수 있음

감사합니다!