04_2 Constructors and Overloading

Object-Oriented Programming

constructor와 overloading에 대해 강의하겠습니다.

Overloading

- Two or more methods (in the same class) have the same method name

```
public void setDate(int month, int day, int year)
public void setDate(String month, int day, int year)
public void setDate(int year)
```

- Any two definitions of the method name must have **different signatures**
  - Signature = (method name + parameter list)
  - Differing signatures: different numbers and/or types of parameters

```
setDate(int, int, int)
setDate(String, int, int)
setDate(int)
```


페이지 2
overloading이란 같은 class내에 같은 이름의 method가
두 개 이상이 존재하는 것을 말합니다.
이 setDate라는 method는 세가지 다른 형태로 define되고 있습니다.
overloading이 되기 위해서는 method들이 서로 다른 signature들을 가지고 있어야만 합니다.
여기서 signature는 method 이름과 parameter list를 말하는데
signature가 다르다는 것은 parameter의 갯수 또는 type이 서로 다르다는 뜻입니다.
우리의 setDate의 예에서는
첫번째와 두번째 것은 세개의 parameter들로 갯수는 같으나
첫번째 parameter의 type이 int와 String으로 다릅니다.
세번째 setDate method는 parameter의 갯수가 하나로
다른 두개의 method와 다릅니다.
이와 같이 signature가 다른 것을 바탕으로
compiler는 어떤 method call이 어떤 method definition을 말하는 것인지를
구별할 수 있게 됩니다.

Example: Overloading (1/3)

```java
public class CarOverloading {
    public static void main(String[] args) {
        Car3 c1 = new Car3();
        Car3 c2 = new Car3();
        Car3 c3 = new Car3();
        c1.set("Hyundai Grandure", 2024);
        c2.set(2021);
        c3.set("Kia Niro");
        System.out.println("c1: " + c1);
        System.out.println("c2: " + c2);
        System.out.println("c3: " + c3);
    }
}
```

```
OUTPUT:
c1: Car3{model='Hyundai Grandure', year=2024}
c2: Car3{model='NO_MODEL', year=2021}
c3: Car3{model='Kia Niro', year=0}
```


페이지 3
Overloading을 사용하는 예로
Car3 class의 구현을 살펴 보겠습니다.
CarOverloading class의 main에서
c1, c2, c3라는 세개의 Car3 object들을 생성합니다.
그리고 method set을 세번 call하는데
모두 다른 signature들을 가지고 있는 것을 볼 수 있습니다.
c1.set의 parameter들은 String과 int.
c2.set의 parameter는 단 한개로 int.
c2.set의 parameter는 역시 한개로 String 입니다.
set method는 Car3 object의 instance variable의 value에
주어진 값들을 assign합니다.
마지막 println들에서 c1, c2, c3를 parameter에서 access하는데
지난 slide에서 공부한 것과 같이
이런 경우 Car3 class의 toString() method가 실행되어
미리 정해놓은대로 object의 info를 String으로 만들어 return 받게 됩니다.

Example: Overloading (2/3)

```
class Car3 {
    String model;
    int year;

    void set(String model, int year) {
        this.model = model;
        this.year = year;
    }
    void set(String model) {
        this.model = model;
        this.year = 0;
    }
```

```
    void set(int year) {
        this.model = "NO_MODEL";
        this.year = year;
    }
    void set() {
        this.model = "NO_MODEL";
        this.year = 0;
    }
```


페이지 4
그럼 set method들이 어떻게 구현되어 있는지 보겠습니다.
Car3는 model과 year 두개의 instance variable들이 있습니다.
이 프로그램에서 사용한 원칙은
model과 year가 다 주어지면 그것들을 assign 하되
parameter로 주어지지 않으면 model은 NO_MODEL,
year가 없으면 0으로 setting하는 것입니다.
우리는 set method의 서로다른 네개의 버전을 여기에서 볼 수 있습니다.

Example: Overloading (3/3)

```
    // equals method
    public boolean equals(Car3 other) {
        return model.equals(other.model) && (year == other.year);
    }


    // toString method
    public String toString() {
        return "Car3{" + "model='" + model + '\'' + ", year=" + year + '}';
    }
}
```


페이지 5
다른 method들은 이전 슬라이드에서 봤듯이
새로 다시 define하는 equals와 toString 입니다.

Cannot Overload Based on the Type Returned

- Return type is not the part of signature

```
public class SampleClass {
    public int computeSomething(int n) { ... }
    public double computeSomething(int n) { ... }  // Compile ERROR!!
}
```


페이지 6
Overloading 가능 여부를 결정하는 method signature에
return type의 차이는 포함되지 않습니다.
즉, return type만 다를 경우 overloading 가능한 method들이 되지는
않는다는 것입니다.
이 예에서 computeSomething(int n) 이 두 개인데,
parameter의 수와 type은 같고,
다만 return type이 int와 double로 다른데
이런 경우 compile error가 나면서
overloading을 허용하지 않습니다.

Constructors

- Special kind of method
- Initializing the instance variables when object created
- Syntax: `public Classname(anyParameters) { code }`
  - A constructor must have the same name as the class
  - A constructor has no type returned, not even `void`
  - Constructors are typically overloaded

페이지 7
Constructor는 특별한 종류의 method이며
object가 생성될 때 instance variable을 초기화하는데 사용합니다.
Constructor의 문법은 일반적인 method와 비슷합니다만 몇가지 다른 점이 있습니다.
먼저 constructor의 name은 class의 name과 정확히 같아야 합니다.
또 constructor는 return type이 없습니다. void도 쓰지 않습니다.
constructor는 일반적으로 overloaded되어서
여러가지 버전의 constructor를 준비하고 사용할 수 있도록 합니다.

Calling Constructor

- Called when an object of the class is created using **new**

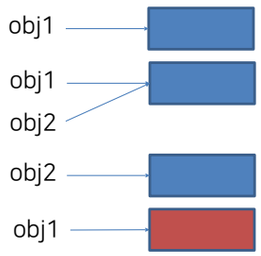
```
ClassName objectName = new ClassName(anyArgs);
```

- If a constructor is invoked again (using **new**), the first object is discarded and an entirely new object is created

```
Class1 obj1 = new Class1(anyArgs);

Class1 obj2 = obj1;

obj1 = new Class1(anyArgs);
```



페이지 8
Constructor는 object가 만들어질 때 new 명령에 의해 실행됩니다.
즉 new 다음에 나오는 부분이 constructor call인 것입니다.
만약 constructor가 여러번 call된다면 첫 object는 무시되고
완전히 새로운 object가 생성되게 됩니다.
Class1 type의 object obj1이 new 명령으로
constructor가 실행되면서 생성됩니다.
이어서 obj2는 새로운 object를 생성하지 않고
obj1의 reference가 그대로 assign됩니다.
그리고 나서 Class1의 constructor가 다시한번 실행되면
obj1은 완전히 새로운 object를 가리키게 됩니다.
물론 obj2는 그대로 같은 object를 가리키도록 남아 있습니다.

Example: Condition Test in Constructor (1/6)

- Test for the conditions that an instance variable should have (e.g., scope)

```
public class Date {
    final static String[] monthName = {"JAN", "FEB", "MAR", "APR", "MAY",
            "JUN", "JUL", "AUG", "SEP", "OCT", "NOV", "DEC"};
    int day;
    int month;
    int year;

    public Date(int day, int month, int year) {
        boolean leapYear = false;

        // testing leaf year
        if ((year % 4 == 0 && year % 100 == 0) || (year % 400 == 0))
            leapYear = true;
```


페이지 9
Constructor는 단순히 instance variable들을 initialize하는 것 뿐 아니라
parameter로 받는 value들이 적절한 것인지 테스트를 하는 데 유용합니다.
예를 들면 value의 범위 등이 정해진 룰에 맞는지를 테스트 할 수 있습니다.
class Date는 day, month, year의 세개의 int variable을 가지고 있으며
날짜를 나타내는 class입니다.
Constructor에서는 먼저 주어진 year가 윤년인지를 결정합니다.
윤년인지에 따라 2월의 day 수가 달라지기 때문입니다.
윤년 여부를 테스트하는 rule은
year가 4로 나누어 떨어질 때는 윤년이며,
다만 year가 4와 100으로 동시에 나누어 떨어지면 윤년이 아닙니다.
다만 year가 4, 100, 400으로 동시에 나누어 떨어지면 윤년입니다.
이에 따라 year가 윤년인 경우에는 leapYear variable이 true가 됩니다.

Example: Condition Test in Constructor (2/6)

```
        // checking month
        if (month < 1 || month > 12) {
            if (month < 1) month = 1;
            else month = 12;
            System.out.print("Date Constructor: Wrong month < 1 or > 12");
            System.out.println(" month fixed to = " + month);
        }

        // checking day
        if (day < 1) {
            day = 1;
            System.out.print("Date Constructor: Wrong day < 1");
            System.out.println(" day fixed to = 1");
        }
```


페이지 10
month는 1부터 12 이내의 범위에 있어야 하며
1보다 작은 경우 month를 1로 만들고
12보다 큰 경우 month를 12로 만듭니다.
이런 경우들에서는 month가 잘못되었다는 메시지와
그래서 고쳤다는 메시지를 프린트 합니다.
day는 음수가 아니어야 하며
음수인 경우 강제로 1로 만듭니다.
또 이 때, 메시지들이 프린트 됩니다.

Example: Condition Test in Constructor (3/6)

```
        switch (month) {
            case 1:
            case 3:
            case 5:
            case 7:
            case 8:
            case 10:
            case 12:
                if (day > 31) {
                    System.out.print("Date Constructor: Wrong day > 31");
                    System.out.println(" day fixed to 31");
                    day = 31;
                }
                break;
```


페이지 11
이제 day의 수를 정하는 부분입니다.
1월, 3월, 5월, 7월, 8월, 10월, 12월은 31일까지 이기 때문에
day가 31보다 더 큰 경우는 31로 만듭니다.

Example: Condition Test in Constructor (4/6)

```
            case 2:
                if ((leapYear && day > 29)) {
                    System.out.print("Date Constructor:(Leap Year) Wrong day:" + day);
                    System.out.println(" day fixed to 29");
                    day = 29;
                }
                else if (!leapYear && day > 28) {
                    System.out.print("Date Constructor:(Normal Year) Wrong day:” +day);
                    System.out.println(" day fixed to 28");
                    day = 28;
                }
                break;
            default: // month = 4, 6, 9, 11
                if (day > 30) {
                    System.out.print("Date Constructor: Wrong day > 30");
                    System.out.println(" day fixed to 30");
                    day = 30;
                }
        }
```


페이지 12
윤년인 경우 2월의 day 수가 29보다 크거나
평년인 경우 2월의 day 수가 28보다 클 때
day 수를 적절히 고쳐 주며
나머지 달들, 즉, 4, 6, 9, 11월은
day가 30 이내에 있는지를 테스트 합니다.

Example: Condition Test in Constructor (5/6)

```
        this.year = year;
        this.month = month;
        this.day = day;
    }

    public String toString() {
        String acbc = "A.C.";
        if (year < 0) {
            year = -1 * year;
            acbc = "B.C.";
        }
        String answer = monthName[month - 1] + "." + day + ", " + year + " " + acbc;
        return answer;
    }
}
```


페이지 13
check가 다 끝나면
유효한 year, month, day를 instance variable들에 assign합니다.
Date class도 toString method를 재정의하여
적절한 info string을 return하게 하였습니다.

Example: Condition Test in Constructor (6/6)

```
public class DateTest {
    public static void main(String[] args) {
        System.out.println("\n25, 4, -3295   ");
        Date date1 = new Date(25, 4, -3295);
        System.out.println(date1);


        System.out.println("\n29, 2, 1900   ");
        Date date2 = new Date(29, 2, 1900);
        System.out.println(date2);


        System.out.println("\n29, 2, 1898   ");
        Date date3 = new Date(29, 2, 1898);
        System.out.println(date3);


        System.out.println("\n5, -3, 1995   ");
        Date date4 = new Date(5, -3, 1995);
        System.out.println(date4);
    }
}
```

```
OUTPUT:
25, 4, -3295
APR.25, 3295 B.C.


29, 2, 1900
FEB.29, 1900 A.C.


29, 2, 1898
Date Constructor: (Normal Year) Wrong Feb. day: 29
day fixed to 28
FEB.28, 1898 A.C.

5, -3, 1995
Date Constructor: Wrong month < 1 or > 12 month
fixed to = 1
JAN.5, 1995 A.C.
```


페이지 14
이 부분은 여러가지 경우의 data들이
constructor의 parameter들로 주어지는 경우입니다.
여기에서는 year가 음수인 경우
B.C. 몇년으로 표시하도록 하였는데
그 부분은 앞의 constructor 구현에 빠져 있습니다.
배포한 source code를 참고하기 바랍니다.

A Constructor: `this` (1/2)

```java
public class ThisInConstructor {
    private String name;
    private int age;

    public ThisInConstructor() {    // Default constructor
        this("Unknown", 0); // Call another constructor
    }

    public ThisInConstructor(String name) {    // Constructor with name only
        this(name, 0); // Call another constructor
    }

    public ThisInConstructor(String name, int age) {// with name and age
        this.name = name;  // Here, 'this' is not for calling constructor
        this.age = age;
    }
```


페이지 15
this는 constructor 내에서 다른 overloaded된 constructor를
call하는 때에 사용될 수 있습니다.
class ThisInConstructor에는
세가지 version의 overloaded된 constructor들이 있습니다.
이들 중 첫번째와 두번째 것들은
세번째 constructor를 call하므로써
구현을 쉽게 만들어 가고 있습니다.

A Constructor: `this` (2/2)

```
    public String toString() {
        return "Name: " + name + ", Age: " + age;
    }

    public static void main(String[] args) {
        ThisInConstructor person1 = new ThisInConstructor();
        System.out.println("Person1: " + person1);

        ThisInConstructor person2 = new ThisInConstructor("Alice");
        System.out.println("Person2: " + person2);

        ThisInConstructor person3 = new ThisInConstructor("Bob", 30);
        System.out.println("Person3: " + person3);
    }
}
```

```
OUTPUT:
Person1: Name: Unknown, Age: 0
Person2: Name: Alice, Age: 0
Person3: Name: Bob, Age: 30
```


페이지 16
이 슬라이드는 앞 슬라이드의 ThisInConstructor class의 constructor들을 테스트하기 위한 code 입니다.

Provide Your Default Constructor! (1/2)

- Default Constructor
  - A constructor having no argument, ex) Date x = new Date();

- If no constructor in the class, Java will **automatically create a default constructor** (but doesn't do anything).

- If one or more constructor exist in the class, Java will **not provide default constructor**.

페이지 17
Default constructor는 parameter가 없는 constructor를 말합니다.
example에서는 Date의 default constructor가 call되는 것을 보여줍니다.
그런데 Java에서는 어떤 class에
programmer가 define한 constructor가 전혀 없는 경우,
자동으로 default constructor를 하나 만들어 줍니다.
물론 자동으로 생성된 default constructor는
아무 일도 하지 않습니다.
그러나 programmer가 어떤 constructor라도 하나 이상 구현했다면
자동으로 default constructor를 생성해 주지 않습니다.
이것은 programmer가 define한 constructor가
default constructor인지 여부에 상관없이
일관되게 적용되는 rule입니다.

Provide Your Default Constructor! (2/2)

```java
public class Date2Test {
    public static void main(String[] args) {
        Date2 d = new Date2();  // ERROR! no default constructor provided
    }
}

class Date2 {
    private int month;
    private int day;
    private int year;

    public Date2(int month, int day, int year) { // user-written constructor
        this.month = month;
        this.day = day;
        this.year = year;
    }
}
```


페이지 18
이 slide의 제목처럼
Java programming에서는
programmer가 class를 구현할 때
아무 하는 일이 없더라도
default constructor를 반드시 define하도록 권장하고 있습니다.
그 이유는 무엇일까요?
이 slide의 경우 Date2Test의 main에서
Date2 class의 object 하나를 default constructor로 생성하려 시도합니다.
그런데 Date2의 definition을 보니
programmer가 작성한 constructor가
default constructor가 아닌 세개 parameter constructor 입니다.
일단 이 constructor 하나를 define해 놓았기 때문에
Java는 default constructor를 자동으로 만들어 주지 않습니다.
즉, Date2에는 default constructor가 없는 것입니다.
따라서 아까 Date2의 default constructor를 실행하려 한 부분에서
compile error가 발생하게 됩니다.
programmer는 간혹 이런 상황의 발생 이유를
잘 모를 수도 있습니다.
따라서 이런 상황을 막기 위해
class마다 항상 default constructor를 구현하는
습관을 가지는 것이 좋습니다.

Default Variable Initializations (1/2)

- Instance variables are automatically initialized
  - **`boolean`** types are initialized to false
  - Other primitives are initialized to the zero of their type
  - Class types are initialized to **`null`**
- However, it is a better practice to explicitly initialize instance variables in a constructor
- Note: Local variables are not automatically initialized

페이지 19
Class의 instance variable들은
정해진 default value들로 자동으로 initialize됩니다.
boolean type일 경우에는 false
이외의 다른 primitive type일 경우에는 zero.
Reference type일 경우에는 null이 default value가 됩니다.
하지만 더 좋은 습관은
instance variable들을 constructor안에서
명시적으로 initialize해 주는 것입니다.
JVM을 구현할 때, instance variable들을 자동 초기화 시키는 것을
잊어버려서 구현이 안 되어 있을 수도 있기 때문입니다.
한편, instance variable들이 아닌 local variable들은
자동 초기화가 일어나지 않습니다.
따라서 어떤 값을 assign하지 않고 그 값을 read하려고 하면
error가 일어나게 됩니다.

Default Variable Initializations (2/2)

```
public class ATest4 {
    public static void main(String[] args) {
        int x, y;
        FooClass f = new FooClass();
        System.out.println(f.a + " " + f.b + " " + f.c);
        System.out.println(x + " " + y); // ERROR!! x, y not initialized
    }
}

class FooClass {
    int a;
    boolean b;
    double c;
}

OUTPUT:
0 false 0.0
```


페이지 20
ATest4 class의 main method는
x와 y라는 local variable들을 가지고 있는데
두번째 println 문에서 x와 y가 초기화 되지 않았는데
value를 읽으려 했기 때문에
error가 나게 됩니다.

Example: StringTokenizer Class

```
import java.util.StringTokenizer;

public class StringTokenizerDemo {
    public static void main(String[] args) {
        // Example input string
        String input = "Java is a high-level, class-based; object-oriented programming
language.";

        // Creating a StringTokenizer with space, comma, and semicolon as delimiters
        StringTokenizer tokenizer = new StringTokenizer(input, " ,;");

        // Counting tokens
        int tokenCount = tokenizer.countTokens();
        System.out.println("Total number of tokens: " + tokenCount);

        // Iterating through tokens and printing each token
        while (tokenizer.hasMoreTokens()) {
            String token = tokenizer.nextToken();
            System.out.println(token);
        }
    }
}
```


페이지 21
StringTokenizer class에 대해 간단히 알아보겠습니다.
StringTokenizer class는 주어진 string에서
token (word)를 하나씩 분리하여 처리할 수 있도록 도와줍니다.
StringTokenizer를 이용하기 위해서는
먼저 java.util.StringTokenizer를 import해야 합니다.
이제 input이라는 String이 code와 같이 주어졌다고 가정합니다.
StringTokenizer object를 하나 생성합니다.
constructor의 parameter로는
입력으로 쓸 String과 delimiter가 될 문자들을 묶은 String을 pass합니다.
여기서는 delimiter로 space, comma, semicolon을 사용하게 됩니다.
이제 위의 delimiter로 분리한 token들이
input string 안에 총 몇개나 있는지를 알아내기 위해
StringTokenizer의 countTokens() 라는 method를 사용할 수 있습니다.
그 아래의 while 문에서 condition으로 사용한 tokenizer.hasMoreTokens() 는
아직 token들이 하나라도 남아 있을 때에는 true를 return하기 때문에
모든 token들을 하나씩 하나씩 처리할 수 있게 됩니다.
다음의 token을 가져오기 위한 명령은 StringTokenizer의 nextToken() 입니다.
여기서는 단순이 token을 하나씩 프린트하기만 합니다.

Example: OUTPUT

```
"Java is a high-level, class-based; object-oriented programming language.";
```

```
Java
is
a
high-level
class-based
object-oriented
programming
language.
```


페이지 22
주어진 input String은 delimiter들에서 나누어져
여러개의 token String을 차례차례 처리할 수 있습니다.
이것이 while문에서 token들을 하나씩 프린트한 결과입니다.