07_1 Interface

Object-Oriented Programming

Interface에 대해 강의하겠습니다.

Interfaces

- Not a class but very **similar to abstract class**
- Any class can **implement** the interface
- **Multiple inheritance** possible

```
interface Name {
    public static final Type Constant_Variable = Value;
    public abstract Type Method(Param1, Param2, ...);
}
```

- "**public static final**" for constants **can be omitted**
- "**public abstract**" for method heading **can be omitted**

페이지 2

Interface는 class는 아니지만 abstract class와 유사합니다.
어떤 class도 interface를 implement 할 수 있습니다.
class hierarchy와는 달리, interface의 경우에는
multiple inheritance가 허용됩니다.
즉, 하나의 class가 여러 개의 interface를
동시에 implement하는 것이 가능합니다.
interface의 grammar를 살펴보면
먼저 interface keyword 다음에 interface의 이름이 오고
interface의 내부에는
public static final로 시작하는 named constant와 그 초기값
그리고 public abstract method의 signature가 존재할 수 있습니다.
named constants를 위한 public static final은 언제나 같기 때문에
실제 interface의 정의에서는 생략되는 경우가 많습니다.
또한 method signature 앞의 public abstract도
언제나 존재해야 하기 때문에
실제로는 생략되는 경우가 많습니다.

Example: `Ordered` Interface (1/4)

```
public interface Ordered {

    public boolean precedes(Object other);
    public boolean follows(Object other);
    // NOTE: o1.follows(o2) == o2.precedes(o1)

}
```


페이지 3

이제 interface의 실제 예로 Ordered interface를 정의해 보겠습니다.
public interface Ordered는
named constants는 가지지 않습니다.
abstract public boolean type인 precedes method가 있는데
abstract가 생략되어 있음에 유의합니다.
또 parameter other의 type이
대문자 O로 시작하는 Object 임에 주목하도록 합니다.
Object는 우리가 사용하는 모든 class의 ancestor이며
우리가 새로 define하는 모든 class들의 parent입니다.
즉, Object type을 parameter로 놓으면
어떤 class type의 parameter라 할지라도
일단 parameter로 받는 것이 가능해 집니다.
Object type에 대해서는 9장에서 더 자세히 살펴보겠습니다.
그 아래 follows도 abstract public boolean type의 method 입니다.
여기서 precedes는 caller object가 parameter인 other보다
앞설때 true를 return하며
follows는 caller가 other보다 뒤에 있을 때 true를 return한다는
의미를 가지고 있습니다.
따라서 o1.follows(o2) 의 return 값은
o2.precedes(o1) 의 return 값과
같다고 볼 수 있습니다.
그러나 "앞서다" "뒤따르다" 라는 것은 의미적인 것만 내포할 뿐이지
어떤 기준으로 object의 순서를 가늠하는지는
이 interface를 implement한 class들 마다
달라질 수 있습니다.
또 한가지, interface에서 주의할 점은
이러한 method와 constant의 의미적인 면들이
comment 등으로 묘사되어 있다 할 지라도
그 의미와 다른 method나 constant로 class에 구현되었을 때
그것이 error는 아니라는 것입니다.
즉, 문법적인 오류만 없으면 프로그램은 실행될 것이며
interface를 정의할 때 내포한 그 의미들은
class 구현에 있어 어떠한 제약도 가하지 않는 다는 점을
잘 알아두도록 합니다.

Example: `Ordered` Interface (2/4)

```java
public class Person implements Ordered {
    private String name;
    private int age;
    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }
    @Override
    public boolean precedes(Object other) {
        if (other instanceof Person) {
            Person otherPerson = (Person) other;
            return this.age < otherPerson.age;
        }
        return false;
    }
```


페이지 4

이제 Ordered Interface를 class로 implement하는 과정을
살펴 보도록 하겠습니다.
class Person 뒤에 implements keyword를 써서
interface Ordered를 implement한다는 정보를 표시하고 있습니다.
class inheritance때 사용했던 keyword인 extends와 다르다는 점을
기억해야 하겠습니다.
Person class에는
private인 String name과 int age의 두 개의 instance variable들이 있습니다.
두 instance variable들의 초기값을 parameter로 받는
constructor가 있습니다.
그 아래에는 Ordered interface의 두 개의 abstract methods들 중 하나인
precedes를 overriding하여 implement하였습니다.
Person이 concrete class가 되려면 Ordered의 모든 abstract methods들을
implement하여야 한다는 점을 기억하도록 합시다.
precedes method에서는 먼저 Object type의 parameter로 받은 other가
caller인 Person class에 대해 "other instanceof Person",
즉, other의 original class type이 Person 이던가
또는 Person의 descendant 일 때만 true가 되는 조건을
만족하는지 먼저 test합니다.
이 구현에서 우리는 caller object (this) 의 age와 other의 age를
비교하여 this.age가 작을 경우 true를 return하게 하려고 합니다.
그런데 other가 Person이나 Person의 descendant가 아니라면
instance variable age를 가지고 있지 않을 수도 있기 때문에
이 구현 자체가 성립되지 않는 것입니다.
이제 (other instanceof Person) 의 condition test가 true로 판명되면
일단 other의 type을 Person type인 otherPerson으로
downcasting 합니다.
이 downcasting은 if 문의 조건 "other instanceof Person" 이
true가 될 때 실행되기 때문에 downcasting의 가능 조건을 만족합니다.
즉, other의 original class가 Person이던지 아니면
Person의 descendant 라는 조건을 만족하는 것입니다.
이제 this.age < otherPerson.age 의 결과값을 return하면 됩니다.
한편 if 문의 조건을 만족하지 못한 경우,
other로 올바른 type의 parameter가 pass되지 않았다는 뜻이 되므로
이 경우에는 무조건 false를 return하게 되겠습니다.

Example: `Ordered` Interface (3/4)

```
    @Override
    public boolean follows(Object other) {
        if (other instanceof Person) {
            Person otherPerson = (Person) other;
            return this.age > otherPerson.age;
        }
        return false;
    }
}
```


페이지 5

이것은 precedes와 거의 비슷하게 구현된 follows method 입니다.
여러분이 직접 따라가면서 어떻게 구현되었는지를 꼭 따져 보시기 바랍니다.

Example: `Ordered` Interface (4/4)

```
public class PersonOrderDemo {
    public static void main(String[] args) {
        Person person1 = new Person("Alice", 25);
        Person person2 = new Person("Bob", 30);

        System.out.println("Person1 precedes Person2: " +
                           person1.precedes(person2)); // true
        System.out.println("Person1 follows Person2: " +
                           person1.follows(person2));  // false
        System.out.println("Person2 precedes Person1: " +
                           person2.precedes(person1)); // false
        System.out.println("Person2 follows Person1: " +
                           person2.follows(person1));  // true
    }
}
```


페이지 6

이제 Person class를 test해 보는 demo 프로그램입니다.
person1과 person2는 각각 “Alice”와 “Bob” 이라는 이름의
Person object들 입니다.
person1.precedes(person2)를 실행하면
Alice의 age가 Bob의 age보다 작으므로
true를 print합니다.
같은 이유로 person1.follows(person2) 는 false
person2.precedes(person1) 은 false
person2.follows(person2) 는 true 입니다.

Example: Interface Hierarchy (1/4)

```java
// Base Interface
public interface Shape {
    public double calculateArea();
}

// Derived Interface
public interface ColoredShape extends Shape {
    public String getColor();
}

// More Specific Interface
public interface TexturedShape extends ColoredShape {
    public String getTexture();
}
```


페이지 7

class와 마찬가지로 interface들도 hierarchy가 있을 수 있습니다.
Base interface로 Shape을 define하는데
calculateArea() method가 abstract method로 포함되어 있습니다.
ColoredShape interface는 Shape을 inherit하는데
이 때 extends keyword를 사용합니다.
이것은 class inheritance 때와 동일합니다.
ColoredShape interface에는 Shape의 method인
calculatedArea() 뿐 아니라 getColor() 라는 abstract method가
하나 더 추가되었습니다.
TexturedShape interface는 ColoredShape을 inherit하는데
getTexture() 라는 abstract method가 하나 더 추가되었습니다.

Example: Interface Hierarchy (2/4)

```
public class TexturedRectangle implements TexturedShape {
    private double width;
    private double height;
    private String color;
    private String texture;

    public TexturedRectangle(double width, double height,
                             String color, String texture) {
        this.width = width;
        this.height = height;
        this.color = color;
        this.texture = texture;
    }
```


페이지 8

TexturedRectangle class는 TexturedShape interface를 implement합니다.
width, height, color, texture의 네개의 private instance variable들이 있습니다.
constructor에서는 네개의 instance variable들을 초기화 합니다.

Example: Interface Hierarchy (3/4)

```
    @Override
    public double calculateArea() {
        return width * height;
    }

    @Override
    public String getColor() {
        return color;
    }

    @Override
    public String getTexture() {
        return texture;
    }
}
```


페이지 9

TexturedRectangle class는 TexturedShape interface를 implement하고 있으므로
TexturedShape interface가 가진 세개의 abstract method
calculatedArea(), getColor(), getTexture()를
모두 implement해야 합니다.

Example: Interface Hierarchy (4/4)

```
public class TexturedRectangleDemo {

    public static void main(String[] args) {
        TexturedRectangle rectangle =
                        new TexturedRectangle(5, 10, "Red", "Smooth");

        System.out.println("Area: "
                           + rectangle.calculateArea()); // Area: 50.0
        System.out.println("Color: "
                           + rectangle.getColor());     // Color: Red
        System.out.println("Texture: "
                          + rectangle.getTexture()); // Texture: Smooth
    }

}
```


페이지 10

이제 demo program에서는 먼저 TextureRectangle class의 object인
rectangle을 생정합니다.
width, height, color, texture가 각각 5, 10, "Red", "Smooth" 로
초기화 됩니다.
이제 implement한 세개의 method를 call하여
area, color, texture를 print합니다.

The `Comparable` Interface

- In the **`java.lang`** package
- Automatically available to any program
- Only one method that must be implemented:
  ```
  public int compareTo(Object other);
  ```
  - return value
    - < 0 : if the calling object "comes before" the parameter other
    - == 0 : if the calling object "equals" the parameter other
    - > 0 : if the calling object "comes after" the parameter other

페이지 11

이번에는 java.lang package에 속한
Comparable interface에 대해 알아보려 합니다.
java.lang에 속해있기 때문에
import 없이 그냥 사용이 가능합니다.
Comparable interface에는
오직 하나의 abstract method가 있습니다.
바로 abstract public int compareTo(Object other) 입니다.
parameter로 Object type을 받는 것에 유의합니다.
return value는 세가지 경우가 있는데
calling object, 즉, this가 other보다 앞에 올 경우 (즉, 작을 경우)
negative integer를 return 합니다.
보통은 -1을 return 하는 경우가 많습니다.
만약 this가 other와 정확히 같다면
0을 return 합니다.
this가 other보다 뒤에 있다면 (즉, 크다면)
positive integer를 return합니다.
보통은 +1을 return합니다.

Other Typical Interfaces in Java API

- `java.lang.Runnable`: having the method '`run()`' to run something
- `java.util.Comparator`: having the method: '`int compare(a,b)`'
  - compare's Output has the same meaning of 'compareTo()' in Comparable
  - i.e., returns negative int (if a < b),
  - positive int (if a > b),
  - zero (if a == b)

페이지 12

이와같이 Java에는 기본 package와 함께 제공되는
interface들이 여러개 있습니다.
그 중에 java.lang.Runnable은 method void run()을 가지고 있습니다.
java.util.Comparator는 method int compare(a, b) 를 가지고 있습니다.
특히 Comparator는 앞쪽에서 살펴본 Comparable interface와 유사하나
그 method compare가 compareTo와는 달리
parameter를 두개 받는 다는 점이 다릅니다.
두 parameter a, b의 관계가 a < b 이면 negative integer를
a == b 이면 0을, a > b 이면 positive integer를 return 합니다.

Example: Comparator Interface (1/3)

```
import java.util.Comparator;

class AgeComparator implements Comparator {
    @Override
    public int compare(Object o1, Object o2) {
        Human h1 = (Human) o1;
        Human h2 = (Human) o2;
        return Integer.compare(h1.getAge(), h2.getAge());
    }
}
```


페이지 13

이제 Comparator Interface를 사용하는 예를 보도록 하겠습니다.
Comparator는 java.util package에 속하기 때문에
먼저 import를 해 주어야 합니다.
AgeComparator class가 Comparator interface를 implement하고 있습니다.
abstract method인 compare() 를 implement하기 위해
Object type의 두 parameter o1과 o2를 사용합니다.
뒤에 나올 Human class로 o1과 o2를 downcasting하고
Integer.compare로 h1과 h2의 age를 비교하여 return합니다.
사실 여기서도 Object인 o1과 o2를
Human class로 downcasting 하기 전에
o1 instanceof Human, o2 instanceof Human과 같이
test를 먼저 한 후 true가 되어야
downcasting을 할 수 있을 것입니다.
그러나 이 example에서는 그 code가 생략되어 있다는 것을
알아두어야 할 것입니다.

Example: Comparator Interface (2/3)

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

    public Human(String name, int age) {
        this.name = name;
        this.age = age;
    }
    public String getName() { return name; }
    public int getAge() { return age; }

    @Override
    public String toString() {
        return name + " (" + age + ")";
    }
```


페이지 14

Comparison에 사용할 Human class를 정의합니다.
name과 age fields가 있고
constructor와
accessor들이 있으며
toString을 override하였습니다.

Example: Comparator Interface (3/3)

```
    public static void main(String[] args) {
        Human human1 = new Human("Alice", 30);
        Human human2 = new Human("Bob", 25);
        AgeComparator comparator = new AgeComparator();
        int comparisonResult = comparator.compare(human1, human2);
        if (comparisonResult < 0) {
            System.out.println(human1.getName() + " is younger than "
                               + human2.getName());
        } else if (comparisonResult > 0) {
            System.out.println(human1.getName() + " is older than "
                               + human2.getName());
        } else {
            System.out.println(human1.getName() + " and "
                           + human2.getName() + " are the same age");
        }
    }
}
```

```
OUTPUT:
Alice is older than Bob
```


페이지 15

main에서는 먼저 Alice와 Bob이라는 두 명의 Human object를 create하고
AgeComparator를 생성하여
comparator의 compare method에 두 Human object를 pass합니다.
나온 결과가 음수인지, 양수인지, 0과 같은지에 따라 적절한 message를 print합니다.

Selection Sort

position 0: [64, 25, 12, 22, 11] : minimum of {64, 25, 12, 22, 11} = 11
[11, 25, 12, 22, 64] : swap 64 and 11
position 1: [11, 25, 12, 22, 64] : minimum of {25, 12, 22, 64} = 12
[11, 12, 25, 22, 64] : swap 25 and 12
position 2 : [11, 12, 25, 22, 64] : minimum of {25, 22, 64} = 22
[11, 12, 22, 25, 64] : swap 25 and 22
position 3 : [11, 12, 22, 25, 64] : minimum of {25, 64} = 25, so no swap
result: [11, 12, 22, 25, 64]

페이지 16

이번에는 Selection sort 알고리즘을 구현해 보겠습니다.
array가 주어졌을 때 position 0부터 n-1까지 증가시키면서
array[positon] 부터 array[n-1] 까지 중에 가장 minimum의 값을 찾아내고
그 minimum 값을 array[position]과 swap 합니다.
이 과정을 position을 증가시키면서 반복합니다.
먼저 position이 0일 때에는 array[0] = 64 부터
array[n-1] 까지의 minimum 을 구하면 110이 됩니다.
이제 64와 11을 exchange하면 array[0] = 11 이 확정됩니다.
다음 step에는 array[1] 부터 array[n-1]까지의 minimum을 찾으면
12가 되고 이를 array[1] = 25와 exchange하여 array[1] = 12가 확정됩니다.
다음 step에는 array[2] 부터 array[n-1]까지의 minum을 찾으면
22가 되고 25와 22를 exchange하여 array[2] = 22로 확정합니다.
마지막 step에서 array[3]부터 array[n-1]까지의 minimum은
25이므로 array[3] = 25로 확정된 후 sorting이 완료됩니다.

Example: Selection Sort (1/4)

```
public class SelectionSort {
    public void sort(Comparable[] array) { // parameter: array of
                                           // interface Comparable
        int n = array.length;
        for (int i = 0; i < n - 1; i++) { // position from 0 to n-1
            int minIndex = i; // index of minimum
            for (int j = i + 1; j < n; j++) { // from i+1 to n-1
                if (array[j].compareTo(array[minIndex]) < 0) {
                    minIndex = j;
                }
            }
            swap(array, i, minIndex);
        }
    }
```


페이지 17

앞에서 보았던 selection sort의 알고리즘을 SelectionSort class의
sort() method에 구현하였습니다.
여기서 눈여겨 보아야 할 것은 sort method의 parameter가
Comparable interface의 array type 이라는 것입니다.
interface를 parameter로 한다는 것은
그 interface를 implement한 모든 concrete class들을
다 parameter로 받을 수 있다는 뜻이됩니다.
Comparable interface에는 compareTo() method가 존재하기 때문에
두 object간의 크고 작은 관계를 test해 볼 수 있는 수단이 존재합니다.
position i를 0부터 n-1 까지 증가시키면서
각 position i 마다 j = i + 1 부터 n-1 까지 중에
가장 작은 object를 compareTo method를 이용하여 찾습니다.
그리고 나서 j에 관한 for loop가 끝나면
array[i] 와 array[minIndex]의 두 element를 exchange합니다.

Example: Selection Sort (2/4)

```
    private void swap(Comparable[] array, int i, int j) {
        Comparable temp = array[i];
        array[i] = array[j];
        array[j] = temp;
    }
}
```


페이지 18

이것은 swap() method인데
swapping을 위해서는 temp variable 하나에
우선 한쪽 element인 array[i]를 assign해 놓고
array[i] = array[j] 로 assign하고
array[j]는 아까 저장해 둔 temp를 assign 하면됩니다.

Example: Selection Sort (3/4)

```java
public class SelectionSortDemo {
    public static void main(String[] args) {

        // Integer array demo
        Comparable[] intArray = {64, 25, 12, 22, 11};
        SelectionSort sorter = new SelectionSort();
        sorter.sort(intArray);
        System.out.print("Sorted Integer Array: ");
        printArray(intArray); // 11 12 22 25 64
```


페이지 19

이 슬라이드에서는 SelectionSort를 test하기 위해
Comparable의 array인 intArray가 주어집니다.
intArray를 sorting하기 위하여
우선 SelectionSort object인 sorter를 생성하고
sorter.sort(intArray) 를 call 하여
sorting된 intArray를 구하게 됩니다.
처음의 input array = {64, 25, 12, 22, 11} 이
output array = {11, 12, 22, 25, 64}
로 sorting되게 됩니다.

Example: Selection Sort (4/4)

```
        // String array demo
        Comparable[] stringArray =
                        {"apple", "orange", "banana", "kiwi", "grape"};
        sorter.sort(stringArray);
        System.out.print("Sorted String Array: ");
        printArray(stringArray); // apple banana grape kiwi orange

    }

    public static void printArray(Comparable[] array) {
        // print the array ...
    }
}
```


페이지 20

integer array 뿐 아니라 대소관계를 정의할 수 있는
과일 이름 String array를 Comparable[] array type으로 정의하였습니다.
sorter.sort(stringArray); call을 통해 sorting을 실행합니다.

Inconsistent Interfaces

```java
interface Inter1 {    int NUMBER = 25;   }
interface Inter2 {    int NUMBER = 32;   }

// Using multiple inheritance
public class InconsistentInterfaceDemo implements Inter1, Inter2 {
    public static void main(String[] argc) {
        int x = NUMBER;  // Compile error, 25? 32? which one?
    }
}

public class InconsistentInterfaceDemo implements Inter1, Inter2 {
    public static void main(String[] argc) {
        // But if we don't use NUMBER, then no compile error
    }
}
```


페이지 21

이번 예에서는 두 interface Inter1과 Inter2에 같은 이름의 constant인
NUMBER가 정의되어 있습니다.
이 때 InconsistentInterfaceDemo class는 두 interface들인
Inter1과 Inter2를 모두 inherit 하는 multiple inheritance로 정의되어 있습니다.
keyword implements 뒤에 Inter1, Inter2 의 두 interface를 볼 수 있습니다.
이때 이 class안에서 NUMBER라는 constant를 사용할 경우
이것은 inter1과 inter2에 모두 존재하기 때문에
NUMBER의 값이 25인지 32인지 알수가 없게 되며
따라서 compile error가 나게 됩니다.
그러나 Inter1과 Inter2를 모두 multiple로 inherit하더라도
NUMBER constant를 사용하지만 않는다면
compile error가 나지 않습니다.

Polymorphism Using Interfaces (1/3)

- An instance of a class implementing an interface can be assigned to the variable of the interface

```
interface Fightable {
    public void move(int x, int y);
    public void attack(Fightable f);
}

class Fighter implements Fightable {
    public void move(int x, int y) {  //. . .   }
    public void attack(Fightable f) {  //. . .   }
}

Fighter f = new Fighter();
Fightable f = new Fighter();
```


페이지 22

이번에는 interface를 사용하는 polymorphism에 대해 설명하겠습니다.
interface를 implement하는 class의 instance는
모두 interface variable에 assign이 가능합니다.
먼저 interface Fightable을 고려해 봅니다.
move와 attack의 두 abstract methods를 가지고 있습니다.
attack method의 parameter로는
자기 자신인 Fightable interface를 사용하였습니다.
이제 Fighter class로 Fightable interface를 implement합니다.
methods의 body는 편의상 생략하였습니다.
Fighter class object를 new Fighter() 로 생성하여
Fighter variable f에 assign하는 것은 당연히 가능합니다.
또 new Fighter() 만들어진 Fighter object를
Fightable interface의 variable인 f에 assign하는 것도 가능합니다.

Polymorphism Using Interfaces (2/3)

- An interface type can be used as **parameter type** in a method

```
interface Fightable {
    public void move(int x, int y);
    public void attack(Fightable f);
}

class Fighter implements Fightable {
    public void move(int x, int y) {
        // implement here
    }
    public void attack(Fightable f) {
        // implement here
    }
}
```


페이지 23

Fightable interface의 attack method의 parameter가
Fightable interface 자신있었던 것을 기억할 수 있을 겁니다.
이처럼 interface type은
method의 parameter type으로도 사용이 가능합니다.

Polymorphism Using Interfaces (3/3)

- An interface can be used as the **return type** of a method

```
Fightable method() {
    // ...
    return new Fighter();
}
```


페이지 24

Interface type은 또한 method의 return type으로도 가능합니다.
이 method는 return type으로 Fightable을 지정해 놓았는데
실제로는 Fightable을 implement한 어떤 class라도
return 가능합니다.