Java프로그래밍

5강. 인터페이스와 다형성 (교재 4장)

컴퓨터과학과 김희천 교수

한국방송통신대학교

오늘의 학습목차

1. 추상 클래스
2. 인터페이스
3. 다형성
4. 열거 자료형
5. 익명 클래스

1. 추상 클래스

1) 추상 메소드

- 메소드 선언에 abstract 키워드를 사용함
- 몸체의 구현이 없이 형식만 존재하는 메소드
  - ✓ 반환형, 이름, 인자 선언만 존재함
  - ✓ 자식 클래스에 상속될 때, 몸체의 구현이 필요함
  - ✓ 상반된 의미의 final과 함께 사용할 수 없음

```
abstract public class Shape {
        ......
        //모양이 정해지지 않았기 때문에 면적을 계산할 수 없음
        abstract public double getArea( );
}
```

2) 추상 클래스

◆ 클래스 정의에 abstract 키워드를 사용함

✓ 물론 데이터 필드나 일반 메소드를 포함할 수 있음

✓ 객체 생성을 할 수 없음

- 구체적이지 못한 불완전한 클래스라는 의미

✓ 추상 메소드를 포함하는 클래스는 반드시 추상 클래스라야 함

```
abstract public class Shape {
   …   …
    abstract public double getArea( );
}
```

✓ Shape s = new Shape(“red”); //컴파일 오류

3) 추상 클래스의 사용

- 의미적으로 유사한 클래스를 묶고자 할 때 사용
  - ✓ 공통으로 사용할 데이터 필드와 메소드를 정의
- 추상 클래스는 불완전한 클래스
  - ✓ 기능적으로 구현하기 어려운 메소드가 존재
- 추상 클래스는 자식 클래스로 상속되어 사용됨
  - ✓ 자식 클래스에서 추상 메소드를 구현해야 함
- 그러면 자식 클래스는 객체 생성이 가능
  - ✓ 자식 클래스가 추상 메소드를 구현하지 않으면
    계속해서 자식 클래스도 추상 클래스로 남음
- 추상 클래스는 일반 클래스와 인터페이스의 중간적 성격을 가짐

2. 인터페이스

1) Java의 인터페이스

- 100% 추상적 클래스
  - ✓ 인터페이스의 모든 메소드가 추상 메소드(public abstract)
  - ✓ 단, 몸체가 구현된 default 메소드와 static 메소드도 포함 가능
    - 모든 메소드의 기본 접근 제어자는 public
  - ✓ 데이터 필드는 클래스 상수만 가능(public static final)
- 참조 자료형이며 직접적 객체 생성은 불가
- 인터페이스의 이름은 보통 형용사임
  - ✓ Runnable, Serializable, Comparable

2) 인터페이스의 정의

- 문법은 클래스 정의와 유사함
- 정의할 때 키워드 class 대신에 interface를 사용
  - ✓ abstract는 생략하는 것이 보통임
- 메소드는 기본적으로(생략하더라도) public abstract임
  - ✓ 몸체가 없으며, 반환형,이름, 매개변수 목록만 표시
- default 메소드와 static 메소드도 가능
  - ✓ 이 경우 몸체를 구현해야 함
  - ✓ 기본적으로(생략하더라도) public임
- 데이터 필드는 항상(생략 가능) public static final임
  - ✓ 클래스 상수만 가능함

3) 인터페이스의 사용

- 추상 클래스와 마찬가지로 자식 클래스에 상속되어 사용됨
  - ✓ 인터페이스를 상속받는 자식 클래스는 **모든 추상 메소드를 구현**해 주어야 함
- 의미적으로는 관련이 없으나 기능적으로 유사한 클래스들을 묶을 때 인터페이스를 사용할 수 있음
  - ✓ 예: 대소 비교가 가능한 객체들의 자료형을 묶을 때
- 인터페이스를 상속받아 자식 인터페이스를 정의할 수 있음
  - ✓ 인터페이스의 상속(또는 확장)

4) 인터페이스의 상속

◆ 자식 인터페이스가 부모 인터페이스를 상속받는 경우

✓ 인터페이스를 상속받아 인터페이스를 정의할 때,
키워드 extends를 사용

✓ 여러 인터페이스를 상속받는 다중 상속도 가능

✓ 예

```
interface 자식인터페이스 extends 부모인터페이스 { … }
```

5) 인터페이스의 구현

- 자식 클래스가 부모 인터페이스를 상속받는 경우
  - ✓ 자식은 부모가 나열한 기능(추상 메소드)을 구현해야 함
  - ✓ 구현을 통해 클래스를 정의할 때 implements를 사용
- 예

```
class MovablePoint implements Movable { … }

class 자식클래스 extends 부모클래스
    implements 부모인터페이스1, 부모인터페이스2 { … }
```

6) 인터페이스의 구현 예

```
interface Movable {
   void moveUp( );
   void moveDown( );
   void moveLeft( );
   void moveRight( );
}

public class MovableTest {
  public static void main(String[] args) {
     Movable m1 = new MovablePoint(5, 5);
     System.out.println(m1);
     m1.moveUp( );
     System.out.println(m1);
     m1.moveRight( );
     System.out.println(m1);
  }
}
```

```
Point at (5,5)
Point at (5,6)
Point at (6,6)
```

```
class MovablePoint implements Movable {
   private int x, y;
   public MovablePoint(int x, int y) {
        this.x = x;
        this.y = y;
   }
   public String toString( ) {
      return "Point at (" + x + "," + y + ")";
   }
   public void moveUp( ) { y++; }
   public void moveDown( ) { y--; }
   public void moveLeft( ) { x--; }
   public void moveRight( ) { x++; }
}
```

7) 디폴트 메소드

- 인터페이스에서 선언하는 메소드에 기본 구현을 넣을 수 있음
  - ✓ 자식 클래스에서 상속받을 때, 디폴트 메소드를 그대로 사용하거나 몸체를 다시 정의해 줄 수 있음
  - ✓ **메소드 선언시 default를 사용하고 몸체를 구현해 줌**
- 인터페이스에 나열된 기능을 확장할 때, 기존 코드의 수정을 피하기 위함
  - ✓ 단순히 추상 메소드가 추가된다면, 이전 인터페이스를 구현한 클래스를 수정해야 함

```
interface DoIt {
   void doSomething( );
   int doSomethingElse(String s);
   // 아래를 새로 추가한다면?
   default boolean didItWork(int i, String s) {
       … …
  }
}
```

8) 추상 클래스, 인터페이스, 클래스의 형변환

- 인터페이스와 클래스는 모두 사용자 정의형
- extends와 implements에 따라 상위/하위 자료형 관계가 설정됨
- 상위 유형의 변수는 하위 객체의 참조값을 가질 수 있음
- 상위 유형의 변수가 가리키는 객체의 실제 유형에 따라 수행되는 메소드가 결정됨(동적 바인딩)
  - ✓ 메소드 호출 시, 변수의 선언 유형으로 정하지 않음
  - ✓ 예: SuperClass super = new SubClass( );

    super.method( ); // SubClass에서 찾음

3. 다형성

1) 다형성

◆ 다형성

✓ 유사하지만 다양한 형상이나 다양한 기능을 가진다는 뜻

- 한 부모에서 나온 두 자식 객체는 비슷하지만 다름
- 하나의 클래스에서 오버로딩된 메소드들은 유사하지만 조금씩 다른 기능을 수행함
- 자식 클래스에서 재정의된 메소드는 부모의 것과 유사하지만 다른 기능을 수행함

2) 다형성과 형변환

◆ 형 변환

✓ 상속 관계에 있는 클래스 간에는 타입 변환이 가능함

- 전혀 다른 두 클래스 간에는 타입 변환이 금지됨

✓ 하위 클래스에서 상위 클래스로의 형 변환은 문제없음

- 업캐스팅이라 하며 자동으로 형 변환 가능함
- 참조형 변수는 같은 유형의 객체 또는 하위 객체를 참조할 수 있음
- 예:
  Animal animal = (Animal) new Dog( ); //하위 객체 참조

3) 다형성과 오버라이딩

◆ 클래스의 다형성

✓ 부모 클래스로부터 상속받은 메소드를 자식 클래스에서 오버라이딩할 수 있음

✓ 부모와 자식에서 같은 이름의 메소드가 다른 기능을 수행

- 같은 이름과 매개 변수 및 반환형을 가지나 몸체가 다름

◆ 인터페이스의 다형성

✓ 자식 클래스들에서 상위 인터페이스의 메소드를 다르게 구현함

4) 클래스 상속과 다형성(1)

```
class A {
  public void func( ) {
      System.out.println( “ a ” );
  }
}
class B extends A {
  public void func( ) {
      System.out.println( “ b ” );
  }
}
class C extends B {
  public void func( ) {
      System.out.println( “ c ” );
  }
}
```

```
public class PolymorphTest {
  public static void main(String args[ ])
{
    A a = new B( );
    a.func( );
    a = new C( );
    a.func( );
  }
}
```

```
b
c
```

4) 클래스 상속과 다형성(2)

```
class Employee {
   int nSalary;
   String szDept = null;
   public void doJob( ) {
     System.out.println("Do something");
   }
}
class Sales extends Employee {
   public Sales( ) { szDept = "Sales Dept"; }
   public void doJob( ) {
     System.out.println("Do sales");
   }
}
class Development extends Employee {
  public Development( ) { szDept = "Sales Dept";}
  public void doJob( ) {
     System.out.println("Do development");
  }
}
```

```
public class Company1 {
   public static void main(String args[ ]) {
         Employee emp1, emp2;
         emp1 = new Sales( );
         emp2 = new Development( );
         emp1.doJob( );
         emp2.doJob( );
   }
}
```

```
Do sales
Do development
```

4. 열거 자료형

1) 열거형 정의

- 열거형은 미리 정의된 상수값을 만들기 위한 자료형
- enum을 사용하여 정의
- 열거형으로 선언된 변수에는 미리 지정된 값만 대입 가능
- 상수값을 배열로 리턴하는 static 메소드로 values( )를 제공

```
Enum Day {
     SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY
}
```

```
// main 함수에서
Day day = Day.MONDAY;
for (Day d : Day.values( )) {
       System.out.println(d);
}
```

2) 열거형의 생성자와 메소드

- 상수 선언이 필드나 메소드보다 먼저 정의되어야 하며 세미콜론(;)으로 끝나야 함
- 열거형 정의에 필드와 메소드를 포함할 수 있음
- 생성자는 열거형과 같은 이름을 가지며 접근 제어자는 생략 또는 private이어야 함
- 열거형에서 상수값은 마치 하나의 객체와 같음
- 열거형의 생성자는 상수값을 설정(객체 생성)할 때 자동 호출됨

3) 열거형 사용 예

```
enum BaseballTeam {

   LG(40, 30), SS(30, 40), KT(20, 50),
SK(35, 35), NC(55, 15);

   private final int win;
   private final int lose;

   private BaseballTeam(int win, int
lose) {
                  this.win = win;
                  this.lose = lose;
   }

   public double winsRate( ) {
      return (win * 100.0) / (win + lose);
   }
}
```

```
public class EnumTest2 {

   public static void main(String args[ ]) {

      BaseballTeam bt = BaseballTeam.LG;

      System.out.println(bt.winsRate( ));

    }

}
```

5. 익명 클래스

1) 익명 클래스

- 일회성으로 1개의 객체를 생성하기 위한 클래스
  - ✓ 클래스 정의와 동시에 객체를 생성할 수 있음
- 슈퍼 클래스를 상속받거나 인터페이스를 구현하도록 익명 클래스를 정의함
  - ✓ new 슈퍼클래스 ( ) { … } //슈퍼클래스의 자식 객체 생성
  - ✓ new 인터페이스 ( ) { … } //인터페이스를 구현하는 자식 객체 생성
  - ✓ 중괄호가 익명 클래스의 몸체

2) 클래스를 상속받는 익명 클래스

```
public class AnonymousTest {
     public static void main(String args[ ]) {
          CSuper sub = new CSuper( ) {
             public int b = 20;
             public void method1( )
                { System.out.println( " sub1 " ); }
             public void method3( )
                { System.out.println( " sub3 " ); }
          } ;

          sub.method1( );
          sub.method2( );
          System.out.println(sub.a);
          ... ...
     }
}
```

```
sub1
super2
10
```

```
class CSuper {
   public int a = 10;
   public void method1() {
        System.out.println( " super1 " );
   }
   public void method2() {
         System.out.println( " super2 " );
   }
}
```

3) 인터페이스를 구현한 익명 클래스

```
public class AnonymousTest {
   public static void main(String args[]) {
       MyInterface sub = new MyInterface( ) {
         public void method( ) {
           System.out.println( " sub1 " );
         }
       } ;

       sub.method( );
   }
}
```

```
interface MyInterface {
          public void method( );
}
```

Java프로그래밍

다음시간안내

6강. 제네릭과 람다식