예제 5-4 : instanceof 연산자 활용

1

- 예제 5-4를 다음과 같이 수정하시오

1) 각 클래스에 static void로 print 메소드를 정의하고, print 메소드에서 클래스 이름을 출력하도록 수정 (예를 들어, Person 클래스의 print에서는 "Person" 출력)
2) InstanceOfEx 클래스의 print 메소드에서는 해당 클래스의 print 메소드 출력 (Person의 instance인 경우 Person.print() 호출)
3) Main에서는 "1. Person, 2. Student, 3. Researcher, 4. Professor"로 메뉴를 출력하고, 선택된 값에 따라 객체를 생성하여 print() 호출 (1~4가 입력되지 않은 경우 오류 메시지 출력 후 종료)

```
class Person { }
class Student extends Person { }
class Researcher extends Person { }
class Professor extends Researcher { }

public class InstanceOfEx {
   static void print(Person p) {
      if(p instanceof Person)
         System.out.print("Person ");
      if(p instanceof Student)
         System.out.print("Student ");
      if(p instanceof Researcher)
         System.out.print("Researcher ");
      if(p instanceof Professor)
         System.out.print("Professor ");
      System.out.println();
   }
   public static void main(String[] args) {
      System.out.print("new Student() ->₩t"); print(new Student());
      System.out.print("new Researcher() ->₩t"); print(new Researcher());
      System.out.print("new Professor() ->₩t"); print(new Professor());
   }
}
```

예제 5-5 : 메소드 오버라이딩으로 다형성 실현


```
class Shape { // 슈퍼 클래스
    public Shape next;
    public Shape() { next = null; }

    public void draw() {
        System.out.println("Shape");
    }
}

class Line extends Shape {
    public void draw() { // 메소드 오버라이딩
        System.out.println("Line");
    }
}

class Rect extends Shape {
    public void draw() { // 메소드 오버라이딩
        System.out.println("Rect");
    }
}

class Circle extends Shape {
    public void draw() { // 메소드 오버라이딩
        System.out.println("Circle");
    }
}
```

```
public class MethodOverridingEx {
    static void paint(Shape p) {
        p.draw(); // p가 가리키는 객체 내에 오버라이딩된 draw() 호출.
                  // 동적 바인딩
    }

    public static void main(String[] args) {
        Line line = new Line();
        paint(line);
        paint(new Shape());
        paint(new Line());
        paint(new Rect());
        paint(new Circle());
    }
}
```

오버라이딩 활용

```
// 예제 5-5의 Shape, Line, Rect, Circle 클래스 활용
public class UsingOverride {
    public static void main(String [] args) {
        Shape start, last, obj;
        // 링크드 리스트로 도형 생성하여 연결
        start = new Line(); // Line 객체 연결
        last = start;
        obj = new Rect();
        last.next = obj; // Rect 객체 연결
        last = obj;
        obj = new Line(); // Line 객체 연결
        last.next = obj;
        last = obj;
        obj = new Circle(); // Circle 객체 연결
        last.next = obj;
        // 모든 도형 출력
        Shape p = start;
        while(p != null) {
            p.draw();
            p = p.next;
        }
    }
}
```

➔ 위 프로그램을 수정하여 다음과 같은 프로그램을 작성한다.

1) "1. Line 2. Circle 3. Rect" 라는 메뉴를 출력하고 1~3 중에 선택하여 입력하도록 함
2) 입력 된 값이 1~3이 아닌 경우 입력을 다시 하도록 함
3) 입력된 번호에 따라 객체를 생성하여 linked list에 연결 (예를 들어, 1, 3, 2 순서로 입력한다면 Line, Rect, Circle 순서로 객체 연결)
4) 1)~3)을 4회 반복 후 모든 도형에 대하여 출력 (위 프로그램의 while 문 부분 사용)

예제 5-6 : 메소드 오버라이딩

게임에서 무기를 표현하는 Weapon 클래스를 만들고 살상능력을 리턴하는 fire() 메소드를 작성하면 다음과 같다. fire()은 1을 리턴한다.

```
class Weapon {
    protected int fire() {
        return 1; // 무기는 기본적으로 한 명만 살상
    }
}
```

대포를 구현하기 위해 Weapon을 상속받는 Cannon 클래스를 작성하라. Cannon은 살상능력이 10이다. fire() 메소드를 이에 맞게 오버라이딩하라. main()을 작성하여 오버라이딩을 테스트하라.

```
class Cannon extends Weapon {
    @Override
    protected int fire() { // 오버라이딩
        return 10; // 대포는 한 번에 10명을 살상
    }
}
```

```
public class Overriding {
    public static void main(String[] args) {
        Weapon weapon;
        weapon = new Weapon();
        System.out.println("기본 무기의 살상 능력은 " +
                                    weapon.fire());
        weapon = new Cannon();
        System.out.println("대포의 살상 능력은 " +
                                    weapon.fire());
    }
}
```

```
기본 무기의 살상 능력은 1
대포의 살상 능력은 10
```