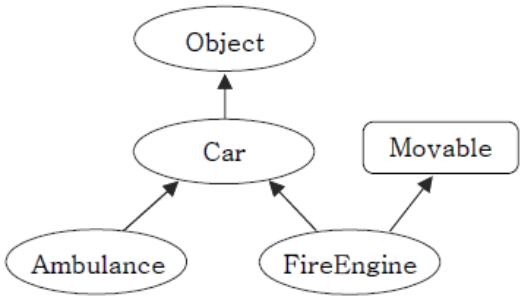
[해설] instanceof연산자는 실제 인스턴스의 모든 조상이나 구현한 인터페이스에 대해 true를 반환한다. 그래서, 아래 그림에서 알 수 있듯이 FireEngine인스턴스는 Object, Car, Movable, FireEngine타입에 대해 instanceof연산을 하면 결과로 true를 얻는다. 어떤 타입에 대해 instanceof연산결과가 true라는 것은 그 타입으로 형변환이 가능하다는 것을 뜻한다. 참조변수의 형변환을 하기에 앞서 instanceof연산자로 형변환이 가능한지 미리 확인해 보는 것이 좋다.



```java
class Robot {}
class DanceRobot extends Robot {
    void dance() {
        System.out.println("춤을 춥니다.");
    }
}
class SingRobot extends Robot {
    void sing() {
        System.out.println("노래를 합니다.");
    }
}
class DrawRobot extends Robot {
    void draw() {
        System.out.println("그림을 그립니다.");
    }
}

class Exercise7_18 {
    static void action(Robot r) {
        if (r instanceof DanceRobot) {
            DanceRobot da = (DanceRobot) r;
            da.dance();
        } else if (r instanceof SingRobot) {
            SingRobot s = (SingRobot) r;
            s.sing();
        } else if (r instanceof DrawRobot) {
            DrawRobot dr = (DrawRobot) r;
            dr.draw();
        }
    }

    public static void main(String[] args) {
        Robot[] arr = { new DanceRobot(), new SingRobot(), new DrawRobot()};

        for(int i=0; i< arr.length;i++) {
            action(arr[i]);
        }
    } // main
}
```

[해설] action메서드의 매개변수가 Robot타입이므로 Robot클래스의 자손클래스인 DanceRobot, SingRobot, DrawRobot의 인스턴스는 모두 매개변수로 가능하다.

```
Robot[] arr = { new DanceRobot(), new SingRobot(), new DrawRobot()};

for(int i=0; i< arr.length;i++)
    action(arr[i]);
```

action메서드 내에서는 실제로 받아온 인스턴스가 어떤 것인지 알 수 없다. 단지 Robot클래스 또는 그 자손클래스의 인스턴스일 것이라는 것만 알 수 있다. 그래서 instanceof연산자를 이용해야만 실제 인스턴스의 타입을 확인할 수 있다.

```
public static void action(Robot r) {
    if(r instanceof DanceRobot) {
        DanceRobot dr = (DanceRobot)r;
        dr.dance();
    } else if(r instanceof SingRobot) {
        SingRobot sr = (SingRobot)r;
        sr.sing();
    } else if(r instanceof DrawRobot) {
        DrawRobot dr = (DrawRobot)r;
        dr.draw();
    }
}
```

```
class Product {
    int price; // 제품의 가격

    Product(int price) {
        this.price = price;
    }
}

class Tv extends Product {
    Tv() {
        super(100);
    }

    public String toString() {
        return "Tv";
    }
}

class Computer extends Product {
    Computer() {
        super(200);
    }

    public String toString() {
        return "Computer";
    }
}

class Audio extends Product {
    Audio() {
        super(50);
    }
```

```
    public String toString() {
        return "Audio";
    }
}

class Buyer {
    int money = 1000;
    Product[] cart = new Product[3]; // 구입한 제품을 저장하기 위한 배열
    int i = 0; // Product배열 cart에 사용될 index

    void add(Product p) {
        /* (2) 아래의 로직에 맞게 코드를 작성하시오.
        1.1 i의 값이 장바구니의 크기보다 같거나 크면
        1.1.1 기존의 장바구니보다 2배 큰 새로운 배열을 생성한다.
        1.1.2 기존의 장바구니의 내용을 새로운 배열에 복사한다.
        1.1.3 새로운 장바구니와 기존의 장바구니를 바꾼다.
        1.2 물건을 장바구니(cart)에 저장한다. 그리고 i의 값을 1 증가시킨다. */
        if (i >= cart.length) {
            Product[] big_cart = new Product[cart.length * 2];
            System.arraycopy(cart, 0 , big_cart, 0, cart.length);
            cart = big_cart;
        }
        cart[i++] = p;
    } // add(Product p)

    void buy(Product p) {
        /* (1) 아래의 로직에 맞게 코드를 작성하시오.
        1.1 가진 돈과 물건의 가격을 비교해서 가진 돈이 적으면 메서드를 종료한다.
        1.2 가진 돈이 충분하면, 제품의 가격을 가진 돈에서 빼고
        1.3 장바구니에 구입한 물건을 담는다.(add메서드 호출) */
        if (money < p.price) {
            System.out.println("잔액이 부족하여 " + p + "을/를 살 수 없습니다.");
            return;
        } else {
            money -= p.price;
            add(p);
        }
    }

    void summary() {
        /* (3) 아래의 로직에 맞게 코드를 작성하시오.
        1.1 장바구니에 담긴 물건들의 목록을 만들어 출력한다.
        1.2 장바구니에 담긴 물건들의 가격을 모두 더해서 출력한다.
        1.3 물건을 사고 남은 금액(money)를 출력한다.*/

        String itemList = "";
        int sum = 0;

        for (int i = 0; i < cart.length; i++) {
            if (cart[i] == null) {
                break;
            }
            itemList += cart[i] + ",";
            sum += cart[i].price;
        }
        System.out.println("구입한 물건:" + itemList);
        System.out.println("사용한 금액:" + sum);
        System.out.println("남은 금액:" + money);
```

```
//         System.out.print("구입한 물건:");
//         for (int i = 0; i < cart.length; i++) {
//             System.out.print(cart[i]+",");
//         }
//         System.out.println();
//         System.out.print("사용한 금액:");
//         int sum = 0;
//         for (int i = 0; i < cart.length; i++) {
//             sum += cart[i].price;
//         }
//         System.out.print(sum);
//         System.out.println();
//         System.out.print("남은 금액:");
//         System.out.println(money);
    } // summary()
}

class Exercise7_19 {
    public static void main(String args[]) {
        Buyer b = new Buyer();
        b.buy(new Tv());
        b.buy(new Computer());
        b.buy(new Tv());
        b.buy(new Audio());
        b.buy(new Computer());
        b.buy(new Computer());
        b.buy(new Computer());
        b.summary();
    }
}
```

[7-20] 다음의 코드를 실행한 결과를 적으시오.

[연습문제]/ch7/Exercise7_20.java

```
class Exercise7_20 {
  public static void main(String[] args) {
      Parent p = new Child();
      Child  c = new Child();

      System.out.println("p.x = " + p.x);
      p.method();

      System.out.println("c.x = " + c.x);
      c.method();
  }
}

class Parent {
  int x = 100;

  void method() {
      System.out.println("Parent Method");
  }
}

class Child extends Parent {
  int x = 200;

  void method() {
      System.out.println("Child Method");
  }
}
```

[정답]

[실행결과]

```
p.x = 100
Child Method
c.x = 200
Child Method
```

[해설] 조상 클래스에 선언된 멤버변수와 같은 이름의 인스턴스변수를 자손 클래스에 중복으로 정의했을 때, 조상타입의 참조변수로 자손 인스턴스를 참조하는 경우와 자손타입의 참조변수로 자손 인스턴스를 참조하는 경우는 서로 다른 결과를 얻는다.

메서드의 경우 조상 클래스의 메서드를 자손의 클래스에서 오버라이딩한 경우에도 참조변수의 타입에 관계없이 항상 실제 인스턴스의 메서드(오버라이딩된 메서드)가 호출되지만, 멤버변수의 경우 참조변수의 타입에 따라 달라진다.

[7-21] 다음과 같이 attack메서드가 정의되어 있을 때, 이 메서드의 매개변수로 가능한 것 두 가지를 적으시오.

```
interface Movable {
  void move(int x, int y);
}

void attack(Movable f) {
  /* 내용 생략 */
}
```

[정답] null, Movable인터페이스를 구현한 클래스 또는 그 자손의 인스턴스
[해설] 매개변수의 다형성을 잘 이해하고 있는지를 확인하는 문제이다. 매개변수의 타입이 인터페이스라는 것은 어떤 의미일지 이해하지 못하는 경우가 많은데, 이것을 이해하는 것은 매우 중요하다.

```
// Shape클래스를 조상으로 하는 Circle클래스와 Rectangle클래스 작성하기
// 생성자도 각 클래스에 맞게 적절히 추가해야 한다
```

```java
class Point {
    int x;
    int y;

    Point(int x, int y) {
        this.x=x;
        this.y=y;
    }

    Point() {
        this(0,0);
    }

    public String toString() {
        return "["+x+","+y+"]";
    }
}

abstract class Shape {
    Point p;

    Shape(Point p) {
        this.p = p;
    }

    Shape() {
        this(new Point(0,0));
    }

    abstract double calcArea(); // 도형의 면적을 계산해서 반환하는 메서드

    Point getPosition() {
        return p;
    }

    void setPosition(Point p) {
        this.p = p;
    }

}

class Circle extends Shape {
    private double r;

    Circle(Point p, double r) {
        super(p);       // 조상의 멤버는 조상의 생성자가 초기화하도록 한다
        this.r = r;
    }

    Circle(double r) {
        this(new Point(0, 0), r);
    }

    double calcArea() {
        return r * r * Math.PI;
    }
}
```

```java
class Rectangle extends Shape {
    private double width;
    private double height;

    Rectangle(Point p, double width, double height) {
        super(p);       // 조상의 멤버는 조상의 생성자가 초기화하도록 한다
        this.width = width;
        this.height = height;
    }
    Rectangle(double width, double height) {
        this(new Point(0, 0), width, height);
    }

    double calcArea() {
        return width * height;
    }

    boolean isSquare() {
        // width나 height가 0이 아니고 width와 height가 같으면 true 반환
       return width * height != 0 && width == height;
    }

}

class Exercise7_23 {
    // 주어진 배열에 담긴 도형들의 넓이를 모두 더해서 반환하는 메서드 작성
    static double sumArea(Shape[] arr) {
        double sum = 0;

        for (int i = 0; i < arr.length; i++) {
            sum += arr[i].calcArea();
        }

        return sum;
    }

    public static void main(String[] args) {
        Shape[] arr = {new Circle(5.0), new Rectangle(3,4), new Circle(1)};
        System.out.println("면접의 합:" + sumArea(arr));
    }
}
```

```java
class Outer {
    class Inner {
        int iv=100;
    }
}
class Exercise7_25 {
    public static void main(String[] args) {
        Outer outer = new Outer(); // 외부 클래스의 인스턴스를 먼저 생성해야 한다.
        Outer.Inner inner = outer.new Inner(); // 인스턴스 내부 클래스의 인스턴스를 생성하려
        System.out.println(inner.iv);

    }
}
```

[해설] 내부 클래스(인스턴스 클래스)의 인스턴스를 생성하기 위해서는 먼저 외부클래스의 인스턴스를 생성해야한다. 왜냐하면 '인스턴스 클래스'는 외부 클래스의 '인스턴스 변수'처럼 외부 클래스의 인스턴스가 생성되어야 쓸 수 있기 때문이다.

```
class Outer {
    static class Inner {
        int iv=200;
    }
}
class Exercise7_26 {
    public static void main(String[] args) {
        Outer.Inner inner = new Outer.Inner();
        // 스태틱 내부 클래스의 인스턴스는 외부 클래스를 먼저 생성하지 않아도 된다.
        // (하지만 내부 클래스의 인스턴스를 만드는 이유는 iv가 내부 클래스의
        // 인스턴스 변수이기 때문이다.) 따라서 내부 클래스의 인스턴스만 생성하면 된다.
        System.out.println(inner.iv);
    }
}
```

[해설] 스태틱 클래스(static inner class)는 인스턴스 클래스와 달리 외부 클래스의 인스턴스를 생성하지 않고도 사용할 수 있다. 마치 static멤버를 인스턴스 생성없이 사용할 수 있는 것처럼.

```
class Outer {
    int value=10;
    class Inner {
        int value=20;
        void method1() {
            int value=30;
            System.out.println(value);          // 30
            System.out.println(this.value);     // 20
            System.out.println(Outer.this.value);   // 10
        }
    } // Inner클래스의 끝
} // Outer클래스의 끝
class Exercise7_27 {
    public static void main(String args[]) {
        Outer outer = new Outer();
        Outer.Inner inner = outer.new Inner();
        inner.method1();
    }
}
```

[해설] 외부 클래스와 내부 클래스에 같은 이름의 인스턴스 변수(value)가 선언되었을 때 어떻게 구별하는가에 대한 문제이다. 외부 클래스의 인스턴스 변수는 내부 클래스에서 '외부클래스이름.this.변수이름' 로 접근할 수 있다.

[7-28] 아래의 EventHandler를 익명 클래스(anonymous class)로 변경하시오.

[연습문제]/ch7/Exercise7_28.java

```
import java.awt.*;
import java.awt.event.*;

class Exercise7_28
{
   public static void main(String[] args)
   {
       Frame f = new Frame();
       f.addWindowListener(new EventHandler());
   }
}

class EventHandler extends WindowAdapter
{
   public void windowClosing(WindowEvent e) {
       e.getWindow().setVisible(false);
       e.getWindow().dispose();
       System.exit(0);
   }
}
```

[정답]

[연습문제]/ch10/Exercise7_28_2.java

```
import java.awt.*;
import java.awt.event.*;

class Exercise7_28_2
{
   public static void main(String[] args)
   {
       Frame f = new Frame();
       f.addWindowListener(new WindowAdapter() {
           public void windowClosing(WindowEvent e) {
               e.getWindow().setVisible(false);
               e.getWindow().dispose();
               System.exit(0);
           }
       });
   } // main
}
```

[7-29] 지역 클래스에서 외부 클래스의 인스턴스멤버와 static멤버에 모두 접근할 수 있지만, 지역변수는 final이 붙은 상수만 접근할 수 있는 이유| 무엇인가?

[정답] 메서드가 수행을 마쳐서 지역변수가 소멸된 시점에도, 지역 클래스의 인스턴스가 소멸된 지역변수를 참조하려는 경우가 발생할 수 있기 때문이다.

[해설] 아직 쓰레드를 배우지 않았지만, 쓰레드를 사용해서 상황을 만들어 보았다.

[연습문제]/ch10/Exercise7_29.java

```
import java.awt.*;
import java.awt.event.*;

class Exercise7_29
{
  public static void main(String[] args)
  {
      final int VALUE = 10; // 외부 클래스의 지역변수

      Thread t = new Thread(new Runnable() { // 익명 클래스(내부 클래스)
          public void run() {
              for(int i=0; i < 10;i++) { // 10번 반복한다.
                  try {
                      Thread.sleep(1*1000); // 1초간 멈춘다.
                  } catch(InterruptedException e) {}

                  System.out.println(VALUE); // 외부 클래스의 지역변수를 사용
              }
          } // run()
      });

      t.start(); // 쓰레드를 시작한다.
      System.out.println("main() - 종료.");
  } // main
}
```

[실행결과]

```
main() - 종료.
10
10
10
10
10
10
10
10
10
10
```

실행결과를 보면 main메서드가 종료된 후에도 지역변수 VALUE의 값을 사용하고 있다는 것을 알 수 있다. 지역변수는 메서드가 종료되면 함께 사라지지만, 상수의 경우 이미 컨스턴트 풀(constant pool, 상수를 따로 모아서 저장해 놓는 곳)에 저장되어 있기 때문에 사용할 수 있는 것이다.

Resources: 자바의 정석