

Java프로그래밍

4강. 클래스와 상속 (교재 3장)

컴퓨터과학과 김희천 교수

오늘의 **학습목차**

1. 클래스 정의와 사용
2. 상속

1. 클래스 정의와 사용

1. 클래스 정의와 사용

Java프로그래밍

4강. 클래스와 상속

1) 메소드 정의

- ◆ 클래스 정의 내부에 존재함
- ◆ 헤더와 몸체로 구성됨

메소드 정의 문법

[접근 제어자] 반환형 메소드이름([자료형 인자[, 자료형 인자...]])
[throws 예외이름]

```
{  
    문장 ...  
}
```

```
public double getArea( ) {  
    return radius * radius * PI;  
}  
public void setRadius(int r) {  
    radius = r;  
}
```

1. 클래스 정의와 사용

2) 생성자

- ◆ 객체가 생성될 때 자동으로 실행되는 메소드
 - ✓ 객체의 필드 값을 초기화하거나 메모리 할당 등의 작업
- ◆ 객체 생성 방법은 **new** 클래스이름(인자...)
- ◆ 예
 - ✓ `Circle c = new Circle(5);`
 - new 연산자를 이용하여 객체를 생성(메모리 할당)하고
 - 생성자가 호출(데이터 필드의 초기화)되면서
 - 객체의 참조값을 변수에 대입(=)

1. 클래스 정의와 사용

3) 생성자 정의(1)

◆ 보통의 메소드와 정의 방법이 다름

✓ 생성자는 new로 객체를 생성할 때 자동 호출됨

정의 방법

◆ 생성자 이름은 클래스 이름과 같음

◆ 반환형을 선언하지 않음

◆ 여러 생성자를 정의할 수 있음(생성자 오버로딩)

✓ 인자의 개수와 인자의 자료형으로 구분

◆ 접근 제어자는 보통 public

1. 클래스 정의와 사용

3) 생성자 정의(2)

Java프로그래밍

4강. 클래스와 상속

```
class Circle {  
    double r;  
    public Circle(double a) {  
        r = a;  
    }  
    public double getArea( ) {  
        return r * r * 3.14;  
    }  
}
```

```
public class CircleArea2 {  
    public static void main(String args[ ])  
    {  
        Circle c = new Circle(5.0);  
        System.out.println(c.r);  
        System.out.println(c.getArea( ));  
    }  
}
```

4) 기본 생성자

- ◆ 인자가 없는 생성자, 디폴트 생성자(default constructor)
- ◆ 클래스 정의에 한 개의 생성자 정의도 없으면 컴파일러가 다음과 같은 것을 자동으로 만들어 줌
 - ✓ `public Circle() { }`
 - ✓ 생성자 몸체의 첫 줄에 부모 생성자의 명시적 호출이 없다면 다음 코드가 자동으로 들어감
 - `super();` //부모 클래스의 기본 생성자를 호출
 - 따라서 부모 클래스에서 기본 생성자의 존재를 확인해야 함

1. 클래스 정의와 사용

5) 클래스의 사용

◆ 상속을 위해 사용하는 것

✓ `class CSub extends CSuper { ... }`

◆ 변수를 선언하고 객체를 생성

✓ 클래스 참조형 변수의 선언

- `Circle c;`


- `c = new Circle(5);`

6) 객체의 사용

◆ 객체 변수와 점(.) 연산자를 사용하여 멤버에 접근

- ✓ 객체가 소유하는 데이터(인스턴스 변수)를 읽거나 쓰기
- ✓ 객체를 이용하여 메소드(인스턴스 메소드)를 호출하기
- ✓ 예

- `c.r = 5;` //c의 r을 변경
- `c.getArea()` //c에게 `getArea()` 실행을 요청



```
public double getArea() {  
    return this.r * this.r * 3.14;  
}
```

1. 클래스 정의와 사용

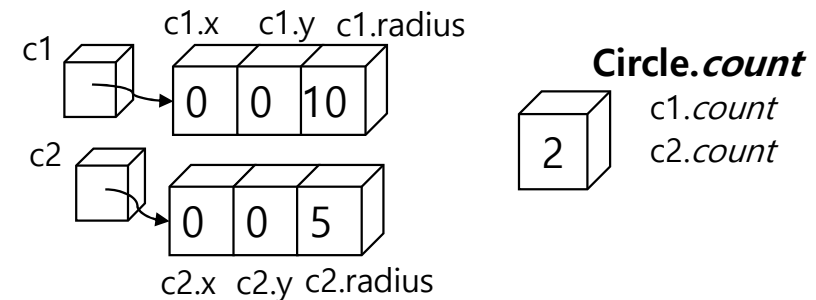
7) static 필드

Java프로그래밍

4강. 클래스와 상속

◆ static 필드

- ✓ 정적 필드 or 클래스 변수
- ✓ 클래스의 모든 객체가 공유하는 데이터
 - 객체의 생성이 없어도 항상 사용 가능
 - 어떤 객체도 값을 변경할 수 있음
- ✓ 사용 방법은 클래스이름.정적필드
 - 객체변수.정적필드도 가능



1. 클래스 정의와 사용

8) static 메소드

◆ static 메소드

- ✓ 정적 메소드 or 클래스 메소드
- ✓ 객체와 무관하게 호출되고 실행됨
 - 메소드 몸체에서 this를 사용할 수 없음
- ✓ static 필드와 인자를 가지고 작업함
- ✓ 사용 방법은 클래스이름.정적메소드()
 - Math.sqrt(2.0);
 - Integer.parseInt("120");

1. 클래스 정의와 사용

9) final 필드와 final 메소드

◆ final 필드

- ✓ 상수 데이터를 선언
- ✓ 선언할 때 초기값을 지정해야 함
- ✓ 자주 static과 함께 사용됨
 - `final static double PI = 3.141592;`

◆ final 메소드

- ✓ 자식 클래스로 상속은 가능하나 재정의 할 수 없는 메소드

10) 객체 초기화(1)

- ◆ 객체를 생성할 때, 데이터 필드에 초기값을 지정하는 것
 - ✓ 클래스 변수는 프로그램 시작 시에 자동 초기화됨
- ◆ 데이터 필드는 자동으로 초기값이 주어질 수 있음

방법(순서)

- ◆ static 필드의 선언문에서 초기화
- ◆ static 초기화 블록 실행
- ◆ non-static 필드의 선언문에서 초기화
- ◆ non-static 초기화 블록 실행
 - ✓ 클래스 몸체 내 임의 위치에 포함
 - ✓ 초기값 지정에 위한 코드
 - ✓ static 필드는 static 블록을 사용
- ◆ 생성자 실행

```
class IniTest {  
    int nValue = 1;  
    {  
        nValue = 2;  
    }  
    public IniTest( ) {  
        nValue = 3;  
    }  
}
```

1. 클래스 정의와 사용

10) 객체 초기화(2)

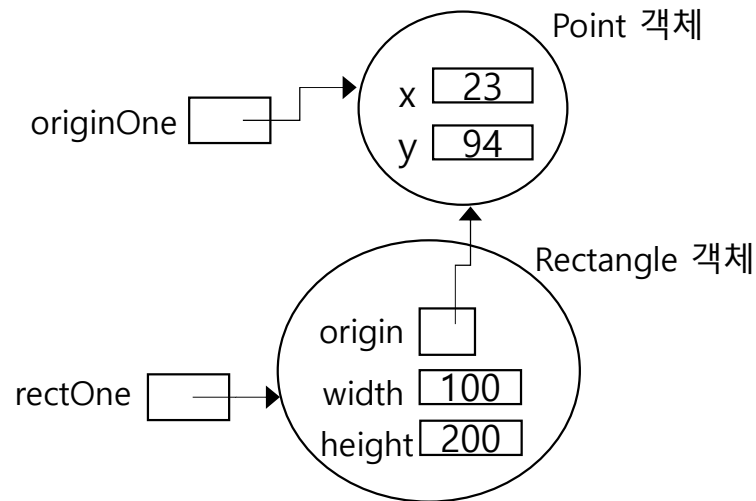
```
import java.awt.Point;

class Rectangle {
    public int width = 0;
    public int height = 0;
    public Point origin;

    public Rectangle( ) {
        origin = new Point(0, 0);
    }

    public Rectangle(Point p, int w, int h) {
        origin = p;
        width = w;
        height = h;
    }
}
```

```
public class Test {
    public static void main(String args[ ]) {
        Point originOne = new Point(23, 94);
        Rectangle rectOne =
            new Rectangle(originOne, 100, 200);
    }
}
```



11) 메소드 오버로딩

- ◆ 인자의 개수나 인자의 자료형이 다르면
같은 이름의 메소드를 한 클래스에서 여러 개 정의할 수 있음
 - ✓ 인자의 개수와 자료형이 정확히 일치하면 중복 정의 불가
- ◆ 메소드를 호출할 때, 가장 가까운 매개변수 목록을 가진 메소드가 호출됨
- ◆ 예
 - ✓ `System.out.println();` //인자 없음
 - ✓ `System.out.println("문자열");` //인자는 String
 - ✓ `System.out.println(241);` //인자는 int
 - ✓ `System.out.println(34.5);` //인자는 double
 - ✓ `mc.add(10, 10.5);` //add(int, int) 와 add(double, double)

1. 클래스 정의와 사용

Java프로그래밍

4강. 클래스와 상속

12) 클래스와 객체의 사용 예

```
class Cylinder {  
    private Circle c;    //원  
    private int h;       //높이  
    public Cylinder(Circle a, int b) {  
        c = a;  
        h = b;  
    }  
  
    public double getVolume( ) {  
        return c.getArea( ) * h;  
    }  
}
```

```
public class CylinderVolume {  
    public static void main(String args[]) {  
        Circle c = new Circle(7);  
        int h = 8;  
        Cylinder cy = new Cylinder(c, h);  
        System.out.println(cy.getVolume( ));  
    }  
}
```

2. 상속

2. 상속

1) 클래스의 재사용

◆ 합성

- ✓ 기존 클래스를 새로운 클래스에서 데이터 필드의 자료형으로 사용
- ✓ "has-a" 관계
- ✓ `class Line { Point begin, end; ... }`

◆ 상속

- ✓ 기존 클래스(부모)를 사용하여 새로운 클래스(자식)를 정의
- ✓ 코드의 중복 작성을 줄이고 프로그램의 확장성이 좋아짐
- ✓ 상속은 기존 클래스를 확장 or 특화하는 것
- ✓ 자식 "is-a" 부모의 관계

2) 클래스의 상속(1)

- ◆ 상속은 부모 클래스와 자식 클래스 간의 관계
 - ✓ 자식 클래스가 부모 클래스의 필드와 메소드를 상속 받음
 - ✓ 기존 클래스를 상속받을 때 키워드 `extends`를 사용함
 - ✓ 예: `class Manager extends Employee { ... }`
 - ✓ 자식 클래스에서 상속받은 메소드를 오버라이딩할 수 있음
- ◆ 클래스의 상속은 단일 상속만 가능
 - ✓ 인터페이스 상속의 경우는 다중 상속 가능

2. 상속

2) 클래스의 상속(2)

```
class CSuper {  
    private int f1;  
    public int f2;  
    public void setPrivate( ) { f1 = 10; }  
    public void setPublic( ) { f2 = 20; }  
    private void mPrivate( ) { f1 = 30; }  
}  
  
class CSub extends CSuper {  
    private int f3;  
    public int f4;  
}
```

```
public class InheritTest {  
    public static void main(String args[ ]) {  
        CSub sub = new CSub( );  
        sub.f1 = 40; // 오류  
        sub.f2 = 50;  
        sub.f3 = 60; // 오류  
        sub.f4 = 70;  
        sub.setPrivate( );  
        sub.setPublic( );  
        sub.mPrivate( ); // 오류  
    }  
}
```

3) 메소드 오버라이딩(1)

- ◆ 부모로부터 상속받은 메소드의 몸체를 자식 클래스에서 재정의하는 것
- ◆ 부모와 자식에서 같은 이름의 메소드가 다른 기능을 수행하게 됨

방법

- ◆ 메소드의 이름, 인자의 개수와 자료형, 반환형이 같은 메소드를 정의
- ◆ 반환형은 서브 타입(상속 관계에서 자식 클래스)도 가능함
- ◆ 접근 제어자의 가시성(접근 범위)은 같거나 커져야 함
 - ✓ protected인 경우 protected 또는 public
 - ✓ public인 경우 public만 가능

2. 상속

3) 메소드 오버라이딩(2)

```
class Shape {  
    public double getArea(double h, double w) { return h * w; }  
}  
  
class Triangle extends Shape {  
    public double getArea(double h, double w) { return h * w * 0.5; }  
}  
  
public class OverridingTest {  
    public static void main(String args[ ]) {  
        Triangle t = new Triangle( );  
        System.out.println( t.getArea(3.0, 4.0) );  
    }  
}
```

2. 상속

4) this

- ◆ 메소드 호출 시, 숨은 인자로 this가 메소드에 전달됨
 - ✓ this는 현재 객체에 대한 참조값을 가지고 있음
 - ✓ c1.display()과 c2.display()의 결과가 다른 이유임
- ◆ 인스턴스 메소드나 생성자에서만 사용 가능

```
class Circle {  
    static double PI = 3.14;  
    double radius;  
    ... ..  
    public double getArea( ) {  
        return this.radius * this.radius * PI;  
    }  
    public void display( ) {  
        System.out.println("반지름:" + this.radius + " 면적:" +  
this.getArea( ));  
    }  
}
```


5) super

- ◆ this와 같으나 자료형이 부모 클래스 유형임
- ◆ 자식 클래스의 인스턴스 메소드나 생성자에서 사용됨
 - ✓ this와 마찬가지로 static 메소드에서 사용할 수 없음
- ◆ 부모 클래스에서 오버로딩 당한 메소드를 호출하거나 상속되었으나 감춰진 필드에 접근할 때 필요함
 - ✓ super.메소드(인자)
 - ✓ super.필드

2. 상속

6) this와 super의 사용 예

```
class CSuper {  
    public double x;  
}  
class CSub extends CSuper {  
    public double x;  
  
    public CSub(double new_x) {  
        this.x = new_x;  
        super.x = new_x * 10;  
    }  
  
    public double getSuper() {  
        return super.x;  
    }  
  
    public double getSub() {  
        return this.x;  
    }  
}
```

```
public class ThisSuperTest {  
    public static void main(String args[]) {  
        CSub sub = new CSub(10.0);  
        System.out.println(sub.x);  
  
        System.out.println(sub.getSuper());  
  
        System.out.println(sub.getSub());  
    }  
}
```

7) this()와 super()

◆ this()

- ✓ 같은 클래스의 다른 생성자를 호출하는 것

◆ super()

- ✓ 부모 클래스의 생성자를 호출하는 것
- ✓ 상속받은 데이터 필드를 초기화하기 위한 것
- ✓ 생성자 몸체에서 부모 클래스 생성자의 명시적 호출이 없다면, 인자가 없는 생성자인 super()가 자동 호출됨

◆ 둘 다 생성자 몸체의 첫 번째 문장에서만 사용 가능함

2. 상속

8) super와 super()의 사용 예

```
public class Cylinder extends Circle {
    private double height;

    public Cylinder( ) {
        super( );
        height = 1.0;
    }
    public Cylinder(double radius, double h) {
        super(radius);
        this.height = h;
    }

    public double getHeight( ) {
        return height;
    }
    public void setHeight(double h) {
        this.height = h;
    }
}
```

```
    public double getArea( ) {
        return 2 * PI * getRadius( ) * height +
        2 * super.getArea( );
    }

    public double getVolume( ) {
        return super.getArea( ) * height;
    }

    public String toString( ) {
        return "Cylinder of radius = " +
        getRadius( ) + " height = " + height;
    }
}
```

Java프로그래밍
다음시간안내

5강. 인터페이스와 다형성