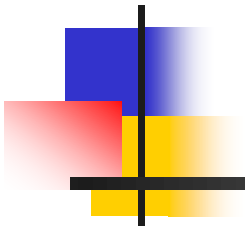
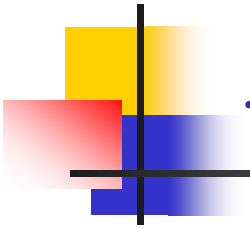


java 7강-오버라이딩

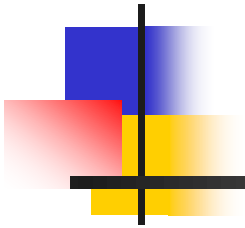
양 명 숙
[now4ever7@gmail.com]



목차

- JVM의 메모리 구조

- 오버라이딩(Overriding)
- this/super
- final
- 클래스간의 관계-포함관계(Has a)

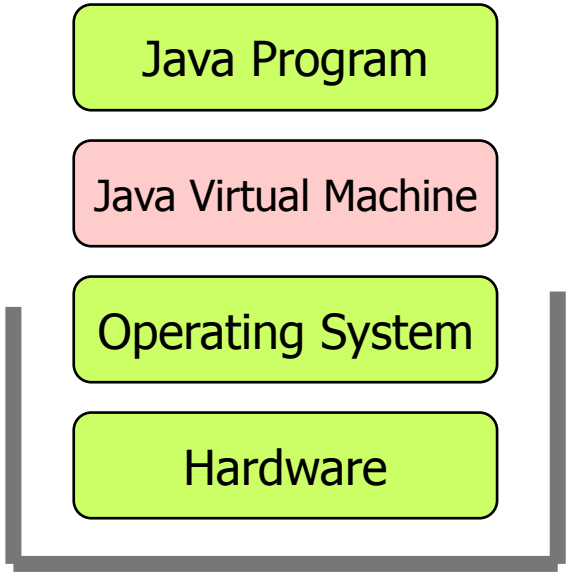


JVM의 메모리 구조

자바 가상머신의 메모리 모델

- 자바 가상머신은 운영체제 위에서 실행되는 하나의 프로그램이다
- 자바 프로그램은 자바 가상머신 위에서 실행되는 프로그램이다

- 자바 가상머신의 실행에 필요한 메모리는 운영체제가 할당해줌
- 프로그램의 실행에 필요한 메모리 => 메인 메모리, 물리적으로 램(RAM)을 의미함
- 운영체제가 메인 메모리를 관리함
- => **운영체제가 응용 프로그램에게 메모리를 할당해줌**
- 자바 가상 머신은 운영체제가 할당해 주는 메모리 공간을 기반으로 자기 자신(가상머신)도 실행을 하면서, 자바 응용 프로그램의 실행도 돕는다



자바 가상머신의 메모리 모델

- 자바 가상머신은 운영체제로부터 할당 받은 메모리 공간의 효율적인 사용을 위해 수납장처럼 메모리 공간을 나눠서 데이터의 특성에 따라 분류해서 저장함
  - 데이터의 성격이 다르면 별도의 메모리 공간에 저장을 해야 관리가 용이함
- 자바 가상머신도 자바 프로그램의 실행을 위해서 메모리 관리를 해야 함
- 자바 가상머신은 메모리 공간을 크게 세 개의 영역으로 나눔

1) 메서드 영역 - 메서드의 바이트 코드, static 변수
2) 스택 영역 - 지역변수, 매개변수
3) 힙 영역 - 인스턴스

JVM의 메모리 구조

- 응용 프로그램이 실행되면, JVM은 운영체제로부터 프로그램을 수행하는 데 필요한 메모리를 할당받고 JVM은 이 메모리를 용도에 따라 여러 영역으로 나누어 관리함
- 3가지 주요 영역
  - 1. 메서드 영역(method area)
    - 프로그램 실행 중 어떤 클래스가 사용되면, JVM은 해당 클래스의 클래스 파일(*.class)을 읽어서 분석하여 클래스에 대한 정보(클래스 데이터)를 이곳에 저장함
    - 이 때, 이 클래스의 클래스변수(static 변수)도 이 영역에 함께 생성됨
  - 2. 힙(heap)
    - 인스턴스가 생성되는 공간.
    - 프로그램 실행 중 생성되는 인스턴스는 모두 이곳에 생성됨
    - 즉, 인스턴스 변수들이 생성되는 공간임
  - 3. 호출 스택(call stack 또는 execution stack)
    - 메서드의 작업에 필요한 메모리 공간을 제공함
    - 메서드가 호출되면, 호출스택에 호출된 메서드를 위한 메모리가 할당되며, 이 메모리는 메서드가 작업을 수행하는 동안 지역변수(매개변수 포함)들과 연산의 중간 결과 등을 저장하는 데 사용됨
    - 메서드가 작업을 마치면 할당되었던 메모리 공간은 반환되어 비워짐

JVM의 메모리 구조

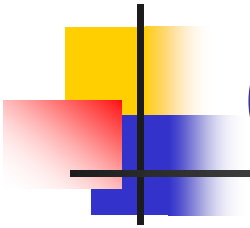
- 각 메서드를 위한 메모리상의 작업공간은 서로 구별되며, 첫 번째로 호출된 메서드를 위한 작업공간이 호출 스택의 맨 밑에 마련되고, 첫 번째 메서드 수행 중에 다른 메서드를 호출하게 되면, 첫 번째 메서드의 바로 위에 두 번째로 호출된 메서드를 위한 공간이 마련됨
- 이 때 첫 번째 메서드는 수행을 멈추고, 두 번째 메서드가 수행되기 시작함
- 두 번째로 호출된 메서드가 수행을 마치게 되면, 두 번째 메서드를 위해 제공되었던 호출 스택의 메모리 공간이 반환되며, 첫 번째 메서드는 다시 수행을 계속하게 됨
- 첫 번째 메서드가 수행을 마치면, 역시 제공되었던 메모리 공간이 호출 스택에서 제거되며 호출 스택은 완전히 비워지게 됨
- 호출 스택의 제일 상위에 위치하는 메서드가 현재 진행 중인 메서드이며, 나머지는 대기 상태에 있게 됨

JVM의 메모리 구조

- 호출스택의 특징

• 메서드가 호출되면 수행에 필요한 만큼의 메모리를 스택에 할당 받는다
• 메서드가 수행을 마치고 나면 사용했던 메모리를 반환하고 스택에서 제거됨
• **호출 스택의 제일 위에 있는 메서드가 현재 실행 중인 메서드임**
• 아래에 있는 메서드가 바로 위의 메서드를 호출한 메서드임

- 반환타입이 있는 메서드는 종료되면서 결과값을 자신을 호출한 메서드에게 반환함
- 대기상태에 있던 호출한 메서드는 넘겨받은 반환값으로 수행을 계속 진행하게 됨

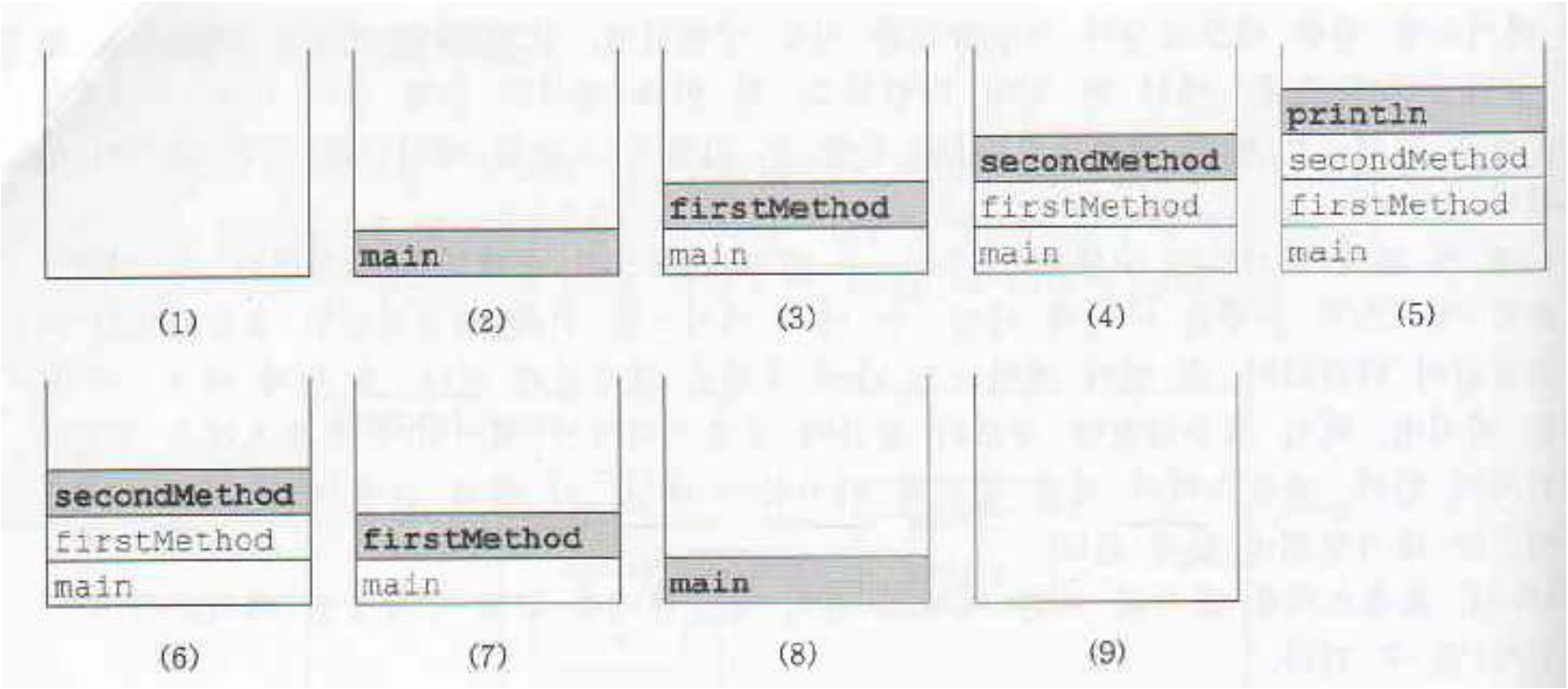


예제

```
class CallStackTest {
	public static void main(String[] args) {
		firstMethod();
	}

	static void firstMethod() {
		secondMethod();
	}

	static void secondMethod() {
		System.out.println("secondMethod()");
	}
}
```

호출스택의 변화

- 예제 실행시, 프로그램이 수행되는 동안 호출스택의 변화



호출스택의 변화

- (1)~(2) 예제를 컴파일한 후 실행시키면, JVM에 의해서 main 메서드가 호출됨으로써 프로그램이 시작됨
  - 이 때, 호출스택에는 main 메서드를 위한 메모리공간이 할당되고 main 메서드의 코드가 수행되기 시작함
- (3) main 메서드에서 firstMethod()를 호출한 상태
  - 아직 main메서드가 끝난 것은 아니므로 main 메서드는 호출스택에 대기상태로 남아있고 firstMethod()의 수행이 시작됨
- (4) firstMethod()에서 다시 secondMethod()를 호출했다
  - firstMethod()는 secondMethod()가 수행을 마칠 때까지 대기상태에 있게 됨.
  - secondMethod() 가 수행을 마쳐야 firstMethod()의 나머지 문장들을 수행할 수 있기 때문이다
- (5) secondMethod()에서 println() 메서드를 호출했다
  - println()메서드에 의해서 'secondMethod()'가 화면에 출력됨

호출스택의 변화

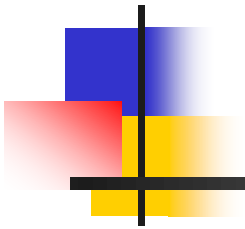
- (6) println() 메서드의 수행이 완료되어 호출스택에서 사라지고 자신을 호출한 secondMethod()로 되돌아감
  - 대기 중이던 secondMethod()는 println() 를 호출한 이후부터 수행을 재개함
- (7) secondMethod()에 더 이상 수행할 코드가 없으므로 종료되고, 자신을 호출한 firstMethod() 로 돌아감
- (8) firstMethod()에도 더 이상 수행할 코드가 없으므로 종료되고, 자신을 호출한 main 메서드로 돌아감
- (9) main 메서드에도 더 이상 수행할 코드가 없으므로 종료되어, 호출스택은 완전히 비워지게 되고, 프로그램은 종료됨

예제2

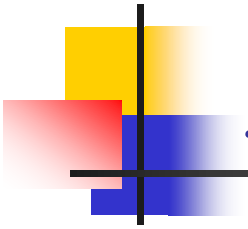
```
main(String[] args)이 시작되었음.
firstMethod()이 시작되었음.
secondMethod()이 시작되었음.
secondMethod()이 끝났음.
firstMethod()이 끝났음.
main(String[] args)이 끝났음.
```

```
class CallStackTest2 {
    public static void main(String[] args) {
        System.out.println("main(String[] args)이 시작되었음.");
        firstMethod();
        System.out.println("main(String[] args)이 끝났음.");
    }

    static void firstMethod() {
        System.out.println("firstMethod()이 시작되었음.");
        secondMethod();
        System.out.println("firstMethod()이 끝났음.");
    }

    static void secondMethod() {
        System.out.println("secondMethod()이 시작되었음.");
        System.out.println("secondMethod()이 끝났음.");
    }
}
```

앞의 예제에 출력문을 추가해서 각 메서드의 시작과 종료의 순서를 확인하는 예제



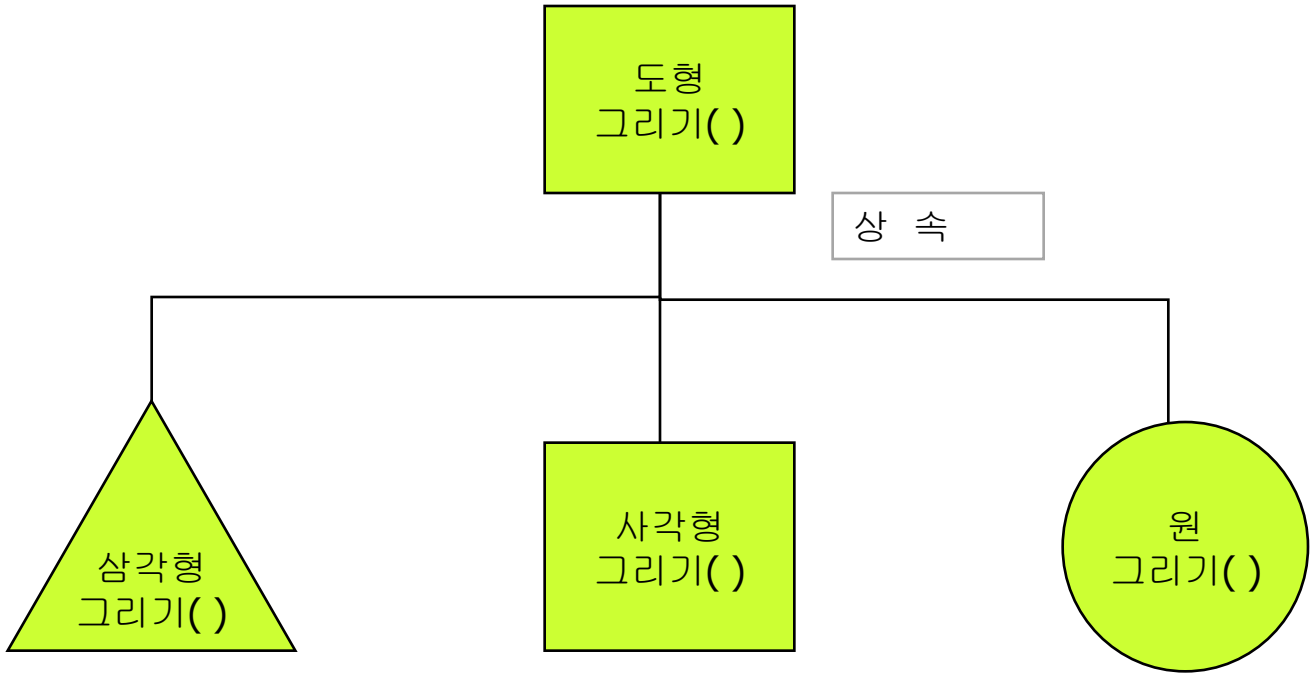
오버라이딩



오버라이딩(Overriding)

- 오버라이딩
  - 부모 클래스로부터 **상속받은 메서드의 내용을 변경**하는 것
  - 상속 받은 메서드를 그대로 사용하기도 하지만, 자식 클래스 자신에 맞게 변경해야 하는 경우, 부모의 메서드를 오버라이딩함
  - 메서드 재정의
- overriding
  - 사전적 의미
    - ~위에 덮어쓰다, ~에 우선하다
    - 무시하다, 짓밟다 의 뜻

메서드 오버라이딩(Overriding)



예제 1

```
class Points
{
      protected int x;
      protected int y;

      public String findLocation()
      {
            String result = "x:"+ x +", y:"+y;
            return result;
     }
}

class Points3D extends Points
{
      private int z;

      public String findLocation()
      {
            String result = "x:"+ x +", y:"+y + ", z:"+z;
            return result;
     }
}//class
```

2차원 좌표계의 한 점을 표현하기 위한 Point 클래스

한 점의 x,y좌표를 문자열로 반환

3차원 좌표계의 한 점을 표현하기 위한 Point3D 클래스

예제 1

```
3차원 좌표=x: 0,y: 0,z: 0
2차원 좌표=x: 0,y: 0
```

```
class  OverridingTest1
{
    public static void main(String[] args)
    {
            Points3D p = new Points3D();
            String r = p.findLocation();  //자식클래스(Points3D)의 메서드 호출
            System.out.println("3차원 좌표="+ r);

            Points p2 = new Points();
            r = p2.findLocation();  //부모클래스(Points)의 메서드 호출
            System.out.println("2차원 좌표="+ r);
    }
}//
```

오버라이딩의 조건

- 오버라이딩은 메서드의 내용만을 새로 작성하는 것이므로 **메서드의 선언부는 부모의 것과 완전히 일치해야** 함

자식 클래스에서 오버라이딩하는 메서드는 부모 클래스의 메서드와
- 이름이 같아야 한다.
- 매개변수가 같아야 한다.
- 리턴타입이 같아야 한다.

- 접근 제한자와 예외(Exception)는 제한된 조건 하에서만 다르게 변경할 수 있음

1. **접근 제한자는** 부모 클래스의 메서드보다 **좁은 범위로 변경할 수 없다.**
   - 부모 클래스에 정의된 메서드의 접근 제한자가 **protected** 라면, 이를 오버라이딩하는 자식 클래스의 메서드는 **protected**나 **public**이어야 함
2. 부모 클래스의 메서드보다 **많은 수의 예외를 선언할 수 없다.**
3. 인스턴스 메서드를 **static** 메서드로 또는 그 반대로 변경할 수 없다.

오버라이딩의 조건

```
class Parent{
        void parentMethod() throws IOException, SQLException{
                .....
        }
}

class Child extends Parent{
        void parentMethod() throws IOException{
                .....
        }
}
```

```
class Child extends Parent{
        void parentMethod() throws Exception{
                .....
        }
}
```

- 잘못된 오버라이딩

<= Exception은 모든 예외의 최고 조상이므로
가장 많은 개수의 예외를 던질 수 있도록 선언한 것이 됨

오버로딩 vs 오버라이딩

오버로딩(overloading) - 한 클래스 내에 같은 이름의 메서드를 여러 개 정의하는 것
(매개변수의 개수나 자료형이 달라야 함)
기존에 없는 새로운 메서드를 정의하는 것(new)
오버라이딩(overriding) - 부모로부터 상속받은 메서드의 내용을 변경하는 것
(change, modify, 재정의)

```
class Parent
{
    void parentMethod(){}
}
class Child extends  Parent
{
    void parentMethod(){}
    void parentMethod(int i){}

    void childMethod(){}
    void childMethod(int i){}
    void childMethod(){}  //에러. 중복정의 되었음
}
```

예제2

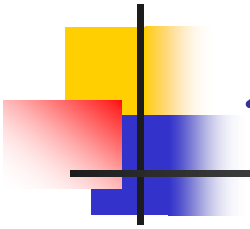
```
class Parent
{
    public void display()
    {
        System.out.println("Parent Method()");
    }
}
class Child extends Parent
{
    public void display()
    {
            System.out.println("Child Method()");
    }
}
class  OverridingTest2
{
    public static void main(String[] args)
    {
            Child c = new Child();
            c.display();  //Child의 메서드 호출

            Parent obj = new Child();
            obj.display();  //Child의 메서드 호출

            Parent p = new Parent();
            p.display();  //Parent의 메서드 호출
    }
}
```

```
Child Method()
Child Method()
Parent Method()
```

```
도형을 그린다
삼각형를 그린다
```

예제3

```
class Shape
{
	public void draw()
	{
		System.out.println("도형을 그린다");
	}
}
class Triangle extends Shape
{
	public void draw()
	{
		System.out.println("삼각형를 그린다");
	}
}
class  OverridingTest3
{
	public static void main(String[] args)
	{
		Shape s = new Shape();
		s.draw();
		Triangle t = new Triangle();
		t.draw();
	}
}
```



실습

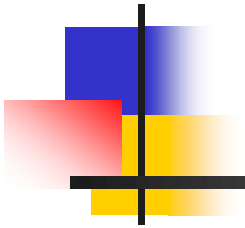
```
가르치는 일을 합니다

프로그래밍을 합니다
```

- Human 클래스 (부모 클래스)
  - 메서드 : work()
    - 하는 일을 기술한다.
- Teacher 클래스 (자식 클래스)
  - 메서드 : work() => 부모 클래스의 메서드 오버라이딩
    - 가르친다(System.out.println으로 가르친다는 내용을 화면 출력)
- Programmer (자식 클래스)
  - 메서드 : work() => 부모 클래스의 메서드 오버라이딩
    - 프로그래밍한다.

```
원의 면적 : 314.0
사각형의 면적 : 35.0
```

실습2

- Shape 클래스 (부모 클래스)
  - 메서드 : findArea()
    - 면적을 구한다. => return 0;
- Circle 클래스 (자식 클래스)
  - 멤버변수 : 반지름
  - 메서드 : findArea() => 부모 클래스의 메서드 오버라이딩
    - 원의 면적을 구해서 return (원의 면적 : 3.14 * 반지름 * 반지름)
- Rectangle 클래스 (자식 클래스)
  - 멤버변수 : 가로, 세로
  - 메서드 : findArea() => 부모 클래스의 메서드 오버라이딩
    - 사각형의 면적을 구해서 return (가로*세로)



this/ super

생성자에서 다른 생성자 호출하기-this()

- 같은 클래스의 멤버들 간에 서로 호출할 수 있는 것처럼 생성자 간에도 서로 호출이 가능함
- 다음의 두 조건을 만족시켜야 함

- 생성자의 이름으로 클래스 이름 대신 **this**를 사용한다.
- 한 생성자에서 다른 생성자를 호출할 때는 반드시 **첫 줄에서만 호출**이 가능하다.

this(), this(매개변수**)** - 같은 클래스의 다른 생성자를 호출할 때 사용한다.

생성자를 호출하는 this()

- **this() - 클래스 자신의 생성자를 호출할 때 사용**
  - 클래스 내에서 유일하게 호출할 수 없는 메서드 – 생성자
  - 생성자를 호출하는 방법을 제공하는 것이 this()
  - 자신의 생성자를 재이용하는 것

- this를 사용하는 곳

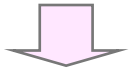
  • 자신의 멤버를 참조하는 this => this.멤버
  • 생성자를 호출하는 this()

```
color:white, gearType:auto, door:4
color:blue, gearType:auto, door:4
```

예제

```
class Car {
	private String color;			// 색상
	private String gearType;		// 변속기 종류 - auto(자동), manual(수동)
	private int door;			// 문의 개수

	Car() {
		this("white", "auto", 4);
	}
	Car(String color) {
		this(color, "auto", 4);
	}
	Car(String color, String gearType, int door) {
		this.color = color;
		this.gearType = gearType;
		this.door = door;
	}
	public void showInfo(){
		System.out.println("color:" + color + ", gearType:" + gearType+ ", door:"+door);
	}
}//
class CarTest2 {
	public static void main(String[] args) {
		Car c1 = new Car();
		Car c2 = new Car("blue");
		c1.showInfo();
		c2.showInfo();
	}
}
```

```
Car() {
		color="white";
		gearType="auto";
		door=4;
}
```



```
Car() {
		this("white", "auto", 4);
}
```

super

- 부모의 멤버를 참조하는 super => super.멤버
- 부모 생성자를 호출하는 super()

- super
  - 자식 클래스에서 **부모 클래스**로부터 상속받은 **멤버를 참조**하는데 사용되는 참조변수
    - cf : 멤버변수와 지역변수의 이름이 같을 때 this 를 사용해서 구별
  - 상속받은 멤버와 자신의 클래스에 정의된 멤버의 이름이 같을 때 super를 사용해서 구별할 수 있음

  - 부모의 멤버와 자신의 멤버를 구별하는 데 사용된다는 점을 제외하고는 super와 this는 근본적으로 같다
  - 모든 인스턴스 메서드에는 자신이 속한 인스턴스의 주소가 지역변수로 저장되는데, 이것이 참조변수인 this와 super의 값이 됨
  - static메서드는 인스턴스와 관련이 없으므로 **this나 super는 static 메서드에서는 사용할 수 없고**, 인스턴스 메서드에서만 사용할 수 있음

예제

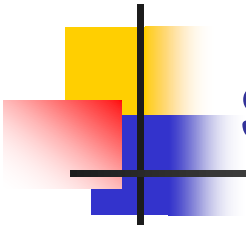
```
x=10
this.x=10
super.x=10
```

```
class SuperTest1_1 {
    public static void main(String args[]) {
        Child c = new Child();
        c.method();
    }
}

class Parent {
    int x=10;
}

class Child extends Parent {
    void method() {
        System.out.println("x=" + x);
        System.out.println("this.x=" + this.x);
        System.out.println("super.x="+ super.x);
    }
}
```

예제2

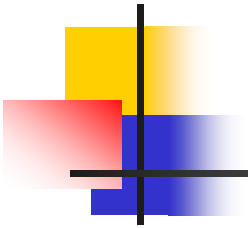
```
x=20
this.x=20
super.x=10
```

```
class SuperTest1_2 {
    public static void main(String args[]) {
        Child c = new Child();
        c.method();
    }
}

class Parent {
    int x=10;
}

class Child extends Parent {
    int x=20;
    void method() {
        System.out.println("x=" + x);
        System.out.println("this.x=" + this.x);
        System.out.println("super.x="+ super.x);
    }
}
```



super

- 변수만이 아니라 메서드 역시 super 를 써서 호출할 수 있음
- 부모 클래스의 메서드를 자식 클래스에서 오버라이딩한 경우에 super를 사용함
- 부모 클래스의 메서드의 내용에 추가적으로 작업을 덧붙이는 경우라면 super를 사용해서 부모 클래스의 메서드를 포함시키는 것이 좋다
- 자식클래스 내부에서 부모클래스의 메서드를 사용하고 싶다면 super 키워드 사용



예제

```
class Points
{
      protected int x;
      protected int y;

      public String findLocation()
      {
          return "x:"+ x +", y:"+y;
     }
}

class Points3D extends Points
{
      private int z;

      public String findLocation()
      {
              //return "x:"+ x +", y:"+y + ", z:"+z;
              return super.findLocation() + ", z:"+z; //부모의 메서드 호출
     }
}//
```

super() – 부모 클래스의 생성자 호출

- this()와 마찬가지로 super() 역시 생성자임
- this()
  - 같은 클래스의 다른 생성자를 호출하는 데 사용
- super()
  - **부모 클래스의 생성자를 호출**하는데 사용됨
- 자식 클래스의 인스턴스를 생성하면, 자식의 멤버와 부모의 멤버가 모두 합쳐진 하나의 인스턴스가 생성됨
  - 이 때 부모 클래스 멤버의 생성과 초기화 작업이 수행되어야 하기 때문에 자식 클래스의 생성자에서 부모 클래스의 생성자가 호출되어야 함
  - **생성자의 첫 줄에서 부모 클래스의 생성자를 호출**해야 하는 이유
    - 자식 클래스의 멤버가 부모 클래스의 멤버를 사용할 수도 있으므로 부모의 멤버들이 먼저 초기화되어 있어야 하기 때문

상속을 받았을 경우 *부모클래스의 생성자가 매개변수를 가지고 있다면* 자식은 부모의 생성자에게 매개변수를 넣어줘야 **함**

super() – 부모 클래스의 생성자 호출

- 부모 클래스 생성자의 호출은 클래스의 상속관계를 거슬러 올라가면서 계속 반복됨
- 마지막으로 모든 클래스의 최고 조상인 Object클래스의 생성자인 Object() 까지 가서야 끝이 남

Object 클래스를 제외한 모든 클래스의 생성자 첫 줄에는 생성자
(같은 클래스의 다른 생성자 또는 부모의 생성자)를 호출해야 한다.
그렇지 않으면 **컴파일러가 자동적으로 super(); 를 생성자의 첫 줄에** 삽입한다.

예제

```
class PointTest {
	public static void main(String args[]) {
		Point3D p3 = new Point3D(1,2,3);
	}
}
class Point {
	int x;
	int y;

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

	String findLocation() {
		return "x :" + x + ", y :"+ y;
	}
}
class Point3D extends Point {
	int z;

	Point3D(int z) {

		this.z = z;
	}
	String findLocation() {		// 오버라이딩
		return "x :" + x + ", y :"+ y + ", z :" + z;
	}
}
```

- 컴파일 에러

=>Point 클래스에 생성자 Point()가 정의되어 있지 않기 때문

- Point 클래스에 생성자 Point()를 추가해주거나

- 생성자 Point3D(int x, int y, int z)의 첫줄에서 Point(int x, int y)를 호출하도록 변경

```
Point3D(int x, int y, int z) {
		super(x, y);
		this.z = z;
}
```

부모 클래스의 멤버변수는 부모의 생성자에 의해 초기화 하도록 한다

생성자 첫 줄에서 다른 생성자를 호출하지 않기 때문에 컴파일러가 super();를 여기에 삽입함

super() => Point3D의 부모인 Point클래스의 기본 생성자인 Point()를 의미함

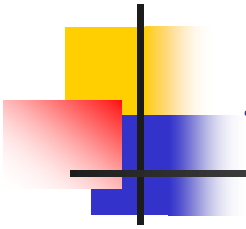
예제2

```
p3.x=100
p3.y=200
p3.z=300
```

```
class PointTest2 {
	public static void main(String args[]) {
		Point3D p3 = new Point3D();
		System.out.println("p3.x=" + p3.x);
		System.out.println("p3.y=" + p3.y);
		System.out.println("p3.z=" + p3.z);
	}
}//
class Point {
	int x=10;
	int y=20;

	Point(int x, int y) {
		//super(); 컴파일러가 자동적으로 삽입, 조상인 Object 클래스의 생성자인 Object()를 호출
		this.x = x;
		this.y = y;
	}
}//
class Point3D extends Point {
	int z=30;

	Point3D() {
		this(100, 200, 300);	// Point3D(int x, int y, int z)를 호출한다.
	}
	Point3D(int x, int y, int z) {
		super(x, y);		// Point(int x, int y)를 호출한다.
		this.z = z;
	}
}//
```

부모클래스는 생성자를 만들지 않는 게 원칙

부모의 값을 매개변수로 받아온 후 super을 써야한다!



부모클래스의 생성자를 호출하는 super()

- 부모클래스의 생성자가 매개변수를 가지고 있다면 자식클래스에서는 부모의 생성자의 매개변수를 넣어주어야 함
  - **super()**
    - 부모 생성자에 매개변수가 존재한다면 생성자의 매개변수의 타입과 개수를 맞추어 주어야만 호출 가능함
    - 부모 생성자의 매개변수 타입이 달라 호출해 줄 수 없을 때 사용
    - 생성자 내에서 부모 생성자를 호출하기 위해 사용

예제

```
class Father {
	private String name;
	Father(String name) {
		this.name = name;
	}
	public void display()
	{
		System.out.println("name : " +  name);
	}
}
class Son extends Father {
	Son(String name){
		super(name);
	}
}

class  SuperTest2
{
	public static void main(String[] args)
	{
		Son s = new Son("아들");
		s.display();
	}
}
```

예제1

```
이름, 나이, 전공을 입력하세요
홍길동
20
영어
==========================================
이름:홍길동
나이:20
전공:영어
```

- person 클래스 정의
  - 필드 : 이름, 나이
  - 생성자 – 이름, 나이를 매개변수로 받아 초기화
  - getter/setter
- person 클래스를 상속받는 student 클래스 정의
  - 필드 : 전공
  - 생성자 : 전공을 매개변수로 받아 초기화
  - getter/setter
- main() 메서드
  - Student 객체생성
  - 이름, 나이, 전공을 getter로 불러서 화면에 출력

예제 1

```
import java.util.Scanner;
class Person
{
	protected String name;
	protected int age;
	public Person(String name, int age)
	{
		this.name = name;
		this.age = age;
	}
	public void setName(String name)
	{
		 this.name = name;
	}
	public String getName()
	{
		return this.name;
	}
	public void setAge(int age)
	{
		this.age = age;
	}
	public int getAge()
	{
		return this.age;
	}
}//
```

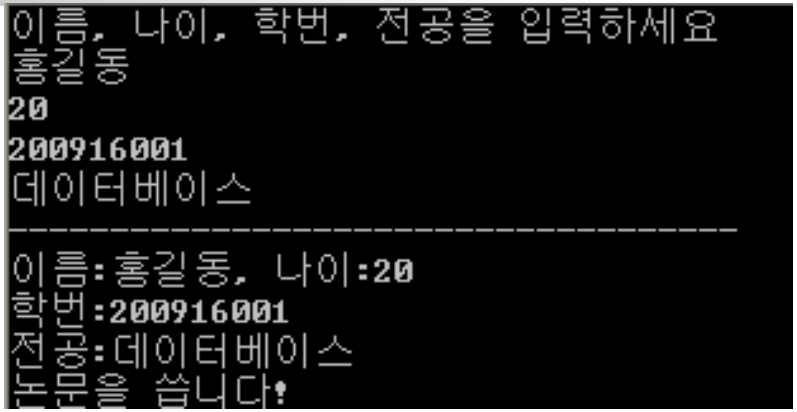
```
class Student extends Person
{
	private String major;
	public Student(String name, int age, String major)
	{
		super(name, age);
		this.major = major;
	}
	public void setMajor(String major)
	{
		this.major = major;
	}
	public String getMajor()
	{
		return this.major;
	}
}//
```

예제 1

```
class  SuperTest3
{
	public static void main(String[] args)
	{
		System.out.println("이름, 나이, 전공을 입력하세요");
		Scanner sc = new Scanner(System.in);
		String name = sc.nextLine();
		int age = sc.nextInt();
		sc.nextLine();
		String major = sc.nextLine();

		System.out.println("====================================");

		Student s = new Student(name,age,major);
		System.out.println("이름 : "+s.getName());
		System.out.println("나이 : "+s.getAge());
		System.out.println("전공 : "+s.getMajor());
	}
}
```

예제2

- Person – 부모 클래스
  - 이름, 나이
  - 생성자
  - 메서드
    - 화면 출력
- Student – Person의 자식 클래스
  - 학번
  - 생성자
  - 메서드
    - 화면 출력 오버라이딩
    - 공부한다
- Graduate – Student의 자식 클래스
  - 전공
  - 메서드
    - 화면출력 오버라이딩
    - 논문을 쓴다



예제2

```
import java.util.Scanner;
class Person
{
	protected String name;
	protected int age;
	public Person(String name, int age)
	{
		this.name = name;
		this.age = age;
	}
	 public void display()
	{
		System.out.println("이름 : " + name + ", 나이 : " +  age);
	}
}

class Student extends Person
{
	protected int stIdNo;
	public Student(String name, int age, int stIdNo)
	{
		super(name, age);
		this.stIdNo = stIdNo;
	}
	public void study()
	{
		System.out.println("공부합니다!");
	}
```

예제2

```
    public void display()
    {
            super.display();
            System.out.println("학번 : " +  stIdNo);
    }
}
class Graduate extends Student
{
    protected String major;
    public Graduate(String name, int age, int stIdNo, String major)
    {
            super(name, age, stIdNo);
            this.major = major;
    }
    public void writeThesis()
    {
            System.out.println("논문을 씁니다!");
    }
     public void display()
    {
            super.display();
            System.out.println("전공 : " +  major);
    }
}
```

예제2

```
class  SuperTest4
{
    public static void main(String[] args)
    {
        System.out.println("이름, 나이, 학번, 전공을 입력하세요");
        Scanner sc = new Scanner(System.in);
        String name = sc.nextLine();
        int age = sc.nextInt();
        int stIdNo = sc.nextInt();
        sc.nextLine();
        String major = sc.nextLine();

        System.out.println("-------------------------------");

        Graduate obj = new Graduate(name, age, stIdNo, major);
        obj.display();
        obj.writeThesis();

    }
}
```

상속관계에 있는 인스턴스의 생성과정

1. 메모리 공간의 할당과 모든 인스턴스 변수의 디폴트 초기화

2. 생성자의 호출
 - 우선 하위 클래스의 생성자가 호출
 - super 문에 의해서 상위 클래스의 생성자가 먼저 실행됨 (상위 클래스의 멤버가 먼저 초기화됨)
 - 하위 클래스의 생성자가 실행됨

[1] 하위클래스의 생성자는 상위클래스의 인스턴스 변수를 초기화할 데이터까지 인자로 전달받아야 한다
[2] 하위클래스의 생성자는 상위클래스의 생성자 호출을 통해서 상위클래스의 인스턴스 변수를 초기화한다
[3] 키워드 super는 상위클래스의 생성자호출에 사용된다. super와 함께 표시된 전달되는 인자의 수와 자료형을 참조하여 호출할 생성자가 결정된다

반드시 호출되어야 하는 상위클래스의 생성자

```
class AAA
{
    int num1;
}

class BBB extends AAA
{
    int num2;
}
```

AAA() { }

자동으로 삽입되는 디폴트생성자의 형태

BBB() { super(); }

```
class AAA
{
    int num1;
}

class BBB extends AAA
{
    int num2;
    BBB() { num2=0; }
}
```

결과적으로 어떠한 형태로든
(프로그래머가 직접 삽입하건, 컴파일러가 삽입하건)
상위클래스의 생성자는 반드시 호출이 이뤄진다

자동으로 삽입되는 상위클래스의 생성자 호출문

super();

실습1

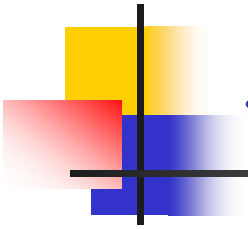
- 은행계좌정보를 담을 수 있는 Account 클래스를 상속하는 KBAccount 클래스 정의하기
- Account 클래스 (부모)
  - 멤버변수 : 계좌번호, 계좌잔액
  - 생성자에서 멤버변수 초기화(계좌번호, 계좌잔액을 인자로 받아서)
  - 출력 메서드 display() : 계좌번호, 계좌잔액 출력
- KBAccount 클래스 (자식)
  - Account 클래스가 지니고 있는 멤버변수 이외에 고객별 이체한도 정보를 담고 있는 멤버변수를 지녀야 함
  - 이체한도 멤버변수 추가
  - 생성자에서 이체한도를 인자로 받아 초기화
  - 출력 메서드 display()를 오버라이딩해서 계좌번호, 계좌잔액, 이체한도 출력
- 메인메서드
  - 매개변수값 넘겨 KBAccount 객체 생성
  - 계좌번호, 잔액, 이체한도를 화면에 출력

```
계좌번호, 잔액, 이체한도를 입력하세요
100-05-2456
790000
2000000
==========================================
계좌번호:100-05-2456
계좌잔액:790000
이체한도:2000000
```

실습2- 급여관리 시스템

- 직원의 고용형태
  - 고용직, 임시직
  - 급여계산방식의 차이
    - 고용직(Permanent) – 연봉제 (매달 기본급여가 정해져 있다)
    - 임시직(Temporary) – 일한시간 * 시간당 급여
- 고용인들이 공통적으로 지니고 있어야 하는 멤버들을 모아서 Employee 클래스로 추상화시킨다

- 고용직 클래스 (Permanent) – Employee클래스를 상속받음
  - 직원의 이름과 급여정보를 저장하기 위한 클래스
    - (고용직 급여는 입사 당시 정해진다고 가정하고 급여인상은 제외)
  - 필드 : 이름, 기본급여, 보너스
  - 생성자, getter/setter
  - 메서드 : 급여계산 => 기본급여 + 보너스
- 임시직 클래스 정의 (Temporary) –Employee클래스를 상속받음
  - 필드 : 이름, 일한시간(time), 시간당 급여(pay)
  - 생성자, getter/setter
  - 메서드 : 급여계산 => time*pay



실습2-정리

- class Employee
  - 필드 : 이름
  - 생성자, getter/setter
  - 메서드 : 급여계산 findPay(){ return 0; }
- class Permanent – 고용직
  - 필드 : 기본급여(salary), 보너스
  - 생성자, getter/setter
  - 메서드 : 급여계산 findPay() 오버라이딩 =>기본급여+보너스
- Class Temporary – 임시직
  - 필드 : 일한시간(time), 시간당 급여(pay)
  - 생성자, getter/setter
  - 메서드 : 급여계산 findPay() 오버라이딩 => time*pay

- main()
  - 결과 화면 출력시 getter 이용하여 출력

실습2

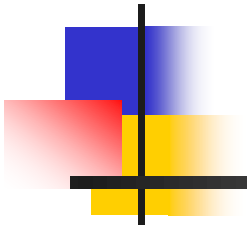
```
고용형태 - 고용직(P), 임시직(T)을 입력하세요
p
이름, 기본급여, 보너스를 입력하세요
홍길동
2500000
300000
------------------------------------------

고용형태:고용직
이름:홍길동
급여:2800000
```

```
고용형태 - 고용직(P), 임시직(T)을 입력하세요
t
이름, 일한시간, 시간당급여를 입력하세요
김연아
20
65000
------------------------------------------------

고용형태:임시직
이름:김연아
급여:1300000
```

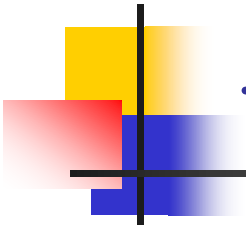
실습3-정규판매와 할인판매를 위한 판매 가격 계산하기

- 학생은 10% 할인하는 분식점에서 판매가격 계산하기
- FoodSale 클래스(부모)
  - 멤버변수 : 메뉴, 수량, 단가, 판매가격, 누적총액(판매가격의 합계)
  - 메서드
    - 판매가격 구하는 메서드 : 판매가격 = 수량*단가
    - 누적 총액 구하는 메서드 : 누적총액 = 판매가격누적하기
- StudentFoodSale 클래스(자식) – 학생인 경우
  - 멤버변수 : 할인금액, 누적 할인금액
  - 메서드 (오버라이딩)
    - 판매가격 구하는 메서드
      - 할인금액 = 수량*단가*할인률
      - 판매가격 = 수량*단가-할인금액
    - 누적 총액 구하는 메서드
      - 누적총액 = 판매가격 누적하기
      - 누적 할인금액 = 할인금액 누적하기

```
메뉴, 수량, 단가, 학생여부(Y/N)를 입력하세요!
김치찌게
4
5000
Y
판매금액=18000, 누적판매금액=18000, 누적할인금액=2000
그만하시겠습니까?(Q)uit

메뉴, 수량, 단가, 학생여부(Y/N)를 입력하세요!
라면
3
3000
N
판매금액=9000, 누적판매금액=27000
그만하시겠습니까?(Q)uit

메뉴, 수량, 단가, 학생여부(Y/N)를 입력하세요!
비빔밥
2
6000
Y
판매금액=10800, 누적판매금액=37800, 누적할인금액=3200
그만하시겠습니까?(Q)uit
q
```



final



final – 마지막의, 변경될 수 없는

- final
  - 변수에 사용되면 값을 변경할 수 없는 상수가 됨
  - 메서드에 사용되면 오버라이딩을 할 수 없게 됨
  - 클래스에 사용되면 자신을 확장하는 자식 클래스를 정의하지 못하게 됨

final 이 사용될 수 있는 곳 - 클래스, 메서드, 멤버변수, 지역변수

final - 마지막의, 변경될 수 없는

제어자	대상	의 미
final	클래스	변경될 수 없는 클래스, 확장될 수 없는 클래스가 된다. final로 지정된 클래스는 다른 클래스의 부모가 될 수 없다.
	메서드	변경될 수 없는 메서드, final로 지정된 메서드는 오버라이딩을 통해 재정의 될 수 없다.
	멤버변수	변수 앞에 final이 붙으면, 값을 변경할 수 없는 상수가 된다.
	지역변수	

대표적인 final 클래스로는 String과 Math가 있다.

예제 1

```
public class  FinalTest1
{
	public static void main(String[] args)
	{
	}
}//
final class Parent
{
	public void func(){
		System.out.println("Parent");
	}
}//
class Child extends Parent  //에러
{
	public void func(){
		System.out.println("Child");
	}
}//
/*---------- javac ----------
FinalTest1.java:16: cannot inherit from final Parent
class Child extends Parent
                    ^
1 error
출력 완료 (1초 경과)*/
```

예제2

```
class FinalTest2
{
	public static void main(String[] args)
	{
	}
}//
class Parent
{
	void func(){
		System.out.println("Parent");
	}
	public final void finalFunc(){
		System.out.println("pppppp");
	}
}//
class Child extends Parent
{
	public void func(){
		System.out.println("Child");
	}
	public void finalFunc(){  //에러
		System.out.println("ccccccc");
	}
}//
```

```
/*---------- javac ----------
FinalTest2.java:22: finalFunc() in Child cannot
override finalFunc() in Parent;
overridden method is final
		public void finalFunc(){
				^
1 error

출력 완료 (1초 경과)*/
```

예제3

```
class MyFinal
{
	final public static double PI=3.14;
	int age=10;
}//
class FinalTest3
{
	public static void main(String[] args)
	{
		MyFinal my=new MyFinal();
		System.out.println("my.age="+my.age);
		System.out.println("MyFinal.PI: "
				+MyFinal.PI);

		my.age=20;
		MyFinal.PI=3.1415;	//에러
		System.out.println("my.age="+my.age);
		System.out.println("MyFinal.PI: "
				+MyFinal.PI);
	}
}//
/*---------- javac ----------
FinalTest3.java:17: cannot assign a value to final variable PI
		MyFinal.PI=3.1415;
			^
1 error
출력 완료 (1초 경과)*/
```

상수

- 상수
  - 한 번 초기화되면 더 이상 값을 변경할 수 없는 멤버
    - 응용 프로그램에서 **절대로 변경되면 안 되는 값**이나 **어떤 수치들의 대표값**
- final 필드
  - 상수 변수를 지정하기 위해 사용하는 예약어
  - 상수 필드
  - final double PI = 3.141592;
    - 이렇게 정해진 PI 란 필드의 값은 절대 변할 수 없음
- final 은 static 와 함께 사용될 수 있음
  - final 필드가 static 필드의 특성까지 띄게 됨
  - 클래스명으로 접근
  - static은 지역변수로 사용할 수 없는 예약어이므로, final 을 지역변수로 선언하고자 한다면 static 과 함께 사용해서는 안됨

상수

```
class AAA{
	//final 변수 - 상수 : 값을 변경할 수 없는 것
	//final static 멤버변수, final 멤버변수
	public final static double PI=3.1415;
	public final int DELIVERY=3000;
	int age=10;
}
class FinalVariable {
	public static void main(String[] args){
		//final 지역변수
		final double INTEREST_RATE=0.03;

		System.out.println("이자율 : " + INTEREST_RATE);
		System.out.println("파이 : " + AAA.PI);
		AAA obj = new AAA();
		System.out.println("배송비 : " + obj.DELIVERY);
		obj.age=20; //변수는 값 변경 가능
		//AAA.PI=3.141592;  //상수는 변경 불가
		//=>cannot assign a value to final variable PI
	}
}
```

예제

```
import java.util.*;
class Sizes
{
    public static final int MEDIUM = 100;
    public static final int LARGE = 105;
    public static final int XLARGE = 110;
}

class Shirt
{
    public static void main(String[] ar)
    {
        Scanner sc = new Scanner(System.in);
        while(true)
        {
            System.out.println("치수 기호를 입력하세요. (M)edium (L)arge (X)Large (q)uit");
            String s = sc.nextLine();

            if(s.toUpperCase().equals("Q")) break;
```

```
치수 기호를 입력하세요. (M)edium (L)arge (X)Large (q)uit
L
105

치수 기호를 입력하세요. (M)edium (L)arge (X)Large (q)uit
M
100

치수 기호를 입력하세요. (M)edium (L)arge (X)Large (q)uit
X
110

치수 기호를 입력하세요. (M)edium (L)arge (X)Large (q)uit
a
그런 치수를 가지고 있지 않습니다.

치수 기호를 입력하세요. (M)edium (L)arge (X)Large (q)uit
q
```

예제

```
                    if(s.toUpperCase().equals("M"))
                            System.out.println(Sizes.MEDIUM);  //클래스이름으로 바로 접근
                    else if(s.toUpperCase().equals("L"))
                            System.out.println(Sizes.LARGE);
                    else if(s.toUpperCase().equals("X")){
                            System.out.println(Sizes.XLARGE);
                    }
                    else
                            System.out.println("그런 치수를 가지고 있지 않습니다.");
            }
    }
}
```

생성자를 이용한 final 멤버변수 초기화

- final 이 붙은 변수는 상수이므로 일반적으로 선언과 초기화를 동시에 하지만, 인스턴스 변수(final 상수)의 경우 생성자에서 초기화되도록 할 수 있음
- 각 인스턴스마다 final이 붙은 멤버변수가 다른 값을 갖도록 하는 것이 가능함

final 변수
- 상수지만 선언과 함께 초기화 하지 않고
  생성자에서 단 한번만 초기화할 수 있다.
- 각 인스턴스마다 다른 값을 갖도록 할 수 있다

static final 변수
- 선언과 함께 초기화
- 클래스 차원에서 하나만 생성, 모든 인스턴스가 같은 값을 갖는다

예제

```
카드 종류 :Heart
카드 숫자 : 7

카드 종류 :Diamond
카드 숫자 : 3
```

```
class Card{
    //인스턴스 final 변수는 선언과 함께 초기화하는 대신, 생성자에서 단 한번
    //초기화할 수도 있다 => 인스턴스별로 상수의 값을 다르게 줄 수 있다
    final String KIND;
    final int NUMBER;

    Card(String kind, int num){
        KIND=kind;
        NUMBER=num;
    }
    public void display(){
        //NUMBER=10;  //에러, 상수값 변경 불가
        System.out.println("카드 종류 :" + KIND);
        System.out.println("카드 숫자 : " + NUMBER+"₩n");
    }
}//class

class FinalVariable2 {
    public static void main(String[] args){
        Card c1 = new Card("Heart", 7);
        c1.display();

        Card c2 = new Card("Diamond", 3);
        c2.display();
    }
}
```

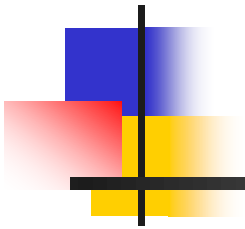
static final int width = 100;
static final int height = 250;

매개변수로 넘겨받은 값으로 KIND와 NUMBER를 초기화한다.

실습

```
입금할 금액을 입력하세요
300000
==================================
원금 : 300000원, 이자율 : 0.02, 이자 : 6000원
```

- 이자율을 상수로 선언하고, 이자 계산하기
  - 은행계좌 클래스 (Account)
    - 상수 – 이자율(INTEREST_RATE) : 2%
    - 메서드 – 이자 계산하는 메서드(입금액을 매개변수로 입력 받아서 이자를 계산한 후 결과를 리턴)
- main() 메서드에서 입금액을 입력 받아서 이자 계산 메서드 호출하여 이자 계산한 후 출력하기



클래스간의 관계-포함관계(Has a)

클래스간의 관계-포함관계(Has a)

- 상속을 통해 클래스 간에 관계를 맺어 주고 클래스를 재사용
- 상속 이외에도 클래스를 재사용하는 또 다른 방법
  - 클래스간에 '포함관계'를 맺어 주는 것
  - 클래스 간의 포함관계를 맺어 주는 것은 **한 클래스의 멤버변수로 다른 클래스를 선언**하는 것을 뜻함
  - 예) 원(Circle)을 표현하기 위한 Circle 이라는 클래스
    좌표상의 한 점을 다루기 위한 Point 클래스

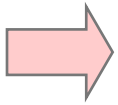
```
class Circle{
    int x; //원점의 x좌표
    int y; //원점의 y좌표
    int r;  //반지름
}
```

```
class Point{
    int x; //x좌표
    int y; //y좌표
}
```

클래스간의 관계-포함관계

- Point 클래스를 재사용(포함관계)해서 Circle 클래스를 작성한다면

```
class Circle{
    int x; //원점의 x좌표
    int y; //원점의 y좌표
    int r;  //반지름
}
```



```
class Circle{
    Point p = new Point(); //원점
    int r; //반지름
}
```

- 하나의 거대한 클래스를 작성하는 것보다 단위 별로 여러 개의 클래스를 작성한 후, 이 단위 클래스들을 포함관계로 재사용하면 보다 간결하고 손쉽게 클래스를 작성할 수 있음

클래스간의 관계 결정하기

- 클래스를 작성하는데 있어서 상속관계를 맺어줄 것인지, 포함관계를 맺어줄 것인지 결정하는 방법

```
class Circle{
    Point p = new Point(); //원점
    int r; //반지름
}
```

```
class Circle extends Point{
    int r;
}
```

[1] 상속관계 : ~ 은 일종의 ~ 이다 (**is a** 관계)

[2] 포함관계 : ~ 은 ~ 을 가지고 있다 (**has a** 관계)

원(Circle) 은 점(Point) 이다 - Circle is a Point.
원은 점을 가지고 있다 - Circle has a Point.

Circle 클래스와 Point 클래스 간의 관계는 상속관계보다 포함관계를 맺어주는 것이 더 옳다.

예) Car 클래스와 SportsCar 클래스는 'SportsCar는 일종의 Car이다' => Car를 부모로 하는 상속관계

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

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

class Circle extends Point{  //상속 관계
	private int r;
	Circle(int x, int y, int r)
	{
		super(x, y);
		this.r=r;
	}
	public void printInfo()
	{
		System.out.println("x=" + x);
		System.out.println("y=" + y);
		System.out.println("r=" + r +"₩n");
	}
}//class
```

```
x=3
y=4
r=10
```

```
class HasaTest2
{
		public static void main(String[] arg)
		{
			Circle c = new Circle(3, 4, 10);
			c.printInfo();

		}
}
```

• 클래스를 재사용하는 방법

[1] 상속 이용 : is a 관계가 성립하면 상속을 이용한다

~ is a ~ => ~ 는 일종의 ~이다

예) SportsCar is a Car

[2] 포함 관계 이용 : has a 관계가 성립하면 포함관계를 이용한다

~ has a ~ => ~는 ~를 가지고 있다

예) Circle has a Point

```
class Circle3  //포함 관계 이용 - 재사용하려는 클래스를 멤버변수로 포함시킨다
{
    private Point p;
    private int r;

    Circle3(Point p, int r)
    {
        this.p = p;
        this.r=r;
    }

    public void printInfo()
    {
        System.out.println("x=" + p.x);
        System.out.println("y=" + p.y);
        System.out.println("r=" + r+"₩n");
    }
}//class
class HasaTest3{
    public static void main(String[] arg)
    {
        Point p = new Point(7,8);
        Circle3 c3 = new Circle3(p, 10);
        c3.printInfo();
    }
}
```

```
x=7
y=8
r=10
```