7장. 상속

이것이 자바다(http://cafe.naver.com/thisjava)

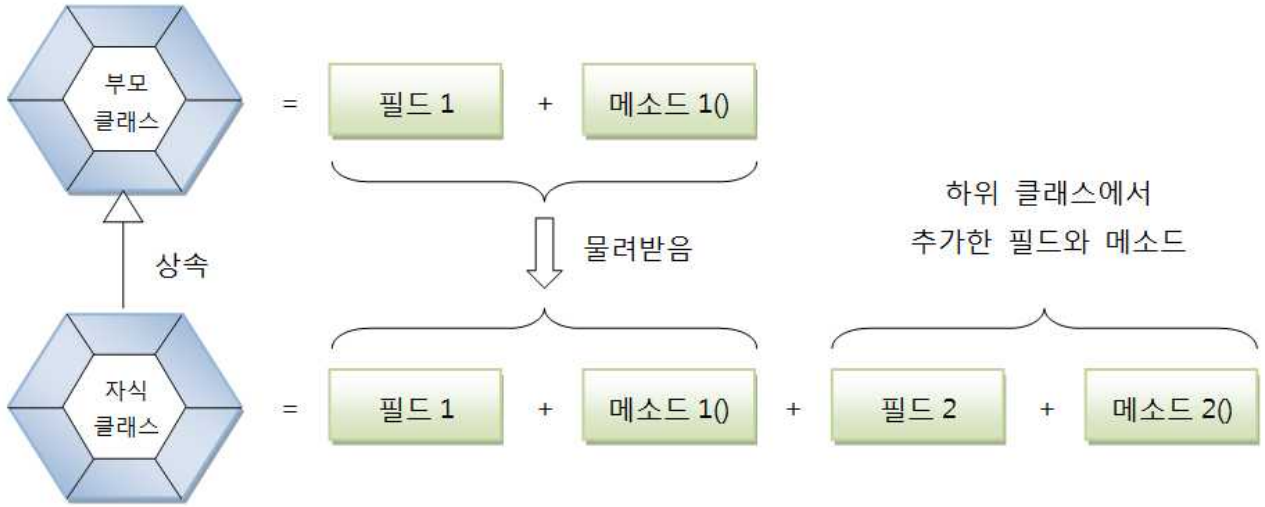
한빛미디어 Hanbit Media, Inc.

Contents

1절. 상속 개념

❖ 상속(Inheritance)이란?

- 객체 지향 프로그램:
  - 자식(하위, 파생) 클래스가 부모(상위) 클래스의 멤버를 물려받는 것
  - 자식이 부모를 선택해 물려받음
  - 상속 대상: 부모의 필드와 메소드



1절. 상속 개념

❖ 상속(Inheritance) 개념의 활용

- 상속의 효과
  - 부모 클래스 재사용해 자식 클래스 빨리 개발 가능
  - 반복된 코드 중복 줄임
  - 유지 보수 편리성 제공

- 상속 대상 제한
  - 부모 클래스의 private 접근 갖는 필드와 메소드 제외
  - 부모 클래스가 다른 패키지에 있을 경우, default 접근 갖는 필드와 메소드도 제외

멤버에 접근하는 클래스	멤버의 접근 지정자			
	private	디폴트 접근 지정	protected	public
같은 패키지의 클래스	×	○	○	○
다른 패키지의 클래스	×	×	×	○
접근 가능 영역	클래스 내	동일 패키지 내	동일 패키지와 자식 클래스	모든 클래스

2절. 클래스 상속(extends)

❖ 상속 선언

- extends 키워드로 선언
  - 부모 클래스를 물려받아 확장한다는 의미
- 부모 클래스 -> 슈퍼 클래스(super class)
- 자식 클래스 -> 서브 클래스(sub class)

```
class BaiscCalculator {
    public String calName = "pCal";
        …
}

// BaiscCalculator 를 상속받는 EnginneringCalculator 클래스 선언
class EnginneringCalculator extends BaiscCalculator {
        …
}
```

- 자바는 단일 상속 - 부모 클래스 나열 불가

예제

```
public class BaiscCalculator
{
        public String calName = "pCal";
        public int add(int x, int y)
        {
            System.out.println("add()");
            int resultAdd=x+y;
            return resultAdd;
        }
        public int sub(int x, int y)
        {
            System.out.println("sub");
            int resultAdd=x-y;
            return resultAdd;
        }
        public int mul(int x, int y)
        {
            System.out.println("mul()");
            int resultAdd=x*y;
            return resultAdd;
        }
        public double div(int x, int y)
        {
            double ret=0;
            ret=x/y;
            return ret;
        }
}
```

```
public class EnginneringCalculator extends
BaiscCalculator
{

}
```

```
public class MainApp
{
    public static void main(String[] args)
    {
        EnginneringCalculator ch = new
EnginneringCalculator();
        int result = ch.add(1, 2);
        String name = ch.calName;
        System.out.println(result);
        System.out.println(name);

    }

}
```

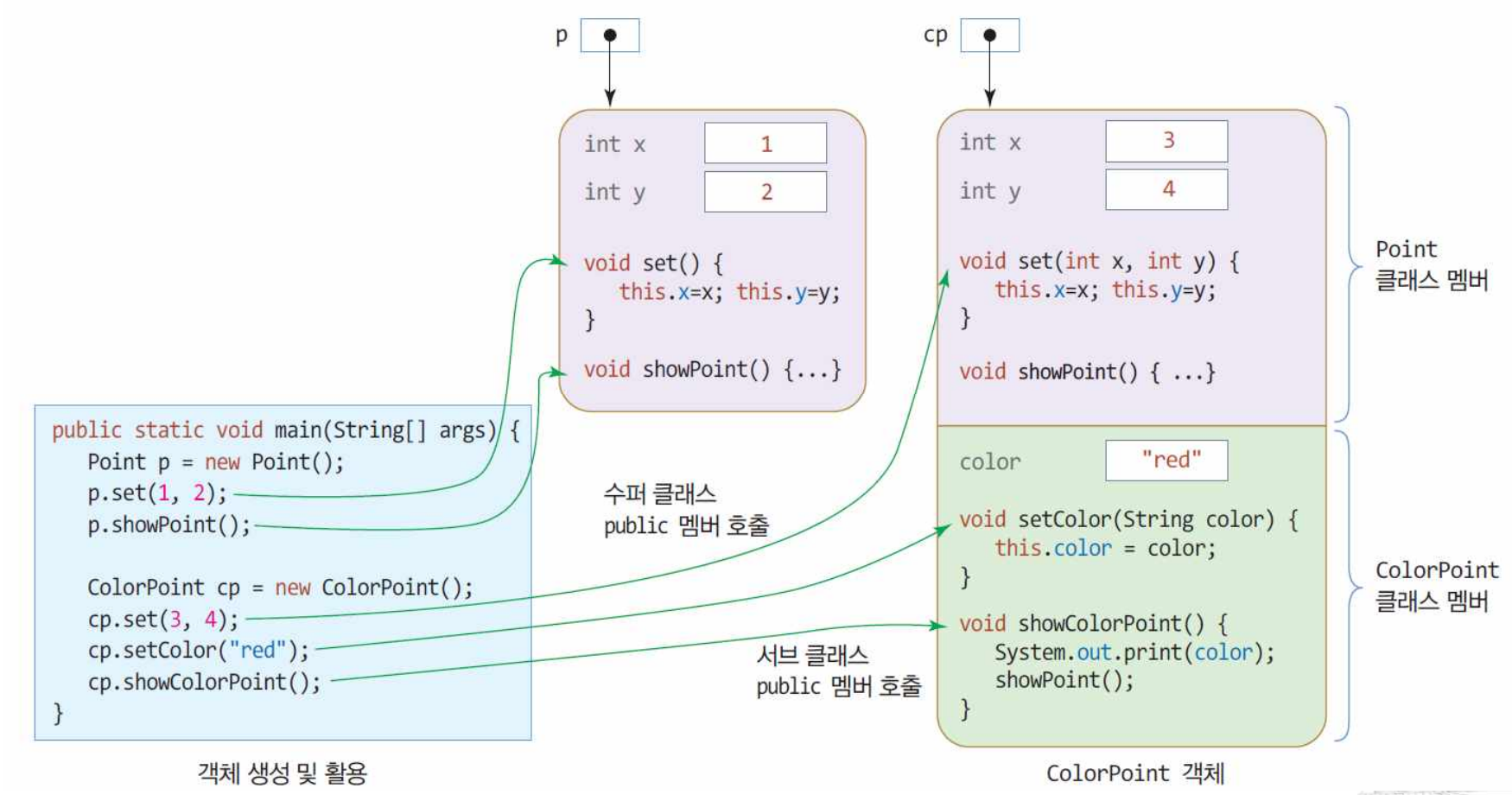
예제

```
class Point {
    int x;
    int y;
    void set(int a, int b) {
        x = a;
        y = b;
    }
    void showPoint() {
        System.out.println(x + "," + y);
    }
}

// Point를 상속받은 ColorPoint 선언
class ColorPoint extends Point {
    String color;
    void setColor(String c) {
        this.color = c;
    }
    void showColorPoint() {
        System.out.print(color);
        showPoint(); // Point의 showPoint() 호출
    }
}
```

```
public class ColorPointEx {
    public static void main(String [] args) {
        Point p = new Point();
        p.set(1, 2);
        p.showPoint();

        ColorPoint cp = new ColorPoint();
        cp.set(3, 4);
        cp.setColor("red");
        cp.showColorPoint();
    }
}
```

서브 클래스/슈퍼 클래스의 생성자 호출과 실행

- 슈퍼 클래스 객체와 서브 클래스의 객체는 별개
- 서브 클래스 객체는 슈퍼 클래스 멤버 포함



실습

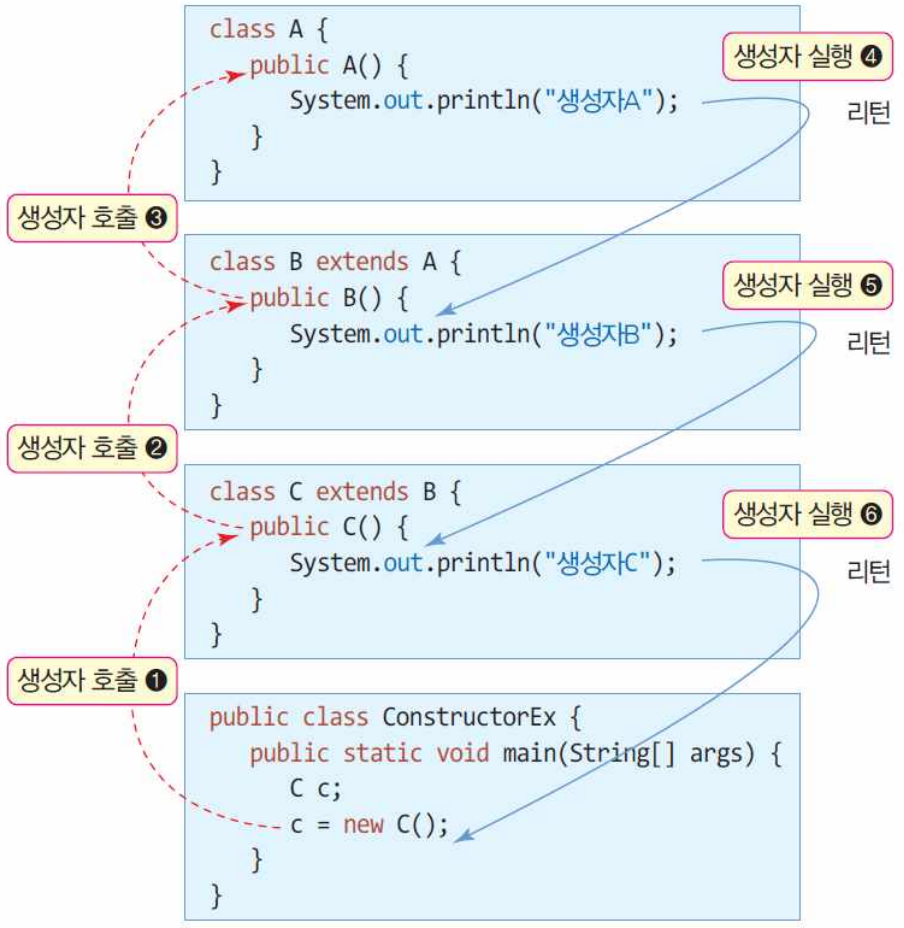
- **package : javab.inherit.basic_1**
  - **Mp3Player은 Player 를 상속 받는다.**
  - **Main()에서 Player의 객체 p, Mp3Player의 객체 mp 객체 생성후 각각play() 호출**
  - **mp를 이용 download() 호출**

class	Player		
	리턴타입	메소드명	매개변수
메소드	int (성공 1, 실패 2 리턴)	play	String name
	int (성공 1, 실패 2 리턴)	stop	없음
	void	fastforward	int time
	void	rewind	int time

class	MP3Player		
	리턴타입	메소드명	매개변수
메소드	int (성공 1, 실패 2 리턴)	download	String url
	int (성공 1, 실패 2 리턴)	onLinePlay	String url

슈퍼 클래스와 서브 클래스의 생성자 호출 및 실행 관계

- **서브 클래스의 객체가 생성될 때**
  - 슈퍼클래스 생성자와 서브 클래스 생성자 모두 실행
  - 실행 순서
    - 슈퍼 클래스의 생성자가 먼저 실행된 후 서브 클래스의 생성자 실행



결과
생성자A
생성자B
생성자C

super()로 슈퍼 클래스의 생성자 명시적 선택

- **super()**
  - 서브 클래스에서 명시적으로 슈퍼 클래스의 생성자 선택 호출
  - 사용 방식
    - super(parameter);
    - 인자를 이용하여 슈퍼 클래스의 적당한 생성자 호출
    - 반드시 서브 클래스 생성자 코드의 제일 첫 라인에 와야 함

예제

```
class A {
    public A() {
        System.out.println("생성자A");
    }
    public A(int x) {
        System.out.println("매개변수생성자A" + x);
    }
}
```

```
class B extends A {
    public B() {
        System.out.println("생성자B");
    }
    public B(int x) {
        super(x);  // 첫 줄에 와야 함
        System.out.println("매개변수생성자B" + x);
    }
}
```

```
public class ConstructorEx4 {
    public static void main(String[] args) {
        B b;
        b = new B(5);
    }
}
```

매개변수생성자A5
매개변수생성자B5

예제

```
class Point {
    private int x, y; // 한 점을 구성하는 x, y 좌표
    Point() {
        this.x = this.y = 0;
    }
    Point(int x, int y) {
        this.x = x; this.y = y;
    }
    void showPoint() { // 점의 좌표 출력
        System.out.println("(" + x + "," + y + ")");
    }
}

class ColorPoint extends Point {
    private String color; // 점의 색
    ColorPoint(int x, int y, String color) {
        super(x, y); // Point의 생성자 Point(x, y) 호출
        this.color = color;
    }
    void showColorPoint() { // 컬러 점의 좌표 출력
        System.out.print(color);
        showPoint(); // Point 클래스의 showPoint() 호출
    }
}
```

x=5, y=6

```
public class SuperEx {
    public static void main(String[] args) {
        ColorPoint cp = new ColorPoint(5, 6, "blue");
        cp.showColorPoint();
    }
}
```

```
blue(5,6)
```

x=5, y=6,
color = "blue" 전달

서브 클래스와 슈퍼 클래스의 생성자 선택

❖ 개발자가 서브 클래스의 생성자에 대해 슈퍼 클래스의 생성자를 명시적으로 선택하지 않은 경우

서브 클래스의 기본 생성자에 대해 컴파일러는 자동으로 슈퍼 클래스의 기본 생성자와 짝을 맺음

```
class A {
    public A() {
        System.out.println("생성자A");
    }
    public A(int x) {
        .....
    }
}
```

```
class B extends A {
    public B() {
        System.out.println("생성자B");
    }
}
```

```
public class ConstructorEx2 {
    public static void main(String[] args) {
        B b;
        b = new B();    // 생성자 호출
    }
}
```

```
생성자A
생성자B
```

슈퍼 클래스에 기본 생성자가 없어 오류 난 경우

```
class A {
    public A(int x) {
        System.out.println("생성자A");
    }
}
```

```
class B extends A {
    public B() {  // 오류 발생
        System.out.println("생성자B");
    }
}
```

```
public class ConstructorEx2 {
    public static void main(String[] args) {
        B b;
        b = new B();
    }
}
```

```
class A {
    public A() {
        System.out.println("생성자A");
    }
    public A(int x) {
        System.out.println("매개변수생성자A");
    }
}
```

```
class B extends A {
    public B() {
        System.out.println("생성자B");
    }
    public B(int x) {
        System.out.println("매개변수생성자B");
    }
}
```

```
public class ConstructorEx3 {
    public static void main(String[] args) {
        B b;
        b = new B(5);
    }
}
```

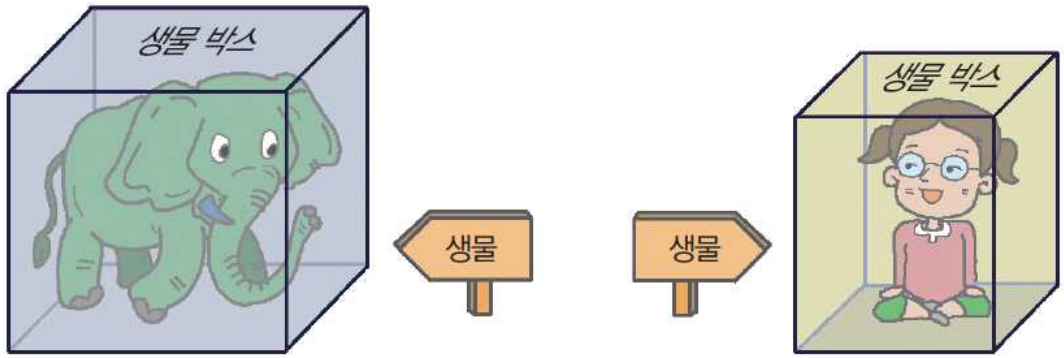
실습

- package : javab.inherit.constructor_1
- 명시적으로 슈퍼클래스 생성자를 호출하지 않은 경우 : 에러 수정
  - MP3Player 에 아래 디폴트 생성자 추가
    - MP3Player(int time)
    - Player(int time)
  - MainClass
    - MP3Player 클래스 객체 생성
- 위의 코드에서 Player(), MP3Player() 각각의 생성자에 출력문을 넣어서 호출되는 순서를 확인

실습

- package : javab.inherit.constructor_2
- 명시적으로 슈퍼클래스 생성자를 명시적으로 호출한경우
- Player 클래스에 아래 생성자 추가
  - Player(String cName, int year)
- MP3Player 에 아래 생성자 추가
  - MP3Player(String cName, int year,int numMusic)
  - MP3Player(String cName, int year,int numMusic) 에서 Player 의 아래 생성자 호출
    - Player(String cName, int year)
- MainClass
  - MP3Player 클래스 객체 생성

업캐스팅



생물이 들어가는 박스에
사람이나 코끼리를 넣어도 무방

* 사람이나 코끼리 모두 생물을
상속받았기 때문

❖ 업캐스팅(upcasting)

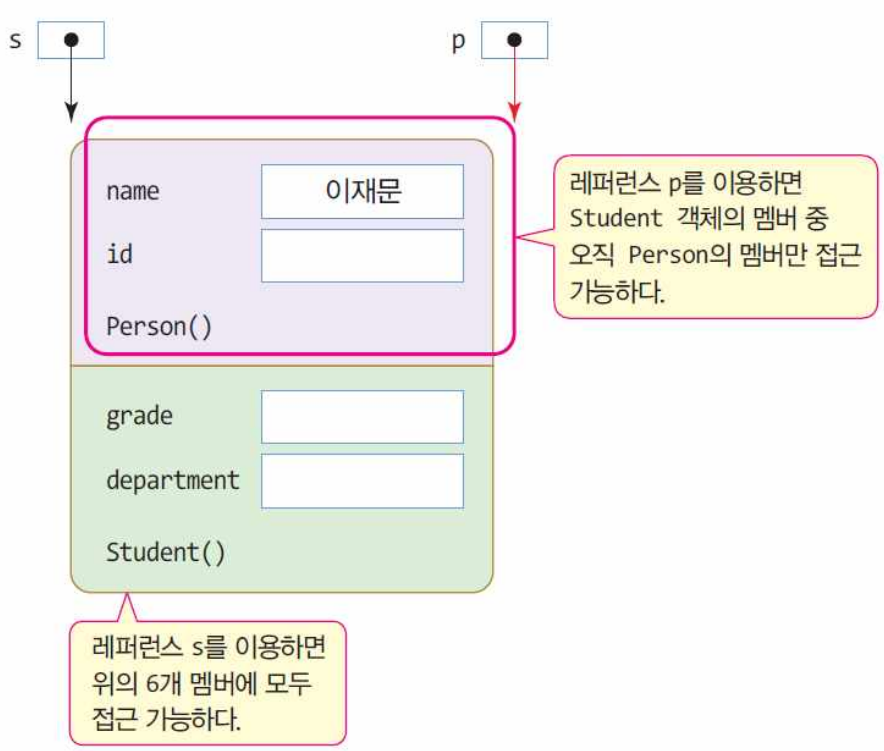
- 서브 클래스의 레퍼런스를 슈퍼 클래스 레퍼런스에 대입
- 슈퍼 클래스 레퍼런스로 서브 클래스 객체를 가리키게 되는 현상
- 슈퍼클래스의 필드와 메소드에만 접근 가능
  - 메소드가 서브클래스에서 오버라이딩 되었다면 서브클래스의 메소드가 대신 호출된다.

```
class Person { }
class Student extends Person { }

Person p;
Student s = new Student();
p = s; // 업캐스팅
```

예제

```
class  Person {
    String name;
    String id;

    public Person(String name) {
        this.name = name;
     }
}

class  Student extends Person {
    String grade;
    String department;

    public Student(String name) {
        super(name);
     }
}

public class UpcastingEx {
    public static void main(String[ ] args) {
        Person  p;
        Student s = new Student("이재문");
        p = s; // 업캐스팅 발생

        System.out.println(p.name); // 오류 없음

        p.grade = "A"; // 컴파일 오류
        p.department = "Com"; // 컴파일 오류
    }
}
```

오류



슈퍼클래스의 필드와 메소드에만 접근 가능

이재문

실습

- package : package : javab.inherit.upcasting_1

- MP3Player 클래스를 이용 mp3p 객체를 생성후, Player 클래스 객체 p에 업캐스팅
- 업캐스팅된 p에서 Player의 play() 메소드, MP3Player의 download() 호출

다운캐스팅

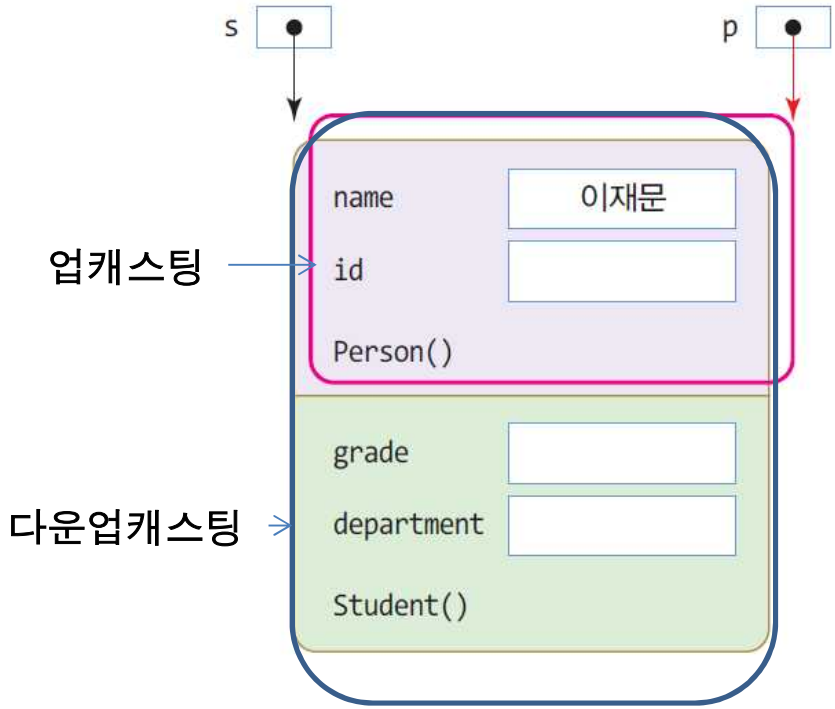
❖ 다운캐스팅(downcasting)

- 서브클래스가 슈퍼 클래스로 타입변환후 다시 서브클래스로 변환될때 사용
- 반드시 명시적 타입 변환 지정
- 서브클래스의 필드 메소드 모두 사용가능
- 업캐스팅으로 객체 사용시 sub class의 필드나 메소드를 사용하고 싶을 경우 상용됨

```
class Person { }
class Student extends Person { }

Person p = new Student("이재문"); // 업캐스팅

Student s = (Student)p; // 다운캐스팅, 강제타입변환
```

예제

```
class  Person {
   String name;
   String id;
   public Person(String name) {
      this.name = name;
    }
}

class  Student extends Person {
   String grade;
   String department;
   public Student(String name) {
      super(name);
    }
}
public class DowncastingEx {
   public static void main(String[] args) {
      Person p = new Student("이재문"); // 업캐스팅
      Student s;

      s = (Student)p; // 다운캐스팅

      System.out.println(s.name); // 오류 없음
      s.grade = "A"; // 오류 없음
   }
}
```



이재문

실습

- package : package : javab.inherit.downcasting_1

- MP3Player 클래스를 이용 mp3p 객체를 생성후, Player 클래스 객체 p에 업캐스팅
- MP3Player 클래스 mp3Smart 선언 후 p를 mp3Smart에 다운캐스팅
- 업캐스팅된 mp3Smart 에서 Player의 play() 메소드, MP3Player의 download(), onLinePlay() 호출

4절. 메소드 재정의(Override)

❖ 메소드 오버라이딩(Method Overriding)

- 서브 클래스에서 슈퍼 클래스의 메소드 중복 작성
- 항상 상속시 서브 클래스에 오버라이딩한 메소드가 실행
- 하나의 메소드명에 서로 다른 구현이 가능
- 슈퍼 클래스의 메소드를 서브 클래스에서 각각 목적에 맞게 다르게 구현

❖ 오버라이딩 조건

- 슈퍼 클래스 메소드의 원형(메소드 이름, 인자 타입 및 개수, 리턴 타입) 동일하게 작성
- 구현부만 틀림

오버로딩과 오버라이딩

비교 요소	메소드 오버로딩	메소드 오버라이딩
선언	같은 클래스나 상속 관계에서 동일한 이름의 메소드 중복 작성	서브 클래스에서 슈퍼 클래스에 있는 메소드와 동일한 이름의 메소드 재작성
관계	동일한 클래스 내 혹은 상속 관계	상속 관계
목적	이름이 같은 여러 개의 메소드를 중복 선언하여 사용의 편리성 향상	슈퍼 클래스에 구현된 메소드를 무시하고 서브 클래스에서 새로운 기능의 메소드를 재정의하고자 함
조건	메소드 이름은 반드시 동일함. 메소드의 인자의 개수나 인자의 타입이 달라야 성립	메소드의 이름, 인자의 타입, 인자의 개수, 인자의 리턴 타입 등이 모두 동일하여야 성립
바인딩	정적 바인딩. 컴파일 시에 중복된 메소드 중 호출되는 메소드 결정	동적 바인딩. 실행 시간에 오버라이딩된 메소드 찾아 호출

예제

```
public class BaiscCalculator
{
	public String calName = "pCal";
	public int add(int x, int y)
	{
	   System.out.println("add()");
	   int resultAdd=x+y;
	   return resultAdd;
	}
	public int sub(int x, int y)
	{
	   System.out.println("sub");
	   int resultAdd=x-y;
	   return resultAdd;
	}
	public int mul(int x, int y)
	{
	   System.out.println("mul()");
	   int resultAdd=x-y;
	   return resultAdd;
	}
	public double div(int x, int y)
	{
	   double ret=0;
	   ret=x/y;
	   return ret;
	}
	public void printValues(int x, int y)
	{
	   System.out.println("x : "+x+"y : "+y);
	}
}
```

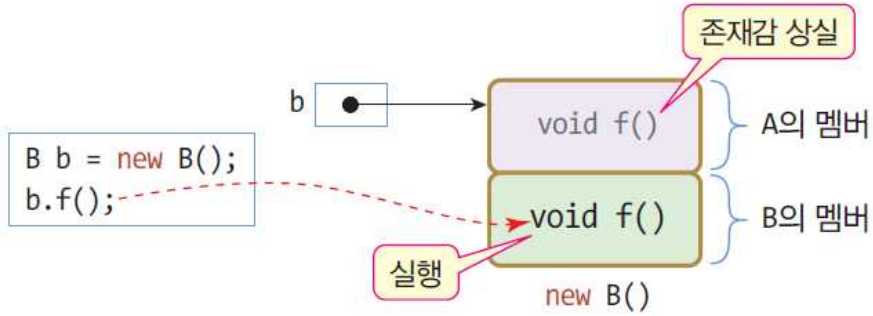
```
public class EnginneringCalculator extends
BaiscCalculator
{
   public void printValues(int x, int y)
   {
      String hx = Integer.toHexString(x);
      String hy = Integer.toHexString(y);
      System.out.println("hex x : "+hx);
      System.out.println("hex y  : "+hy);
   }

}
```

```
public class MainApp extends BaiscCalculator
{
   public static void main(String[] args)
   {
      EnginneringCalculator ch = new
EnginneringCalculator();
      ch.printValues(15, 9);
   }

}
```

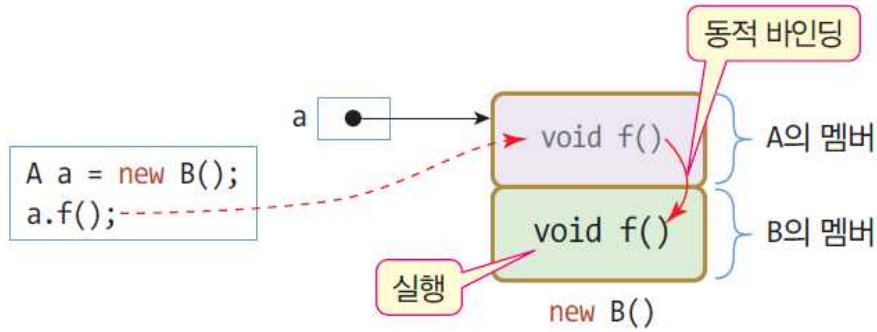
4절. 메소드 재정의(Override)

```
class A {
   void f() {
      System.out.println("A의 f() 호출");
   }
}
class B extends A {
   void f() {
      System.out.println("B의 f() 호출");
   }
}
```

(a) 오버라이딩된 메소드, **B의 f() 직접 호출**



(b) **A의 f()를 호출해도,** 오버라이딩된 메소드, **B의 f()가 실행됨**



동적바인딩 , 정적바인딩

- 정적바인딩(컴파일시 바인딩)
  - 참조변수와 인스턴스를 컴파일때 연결하는 것이라 컴파일시에 모든 연결(호출)이 결정
  - 실행시에 연결을 변경할 수 없음
- 동적바인딩(실행시 바인딩)
  - 프로그램 실행중 함수가 호출될 때 그 메모리 참조를 알아내는 것을 뜻함.
  - 실행시에 참조변수와 인스턴스등을 연결
    - 정적바인딩 대비 굉장한 유연성을 갖게 됨

동적 바인딩 – 오버라이딩된 메소드 호출

정적바인딩

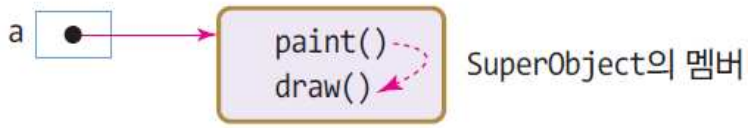
```
class SuperObject {
    protected String name;
    public void paint() {
        draw();
    }
    public void draw() {
        System.out.println("Super Object");
    }
}
public class SubObject extends SuperObject {
    public static void main(String [] args) {
        SuperObject b = new SubObject();
        b.paint();
    }
}
```

동적바인딩

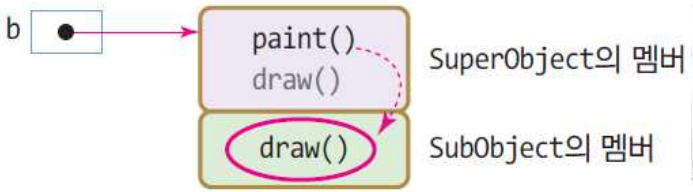
```
class SuperObject {
    protected String name;
    public void paint() {
        draw();
    }
    public void draw() {
        System.out.println("Super Object");
    }
}
public class SubObject extends SuperObject {
    public void draw() {
        System.out.println("Sub Object");
    }
    public static void main(String [] args) {
        SuperObject b = new SubObject();
        b.paint();
    }
}
```

동적바인딩

Super Object



Sub Object



예제

Shape의 draw() 메소드를 Line, Circle, Rect 클래스에서 목적에 맞게 오버라이딩하는 다형성의 사례를 보여준다.

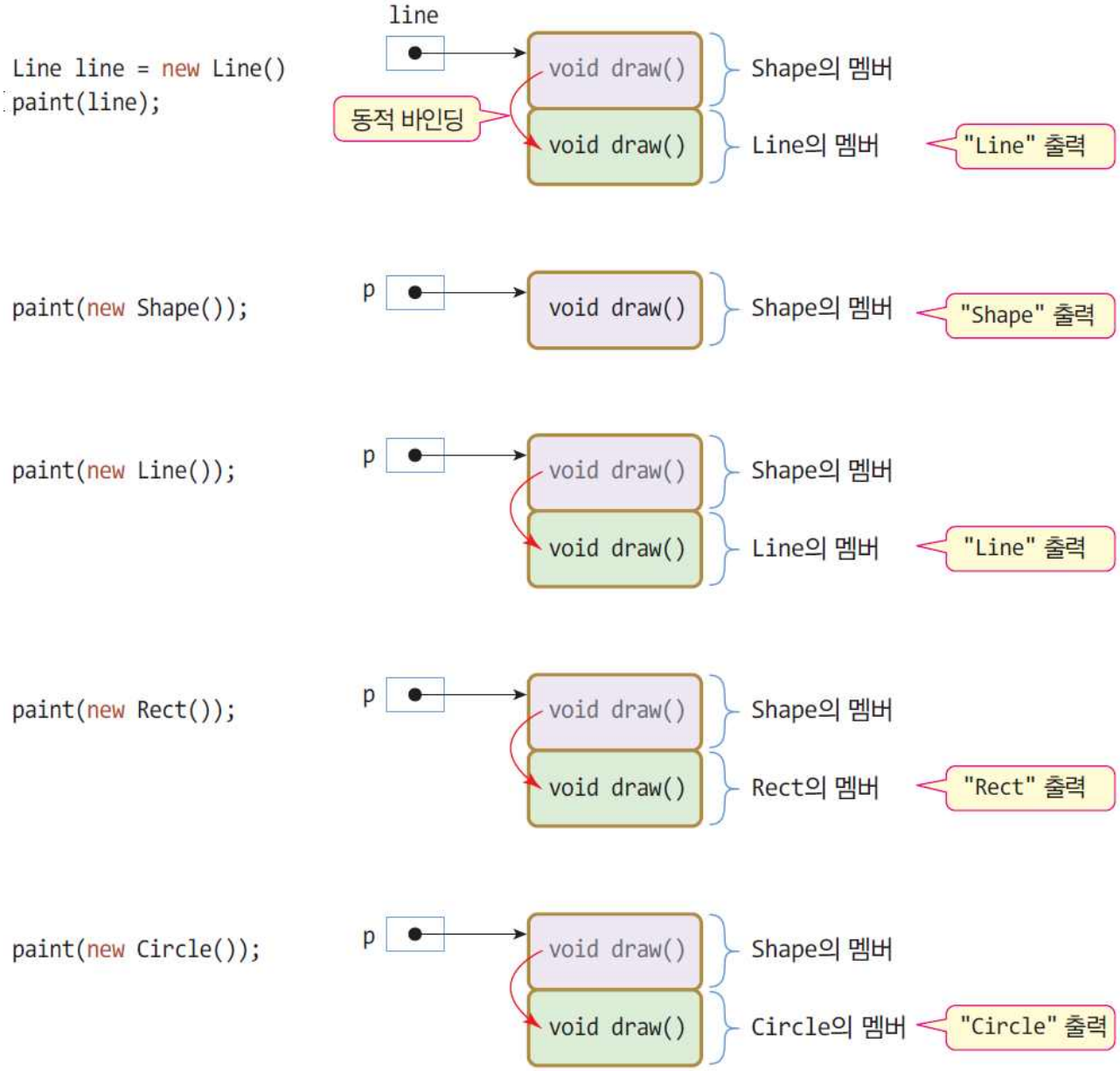
```
class Shape { // 도형의 슈퍼 클래스
    public void draw() {
        System.out.println("Shape");
    }
}
class Line extends Shape {
    public void draw() {
        System.out.println("Line");
    }
}
class Rect extends Shape {
    public void draw() {
        System.out.println("Rect");
    }
}
class Circle extends Shape {
    public void draw() {
        System.out.println("Circle");
    }
}
```

동적바인딩

```
public class MethodOverridingEx {
    static void paint(Shape p) { // Shape을 상속받은 객체들이
                                 // 매개 변수로 넘어올 수 있음
        p.draw(); // p가 가리키는 객체에 오버라이딩된 draw() 호출.
                  // 동적바인딩
    }

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

    }
```

```
Line
Shape
Rect
Circle
```

예제


Line line = new Line()
paint(line);
line
void draw()
Shape의 멤버
동적 바인딩
void draw()
Line의 멤버
"Line" 출력
paint(new Shape());
p
void draw()
Shape의 멤버
"Shape" 출력
paint(new Line());
p
void draw()
Shape의 멤버
void draw()
Line의 멤버
"Line" 출력
paint(new Rect());
p
void draw()
Shape의 멤버
void draw()
Rect의 멤버
"Rect" 출력
paint(new Circle());
p
void draw()
Shape의 멤버
void draw()
Circle의 멤버
"Circle" 출력

실습

- javab.inherit.methOverride_1
- javab.inherit.basic_1 에 다음과 같은 소스 추가
  - MP3Player 클래스에 play() 메소드 추가
    - onLinePlay() 호출
  - MainClass 클래스
    - 업캐스팅을 이용 Play 클래스의 객체 생성
    - p.play();

실습

- javab.inherit.methOverride_2
- javab.inherit.methOverride_1 에 위의 예제와 같이 하나의 인터페이스 메소드를 통해 호출

실습

- package : javab.inherit.methodoverriding_3
- 메소드 오버라이딩을 이용 PhoneCall4G의 각 메소드들을 호출
- PhoneCall2G
  - int call(String num);
  - int disConnectCall();
  - int sendSMS(String pNum);
- PhoneCall4G
  - int call(String pNum);
  - int disConnectCall();
  - int sendSMS(String pNum);
  - public int callVideo(String pNum);

5절. final 클래스와 final 메소드

❖ final 키워드의 용도

- final 필드: 수정 불가 필드
- final 클래스: 부모로 사용 불가한 클래스
- final 메소드: 자식이 재정의할 수 없는 메소드

❖ 상속할 수 없는 final 클래스

- 자식 클래스 만들지 못하도록 final 클래스로 생성

```
public final class 클래스 { ... }
```

```
public final class String { .. }
```

```
public class NewString extends String { ... }
```

❖ 오버라이딩 불가한 final 메소드

- 자식 클래스가 재정의 못하도록 부모 클래스의 메소드를 final로 생성

8절. 추상 클래스(Abstract Class)

❖ 추상 메소드(abstract method)

- abstract로 선언된 메소드 : 구현부가 없고 원형만 선언

```
public abstract String getName(); // 추상 메소드
```

오류

```
public abstract String fail() {
                return "Good Bye";
} // 추상 메소드 아님. 컴파일 오류
```

❖ 추상 클래스(abstract class)

- 추상 메소드를 가지며, abstract로 선언된 클래스
- 추상 메소드 없이, abstract로 선언한 클래스

```
// 추상 메소드를 가진 추상 클래스
abstract class Shape {
   Shape() { … }
   void edit() { … }

   abstract public void draw(); // 추상 메소드
}
```

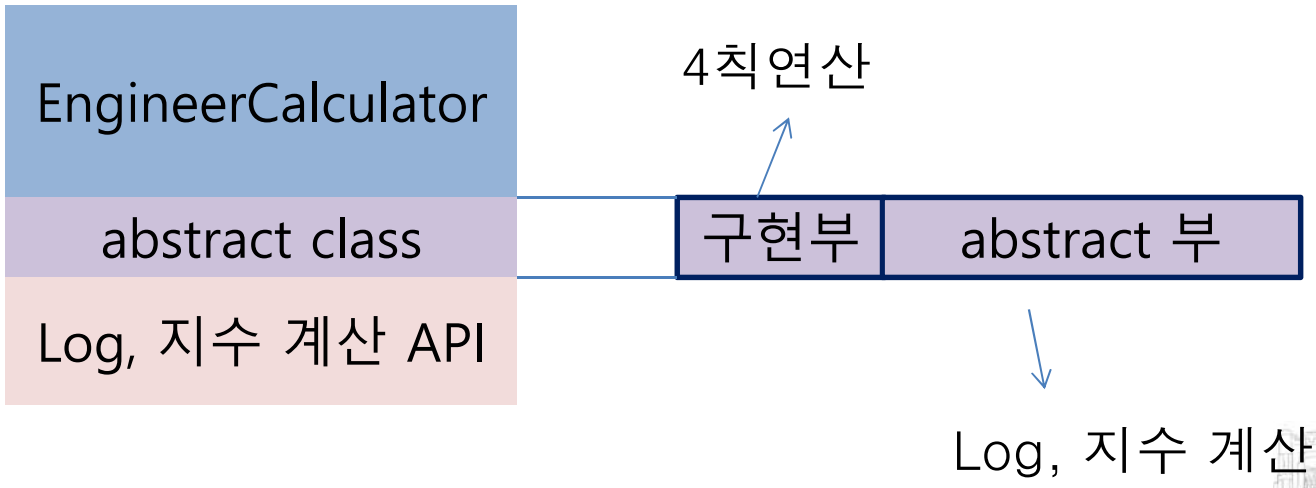
오류

```
class fault { // 오류. 추상 메소드를 가지고 있으므로 abstract로 선언되어야 함
   abstract void f(); // 추상 메소드
}
```

8절. 추상 클래스(Abstract Class)

❖ 추상 클래스의 용도

- 구현 클래스 설계 규격을 만들고자 할 때
  - 구현 클래스가 가져야 할 필드와 메소드를 추상 클래스에 미리 정의
  - 구현 클래스의 공통된 필드와 메소드의 이름 통일할 목적
- 전체 기능중 일부 기능이 달라질수 있을 경우



8절. 추상 클래스(Abstract Class)

❖ 추상 클래스 선언

- 클래스 선언에 abstract 키워드 사용
  - New 연산자로 객체 생성하지 못하고 상속 통해 자식 클래스만 객체 생성 가능

```
public abstract class 클래스 {
  //필드
  //생성자
  //메소드
}
```

❖ 추상 클래스는 온전한 클래스가 아니기 때문에 인스턴스를 생성할 수 없음

```
JComponent p;                  // 오류 없음. 추상 클래스의 레퍼런스 선언
p = new JComponent();          // 컴파일 오류. 추상 클래스의 인스턴스 생성 불가
Shape obj = new Shape();       // 컴파일 오류. 추상 클래스의 인스턴스 생성 불가
```

오류

컴파일 오류 메시지

Unresolved compilation problem: Cannot instantiate the type Shape

8절. 추상 클래스(Abstract Class)

❖ 추상 메소드와 오버라이딩(재정의)

- 메소드 이름 동일하지만, 실행 내용이 실체 클래스마다 다른 메소드
- 구현 방법
  - 추상 클래스에는 메소드의 선언부만 작성 (추상 메소드)
  - 실체 클래스에서 메소드의 실행 내용 작성(오버라이딩(Overriding))

예제

```
public abstract class Phone {
    //필드
    public String owner;


    //메소드
    public void turnOn() {
        System.out.println("폰 전원을 켭니다.");
    }
    public void turnOff() {
        System.out.println("폰 전원을 끕니다.");
    }

    public abstract String saveContact(String phoneNum);
}
```

```
public class NaverContact extends Phone {
    String pb=null;

    public String saveContact(String phoneNum)
    {
        System.out.println("naver contact added : "+phoneNum);
        this.pb+=phoneNum;
        return this.pb;
    }
}
```

```
public class GoogleContact extends Phone {
    String pb=null;

    public String saveContact(String phoneNum)
    {
        System.out.println("google contact added : "+phoneNum);
        this.pb+=phoneNum;
        return this.pb;
    }
}
```

```
public class PhoneExample {
    public static void main(String[] args) {
        //Phone phone = new Phone(); (x)

        Phone smartPhone = new GoogleContact();
        //Phone smartPhone = new NaverContact();
        String pNum="010-2392-2343";
        String retVal=null;
        smartPhone.turnOn();
        retVal=smartPhone.saveContact(pNum);
        System.out.println(retVal);
        smartPhone.turnOff();
    }
}
```

실습

- package : javab.inherit.abstractClass_1

- BasicPhone
  - int call(String num){}
  - int disConnectCall(){}
  - int sendSMS(String pNum){}
  - int receiveSMS(){}
  - abstract : int sendData4G(String data);
  - abstract : int videoCall(String num);

- Phone를 상속받아 abstract 메소드를 구현한다.
  - Phone4G

Thank You !

이것이 자바다(http://cafe.naver.com/thisjava)