JAVA PROGRAMING

상속

기능으로 보는 상속

기존의 코드를 물려받아서 하나의 객체가 손쉽게 새 기능을 추가하는 기법

클래스로 보는 상속

하나의 객체가 다른 클래스의 정보를 가지고 있어서 객체의 type을 유연하게 사용하는 기법

- extends 키워드를 통한 기존 클래스에 새로운 변수나 method를 추가

```
public class AClass {

    public void doA(){
        System.out.println("AAAA");
    }
}
```

```
public class BClass extends AClass {

    public void doB(){
        System.out.println("BBBB");
    }
}
```

상속

AClass의 객체

BClass의 객체

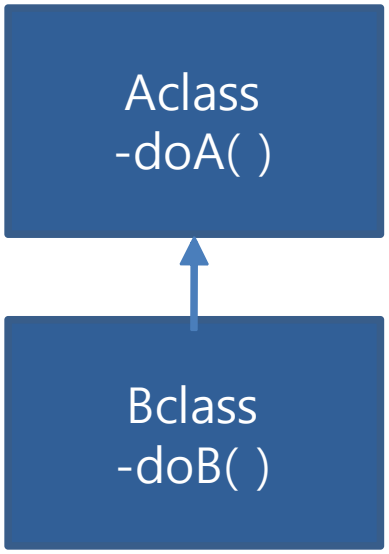
```
Bclass obj = new Bclass( );
```

부모클래스의 객체가 있어야만 자식 클래스의 객체도 온전히 동작하기 때문에 실제로는 두 개의 객체가 생성되는 방식

자식 클래스의 객체에서 부모 클래스의 객체를 찾아가는 묵시적인 변수 – super

super는 자식 클래스에서만 사용할 수 있는 부모 객체의 레퍼런스(리모컨)
- this와 비교

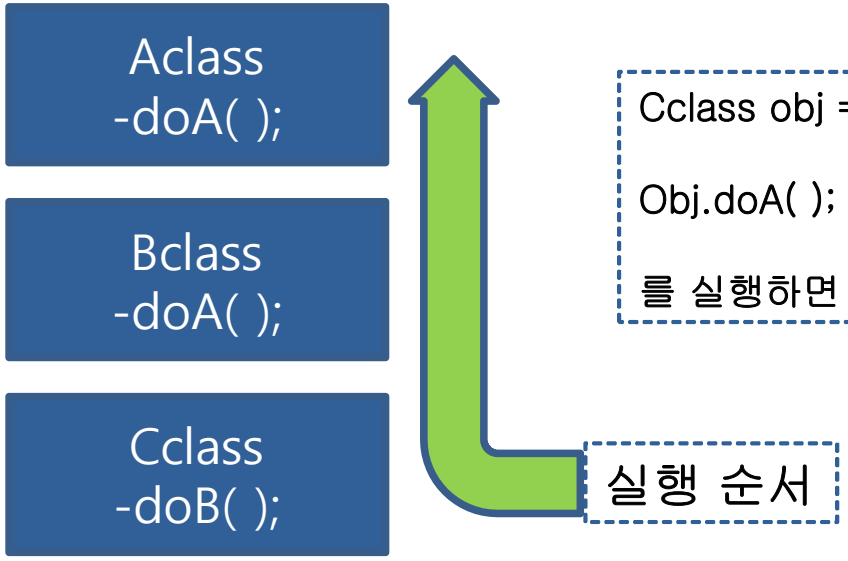
상속




```
BClass obj = new BClass();

obj.doA();
```

현재 객체가 만들어진 클래스의 메소드 확인

없으면 부모 클래스의 메소드 확인



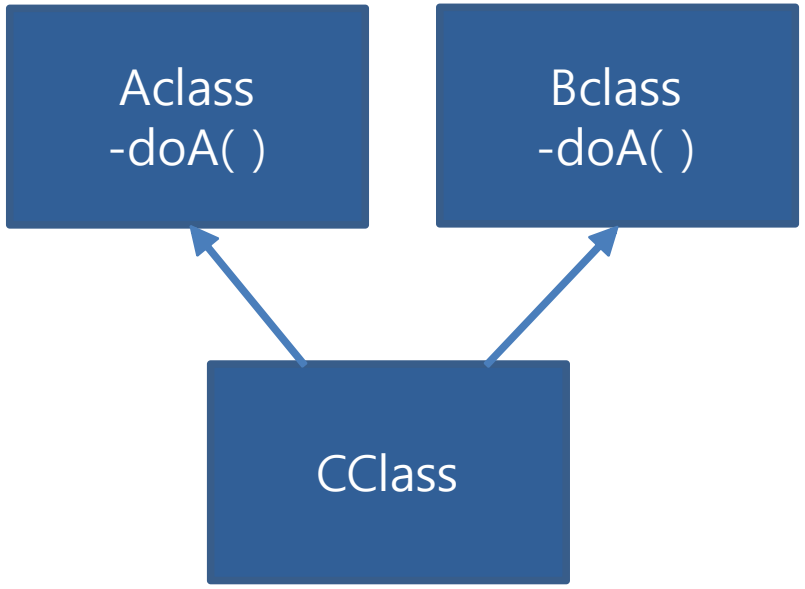

```
Cclass obj = new Cclass( );

Obj.doA( );
```

를 실행하면 어떤 일이 일어 날까?

상속

Java와 단일 상속

하나의 클래스가 두 개 이상의 클래스를 상속하는 경우에 발생하는 문제를 막기 위해서 Java에서는 단일 상속만 지원



Cclass의 doA( )를 호출하면 어떤 부모의 doA( )를 호출해야 하는가?

상속

변수의 타입으로 보는 상속

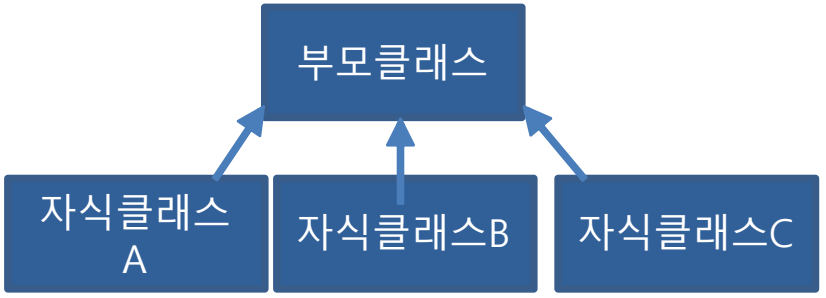
Aclass
-doA( )

Bclass
-doB( )

Bclass 에서 나온 객체는 Bclass라는 정보를 가지는 동시에

Aclass의 정보가 같이 가지고 있게 된다.

따라서 Aclass obj = new Bclass( );와 같은 선언이 가능하다.



어떤 메소드가 부모타입의 파라미터를 받는다면?

부모클래스[ ] arr = new 부모클래스[10];

상속

컴파일러와 실행 객체의 차이

컴파일러가 보는 정보는 변수가 선언될 때의 타입

반면에 실행될 때에는 현재 실행중인 객체의 메소드

컴파일시와 런타임시점에 차이가 존재하는 방식

접근제한자에 대한 환상(?)

접근 제한자는 객체의 정보를 은닉하기 위해서 사용한다?
- 객체가 달라도 동일한 클래스에서는 데이터에 접근할 수 있게 된다.

접근 제한이란 객체의 클래스 정보를 이용하는 필터링(filtering)

접근제한을 정확한 의미에서 보면 객체가 가진 클래스의 정보(패키지이름+클래스의 이름)를 이용해서 변수나 메소드를 사용하는 것을 제한하는 방법일 뿐

private --> 패키지 명과 클래스 명이 전부 일치하는 경우에만 사용가능
default --> 패키지이름 정보만 맞으면 사용가능
public --> 아무 정보가 일치하지 않아도 됨

상속

상속과 접근제한 (protected)

상속의 경우는 자식 클래스의 객체를 만드는 경우에 부모 클래스의 객체도 같이 만드는 방식
따라서 하나의 객체는 해당 클래스의 정보 + 부모 클래스들의 정보를 가지는 구조

protected --> 특정 객체의 정보와 현재 객체의 정보중에 부모클래스의 정보가 일치하면 접근을 허락하는 방식

일반 상속과 추상 클래스

오버라이드가 필요없으면 일반 상속 사용
- 하위에서 반드시 오버라이드 해야하는 내용이 있다면 추상클래스 형식이 좋음

부모 클래스도 객체화 될 때만 일반 상속 사용
- 부모 클래스도 객체로 갈 것이 확실하다면 그 때는 일반 상속 사용

상속은 일반화 이다.

일반적이고 공통적인 것들을 모아 추상화 시킨다는 개념. 즉, 부모클래스를 정의하고 하위 클래스를 정하는 것이 아닌, 여러 개의 클래스를 보면서 상속 구조로 변경하는 것이 더 정확하다.

상속

```
import java.io.*;

class MyPoint {
    private int x;
    private int y;
    protected static BufferedReader in;
    static {
        in = new BufferedReader(new InputStreamReader(System.in));
    }

    protected MyPoint() throws IOException{
        System.out.print("x = ");
        this.x = Integer.parseInt(in.readLine());
        System.out.print("Y = ");
        this.y = Integer.parseInt(in.readLine());
    }

    protected void disp(){
        System.out.println();
        System.out.println("점(x, y) = (" + this.x + "," + this.y + ")");
    }
}

class Circle extends MyPoint{
    private int r;
    public Circle() throws IOException{
        super();    // MyPoint() 호출, 생략 가능
        System.out.print("r = ");
        this.r = Integer.parseInt(in.readLine());
    }

    public void disp(){
        super.disp();
        System.out.println("반지름 r = " + this.r);
    }
}
```

상속

```
class Rect extends MyPoint{
    private int w;
    private int h;
    public Rect() throws IOException{
        super();    // MyPoint() 호출, 생략 가능
        System.out.print("w = ");
        this.w = Integer.parseInt(in.readLine());
        System.out.print("h = ");
        this.h = Integer.parseInt(in.readLine());
    }
    public void disp(){
        super.disp();
        System.out.println("폭 = " + this.w + ", 높이 " + this.h);
    }
}

public class Round13_Ex07 {
    public static void main(String[] ar) throws IOException{
        Circle cc = new Circle();
        cc.disp();
        System.out.println();
        System.out.println();
        Rect rt = new Rect();
        rt.disp();
    }
}
```

추상클래스

override를 적극 활용하는 추상클래스

상속의 장점 - 실제로 존재하는 데이터와 로직을 코드로 물려준다는 장점
override의 장점 - 컴파일러는 클래스의 타입만을 보기 때문에 실제 객체가 동일한 메소드만 가지고 있다면 동작시킬수 있다.

상속의 기능 + 적극적인 override = 추상 메소드를 가진 추상클래스

추상메소드(abstract method)란?

override를 목적으로 부모클래스에 메소드의 형태만 작성해 둔 메소드

```
public abstract doA( );
```

override를 할 목적 & 컴파일러를 속이는 목적
- 문법적으로 하위 클래스들은 강제적인 override가 보장된다.

추상메소드의 접근제한자

추상메소드는 패키지정보가 같거나, 동일한 메타데이터(클래스정보)를 가지고 있거나, 외부에서 접근이 가능해야 한다.

추상 메소드가 있으면 반드시 추상 클래스이므로 패키지 정보 + 클래스명이 일치하는 private 인 사용할 수 없다.

추상클래스

추상클래스는 언제 사용해야 할까?

90%의 상속 + 10%의 오버라이딩을 해야 하는 경우
- 여러 메소드들 중에서 일부분만 매번 다르게 작성해야 하는 경우

유사한 클래스들을 보고 상위 클래스를 추출하는 경우
- 일반화(Generalization)에 의한 공통 데이터나 로직 추출 후에 부모 클래스가 직접 객체 생성을 해야 하는 필요는 없는 경우

추상 클래스는 객체 생성을 안할까?

상속이란 부모클래스의 객체 생성 역시 필요하다.
그렇다면 추상클래스에서도 부모 클래스를 만들까?

```
public class SuperA {
        {
                System.out. println("super a ");
        }
}
```

```
public class SubA extends SuperA {
        {
                System .out.println("sub a
");
        }
        public static void main(String[] args) {
                SubA obj1 = new SubA();
                SubA obj2 = new SubA();
        }
}
```

java.lang.Object

Java와 C++상속의 차이

Java는 C++과는 달리 클래스의 다중 상속을 허용하지 않는다.

클래스의 다중상속의 문제를 막기위한 장치
- 다중상속의 애매모호한 문제

모든 클래스는 extends뒤에 하나의 클래스만 상속할 수 있는 구조

java.lang.Object클래스

모든 클래스들의 암묵적인 부모클래스

모든 객체들을 하나의 타입으로 묶기 위한 설계

모든 객체들이 공통적으로 사용되는 자료구조 혹은 운영체제의 스레드 관련 메소드가 미리 정의되어 있다.

가장 중요한 사실은 JVM이 객체 생성이나 소멸시에 알아야 하는 동작을 물려주는 기능을 하기 위해서 설계되어 있다.

java.lang.Object

java.lang.Objet의 toString( )

System.out.println( )으로 출력되는 결과물의 실제 호출되는 메소드

override해 두면 간편하게 원하는 정보를 출력하는 데 사용할 수 있다.

java.lang.Object의 hashcode( )메소드

Java언어의 레퍼런스(리모컨)는 C언어의 포인터와는 달리 프로그램 내에서 직접 포인터를 핸들링 할 수 없다.

흔히 보는 '@'뒤로 표시되는 값은 16진수는 hashcode값이다.

Java에서 제공되는 자료구조들이 객체를 보관할때 hashcode값을 사용하는 경우가 있다.

java.lang.Object

hashcode값은 주소값이 아니다!

C언어의 주소값과 유사하게 JVM내에 특정 객체에 대한 번호(메모리 위치 값)을 저장해 둔 것을 객체의 hashcode라고 한다.

해당 클래스에서 override하지 않으면 기본적으로 마치 주소값처럼 모든 객체가 다른 hashcode값을 가지게 된다.

클래스에서 override하게 되면 다른 객체라고 해도 동일한 hashcode값을 가지게 된다. 따라서 C언어의 주소값과 혼동하면 안된다.

java.lang.Object의 eqauls( )

원래의 코드는 '=='과 동일한 기능

override를 통해서 동일한 객체가 아니라고 해도 같은 결과가 필요로 하는 경우에 사용

java.lang.Object

예제) toString()을 오버라이드
 - toString() 매소드는 System.out.println()에서 사용됨

```
public class SampleObj {
	public String toString(){
		return "AAA";
	}

	public static void main(String[] args) {
		SampleObj obj = new SampleObj();
		System.out.println(obj);
	}
}
```

```
AAA
```

숙제) 임의의 숫자를 발생시키는 Random()매소드를 상속하여 랜덤하게 만들어진 값 + 100을 돌려주는 매소드를 만드시오.

인터페이스

다양한 도형의 넓이를 구한다면?

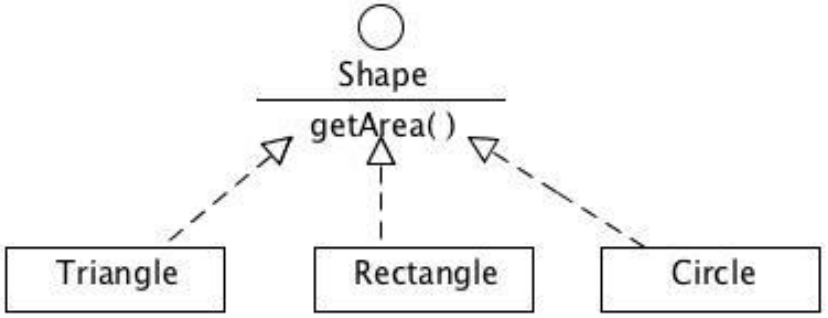
C언어 방식이라면 if ~ else

```
삼각형: 밑변, 높이
사각형: 가로, 세로
원: 반지름
타원: 가로지름, 세로지름
사다리꼴: 윗변, 밑변, 높이
```

Java라면 상속?

Interface 위주의 설계

Interface는 데이터 위주의 설계 방식이 아닌 스펙(spec)을 정하는 '큰그림(big picture)'을 먼저 잡는 방식의 설계



인터페이스

상속이 걸맞지 않는 오버라이딩

상속은 부모클래스의 모든 데이터와 로직을 물려주는 구조이므로, 자식 클래스의 데이터와 상호호환이 되지 않는 경우에는 상속을 사용할 필요가 없게 된다.

데이터와 로직을 물려주지 않는 독립적으로 구현하되 다형성을 만족시키는 상황에서 Interface는 좋은 대안이 된다.

Interface는 객체가 아니다!

Interface라는 개념 자체가 추상화된 개념

객체와 객체 사이에 있는 통신하는 장치

인터페이스

Interface를 먼저 선정한다.

클래스를 만들기 전에 해당 클래스가 가져야 하는 기능을 먼저 Interface로 설계한다.

Interface의 추상 메소드를 설계하는 것은 마치 사물에 대한 스케치를 하는 작업과 비슷하다.

Interface를 사용하는 코드 연습

처음에는 로직 위주의 클래스만이라도 Interface를 적용하는 연습

메소드의 파라미터나 리턴타입을 실제 클래스타입으로 설계하지 않고, Interface타입으로 설계해 보자.

Interface를 사용해도 피할 수 없는 일

아무리 Interface를 사용해도 객체 생성하는 부분은 피해 갈 수 없다.

가능하면 하나의 객체가 다른 객체를 만들어 내는 코드를 작성하지 않도록 한다면 좀더 재사용을 높일 수 있다.

인터페이스

예제)

```
package suture.character;

public interface Character {

}
```

클래스 대신 interface가 들어간것만 빼면 클래스와 비슷

```
package suture.character;

public interface Character {
    public String getName();
}
```

매소드 하나 선언

```
package suture.character;

public class NonPlayerCharacter implements Character {

}
```

실행 시 에러

인터페이스

```
package suture.character;

public class NonPlayerCharacter implements Character {
    String name;

    public String getName() {
        return this.name;
    }

}
```

따라서 위와 같이 해야 합니다.

인터페이스를 왜 설계도라고 했는지 이제 감이 오셨겠죠?
인터페이스에 정의한 메소드들은 해당 인터페이스를 사용하는 클래스에서 반드시 구현해야 합니다.

그리고 인터페이스는 여러개를 사용할 수가 있습니다.
인터페이스에 정의된 메소드는 모두 구현해야 합니다.

```
package suture.character;

public class NonPlayerCharacter implements Character, Monster {
    String name;

    public String getName() {
        return this.name;
    }

}
```

인터페이스

의존성 주입(Dependency Injection) 방식

객체와 객체의 연관 – 의존성

아무리 Interface를 이용해도 객체의 생성을 다른 객체가 담당하게 되면 결국 유연하지 못한 구조를 만들어 낸다는 의견에서 시작

하나의 객체는 다른 객체를 받아들이는 구조로만 설계해주고, 외부에서 실제 객체를 주입(inject)하는 방식의 설계 방법

의존성주입과 제 3의 객체

하나의 객체와 다른 객체의 의존
- 가게의 점원과 주방장의 관계

외부에서 특정 객체에게 주입해 줄 객체가 필요
- 가게의 매니저가 점원과 주방을 매칭시키는 행위