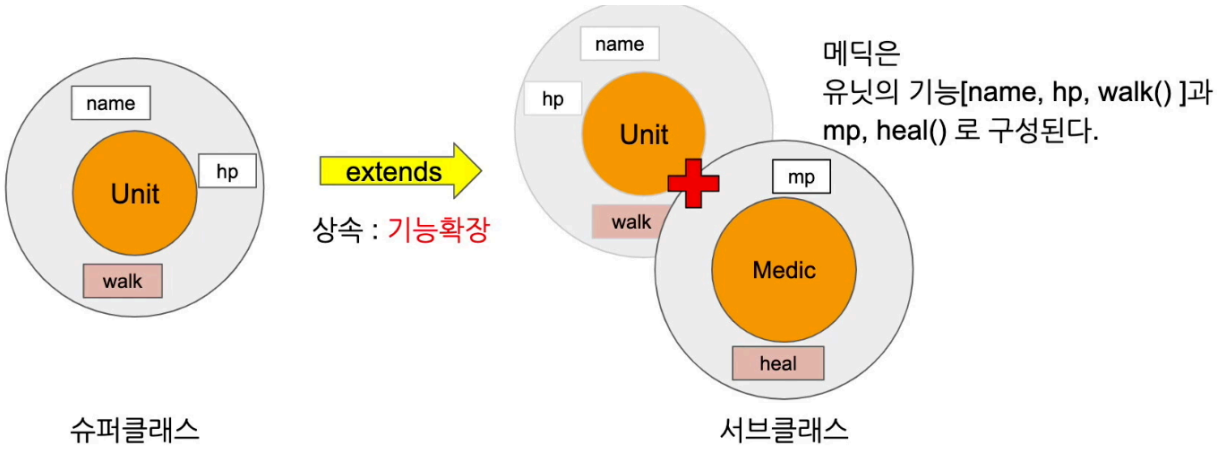
inflearn part2

상속

- 상위클래스의 모든 것이 하위클래스에게 전달되는 것을 뜻함.
- 하지만, 멤버변수와 멤버함수 중 private로 접근제한되면 하위클래스로 전달되지 않음.



- 상속의 장점
  - 재사용성
  - 확장 용이 : 새로운 클래스, 데이터, 메서드
- `extends` 사용, 다중상속은 허용되지 않음.

```
class Book
{
  String title;

  void printBook(){
    System.out.println("제 목 :" + title);
  }
}

//Book 상속받은 Novel
class Novel extends Book{
  String writer;  //추가적으로 저자

  void printNov(){ //부모클래스의 메서드

    printNov();
    System.out.println("저 자 : "+ writer);
  }
}

class Magazine extends Book{
```

```
  int day;

  void printMag(){
    System.out.println("발매일 : "+day + "일");
  }
}


public class Bookshelf{
  public class void main(String[] args){
    Novel nov = new Novel();
    nov.title = "홍길동전";
    nov.writer = "허균";

    Magazine mag = new Magazine();
    mag.title = "월간 자바";
    mag.day = 20;

    nov.printNov();
    System.out.println();
    mag.printMag();

  }
}
```

오버라이딩

- 상속된 메서드와 동일한 이름, 동일한 인수를 가진 메서드를 정의하여 `메서드를 덮어쓰는 것` 이다.
- 반환값의 형도 같아야 함.
- `하위 클래스에서 상위 클래스의 특정 메서드를 다시 정의`
  - 기능의 변경
  - 기능의 추가
- 확장설 실현하는 것에 기여.

오버라이딩

- 상속의 관계에서 발생한다.
- 위(부모)에서 아래(자식)로 연결된다는 점에서 '오버라이딩'의 'ㅏ' 로 위에서 아래로 긋는다는 이미지의 느낌으로 외운다.

오버로딩

- 한 클래스 내에서 동일한 이름의 메서드가 여러 개 존재한다.
- 모든 메서드는 수평적인 관계이므로 '오버로딩'의 'ㅗ'로 옆으로 긋는다의 이미지의 느낌으로 외운다.

```
class Animal{
  String name;
  int age;

  void printPet(){
    System.out.println("이름 : "+ name);
    System.out.println("나이 : "+ age);
  }
}

class Dog extends Animal{
  String variety;

  //함수의 오버라이딩
  void printPet(){ //부모에 있는 메서드명과 동일
    super.printPet(); // super. ->부모의 메서드 명 출력
    System.out.println("종류 :" + variety); //기능 추가
  }
}

public class Pet{
  public static void main(String[] args){
    Dog dog = new Dog();
    dog.name = "진돌이";
    dog.age = 5;
    dog.variety = "진돗개";
    dog.printPet();

  }
}
```

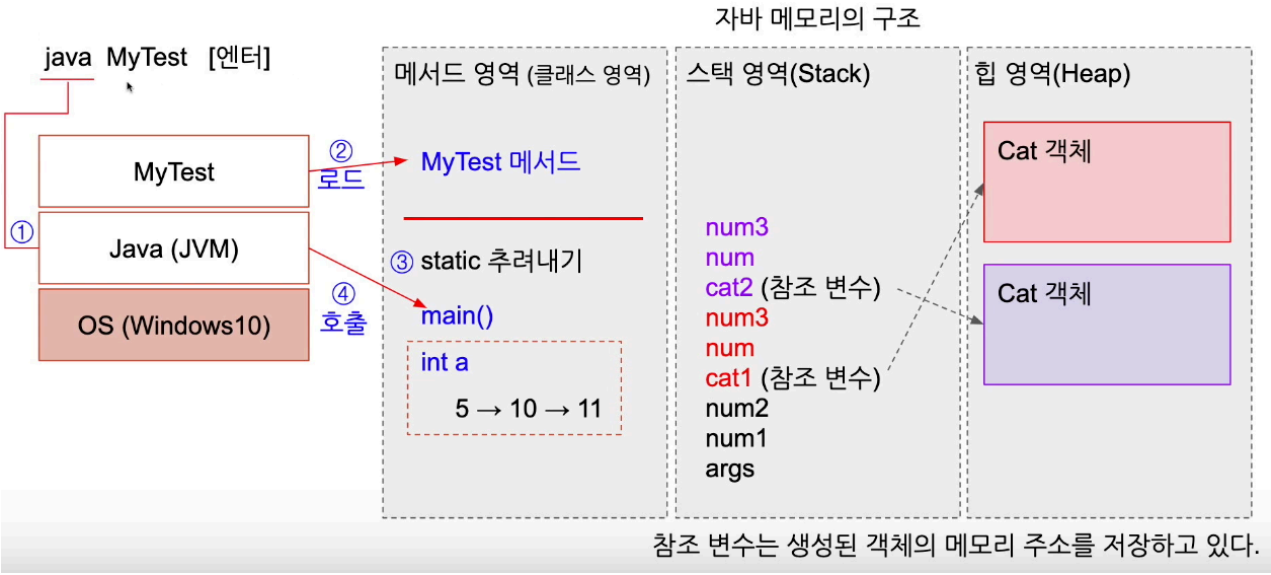
Static 의 이해

- `static` 필드와 메서드에 붙이는 제한자
- 다른 오브젝트에서 참조하기 위해서는 클래스명을 지정해야함.
- 메모리에는 만들어져 있는데, 객체를 생성하지 않아서, `클래스명.어쩌구` 로 불러와야함.
- 같은 클래스에서 생성된 오브젝트들은 static을 붙인 키워드 값을 붙여준 것이 좋음.

- Static 붙인 메서드의 성질
  - `오버라이딩 불가` : `유일하다`
  - 메서드 내에서 동일한 오브젝트 내의 멤버 이용하기 위해서는 필드, 메서드에게 static 붙여야 한다.

```java
// static의 이해

class Cat {
    static int a = 5;
    int num = 3;

    void print(int num3) {
        System.out.println("a:"+a);
        num = num3;
        System.out.println("num:"+num);
    }
}
```

```java
public class StaticEx1 {
    public static void main(String[] args) {
        int num1 = 5;
        int num2 = 2;
        System.out.println(num1 + ", " + num2);

        Cat cat1 = new Cat();
        cat1.num = 1;
        cat1.a = 10;
        cat1.print(20);
        System.out.println(cat1.num);
        System.out.println(cat1.a);

        Cat cat2 = new Cat();
        cat2.num = 2;
        cat2.a = 11;
        cat2.print(20);
        System.out.println(cat2.num);
        System.out.println(cat2.a);
        System.out.println(cat1.a);
    }
}
```



이미 프로그램이 시작도 전에 a 는 5라는 값을 가지고있음.

```java
import java.util.Random;

//인스턴스 생성과 관계 없이 static 변수가 메모리
//공간에 할당될 때 실행이 된다.

public class StaticEx2{
```

```
  static int num;

  //static 초기화 블록
  static{
    Random rand = new Random();
    num = rand.nextInt(100);
  }
  public static void main(String[] args){
    System.out.println(num);
  }
}
```

public static void main(String[] args) {...}

> static인 이유! 인스턴스 생성과 관계없이 제일 먼저 호출되는 메소드이다.

public static void main(String[] args) {...}

> public인 이유! main 메소드의 호출 명령은 외부로부터 시작되는 명령이다.
> 단순히 일종의 약속으로 이해해도 괜찮다.

추상클래스

- 처리 내용을 기술하지 않고, `호출하는 방법`만 정의한 메서드
  - 기능은 하지 않지만, 만들어야 할 메서드 기술해 놓음.
    - 나중에 취향에 맞게 구현한다.
    - 무조건 만들어야 해서 까먹을 일도 없음. 에러 뜨자나
- 한개라도 가지면 그 클래스는 `추상클래스`
- `abstract`

```
abstract class Animal{
  String name;
  abstract void cry();
}
```

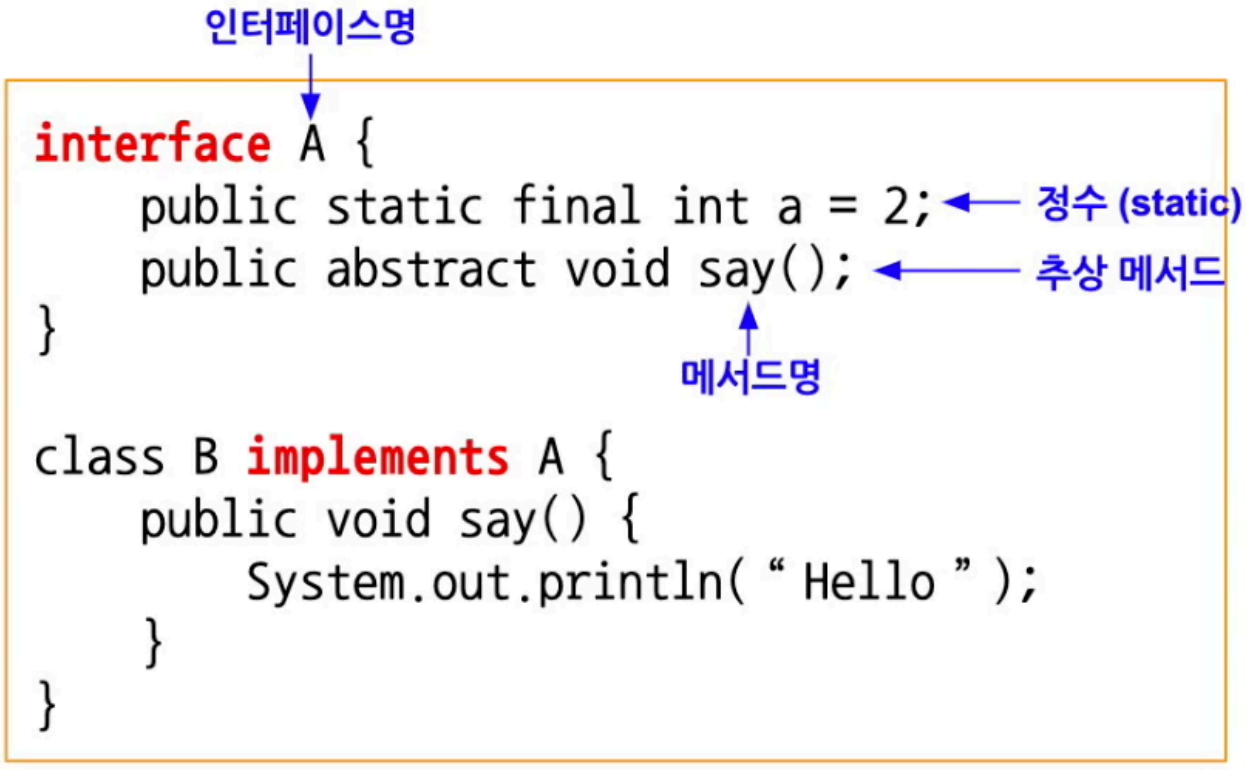
- `오버라이딩` 해서 구현해야 한다.

```
abstract class Animal{
  String name;
  int age;
```

```
  abstract void cry(); //호출만 한 것. 자기 상황에 맞게 바꿔주어야 함
}

class Dog extends Animal{

  @Override  //자동으로 어노테이션 생김.
  void cry(){
    System.out.println("멍멍"); //기능 이렇게 만들어주어야 한다.
  }
}

//사용
public class AbstractClassExam{
  public static void main(String[] args){
    Dog dog = new Dog();
    dog.cry();

  }
}
```

인터페이스

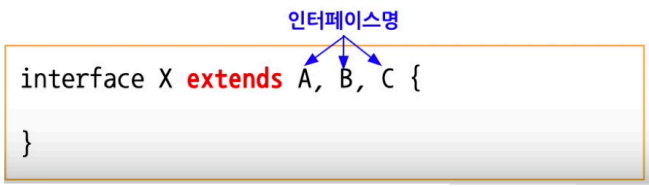
- 상속관계가 아닌 클래스에 기능을 제공하는 구조.
- `정의` 와 `추상 메서드` 만이 멤버가 될 수 있음.
- `implements`
- 여러개 구현 가능, 상속 가능.



인터페이스도 클래스처럼 상속할 수 있다.

```
interface A {
    void greeting();
}

interface B extends A {
    void goodbye();
}
```

복수의 인터페이스를 상속하여 새로운 인터페이스를 만들 수 있다.



상속은 단일 상속만 가능하다.

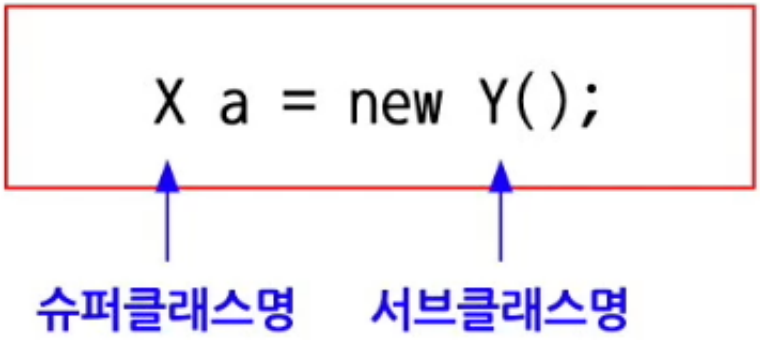
```
class X extends A {

}
```

- `extends` 와 `Implemets`

```
package step1;

interface Greet{
  void greet();
}

interface Bye{
  void bye();
}

class Morning implements Greet,Bye{

  //Greet, Bye 추상 메서드 구현되어야 함.
  public void bye(){
    System.out.println("안녕히 계세요.");
  }
  public void greet(){
    System.out.println("안녕하세요.");
  }
}

public class Meet{
  public static void main(String[] args){
    Morning morning = new Morning();
    morning.greet();
    morning.bye();
  }
}
```

다형성(폴리모피즘)

- 상속한 클래스의 오브젝트는 슈퍼클래스로도 , 서브클래스로도 다룰 수 있음.
  - **하위 클래스 객체를 상위클래스 객체에 대입하여 사용 가능**
- 하나의 오브젝트와 메서드가 `많은 형태`를 가지고 있는 것을 `다형성`이라 한다.
- `같은 이름이라도, 다른 형태 얻을 수 있는 것.`
- 서브클래스의 오브젝트는 슈퍼클래스의 오브젝트에 대입 할 수 있다.




```
abstract class Calc{
  int a = 5;
  int b = 6;

  abstract void plus();
}

class MyClac extends Calc{
  //추상메서드 오버라이딩 하기
  void plus(){
    System.out.println(a+b);
  }
  void minus(){
    System.out.println(a-b);
  }
}

public class Polymorphism1{
  public static void main(String[] args){
    myCalc myCalc1 = new MyCalc();
    myCalc1.plus();
    myCalc1.minus();

    //하위클래스 객체를 상위 클래스 객체에 대입
    Calc myCalc2 = new MyClac();
```

```
    myClac2.plus();
    //다음메서드는 설계도에 없다. 사용 불가능.
    //myClac2.minus();



  }
}
```

클래스 캐스팅 예외

```
class PBoard{}
class CBoard extends PBoard{}

  public class ClassCast{
    pulbic static void main(String[] args){
      pBoard sbd1 = new CBoard(); //서브클래스 만드는 것 슈퍼클래스에 대입
      CBoard sbd2 = (CBoard) sbd1; //Ok

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

      PBoard ebd1 = new PBoard();
      CBoard ebd2 = (CBoard) ebd1;  //Exception
    }
  }
```

은닉화

- 객체의 변수를 Public : 외부에서 마음대로 이 변수 사용 가능
  - 의도하지 않는 범위의 값 넣을 수 있음
  - 이상한 형으로 넣을 수도 있음.
    - 나이를 음수로 넣는다던가...
- 이런점 막기 위해서, 은닉화 사용.
- Private 이용 : getter/setter 통해서 변수에 접근.
  - Getter : get + 변수명 (변수명 첫글자 대문자)
  - Setter : set + 변수명 (변수명 첫글자 대문자)

```
class SimpleBox{
  private int num;
  //은닉화 : 사용할 수 없음-> getter/setter

  SimpleBox(int num){
```

```
    this.num = num;
  }
  public int getNum(){
    return num;
  }
  public void setNum(int num){
    this.num = num;
  }
}

public class ThisUseEx{
  public static void man(String[] args){
    SimpleBox box = new SimpleBox(5);
    box.setNum(10);
    System.out.println(box.getNum());
    //값 가져와서 출력. 10이 정상적으로 출력.
  }
}
```

클래스와 오브젝트의 응용

- `instanceof` 는 오브젝트가 지정한 클래스의 오브젝트인지를 조사하기 위한 연산자.
- boolean flag = c instance of X;
  - c: 오브젝트 명
  - X : 클래스 명

```
interface Cry{
  void cry();
}

class Cat implements Cry{
  public void cry(){
    System.out.println("야옹");
  }
}

class Dog implements Cry{
  public void cry(){
    System.out.println("멍멍");
  }
}

public classs CheckCry{
  public static void main(String[] args){
    Cry test1 = new Cat();
```

```
    if (test1 instanceof Cat){
      test.cry();
    }else{
      System.out.println("고양이가 아닙니다.")
    }
  }
}
```

Class 클래스

- 자바의 모든 클래스와 인터페이스는 컴파일 후 클래스 파일로 생성됨.
- class 파일에는 객체의 정보가 포함되어있음.
- 컴파일된 class 파일에서 객체의 정보를 가져올 수 있음.
- `reflection` 프로그래밍
  - 클래스를 이용하여 클래스의 정보를 가져오고, 이를 활용하여 인스턴스를 생성하고, 메서드를 호출하는 등의 프로그래밍 방식.

Class.forName() 메서드로 동적 로딩하기

- 어떤 클래스 사용할지 모를 때 변수로 처리하고, 실행될 때 해당 변수에 대입된 값의 클래스 실행될 수 있도록 클래스에서 제공하는 `static` 메서드.
- 실행시 동적 로딩 되므로 다른 클래스가 사용될 수 있어 유용하다.
  - `동적 로딩`
    - **컴파일 시 데이터 타입이 모두 binding 되어 자료형이 로딩 되는 것이 아니라, 실행중에 데이터 타입을 알고 binding 되는 방식.**
    - 만약 해당 문자열에 대한 클래스가 없는 경우, 예외 발생 가능.

```
public class MyBook{
  private String title;
  public String author;

  public MyBook(String title){
    this.title = title;
  }

  //private title 가져오는 getTitle()
  public String getTitle(){
    return title;
  }

  public void setTitle(String title){
    this.title = title;
  }
}
```

```java
import java.lang.reflect.Constructor;

public class ClassForNameTest{

  public static void main(String[] args) throws ClassNotFoundxception{
  // ClassNotFoundxception 생기면 throw 해버리겠다는 뜻.

    //위에서 만들어놓은 클래스 string 타입으로 입력.
    Class strClass = Class.forName("MyBook");

    //생성자 뽑기
    //array 로 가져와서 다 가져와줌. (여러개일 수도 있으니까 array로.)
    Constructor[] cons = strClass.getConstructor();
    for (Constructor c : cons){
      System.out.println(c);
    }
    System.out.println("---------------");

    //필드 뽑기 (public 만.)
    Field[] fields = strClass.getFields();
    for (Field f: fieldds){
      System.out.println(f);
    }
    System.out.println("---------------");

    //method 뽑기
    Method[] methods = strClass.getMethods();
    for(Method m: methods){
      System.out.println(m);
    }
  }
}

>>> Exception. 클래스 내에 참조할 수  있는 Mybook 이 없다고 뜸.
>>> class path 에 넣어주어야 함. (같은 폴더에)
```

절차지향 & 객체지향

- 절차지향 프로그래밍 : C
  - 순차적인 처리가 중요시.
  - 프로그램 전체가 연결되어야 함.
  - 함수의 호출 순서가 바뀌면 데이터의 전달 달라짐.
  - 분리 되어서도 안됨.

- 객체지향 프로그래밍 : 나머지!
  - 개발하려는 것을 기능별로 묶어 모듈화, 모듈을 재활용.

- 굳이 순서대로 제작안해도 됨. 그냥 한번에 모였을 때 자기 기능만 제대로 하면 되니까.

두 프로그래밍의 방식이 대립되는 것이 아님.

자동차를 만들기 위해서는 차체, 바퀴, 엔진, 핸들, 의자, 엑셀 등등 … 많은 부품들이 있어야 한다.

절차지향 프로그래밍

집안 대대로 차체→ 바퀴 → 엔진 → 핸들 순으로 작업해 왔다.
순서가 바뀌면 우리 집안의 제품이 아니다.

객체지향 프로그래밍

유리한 조건의 제품을 생산하는 공장과 계약한다.