클래스

기본 구조

```
public class ClassName {클래스 + 클래스명

    public ClassName() {
        //생성자
    }

    public void name() {
        //클래스 멤버 메서드
    }

    public static void main(String[] args) {
        //메인 메서드 => 처음 무조건 시작하는 곳
    }

}
```

내부 클래스(Inner Class)

```
class Outer { // 외부 클래스

    class Inner { // 내부 클래스

        }//Inner클래스 닫음

}//Outer클래스 닫음
```

내부 클래스 장점

1. 내부 클래스에서 외부 클래스의 멤버에 손쉽게 접근
2. 서로 관련 있는 클래스를 논리적으로 묶어 표현(코드의 캡슐화 증가)
3. 외부에서는 내부 클래스에 접근할 수 없음 -> 코드 복잡성 줄임

만약 클래스가 상속을 받은 경우 생성자는 그 클래스의 생성자이니까 객체생성 없이 기능사용가능

객체생성이 가능 한 곳(클래스내 기능 사용을 위해)

클래스 선언

```java
public class APhone { //4byte
    //멤버변수(전역변수) , 자동초기화
    String color; //4byte
    int size; //4byte

    public APhone(String color, int size) {
        //this : 해당 클래스를 의미
        //this.colr : 해당 클래스밑에 바로 선언된 color를 의미
        //this로 전역변수를 지정할 수 있다.
        //변수명이 전역/지역이 동일할 때 전역변수를 지정할 목적으로 사용
        this.color = color;
        this.size = size;
    }

    public void internet() {
        System.out.println("인터넷하다");
    }
    public void text() {
        System.out.println("문자하다");
    }
    @Override //재정의, 오버라이드 => 원래 부모클래스(Object)가 있어서 APhone이
    public String toString() {
        return "APhone [color=" + color + ", size=" + size + "]";
    }
```

클래스를 만들면 기본적으로 Object라는 부모클래스 상속받음

따라서, 기본 메서드 + 클래스 생성하며 만든 게 섞여있음

생성자 (Constructor)
객체 생성시 반드시 해주어야 하는 작업
클래스와 동일한 메서드가 객체 생성시 자동호출
객체의 초기화 담당

스태틱(Static) VS 인스턴스(Instance)

스태틱(Static) – 복사가 안됨(고정된것)/ Static 영역
-객체 생성 없이 사용할 수 있도록 선언
-Static으로 선언된 변수나 메서드는 프로그램이 실행 될 때 자동적으로 메모리에 할당
-클래스이름.변수명 or 메소드명 으로 활용
-스태틱 메소드는 스태틱 변수만 사용 가능

인스턴스(Instance) – 복사됨(동적)/Stack영역
-객체 생성 통해 사용할 수 있도록 선언
-객체 생성 때마다 메모리에 복사되는 거
-Instance으로 선언된 변수나 메서드는 객체 생성시 메모리에 공간을 할당
-Stack영역의 메모리(=> 객체 생성시 계속 쌓임)

```java
public class 스태틱메서드사용 {

    public static void main(String[] args) {
        //자바전체에서 아주 많이 사용되는 기능을 사용하고자 하는 경우
        //new를 사용해서 객체를 생성한 후에 메서드를 사용하면 인스턴스변수들 모두 복사가 되고, 참조형 변수가 생성되는 등
        //그 기능을 쓰기 위해서 너무 많은 불필요한 작업들이 이루어진다.
        //static메서드로 만들어 놓으면, 객체생성하지 않아도 클래스 이름으로만 그 기능을 사용가능 => 따로 Ram에 메모리 공간 할당 할 필요없어서 효과적

        String s = "100";
        int n = Integer.parseInt(s); // Integer라는 클래스에 static 메서드인 parseInt 사용!! -> 객체생성할 필요 없음
    }
```

static 키워드를 사용한다는 것은 **어떠한 값이 메모리에 한번 할당되어 프로그램이 끝날 때 까지 그 메모리에 값이 유지된다는 것을 의미**한다. 쉽게 설명하자면 **특정한 값을 공유를 해야 하는 경우라면 static 사용 시 메모리의 이점**을 얻을 수 있다. 값 보다는 메모리라는 말을 좀 더 생각해 보면 이해가 될 것이다.
https://velog.io/@lshjh4848/static%EB%B3%80%EC%88%98%EC%99%80-static-%EB%A9%94%EC%84%9C%EB%93%9C-final-xpk2l8e7g0

크롤링

Jsoup 라이브러리

-HTML을 파싱 해주는 라이브러리
-파싱 : 내가 원하는 대로 데이터를 가공/변환해주는 것
-HTML 파싱 : HTML 소스를 얻은 후 원하는 부분만 추출하는 목적으로 사용

```
Connection con = Jsoup.connect("https://finance.naver.com/item/main.nhn?code=007570");
Document doc = con.get(); //Document : 페이지 전체 소스 저장하는 객체
Elements spancode = doc.select("span.code");
```

Connection 클래스
Jsoup의 connect 혹은 설정 메소드를 이용해 만들어진 객체, 연결하기 위한 정보를 담아주는 객체

Document 클래스
연결해서 얻어온 HTML 전체 문서(코드)

Element 클래스
Document의 HTML요소

Elements 클래스
Element가 모인 자료형

예외처리

실행 단계에서 에러가 난 경우 처리를 통해 실행시켜주는 부분

System.out.printl(3/0); => 컴파일 에러 오타인 경우
컴퓨터가 실행할 수 있는 형태로 번역(컴파일)
System.out.println(3/0); => 실행(run-time)에러, try~catch~대상
컴퓨터가 실행하면 됨(실행)

try~catch

```java
public class 문제발생클래스 {
    public static void main(String[] args) {
        System.out.println("1. 나는 프린트가 잘 될 예정.");
        try {
            //try부분에 문제가 발생할거 같은 코드 넣기
            System.out.println("2. 문제 발생코드" + 10/0);//문제 발생하는 곳 -> 에러나서 뒷 부분 실행 안됨
        } catch (Exception e) {
            //예외상황이 발생하면, 잡아서 어떻게 처리할 지 코드를 넣어주는 부분    Exception e ~> 참조형 변수
            System.out.println("에러 발생함");
            System.out.println(e.getMessage()); //에러 사유를 알려줌
            e.printStackTrace();
        }
        System.out.println("3. 나는 과연 프린트가 될까요??");
    }
}
```

throws Exception

```java
3  public class 에러발생2 {
4      //예외가 발생하면 메서드가 정의된 곳에서 처리하는 것이 아니라
5      //메서드를 호출한 곳으로 예외처리를 떠넘길 수 있다.
6      //throws Exception 을 사용하면 됨
7      //즉, call메소드 사용하기위해 부른 곳에서
8●     public void call() throws Exception{//에러메소드
9              System.out.println(3/0);
0          }
1      }
```

상속(=재사용)

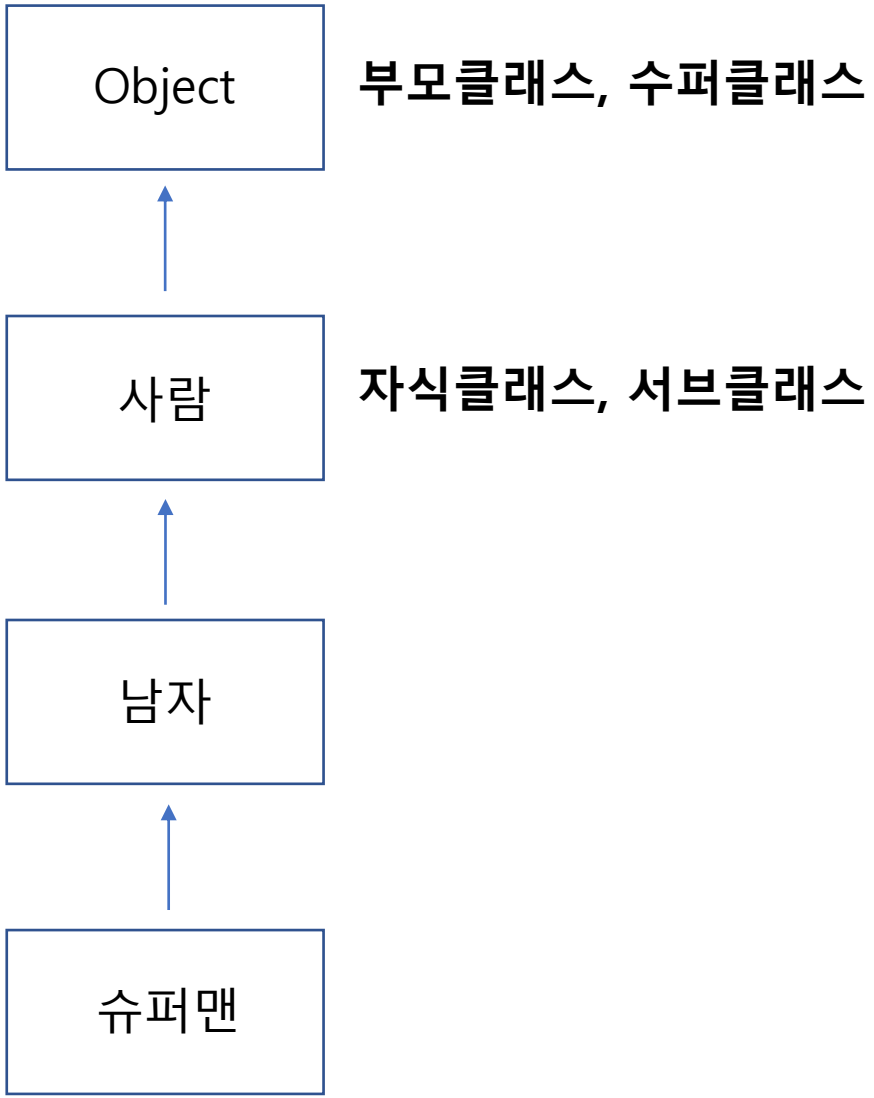
클래스를 만들 때 기존에 있는 것을 사용해(재사용) +추가 기능을 붙혀 확장성을 넓힘

부모클래스 ---상속---> 자식클래스

결합도(반드시 그 클래스만 사용해야함)가 높으면 안 좋은 코드

응집도를 높여야 좋은 코드 : 하나의 기능만

```java
package 상속;
//사람을 상속받아 만듬
public class 맨 extends 사람{
    //extends: 확장하다.
    //사람클래스를 확장해서 맨을 만들겠다.
    //멤버변수 2개, 멤버메서드1개를 갖고 생성됨

    int speed;// 속도

    public void 달리다() {
        System.out.println("빨리 달리다.");
    }
}
```



접근제어자

public : 아무 곳 에서나 접근가능
protected : 같은 패키지 + 상속 받을 때만 다른 패키지에서 사용 가능
default(안쓰는거) : 같은 패키지에서만 접근 가능
private : 해당 클래스 안에서만 사용 가능

```
public String name;
String address;//다른
protected int salary;
private int rrn;
```

스레드(Thread: 프로세스를 잘게 나눠 다중처리가 가능하도록)

프로세스를 세분화 해줘야 다중처리가 가능 => **동시(병행)에 프로그램을 수행시키도록 하기 위해**

```
package 스레드;

public class 동시프로그램 extends Thread { //동시에 처리되는 cpu단위로 만듬 -> 상속받아 상용!!
    //동시에 처리하고 싶은 프로그램 내용이 있으면
    //Thread안에 있는 run()이라는 메서드를 override
    //thread는 부모클래스임, 수퍼클래스

    //알아서 돌아가게됨
    @Override
    public void run() {
        for (int i = 0; i < 100; i++) {
            System.out.println("동시에 돌아가요 -1");
        }
    }
}
```

동시 처리가 필요한 클래스의 경우 Thread를 상속받아 run()메서드를 오버라이딩하여 사용

동시에 처리하고자 하는 기능

```
package 스레드;

public class 동시프로그램실행 {

    public static void main(String[] args) {
        동시프로그램 t1 = new 동시프로그램();
        동시프로그램2 t2 = new 동시프로그램2();

        //cpu한테 스레드 2개 만든 거에 대해 스레드등록 ~> 스케줄러에 의해 다른 프로그램이 사용되면 결과가 달라질 수 있음 (결과가 많이 섞인다 : cpu가 바쁨)
        //start()는 cpu가 스케줄러를 통해 스케줄관리해주면서 사용하게 해줌(병행실행) run을 사용하게 되면 병행 처리를 하지 못하게 되서 스레드의 의미가 없어짐
        t1.start();
        t2.start();

    }
}
```

형변환(캐스팅, casting)

기본형 형변환 : 기본형 데이터의 형변환(정수, 실수, 문자) => 계산할 때 형을 맞추기 위해

강제 형변환(down casting)
상위 데이터 타입 ---------→ 하위 데이터 타입

```
byte z = (byte)y; //기본형 복사 -> 큰거(int)를 작은(byte)로 강제 형변환
//y에 들어있던 100 값이 z에 들어갈 수 있도록
//강제로 byte타입으로 변환시켜 들어가야한다.
//강제형변환(int(큰) ----------> byte(작은))
```

자동 형변환(up casting)
하위 데이터 타입 ---------→ 상위 데이터 타입

```
//정수 :
//byte(1바이트, -128 ~ 127)
//short(2바이트, -30000~ 30000)
//int(4바이트, -2100000000~21000000000)
//long(8바이트, +-21000000000 이상/이하)

byte x = 100;
int y = x;//기본형 복사
//x에 들어있던 100값이 y에 들어갈 수 있도록 자동으로 int타입으로 변환되어 들어감
//자동형변환 (byte(작은) ------> int(큰))
```

형변환(캐스팅, casting)

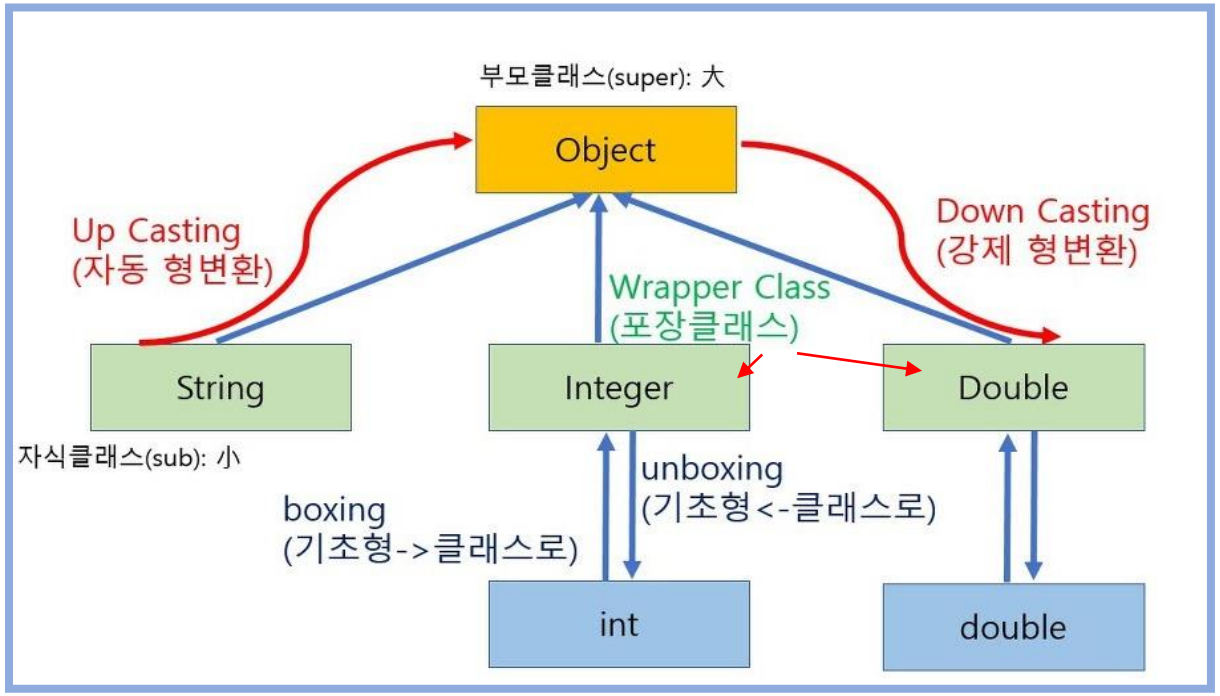
참조형 형변환 : 상속관계(부모-자식)가 있을 때만 가능

참조형타입 : 기본형을 제외한 모든 것
Ex. ArrayList, 배열 …. 혹은 클래스

```
사람 p = new 사람();
슈퍼맨 s = new 슈퍼맨();
맨 m = new 맨();
우먼 w = new 우먼();

//참조형은 상속관계에 있을 때만 가능
//부모자식간의 관계만 형변환이 가능해서 대입할 수 있다.
p = s; //작은(s) -> 큰(p)  : 클래스 자동 형변환
p = w; //작은(w) -> 큰(p) : 자동 형변환(up casting)
//m = w; //형제간의 관계는 형변환하여 대입할 수 없다.
m = s; //작은(s) -> 큰(m) : 자동형변환
m = (맨)p; // 큰(p) -> 작은(m) : 강제형변환 필요( 형변환하고 싶은 타입 변수 설정)
```

```
ArrayList list = new ArrayList(); //<> 를 쓰는 순간 형변환을 하지 않겠다라는 뜻
list.add("홍길동"); //String이었던게 Object로 자동 형변환됨 why? 기본적으로 모든 것들은 다 object 상속받았기 때문에 다 들어감
list.add(new 맨()); //참조형클래스 맨(Object)
list.add(new Random());//
list.add(new 슈퍼맨());//

list.get(0).charAt(0);//String에서 추가된 메서드들은 바로 쓸 수 없다. -> 현재 Object형임

//따라서 위의 기능을 사용하기 위해서는 강제형변환을 통해 사용
String name = (String)list.get(0); // 강제 형변환
맨 man = (맨)list.get(3);
man.run();
```

참조형변수 -> 기본형변수 형변환




```
//입력시 자동형변환 Object로 변환되어 들어감
//자동형변환 단점: 부가적으로 추가된 기능등을 쓸 수 없다.
//add()안에는 다 들어감 왜냐면 object상속을 받은거라 기본적으로 모든 것들은 다 object 상속받았기 때문에 다 들어감
list.add("홍길동"); //0번
list.add(100); //기본형과 오브젝트와의 상속관계가 없는데 들어가짐  ~> int(기본형) ------auto-boxing------> Integer(참조형) --------->upcasting ----->Object
list.add(11.12);
System.out.println(list);

int num =(Integer)list.get(1);//int(기본형) <------auto-boxing <------ Integer(참조형) <--------- downcasting <-------- Object
```

다형성(상속관계에서만 사용가능)

한 타입의 참조변수(슈퍼클래스)로 **여러 타입의 객체(자식클래스)**를 참조할 수 있도록 하는 것

```
/객체 설계할 때  2대 원칙
/1. 객체를 만들때는 하나의 클래스안에는 하나의 역할만을 정의해야한다. => 응집도(Cohesion) 높게!
/2. 결합시 특정한 클래스만 지정하여 설계하면 안된다. => 결합도(Coupling) 낮게!
```

```
//Object는 너무 커서 안되고 사용하는 클래스중 범위가 가장 큰 코드를 왼쪽에 설정해!!!
//결합도가 낮은 코드는 유지보수하기도 좋고 , 더 좋은 성능을 가진 클래스로 변경에 용이!!!
사람 m2 = new 맨(); //자동형변환 됨 ~> 이게 더 좋은 코드 : 결합도가 훨씬 낮음 => 왼쪽을 큰걸로 맞추면 오른쪽을 수정할 때 왼쪽만 맞추면 됨
m2 = new 우먼();
m2 = new 슈퍼맨(); // 이런것 처럼 고쳐가면서 사용하기 편함(수정에 용이)
```

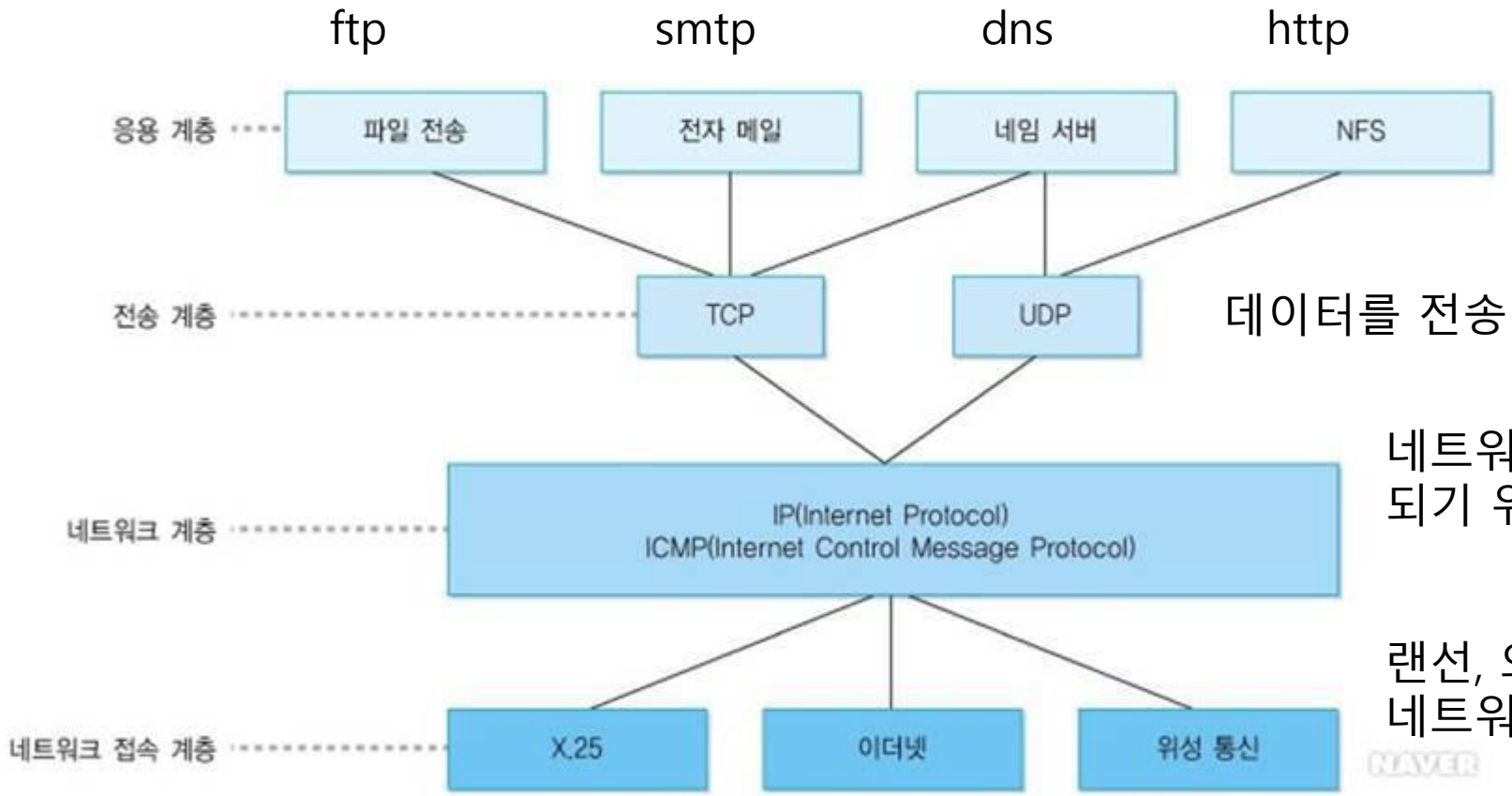
여기서 **m2.run();** => **맨 객체의 메소드를 실행하면 오류 발생**
~> 왜냐면 다형성을 통해 참조된 객체에서는 **상위 객체의 오버라이딩 된 메소드만 사용 가능**
즉, m2는 맨 객체를 참조하고 있지만 실제로 참조 되는 건 상위 객체에 해당하는 부분만 참조할 수 있음

혹은, **형변환을 통해** 하위객체의 메서드를 사용할 수 있다.

```
((맨) m2).run();
```

TCP/IP구조(출처: 네이버)

p.148



데이터를 전송

네트워크의 하나의 노드가 되기 위해 IP설정

랜선, 와이파이
네트워크 접속을 위한 설정

인터페이스

인터페이스 역할&개념 : 극단적으로 **동일한 목적 하에 동일한 기능**을 **수행하도록 강제**하는 것

=> 자바의 다형성을 극대화하여 개발코드 수정을 줄이고 유지보수성을 높이기 위해 사용

예시

교수님이 학생들에게 논문을 쓰라고 했다.

A학생은 PPT로 논문을 11일날 제출했다.
B학생은 EXCEL 2016 로 논문을 12일날 제출했다.
C학생은 EXCEL 2019 로 자기만의 색깔을 입혀 표 형식으로 12일날 제출했다.
D학생은 메모장에 '논문'을 쓰고 교수님이 말해준 당일 바로 제출했다.

응?? 논문을 쓰라고 지시했지만, 결과물이 너무 개성이 넘친다....이건 평가할 수 없을 것 같다. 왜 이런일이 일어날까?
가이드 라인 또는 어떠한 규격이 없기 때문에 발생하는 문제이다.
따라서 교수는 다시 명확한 가이드라인 있는 논문을 쓰라고 지시해야한다. 아래처럼...

학생들에게 2019.12.12일 18:00까지 홈페이지 본인 교수 홈페이지 제출란을 통해 논문을 제출해야하고 논문 파일 형식은 .PPT이며 10Page 안에 작성을 해야하고 논문 주제는 '블록체인을 활용한 공인인증서' 이다.

인터페이스 문법과 다형성 이해

인터페이스는 **interface** 키워드를 통해 **선언**, **implements** 키워드를 통해 일반 클래스에서 **인터페이스 구현**

인터페이스 예시

대한민국에서 은행 사업을 할 때, 금융결제원에서 정의한 가이드라인을 따라야 할 때
아래 코드가 가이드라인에 해당하는 인터페이스(Bank)

```java
package 인터페이스;

public interface Bank {
    // 상수 (최대 고객에게 인출해 줄 수 있는 금액 명시)
    public int MAX_INTEGER = 10000000;

    // 추상메소드(인출하는 메소드)
    void withDraw(int price);

    // 추상메소드(입금하는 메소드)
    void deposit(int price);

    // JAVA8에서 가능한 defualt 메소드(고객의 휴면계좌 찾아주는 메소드 : 필수구현은 선택사항)
    default String findDormancyAccount(String custId) {
        System.out.println("**금융개정법안 00이후 고객의 휴면계좌 찾아주기 운동**");
        System.out.println("**금융결제원에서 제공하는 로직**");
        return "00은행 000-000-0000-00";
    }
    // JAVA8에서 가능한 정적 메소드(블록체인 인증을 요청하는 메소드)
    static void BCAuth(String bankName) {
        System.out.println(bankName + " 에서 블록체인 인증을 요청합니다.");
        System.out.println("전 금융사 공통 블록체인 로직 수행");
    }
}
```

상수: 인터페이스에서 정해준 값으로 변경불가 하며 제공된 값만 참조(**절대적**)

추상메소드: 가이드만 줄 테니 추상 메소드를 **오버라이딩 하여 무조건사용**해라(**강제적**)

디폴트메소드: 인터페이스에서 기본적으로 제공하지만, 맘에 안 들면 알아서 구현해서 써라(**선택적**)

정적메소드: 인터페이스에서 제공해주는 것으로 사용해라 (**절대적**)

절대적: 수정불가
강제적: implements하지 않으면 피할 수 있음

결국 이미 운영되고 있는 시스템에서 추가 요건으로 인해 불가피하게 반영을 해야 할 때 디폴트메소드를 쓰면 효과적
왜냐하면 추상메소드를 추가하면, 이를 implements한 모든 클래스에서 강제적으로 구현 해야하고 구현 안 할 시 에러가 발생하기 때문이다.

클래스 다이어그램의 이해

https://morm.tistory.com/88

추상클래스

추상적인 개념을 표현하고, **완성되지 않은 메서드를 가지고 있는 클래스**
메서드가 미완성되어 있기 때문에 추상 클래스로는 객체 생성 불가
주로 상속관계에서 추상적인 개념을 나타내는 목적으로 사용, 단일 상속가능
추상 메소드를 하나라도 가지면 추상 클래스가 된다.
멤버변수를 갖고 있다.
추상클래스는 추상 메소드, 일반 메소드, 멤버변수, 생성자로 구성

```
public abstract class Animal {      //추상클래스 : 추상메서드를 가지고 있는 클래스
    public abstract void move();   // ';'으로 종료됨을 유의! , 추상메서드 정의
    ...
};
```