JAVA Programming 특강

-목차-

- C언어와 Java언어의 차이 1 <컴파일언어/인터프리트언어>
- C언어와 Java언어의 차이 2 <타입의 차이>
- C언어와 Java언어의 차이 3 <절차지향/객체지향(OOP)>

- 객체(Object)란? <객체와 클래스>
- 생성자(Constructor>
- 접근제어자(Access modifier)
- 캡슐화(Encapsulation) , 정보은닉(information hiding)

- 포함(Has a 관계) / 상속 관계(Is a 관계)
- 오버로딩(over-loading), 오버라이딩(over-riding)
- 업캐스팅(Up-casting), 다운캐스팅(Down-casting)
- 구현(implements)의 이해 <interface>
- 다형성(polymophism)의 이해

- static, final 키워드
- LinkedList<T> 클래스의 활용
- Math클래스의 static Methods 활용
- Wapper Class < Integer, Double, Boolean, Float 클래스>
- Java API 활용법.

지금 과정은 보통 1~2주 수업 내용을 꼭 필요한 내용만 압축해서 진행하기 때문에 처음 배우시는 분들은 이해가 잘 안가거나 멘붕이 오는게 당연한 현상입니다. 그래도 쉬운 예제들을 많이 넣었으니 다 함께 잘 배워 봅시다.

모르면 바로 질문해주세요. 다같이 이해하고 넘어 갑시다.

C 와 Java 언어의 차이 1

C 는 컴파일(Compile,편집) 언어
컴파일러와 링커로 구성
컴파일러 : .c -> .obj(기계어)
링커 : .obj -> .exe

Java는 인터프리트(Interpret,번역) 언어
내부컴파일 , JVM으로 구성
내부컴파일 : .java -> .class(기계어)
JVM : .class -> 실행파일.

C 와 Java 언어의 차이 2

C는 call by value/call by pointer

Java는 call by value/call by reference

C 와 Java 언어의 차이 3

C는 절차지향 언어 이다.
단, 구조체와 함수포인터를 활용하여 객체지향 기법 사용 가능
순차적인 흐름을 가지고 주어진 작업을 실행하는 것

Java는 객체지향 언어 이다.
What is OOP(Object Oriented Programming)?
객체라는 작은 단위로 모든 처리를 기술하는 **프로그래밍** 방법.

객체 와 클래스(class)

- 객체(Object) 클래스(class) / 물리적 객체 (type으로 쓰임)
  인스턴스 객체(임시 객체) / 논리적 객체

&추상화란?

- 클래스에 고유의 값을 넣어(안 넣을 수도 있음),
  인스턴스 객체를 만든다.

```
Person p1 = new Person("함영식", "남자", 27);
Person p2 = new Person("김준환", "남자", 26);
Person p3 = new Person("전효재", "남자", 26);

class Person{
    String name; // String은 문자열 타입
    String sex;
    int age;

    public Person(String name, String sex, int age){
            //생략.
    }
}
```

문제 1. 자신만의 Person 클래스를 만들어 봅시다.

조건 : Person 은 이름과 나이는 필수로 가진다.
이름과 나이 외에 무조건 1개 이상 변수를 추가한다.

문제 2. 자신만의 Person 객체를 만들어 봅시다.

도움말 : Test 클래스를 만들고 다음 형식처럼 적용해보세요.

```
public class Test {
        public static void main(String args[]){
                Person p1 = new Person("함영식","남자",27);
                System.out.println(p1.name+"/"+p1.sex+"/"
                + pl.age);
        }
}
```

& 문자열 + 숫자 => 문자열.

&System.out.println( ) 은 C에서 printf( ) 와 비슷한 출력문.

System.out.println( ) – 출력 후 한 줄 뛴다.
System.out.print( ) – 출력 후 return.

****System.out.println(p1); 은**
System.out.println(p1.toString())와 같다.
=> 인자가 레퍼런스 타입일 때만. toString() 많이 씀.

문제 2-2. toString() 만들어 보고 .println( ) 테스트 해보기.

도움말 : Person 클래스 다음 코드를 추가하자.

```
public String toString();{
    return /* 이곳에  출력하고 싶은 문자열을 만든다.*/;
}
```

클래스(class) 구조

멤버변수
필드
속성

```
public class Person {
	private String name;
	private String sex;
	private int age;

	public Person(String name, String sex, int age) {
		super();
		this.name = name;
		this.sex = sex;
		this.age = age;
	}

	public Person() {
		this("없음","모름",0);
	}


	public String getName() {
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}

	public String getSex() {
		return sex;
	}

	public void setSex(String sex) {
		this.sex = sex;
	}

	public int getAge() {
		return age;
	}

	public void setAge(int age) {
		this.age = age;
	}

	@Override
	public String toString() {
		return "Person [name=" + name + ", sex=" + sex + ", age=" + age + "]";
	}
}
```

생성자

멤버함수
메소드

생성자(Constructor)

- 특징?
- 사용 방법?

생성자 오버로딩(Constructor Over-loading)

생성자의 인자만 다르게 정의
실행 시, 알아서 맞는 생성자가 호출됨.
this 생성자 호출은 키워드는 맨 첫 줄에만.

this 키워드

자기 자신을 가리키는 키워드로써 멤버변수(필드)와 멤버함수(메소드) 와 이름이 같은 외부 정의가 있을 때 구분하기 위해 사용.

```
public class Person {
	private String name;
	private String sex;
	private int age;

	public Person(String name, String sex, int age) {
		super();
		this.name = name;
		this.sex = sex;
		this.age = age;
	}

	public Person() {
		this("없음","모름",0);
	}
```

접근 제어자(access modifier)

- public
- Protected
- private

```
public class Person {
        private String name;
        private String sex;
        private int age;
```

```
public class Person {
        public String name;
        public String sex;
        public int age;
```

정보은닉(information hiding)

private 키워드 등으로 외부에서 함부로 데이터의 값을 바꾸지 못하도록 하는 것.

캡슐화(Encapsulation)

객체를 생성할 때 프로그램 작성자는 숨겨야 하는 정보(private)와 공개해야 하는 정보(public)이 있을 것이다.
이러한 캡슐화를 통하여 객체를 사용하는 사람에게 공개해야 하는 정보만 공개하는 것이다. (정보은닉)

Ex) 알약, 자판기.

만약? 모든 데이터를 private으로 한다면?

getter / setter 필요

문제 3. Person 클래스 생성자 오버로딩 해보기.

이름만 인자로 받는다. 나머지는 기본값으로,
이름과 나이를 인자로 받는다. 나머지는 기본값으로.
모두 인자로 받는다.

*(중요)하나도 인자로 받지 않는다.(디폴트 생성자)
왜 중요한가요?

문제4. 오버로딩한 생성자를 통해서 객체를 만들어 봅시다.
그리고 만든 객체를 문제 2-2번처럼 정보출력 해봅시다.

```
public class Test {
	public static void main(String args[]){
		Person p1 = new Person("위은복");
		Person p2 = new Person("함영식",10);
		Person p3 = new Person(/*모든 인자*/);
		Person p4 = new Person();
		//여기 이후에 정보출력 실행 하는 코드.
	}
}
```

&주석 : // , /* */

문제 5. Person 클래스를 캡슐화 시켜봅시다.

조건 : 이름은 외부에 **공개**해도 되지만,
나이는 **정보은닉**시킨다.
나머지는 알아서.

문제 6. 외부에서 이름과 나이를 수정해봅시다.
결과는?

```
public class Test {
	public static void main(String args[]){
		//스스로 적어넣으셈
		p1.name = "홍길동";
		p1.age = 11;
	}
}
```

포함

A 가 B를 가진다. (A:주인 , B:대상)

ex)
```
class PoliceMan {
      //기존 필드(멤버변수)들 생략.
      private Gun gun;
}
```

보통 A Has a B 가 성립. (역은 성립되지 않는다.)
ex) 경찰은 총을 가진다(0) 총은 경찰을 가진다(X)

문제 7. 다음 테스트를 했을 때 예상 결과가 나오게 만들기.

```
public class Test {
      public static void main(String args[]){
            PoliceMan pm1 = new PoliceMan(new Gun(12));
            pm1.shot(3);
            System.out.println(pm1);
      }
}
```

조건 : 예외 처리 유/무 선택!
예상결과 :

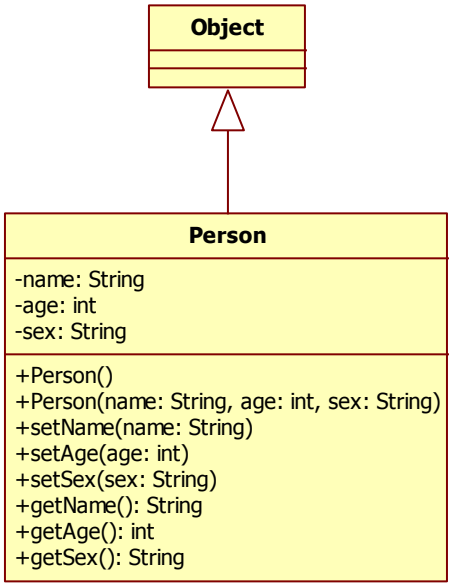
빵!
빵!
빵!
남은 총알 개수는 9개 입니다.

```
public class Gun{
      private  int bullets;

      //생성자
      //setter and getter
      //toString()
}
```

상속(extends)

디폴트 Object 클래스 상속



부모의 성질을 자식이 이어받는 것과 비슷한 맥락

B 가 A를 상속한다. (A:부모 , B:자식)

ex)
class PoliceMan extends Person {
//(중요) 생성자에서 상위타입의 멤버변수 까지 초기화 시킬 수 있다.
→ **super 키워드**를 이용해서.
}

보통 B is A 가 성립. (역은 성립되지 않는다.)
ex) 경찰은 사람이다. (O) 사람은 경찰이다(X)
=> Police extends Person

super 키워드

자기 자신의 부모를 가리키는 키워드 보통 생성자 맨처음에, 부모의 생성자를 호출하는 역할을 함.

Ex) super(10) == PoliceMan(10)

문제 8. super 키워드 문제

PoliceMan이 Person을 상속시키고, PoliceMan 객체가 생성될 때, Person이 가지고 있던, 멤버변수 전체와 PoliceMan의 멤버변수를 같이 초기화 할 수 있는 생성자를 만들어라.

힌트 : 8쪽의 this("없음","모름",0); 은 Person("없음","모름",0); 과 같은 뜻이다. this는 자기 자신을 의미하고 super는 자신의 부모를 의미한다.

Ex) new PoliceMan pm = new PoliceMan("영식","남",27, new Gun(18));

오버로딩(over-loading) : 한 클래스 내에 같은 이름의 메소드를 선언하는 것.
Ex) 생성자 오버로딩.

오버라이딩(over-riding) : 상속 관계의 두 클래스가 존재할 때, 자식클래스가 부모클래스에 이미 정의 혹은 선언이 되어 있는 메소드를 재 정의 하는 것.
Ex) toString() 메소드. toString() 은 Object 클래스의 메소드임.

업 캐스팅(Up-casting) : 자식 또는 하위 클래스는 상위 클래스 타입으로 캐스팅 할 수 있다. Ex) Person p1 = new PoliceMan( );

다운 캐스팅(Down-casting) : 업캐스팅 된 객체를 다시 하위객체로 캐스팅 하는 것 도 가능하다. Ex) PoliceMan p2 = (PoliceMan)p1;

(중요) 어떤 메소드가 오버라이딩이 되어있고 업캐스팅 된 상태에서 해당 메소드가 호출이 되면, 업캐스팅된 타입(클래스)에 있는 메소드가 불리는 것이 아니고, 오버라이딩 된 메소드가 호출이 된다.

문제9. 오버로딩 문제

PoliceMan 클래스에서 shot(int num) 메소드를 오버로딩하여, 문자로 된 숫자가 입력이 되도 발사 될 수 있도록 만들어라.

힌트 : int num = Integer.valueOf("숫자");

문제 10. 업캐스팅&오버라이딩 문제

PoliceMan을 상속하는 SuperPoliceMan 클래스를 정의하고
멤버변수로 int power를 선언하고, shot()메소드를 **오버라이딩**하고
하위 클래스인 PoliceMan으로 **업캐스팅**한 후, shot(int num)을 호출 했을 때,
(power * num) 만큼 총알이 발사 되도록 만들어라.

도움말) 해당 문제를 위해서는 super를 사용하여 초기화를 시키거나,
setter 메소드를 사용하여 초기화를 시켜주는 것이 좋다.

Ex) new PoliceMan pm = new SuperPoliceMan("영식","남",27, new Gun(18), 2);
pm.shot(2);

예상결과 :
슈퍼총 발사!
빵!
빵!
빵!
빵!
남은 총알은 14개 입니다.

문제 11. 오버라이딩 문제2

PoliceMan클래스의 toString() 메소드를 구현하고 업캐스팅&오버라이딩을 강사에게 보여라.

PoliceMan클래스의 toString() 내용 : 이름, 나이, 성별, 총알 수

구현(implements) - 클래스가 interface를 사용할 때

interface 란?

- **함수의 선언만 있고 정의가 없다.(정의부가 없음)**
- **final 변수만 정의할 수 있다.**
- **implements로 구현한다.**
- **하나이상의 interface를 구현한 클래스는 반드시 모든 메소드를 오버라이드 해야한다.**
- **인터페이스끼리 상속이 가능하다.**
- **클래스와 다르게 다중상속이 가능하다.**
- **하지만, class처럼 new를 써서 객체를 만들지 못한다.**

ex) 걷는 행동을 선언을 한 interface

```
public interface Walkable{
    public abstract walk();
}
```

ex) 나는 행동을 선언을 한 interface

```
public interface Flyable{
    public abstract fly();
}
```

interface 와 implements 의 개념

구현 방법

```
public class Person implements Walkable{
      //무조건 walk()를 정의해야 한다.
       public void walk(){
          //구현.
       }
}
```

새는 날 수도 있고, 걸을 수도 있다. (다중 구현)
→ public class Bird implements Walkable,Flyable{
// fly(), walk() 둘 다 정의해야 함.
}

Interface는 new를 써서 객체를 만들지는 못해도, 타입으로써 업캐스팅을 할 때 사용된다. (다형성에 사용)

퀴즈- 실행 가능한 코드 찾기(실제 코드 작성 해보고 결과실행)

Walkable p1 = new Person();
p1.walk(); ( )

Walkable p1 = new Person();
p1.fly(); ( )

Flyable p1 = new Person();
p1.fly(); ( )

Person p1 = new Walkable();
p1.walk(); ( )

Walkable p1 = new Walkable();
p1.walk(); ( )

Person p1 = new Person();
p1.fly(); ( )

Flyable p1 = new Person();
p1.walk(); ( )

다형성

다형성이란? “여러가지 형태를 가질수 있는것"

한가지 타입의 참조변수로 여러 타입의 객체를 참조할 수 있다는 것이다. (단, 상속 관계일 때)

다형성의 3가지 조건
1. **상속**
2. **업 캐스팅**
3. **오버라이딩**

```
PoliceMan p1 = new SuperPoliceMan(“영식”,”남”,27, new Gun(18));
p1.shot();
//SuperPoliceMan 은 PoliceMan 을 상속한다. (조건1)
//SuperPoliceMan 타입인 p1 을 PoliceMan 타입으로 업 캐스팅 하였다.(조건2)
//SuperPoliceMan은 PoliceMan의 shot() 을 오버라이딩 하였다.(조건3)
```

다형성의 활용 #01. Parametric Polymorphism

```
public static void goShot(PoliceMan p){
    //p에 PoliceMan객체도, SuperPoliceMan객체도 들어올 수 있음.
     p.shot();
}
```

```
PoliceMan p1 = new SuperPoliceMan("영식","남",27, new Gun(18),2);
PoliceMan p2 = new PoliceMan("영식","남",27, new Gun(18));

goShot(p1);
goShot(p2);
```

다형성의 활용 #02. Heterogeneous Collection

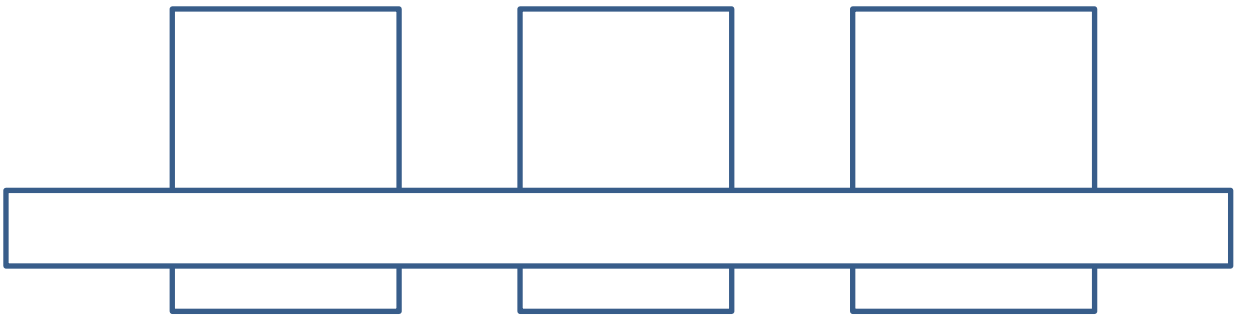
```
ArrayList<PoliceMan> pList = new ArrayList<PoliceMan>();

PoliceMan p1 = new SuperPoliceMan("영식","남",27, new Gun(18),2);
PoliceMan p2 = new PoliceMan("영식","남",27, new Gun(18));

pList.add(p1);// PoliceMan객체도, SuperPoliceMan객체도 add할 수 있음.
pList.add(p2);//

for(PoliceMan p : pList){
    p.shot();
}
```

static / final 키워드



```
public class A{
      private int a=10;
      public final int b=20;
      public static int c=30;
}
```

```
A a1 = new A();
A a2 = new A();
A a3 = new A();
```

왜 main은 public **static** void main(String args[])일까?

final은 언제 언제 선언해야 될까?

static final 은 언제 쓰일까? (#define)

final 키워드는 메소드나 클래스명 앞에서도 유용하게 쓰입니다.
지금은 그것만 알아 두세요.

LinkedList<T> 클래스의 활용

<T>의 의미? (제네릭!)

클래스의 메소드들과 기능설명

add()
clear()
size()
remove()
contains()
get()
lastIndexOf()
removeAll()
isEmpty()
Push()
Pop()
addLast()
addFirst()
removeFirst()
removeLast()

다형성을 구현할 수 있다.

```
LinkedList<Person> plist = new LinkedList<Person>();
plist.add(new PoliceMan());
plist.add(new PowerPoliceMan());
plist.add(new Person);

for(Person p : plist){                    //향상된 for문.
        System.out.println(p);
}
```

Math클래스의 static Methods 활용

Math.abs(숫자); - 절대값
Math.max(숫자,숫자); - 큰 값 구하기
Math.min(숫자,숫자); - 작은 값 구하기
Math.tan(각도); - 삼각함수
Math.cos(각도); - 삼각함수
Math.sin(각도); - 삼각함수
Math.sqrt(숫자); - 숫자의 제곱근값
Math.random(); - 0~1 사이의 무작위 값

Wapper Class < Integer, Double, Boolean, Float 클래스>

unBoxing ex)

Boxing ex)

ArrayList<Integer> 이런 식으로도 선언 가능하다는 말!

toString(), valueOf(),CompareTo() 등 프리미티브 타입일 때 불편한 점들을 객채화 시킴으로써 해결할 수 있다.

Java API 활용법.

검색창에 JavaAPI 치면 많이 나온다.
전체를 다운 받아도 좋고, 필요할 때 마다 검색해서 봐도 좋다.
한가지 부탁하고 싶은점은 한글로 보지 않았으면 한다.
해석이 정확이 안되어 있고, 오히려 혼란을 주는 경우가 있기 때문이다.

자신이 많이 쓰는 패키지 위주로 이리저리 심심할 때 둘러봐도 도움이 많이 될 것이다.

문제 .

Wrapper Class , ArrayList, 향상된 for문 을 이용하여
10개 이상의 수를 더한 결과출력.

힌트 <Integer>