2023.5.25. Java 프로그래밍

<상속자 inherit>

```
package com.day07;

public class Main {

	public static void main(String[] args) {
		Father f = new Father();
		f.fatherMethod();
		System.out.println("-----------------");
		Child c = new Child();
		c.childMethod();
		c.fatherMethod();
		c.grandFatherMethod();
		f.grandFatherMethod();
	}

}
```

```
package com.day07;

public class Child extends Father { // 단일 상속(Father, GrandFather 불가)
	public Child() {
		System.out.println("Child 생성자");
	}
	public void childMethod() {
		System.out.println("childMethod");
	}

	@Override
	public void method() {
		System.out.println("child가 method 수정함");
	}
}
```

```
package com.day07;

public class Father extends GrandFather {
	public Father() {
		System.out.println("Father 생성자");
	}
	public void fatherMethod() {
		System.out.println("fatherMethod");
	}
	/*Overriding( 상위 클래스의 특성과 동작을 하위 클래스에게 상속받았을 때,
	하위 클래스에서 상위 클래스의 메서드를 필요에 맞게 수정하거나 추가적인 동작을
정의하기 위해 사용)*/
	@Override
	public void method() {
		System.out.println("father가 method 수정함");
	}
```

}

<오버 라이딩>

오버라이딩(Overriding)은 객체 지향 프로그래밍에서 상속 관계에 있는 클래스 간에 동일한 이름의 메서드를 재정의하는 개념입니다. 상위 클래스에 정의된 메서드를 하위 클래스에서 동일한 시그니처(이름, 매개변수, 반환 타입)로 다시 구현하는 것을 의미합니다.

오버라이딩은 상속을 통해 상위 클래스의 특성과 동작을 하위 클래스에게 상속받았을 때, 하위 클래스에서 상위 클래스의 메서드를 필요에 맞게 수정하거나 추가적인 동작을 정의하기 위해 사용됩니다. 이를 통해 하위 클래스는 상위 클래스의 메서드를 자신에게 맞게 재정의하여 사용할 수 있습니다.

오버라이딩을 위해서는 다음 조건을 만족해야 합니다:

상속 관계가 있어야 합니다. 즉, 하위 클래스가 상위 클래스를 상속받아야 합니다.
오버라이딩하려는 메서드는 상위 클래스에서 정의된 메서드와 동일한 이름, 매개변수, 반환 타입을 가져야 합니다.
접근 제어자와 예외 처리에 일부 제한이 있습니다. 오버라이딩된 메서드는 상위 클래스의 메서드와 동일한 접근 제어자를 가져야 하며, 더 큰 범위의 예외를 던질 수 없습니다.

<상속자의 범위>
public > protected > default(생략) > private

<다형성>
```
package com.day07.interitance;

class Animal {
	public void move() {
		System.out.println("동물이 움직입니다.");
	}
}
class Human extends Animal {
	@Override
	public void move() {
		System.out.println("사람이 두 발로 걷습니다.");
	}
}

class Tiger extends Animal {
	@Override
	public void move() {
		System.out.println("호랑이가 네 발로 뜁니다.");
	}
}

class Eagle extends Animal {
	@Override
	public void move() {
		System.out.println("독수리가 하늘을 납니다.");
```

```
	}
}

public class AnimalTest {
	public void moveAnimal(Animal animal) {
		animal.move();
		if(animal instanceof Human) {
// class형이 맞는지 물어볼때 instanceof 사용 273p
			System.out.println("Human");
		}
		if(animal instanceof Animal) {
			System.out.println("Animal");
		}
	}
	public static void main(String[] args) {
		AnimalTest at = new AnimalTest();
		at.moveAnimal(new Human());
		at.moveAnimal(new Tiger());
		at.moveAnimal(new Eagle());


	}
}

<ArrayList, Generic, Wrapper Class>
package com.day07;

import java.util.ArrayList;

public class ArrayListTest {
	public static void main(String[] args) {
		// Array
		// int형 크기가 4인배열 arr
		int[] arr = new int[4];
		arr[0] = 1;
		arr[1] = 2;
		// ArrayList : 객체를 다루는 자료 구조
		ArrayList alist = new ArrayList();
		alist.add("one");
		alist.add("1");
		alist.add("two");
		System.out.println(alist.size());
		// 2 추가 => int 내부적으로 Interger로 변환작업 수행(Wrapper Class)
		alist.add(2);
		System.out.println(alist.size());
		alist.add("three");
		System.out.println("alist길이 : " + alist.size());
		// alist.add(new Interger(3); - 사용할 필요 없음
		System.out.println("alist 첫 번째 : " + alist.get(0));
		System.out.println("alist 마지막 번째 : " + alist.get(alist.size() - 1));
```

```
		for (int i = 0; i < alist.size(); i++) {
			System.out.println(alist.get(i));
		}
		System.out.println("------------------------------");
		for (Object obj : alist) {
			System.out.println(obj);
		}
		// 2번째에 딸기를 추가해 보세요.
		alist.add(2, "딸기");
		for (Object obj : alist) {
			System.out.println(obj);
		}
		System.out.println(alist.size());

		alist.remove(0);
		System.out.println("size : " + alist.size());
		String su = (String) alist.get(1);// two
		System.out.println("su : " + su);

		// 객체를 다루는 구조 : Collection
		// Generic
		ArrayList<String> blist = new ArrayList<>();
		blist.add("바나나");
		// blist.add(1);
		System.out.println("--------------------------------");
		for (String s : blist) {
			System.out.println(s);
		}

		System.out.println("--------------------------------");
		ArrayList<Integer>clist = new ArrayList<>();
		clist.add(1); // Wrapper Class
		clist.add(3);
		for(Integer i : clist) {
			System.out.println(i);
		}

	}
}
```

ArrayList는 Java의 내장된 클래스로, 동적 크기 배열을 구현하는데 사용됩니다. 이를 활용하여 데이터를 추가, 제거, 조회하는 등의 작업을 편리하게 수행할 수 있습니다.

Generic(제네릭)은 Java에서 컬렉션 클래스나 메서드의 매개변수, 반환 타입 등을 다룰 때 사용되는 기능입니다. 제네릭을 사용하면 타입 안정성을 보장하고 재사용성을 높일 수 있습니다.

레퍼 클래스(Wrapper Class)는 기본 데이터 타입(primitive data type)을 객체로 감싸는 클래스입니다. 기본 데이터 타입인 int, double, boolean 등은 값을 직접 저장하고 메모리 상에 사용됩니다. 그러나 레퍼 클래스를 사용하면 이러한 기본 데이터 타입을 객체로 다룰 수 있습니다.

<Wrapper Class의 종류>
Integer: int 타입의 래퍼 클래스
Double: double 타입의 래퍼 클래스
Boolean: boolean 타입의 래퍼 클래스
Character: char 타입의 래퍼 클래스
그 외에도 Byte, Short, Long, Float 등이 있습니다.

<ArrayList 활용>

```
package com.day07;

import java.util.ArrayList;
import java.util.Scanner;

public class TeacherMain {
	static Scanner sc = new Scanner(System.in);
	ArrayList<Student> arr = new ArrayList<>();


	public static void showMenu() {
		System.out.println("선택하세요>>");
		System.out.println("1. 입력  2. 전체보기/종료");
		System.out.println("선택>>");
	}

	public void inputData() {
		System.out.println("--- 학생 성적 입력 ----");
		System.out.println("이름>>");
		String name = sc.next();
		System.out.println("국어 영어 수학>>");
		int kor = sc.nextInt();
		int eng = sc.nextInt();
		int math = sc.nextInt();
		arr.add(new Student(name, kor, eng, math));
	}

	public void viewData() {
		// for-each
		for (Student s : arr) {
			System.out.println("이름 : " + s.getName());
			System.out.println("국어 : " + s.getKor());
			System.out.println("영어 : " + s.getEng());
			System.out.println("수학 : " + s.getMath());
			System.out.println("총점 : " + s.getTotal());
			System.out.println("평균 : " + s.getAvg());
			System.out.println();
		}

}

	public static void main(String[] args) {
		TeacherMain t = new TeacherMain();
		while (true) {
			TeacherMain.showMenu();
			int num = sc.nextInt();
			switch (num) {
			case 1:
				t.inputData();
				break;
			case 2:
				t.viewData();
				System.exit(0); // 종료
			default:
				System.out.println("입력오류");
```

```
                    }// switch
                } // while
        }// main

} // class
```