Week 5-1: ADTDesign

Part1. ADT Design

클래스로 정의됨.
모든 객체들은 힙 영역에 할당됨.
캡슐화(Encapsulation) : Data representation + Operation
정보은닉(Information Hiding) : opertion부분은 가려져있고, 사용자가 operation으로만 사용 가능해야 함.

- 클래스 정의의 형태

```
public  class  Person {
        private String name;
        public int  age;
        public  Person() {
        }
        public Person(String s) {
                name  = s;
        }
        public String getName() {
                return name;
        }
 }
```

생성자 : 클래스의 이름과 동일한 메소드, 클래스의 객체가 생성될 때 호출되는 메소드

메소드 : 실행 가능한 함수, 객체의 행위를 구현

메소드 오버로딩 :

① 메소드 이름 동일

② 메소드 인자의 개수가 서로 다르거나, 메소드 인자의 타입이 서로 달라야 함.

③ 리턴 타입만 다른 경우에는 에러

- 객체의 생성

```
public static void main (String args[]) {
        Person aPerson;
        aPerson = new Person("홍길동");
        aPerson.age = 30;
        String s = aPerson.getName();
}
```

객체에 대한 레퍼런스 변수 aPerson을 선언한 뒤, new를 통해서 Person 객체를 생성하고 있다.

- 멤버 접근 지정자

멤버에 접근하는 클래스	멤버의 접근 지정자			
	default	private	protected	public
같은 패키지의 클래스	O	X	O	O
다른 패키지의 클래스	X	X	X	O

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

```
        public  int  getAge() {
                return  age;
        }

        public void setAge(int value) {
                age  = value;
        }
}

public  class  Access {
        public static void main(String[] args) {
                Person aPerson = new Person();
                aPerson.name = "홍길동";
                aPerson.setAge(20);
                aPerson.num  = 10;
          }
}
```

public인 name에 대해서는 바로 값 지정이 가능하지만, private으로 선언된 age의 경우 바로 선언하려고 하면 에러가 발생한다. 이는 private의 경우, 서로 다른 클래스에 선언되어 있기 때문이다. 때문에 클래스 내부에 서 get/set 메소드를 만들어서 접근해야 한다.

Week 5-2 : Class Hierarchy

Part2. Casting

- 객체의 타입 변환

업캐스팅 : 서브클래스 객체가 슈퍼 클래스 타입으로 변환되는 것
다운캐스팅 : 업캐스팅 된 것을 다시 원래대로 되돌리는 것, 명시적으로 타입 지정이 필요.

```
class  Person {
        String name;
        String id;

        public Person(String name)
                { this.name  = name;

}

class Student extends Person {
        String grade;
        String department;

        public Student(String name) {
                super(name);
        }

}

public  class  Casting {
        public  static  void  main(String[]  args) {
                Person p = new Student("홍길동"); // ① 업캐스팅
                System.out.println(p.name);
                p.grade = "A";
                p.department  = "컴퓨터";

                Student s = (Student)p; // ② 다운캐스팅
                System.out.println(s.name);
                s.grade  = "A";
```

```
        }
}
```

① 업캐스팅

Person()
name : 홍길동
id

Student()
grade
depertment

② 다운캐스팅

Person()
name : 홍길동
id

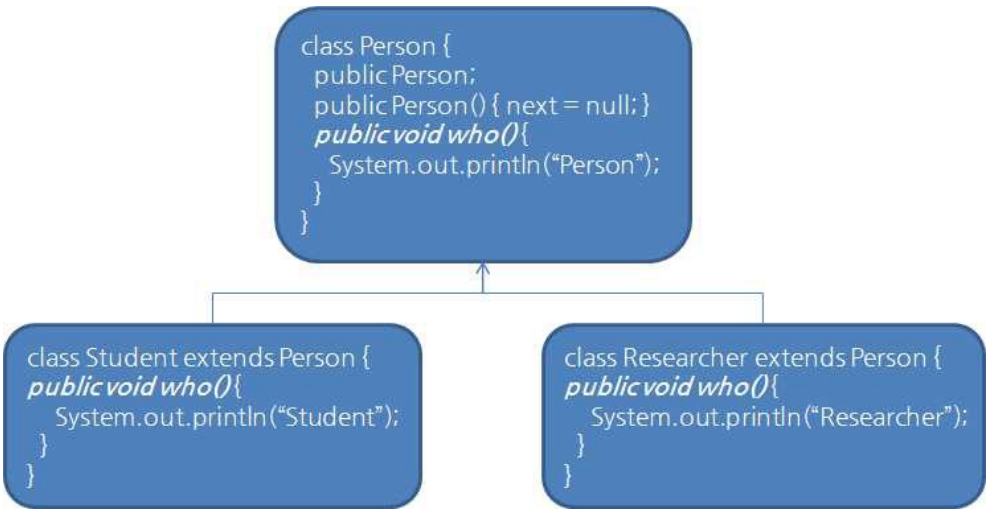
Student()
grade : A
depertment

Part3. Overriding

슈퍼클래스와 서브 클래스의 메소드 사이에 발생하는 관계.
슈퍼클래스의 메소드를 동일한 이름으로 서브 클래스에서 재작성하는 것.
super 키워드를 통해서 슈퍼 클래스에 대한 멤버와 메소드에 접근 가능.

- 오버라이딩의 조건

① 슈퍼 클래스의 메소드와 완전히 동일한 메소드를 재정의.
② 슈퍼 클래스 메소드의 접근 지정자보다 접근의 범위가 좁아질 수 없음.
③ 리턴 타입만 다를 수 없음.



```
public  class  Overriding {
        public static void main(String[] args) {
                Person p = new Person();
                Student st = new Student();
                Person p1 = new Researcher();
                Person  p2  = st;

                p.who();
                st.who();
                p1.who();
                p2.who();
        }
}
```

☞ 출력결과 : Person / Student / Researcher / Student

Student와 Researcher의 who() 메소드는 오버라이딩 된 형태이다. 때문에 자식 클래스에 있는 함수가 실행된다.

- 메소드 오버로딩 vs 메소드 오버라이딩

	오버로딩	오버라이딩
정의	같은 클래스나 상속 관계에서 동일한 이름의 메소드 중복 작성	서브 클래스에서 슈퍼 클래스에 있는 메소드와 동일한 이름의 메소드 작성
관계	같은 클래스나 상속 관계	상속관계
목적	이름이 같은 여러 개의 메소드를 중복 정의하여 사용의 편리성 향상	서브 클래스에서 새로운 기능의 메소드를 재정의
조건	메소드 이름 동일, 메소드의 인자의 개수나 인자의 타입이 달라야 함.	메소드의 이름, 매개변수의 형태 등이 모두 동일
바인딩	정적 바인딩	동적 바인딩