Week 5-1: ADTDesign

Part1. ADT Design

Define as class.

Every obejects are allocated in heap space.

Encapsulation : Data representation + Operation

Information Hiding : Object's representation part hides, and user access object by opertion.

- Form of class

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

Constructor : The method that has same name with class name.
Method : Possibly execute. an object behavior is implemented in Method.
Method Overloading :

① Same method name

② Different number of the parameter or different type of the parameter.

③ Error : Only different return type.

- Generating Object

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

After declaring reference variable about object 'aPerson', Person object is generated by 'new' keyword.

- Member Access Modifier

Accessing class	Member Access Modifier			
	default	private	protected	public
Same package class	O	X	O	O
Different package class	X	X	X	O

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

```
        int num;

        public int getAge() {
                return age;
        }

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

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

You can define public declared name instantly. When private you define private declared age instantly, then an error occurs. Because age is declared at different class. so, you have to make a get/set method to access private variable.

Week 5-2: Class Hierarchy

Part2. Casting

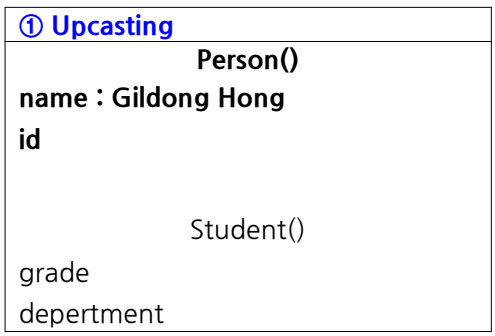
- Change an object type

Upcasting : a subclass' object change to a superclass' object.
Downcasting : Changing an Upcasted object to original. Need to type defining explicitly.

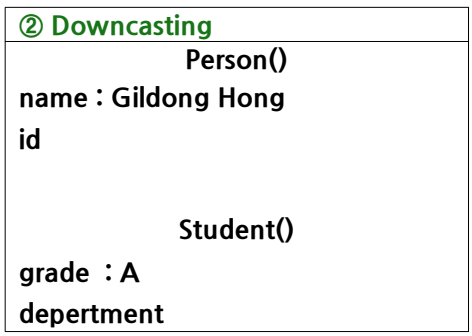
```
class Person {
        String name;
        String id;

        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("Gildong Hong"); // ① Upcasting
                System.out.println(p.name);
                p.grade = "A";
                p.department = "Computer";

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

```
        }
}
```

① Upcasting



② Downcasting



Part3. Overriding

Relation between superclass and subclass' methods.

Rewriting a subclass' method which have same name with superclass'.

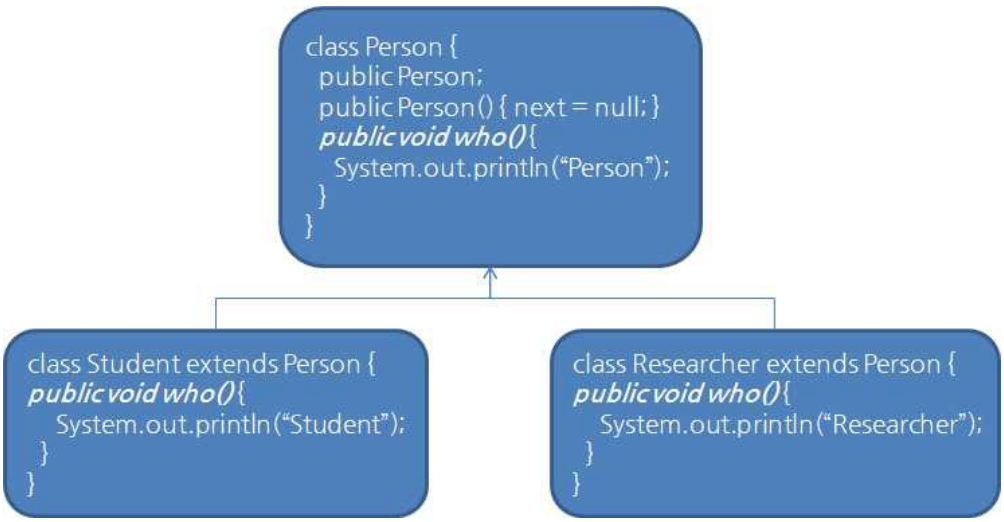
Possibly accessing superclass' member and method using 'super' keyword.

- Conditions of overriding

① Re-defining a subclass' method which is same with superclass'.

② Not possibly have a small scope than an access modifier of superclass' methods.

③ Not possibly have a different return type only.




```
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();
        }
}
```

☞ Result : Person / Student / Researcher / Student

Student and Researcher's who() method is overriding. so, a subclass' method executes.

- Method overloading vs Method overriding

	Overloading	Overriding
Definition	Rewrite a same named method in a same class or an inheritance relationship	Rewriting a subclass' method which have same name with superclass'.
Relation	a same class or an inheritance relationship	an inheritance relationship

Goal	Improving using the several same named methods.	Re-defining a new method in a subclass.
Condition	a same named method, but a different name, number, or type of method's paramter.	Everything is same form except implementing.
Biding	Static binding	Dynamic binding