Week 5-1: ADTDesign

Part1. ADT Design
Define as class.
Every objects are allocated in heap space.
Encapsulation: Data representation + Operation
Information Hiding: Object's representation part hides, and user access object by operation.

- Form of class

```java
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 the 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

```java
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

<table>
<thead>
<tr>
<th>Accessing class</th>
<th>Member Access Modifier</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>default</td>
</tr>
<tr>
<td>Same package class</td>
<td>X           O</td>
</tr>
<tr>
<td>Different package class</td>
<td>X               X</td>
</tr>
</tbody>
</table>

```
class Person {
    public String name;
    private int age;
}
```
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";
    }
}
Part 3. 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.

### Upcasting

<table>
<thead>
<tr>
<th>Person()</th>
</tr>
</thead>
<tbody>
<tr>
<td>name : Gildong Hong</td>
</tr>
<tr>
<td>id</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Student()</th>
</tr>
</thead>
<tbody>
<tr>
<td>grade</td>
</tr>
<tr>
<td>department</td>
</tr>
</tbody>
</table>

### Downcasting

<table>
<thead>
<tr>
<th>Person()</th>
</tr>
</thead>
<tbody>
<tr>
<td>name : Gildong Hong</td>
</tr>
<tr>
<td>id</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Student()</th>
</tr>
</thead>
<tbody>
<tr>
<td>grade : A</td>
</tr>
<tr>
<td>department</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th></th>
<th>Overloading</th>
<th>Overriding</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Definition</strong></td>
<td>Rewrite a same named method in a same class or an inheritance relationship</td>
<td>Rewriting a subclass' method which have same name with superclass'.</td>
</tr>
<tr>
<td><strong>Relation</strong></td>
<td>a same class or an inheritance relationship</td>
<td>an inheritance relationship</td>
</tr>
<tr>
<td>Goal</td>
<td>Improving using the several same named methods.</td>
<td>Re-defining a new method in a subclass.</td>
</tr>
<tr>
<td>------------------------------------------</td>
<td>------------------------------------------------</td>
<td>----------------------------------------</td>
</tr>
<tr>
<td>Condition</td>
<td>a same named method, but a different name, number, or type of method's parameter.</td>
<td>Everything is same form except implementing.</td>
</tr>
<tr>
<td>Biding</td>
<td>Static binding</td>
<td>Dynamic binding</td>
</tr>
</tbody>
</table>