Lab 06 Interface and State-Pattern

State-Pattern - what is it and **WHY**?

A casual story for introduction:

We now need to: Model 3 levels of happiness
Later on, we: *may add new levels !!* (software change)

An OOP principle: Open-closed Principle (OCP)
Modules should be open for extension, but closed for modification
If OCP is applied well, then further changes can be done by adding new code modules, not by changing old modules that already work.

Approach 1: if-then-else **[Bad]**

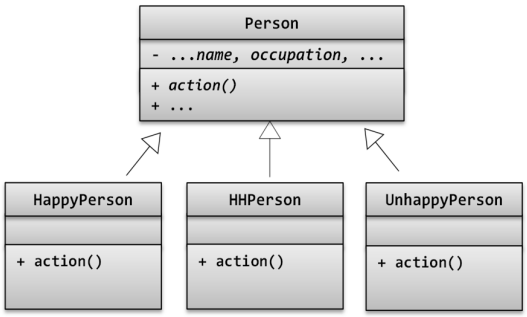
```
class Person
{
    private type happinessLevel;
    ... other fields: name, occupation..

    public void setHappinessLevel(..) {..};
    public void action() {..};
    ... other methods
}
```

```
If happy
    Output "Willing to help others";
Else if half_and_half
    Output "Living well";
Else/*unhappy*/
    Output "Seek positive growth";
```

Why **bad**?
Reason: to add new levels, need to change the if-elseif block.
That means existing code needs to be open for change.

Approach 2: Subclasses extend (inherit) superclass **[Bad]**



Why **bad**?
Reason: When happiness change, we need to *recreate* the person object (and update related links):

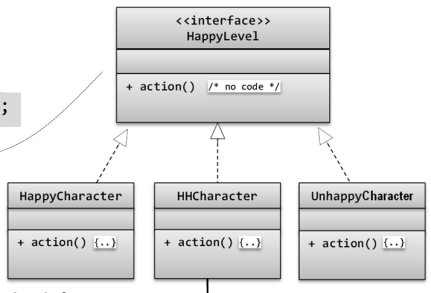
.. Manager m is previously unhappy
m = new HappyPerson(); //to change
company.changeManager(m); 😱

State-Pattern approach **[Good!]**

An object can change its behavior by changing the class of its state: `person.setHappyLevel(new HappyCharacter());`

```
class Person
{
    private HappyLevel happyLevel;
    ... other fields: name, occupation..

    public void setHappyLevel(HappyLevel h) { happyLevel = h;}
    public void action() { happyLevel.action(); }
    ... other methods
}
```



Q1. Complete the given program which lists the messages for team contacts:

- Sample rundown (underlined contents are input by the user):

```
Please input the file pathname: m1.txt

There are 4 members in the team: Helena Peter Mary Paul

Messages for team contacts:
Dear Helena,  please contact your teammates:  Peter Mary Paul
Dear Peter,  please contact your teammates: Helena  Mary Paul
Dear Mary,  please contact your teammates: Helena Peter  Paul
Dear Paul,  please contact your teammates: Helena Peter Mary
```

- Sample data file (first line is the number of members):

m1.txt
```
4
Helena
Peter
Mary
Paul
```

- Implementation of classes (Also see given code):

1. Member - Encapsulated private data:
   private String name;

   - Provide the public methods:
   public Member(String aName)
   public String getName()

2. Team - Encapsulated private data:
   private Member[] allMembers;

   - Provide the public methods:
   public Team(String filepathname) - constructor which gets member list from the file
   public int getMemberCount()
   public String getStringOfAllMembers()
   public void printTeamContactMessages()

Q2. Extend the program - create output based on the roles of the members: *normal member* or *leader*

- Sample rundown (underlined contents are input by the user):

```
Please input the file pathname: m2.txt

There are 4 members in the team: Helena Peter[Leader] Mary Paul

Messages for team contacts:
Dear Helena, Please contact your leader: Peter
Dear Peter, Please contact your members: Helena Mary Paul
Dear Mary, Please contact your leader: Peter
Dear Paul, Please contact your leader: Peter
```

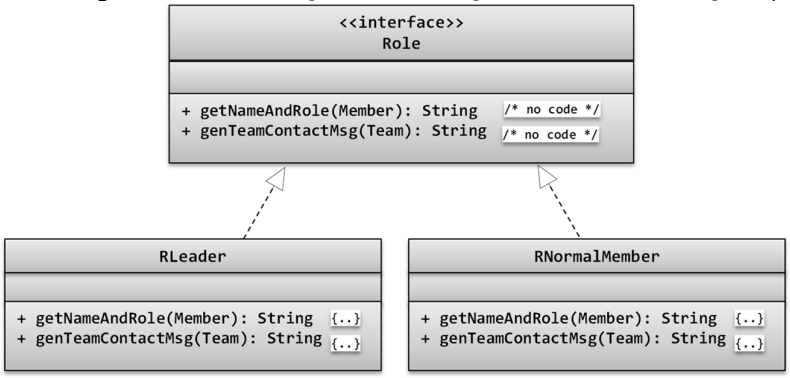
- Sample data file (Each name is followed by 'n'/'l': *normal member* or *leader*):

m2.txt
```
4
Helena n
Peter l
Mary n
Paul n
```

- Implementation (next page)

Implementation - Apply state-pattern approach:

(1) Add the following modules in **`Role.java`**, **`RLeader.java`**, **`RNormalMember.java`** (Also see given code):



(2) Change the original Classes:

1. Member - Encapsulated private data:

```
private String name;
private Role role;
```

- Public methods:

```
public Member(String aName, Role aState) {..}
public String getName() {..}
public Role getRole() {return role;}
public String getNameAndRole() {return role.getNameAndRole(this);}
```

Call the role's getNameAndRole() method to work out the the string like "Helena", "Peter[Leader]".

2. Team — See givenCode.txt

- Encapsulated private data:

```
an array of Members
```

- Modify original methods:

```
public Team(String filepathname)
public int getMemberCount()
public String getStringOfAllMembers()
public void printTeamContactMessages()
```

Now need to handle the roles

Now use the members' getNameAndRole()

Now use the roles' genTeamContactMsg ()

- Add new methods (used for role-specific messages)

```
public Member getLeader()
public String getStringOfNormalMembers()
```

Notes: ↙ More to learn in Topic05

- Role.java contains the Role interface. An interface is just like a class, but no implementation code for methods.
- RLeader.java and RNormalMember.java : The classes which *implement* the Role interface

Role.java

```
public interface Role
{
  public String getNameAndRole(Member member);
  public String genTeamContactMsg(Team team);
}
```

RLeader.java

```
public class RLeader implements Role
{
}
```

RNormalMember.java

```
public class RNormalMember implements Role
{
  public String genTeamContactMsg(Team team) {..}
  public String getNameAndRole(Member member) {..}
}
```

- You will need to use the **instanceof** operator :
  It checks whether an object is an instance of a class.
  E.g. `if (allMembers[i].getRole() instanceof RNormalMember)`

Submit your work to PASS. You can use a zip file (but Main.java should be separated):

H:\TestingLab05\src\src.zip

H:\TestingLab05\src\Main.java

For Mac users: Please read the announcement about using online ZIP tools.

Q3. Extend the program - Add the "Disappeared" Role

- Sample rundown (underlined contents are input by the user):

```
Please input the file pathname: m3.txt

There are 4 members in the team: Ann Daisy[Leader] Jack Joey[x]

Messages for team contacts:
Dear Ann, Please contact your leader: Daisy
Dear Daisy, Please contact your members: Ann Jack
Dear Jack, Please contact your leader: Daisy
Dear Joey, When you are back, please contact your leader: Daisy
```

- Sample data file
  (Each name is followed by 'n'/'l'/'d': *normal member* or *leader* or *disappeared*):

M3.txt

```
4
Ann n
Daisy l
Jack n
Joey d
```

- Implementation:
  (1) Add the `RDisappeared` class: `public class RDisappeared implements Role`
  (2) Modify the original code where necessary.

Submit your work to PASS.

Lab06 - Progress

• I have seen the basic use of **State Pattern** in _____(Q1 / Q2)

• In Q3, I have added the _____(RDisappeared/RLeader/RNormalMember) class to model the new kind of role.

The previous classes (RLeader and RNormalMember) _____(do not need / have) to change.
Therefore OCP is fullfulled:

[Review page 1]
An OOP principle: Open-closed Principle (OCP)
Modules should be _____(open/close) for extension, but _____(open/close) for modification
If OCP is applied well, then further changes can be done by _____(adding new / changing old) code modules, not by _____(adding new / changing old) modules that already work.

• I've applied _____(proper / improper ☺) formatting (**especially** Indentations) in the programs.

Anything unclear? Ask!

Q4. [Homework] Rewrite the program - Change of the leader

- Sample rundown (underlined contents are input by the user):

```
Please input the file pathname: m3.txt
There are 4 members in the team: Ann Daisy[Leader] Jack Joey[x]
Enter new leader: Jack
Result: Ann Daisy Jack[Leader] Joey[x]
```

- Implementation:

(1) Add a `setRole` method in `Member.java`

(2) Add the following methods in `Team.java`

```
//Return the member with the given name
public Member findMember(String name) {..}
Note: please use .equals for comparing strings (Topic02 P.9)

//Change of leader
public void changeLeader(String newLeaderName) {..}
Note: change the roles like: originalLeader.setRole(new RNormalMember()); newLeader.setRole(new RLeader());
```

(3) Change `main()`according to the sample rundown.

- Submit your work to PASS.

Suggested practices: (a) use for-each loop, (b) use `ArrayList<Member>` instead of `Member[]`

Q5. [Homework] Rewrite the program for searching a member within some teams.p

- Sample rundown #1 (underlined contents are input by the user):

```
Please input the file pathname of each team: c:\m3.txt c:\m3a.txt c:\m3b.txt
Listing of teams:
[Team 1] 4 members: Ann Daisy[Leader] Jack Joey[x]
[Team 2] 5 members: Helena[x] Peter[x] Mary Paul Jacky[Leader]
[Team 3] 3 members: Amy Jane[Leader] Johnson[x]
Enter a name for searching: Helena
Result: Helena is a disappeared member in Team 2
```

↙ Assume no space in each file pathname

Depend on the role, it should show
"the leader",
"a disappeared member", or
"a normal member"

- Sample rundown #2 (underlined contents are input by the user):

```
Please input the file pathname of each team: c:\m3.txt c:\m3a.txt
Listing of teams:
[Team 1] 4 members: Ann Daisy[Leader] Jack Joey[x]
[Team 2] 5 members: Helena[x] Peter[x] Mary Paul Jacky[Leader]
Enter a name for searching: Johnson
Result: Not found
```

- Implementation:

(1) The user's input line of file pathnames may contain 1 or more files.
Therefore you need to use a loop to ***extract each one*** from ***the string of the whole input line***
- Refer to "**Reading input from another string**" in Topic02 P.10; and the given code for Lab05Q6(aii)

(2) Store the teams in **an array list of teams**. (You may review `ArrayList` in Lab05 Page 2)

(3) Proper design to get the role description:
- Add the following public method in the **Member** class:

```
public String getRoleDescription() {return role.getRoleDescription();}
```

Also add such a method in **Role.java**, and implement it in **RLeader.java etc..,** to return "the leader", "a disappeared member", or "a normal member"

-- END --