Week 6: Interfaces

Part 1. Abstract Class
- Abstract class' inheritance
  When subclass is inherited from an abstract class, a subclass become an abstract class.
  For preventing a subclass from becoming an abstract class, abstract methods have to be overrided.

```java
abstract class Person {
    public Person();
    public Person() { next = null; }
    abstract public void who();
}
```

- Feature of interface
  ① Only Being composed of abstract methods and constants.
  ② Every methods can omit 'abstract public'.
  ③ Constants can omit 'public static final'.
  ④ Not possibly generating interface's objects.

Part 2. Interfaces
  One of the abstract classes, make a multiple inheritance possibly. 'implements' keyword means implementing abstract methods in the class.
5. Possibly inherited from other interface.
6. Interface can be used as a type of reference variable.

- Example

```java
interface CanFight {
    void fight();
}

interface CanSwim {
    void swim();
}

interface CanFly {
    void fly();
}

class ActionCharacter {
    public void fight() {
        System.out.println("fight");
    }
}

class Hero extends ActionCharacter implements CanFight, CanSwim, CanFly {
    public void swim() {
        System.out.println("swim");
    }

    public void fly() {
        System.out.println("fly");
    }
}

public class Adventure {
    public static void t(CanFight x) {
        x.fight();
    }

    public static void u(CanSwim x) {
        x.swim();
    }
}
```
public static void v(CanFly x) {
    x.fly();
}

public static void w(ActionCharacter x) {
    x.fight();
}

public static void main(String[] args) {
    Hero h = new Hero();
    t(h); // Treat it as a CanFight
    u(h); // Treat it as a CanSwim
    v(h); // Treat it as a CanFly
    w(h); // Treat it as an ActionCharacter
}

-Example: interface including inheritance

interface Monster {
    void menace();
}

interface DangerousMonster extends Monster {
    void destroy();
}

interface Lethal {
    void kill();
}

class DragonZilla implements DangerousMonster {
    public void menace() {
        System.out.println("menace1");
    }

    public void destroy() {
        System.out.println("destory1");
    }
}
interface Vampire extends DangerousMonster, Lethal {
    void drinkBlood();
}

class VeryBadVampire implements Vampire {
    public void menace() {
        System.out.println("menace2");
    }

    public void destroy() {
        System.out.println("destroy2");
    }

    public void kill() {
        System.out.println("kill2");
    }

    public void drinkBlood() {
        System.out.println("drinkBlood2");
    }
}

public class HorrorShow {
    static void u(Monster b) {
        b.menace();
    }

    static void v(DangerousMonster d) {
        d.menace();
        d.destroy();
    }

    static void w(Lethal l) {
        l.kill();
    }

    public static void main(String[] args) {
        DangerousMonster barney = new DragonZilla();
    }
}
```java
System.out.println("1");
u(barney);
System.out.println("2");
v(barney);
Vampire vlad = new VeryBadVampire();
System.out.println("3");
u(vlad);
System.out.println("4");
v(vlad);
System.out.println("5");
v(vlad);
}
```

Possibly inheriting a multiple interface, making a interface. You can check a multiple inheritance using interface.

- **Interface vs Abstract class**

<table>
<thead>
<tr>
<th></th>
<th>Abstract class</th>
<th>Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>- Including not abstract methods.</td>
<td>- Every methods are abstract methods.</td>
</tr>
<tr>
<td></td>
<td>- Constant and variable fields can include.</td>
<td>- Only constant fields include.</td>
</tr>
<tr>
<td></td>
<td>- Abstract class is suitable when every subclasses</td>
<td>- supporting a multiple inheritance.</td>
</tr>
<tr>
<td></td>
<td>have a same method.</td>
<td></td>
</tr>
</tbody>
</table>

Distributed Computing Systems Lab
CSE, SNU