

# Inheritance

Hsuan-Tien Lin

Department of CSIE, NTU

OOP Class, March 29-30, 2010

# Has-A (1/3)

```
1  class Person{
2      String name; Pet pet;
3      void sing(){
4          System.out.print(name + "_has_a_" + pet.name + ",_");
5          System.out.println(pet.type + ",_" + pet.type + ".");
6      }
7  }
8  class Pet{ String type; }
9  public class PetDemo{
10     public static void main(String [] argv){
11         Person mary = new Person();
12         mary.name = "Mary";
13         mary.pet = new Pet();
14         mary.pet.type = "little_lamb";
15         mary.sing();
16     }
17 }
```

*type*

- has-a: a basic way of re-using variables/methods in other classes
- “Mary has a little lamb.”

## Has-A (2/3)

```
1  class POOBBS{
2      POOBoard board;
3      POOMessage message;
4      POOVote vote;
5      POOCasino casino;
6      POOMail mail;
7  }
```

- has-a: a basic way of defining the components of a system
- “POOBBS has a casino system.”

## Has-A (3/3)

```
1  class User{
2      User[] friendlist;
3      int friendcount;
4      void addFriend(User friend){
5          friendlist[friendcount++] = friend;
6      }
7  }
```

- has-a: a basic mechanism for objects to “connect” with each other
- “He has two friends: George and Mary.”

most basic design component of OOP

# Is-A (1/2)

```
1  class SYSOP{
2      User user;
3      void talk_to_friend(User a_user){
4          user.talk_to_friend(a_user);
5      }
6      void talk_to_sysop(SYSOP a_sysop){
7          talk_to_friend(a_sysop.user);
8      }
9      void reset_user_money(User a_user){
10         a_user.money = 0;
11     }
12     void reset_sysop_money(SYSOP a_sysop){
13         reset_user_money(a_sysop.user);
14     }
15 }
```

- **SYSOP is a user**
- can be implemented via **has-a**  
—everyone has a child living in his heart—  
but complicated

## Is-A (2/2)

```
1  class User{
2      int money;
3      void talk_to_friend(User a_user){ }
4  }
5  class SYSOP extends User{
6      //money "inherited" from the definition above
7      //talk_to_friend "inherited" from the definition above
8
9      void reset_user_money(User a_user){
10         a_user.money = 0;
11     }
12     //reset_sysop_money no need, because SYSOP is a User
13 }
```

- TypeA **extends** TypeB: TypeA is a (special case of) TypeB
- TypeSubClass (TypeDerivedClass or ChildClass) **extends** TypeSuperClass (or TypeBaseClass or ParentClass)
- another way of reusing code

is-a  $\neq$  has-a  
**extends** better suited for the former



## Uses of Is-A (1/4)

```
1  class CassetePlayer{
2      void play(Cassete c){ }
3  }
4  class CDandCassetePlayer extends CassetePlayer{
5      //play(Cassete) inherited
6      void play(CD c){ }
7  }
```

- CDandCassetePlayer is a (extended) CassetePlayer

## Uses of Is-A (2/4)

```
1  class Player{
2      void play(){ }
3  }
4  class CassetePlayer extends Player{
5      void play(Cassete c){ insert(c); play();}
6  }
7  class CDPlayer extends Player{
8      void play(CD c){ insert(c); play();}
9  }
```

- CassetePlayer is a (concrete description of) Player
- CDPlayer is a (concrete description of) Player

## Uses of Is-A (3/4)

```
1  class CassetePlayer{
2      void play(Cassete c){ }
3  }
4  class CDPlayer extends CassetePlayer{
5      void play(Cassete c){
6          System.out.println("I_cannot_play_Cassete");
7      }
8      void play(CD c){ }
9  }
```

- CDPlayer is a (update of my) CassetePlayer
- some behaviors “changed” (will discuss in more detail later)
- not semantically as clear as the previous cases, but a potential trick in the OO World

## Uses of Is-A (4/4)

```
1  class Player{
2      void play(){ }
3  }
4  class CassetePlayer extends Player{
5      void play(Cassete c){ insert(c); play(); }
6  }
7  class CDPlayer extends Player{
8      void play(CD c){ insert(c); play(); }
9  }
10 class CDandCassetePlayer extends CassetePlayer and CDPlayer{
11     //play(Cassete) inherited
12     //play(CD) inherited
13 }
```

- multiple inheritance: **not** supported in Java
- think: Professor CharlieL and Alumnus CxxxxxxL conflict
- Java: basically, single inheritance

## Use of Is-A: Key Point

- is an extended type of
  - FunProfessor is Professor who can also tell jokes
- is a more concrete/restricted description of
  - YoungProfessor is Professor who is young
- (is an update of)
  - NewProfessor replaces ExistingProfessor
- is a TypeA and is a TypeB (no!)
  - both Fun and Young?

## More on Is-A (1/2)

```
1  class CSIEProfessor{ String name; int office_num; }
2  class HTLin extends CSIEProfessor{
3      //HTLin is a CSIEProfessor
4      HTLin(){ name = "HTLin"; office_num = 314; }
5  }
6  class CLChen extends CSIEProfessor{
7      //CLChen is a CSIEProfessor
8      String title;
9      CLChen(){
10         name = "CLChen"; office_num = 500; title = "Vice_Chair";
11     }
12 }
13 class YDLyuu extends CSIEProfessor{
14     //YDLyuu is a CSIEProfessor
15     String title;
16     YDLyuu(){
17         name = "YDLyuu"; office_num = 200; title = "Chair";
18     }
19 }
```

- over-use of **extend** for is-a: one class describes one instance
- usual goal of OOP: one class, **many** instances

## More on Is-A (2/2)

```
1  class CSIEProfessor{
2      String name; String title ;
3      int office_num;
4      CSIEProfessor(String name, String title , int office_num){ }
5      bool is_chair(){ return title.equals("Chair"); }
6      bool decide_budget(){ if (is_chair()){ /* ... */ } }
7  }
8
9  class CSIEProfessorDemo{
10     public static void main(String[] argv){
11         CSIEProfessor = new CSIEProfessor("YDLyuu", "Chair",
12             200);
13         YDLyuu.decide_budget();
14     }
15 }
```

- under-use of **extend** for is-a: overly complicated class CSIEProfessor
- potential for bugs/hacks  
what if `new CSIEProfessor("YAOMMENT", "Chair", 0)`?

## More on Is-A: Key Point

```
class ViceChair extends Professor
    is a kind (type) of
Professor HTLin = new Professor()
    is an instance of
```




# Instance Variables and Inheritance (1/3)

```
1  class Professor{
2      public String name;
3
4      public String get_name() {
5          return name;
6      }
7  }
8  class CSIEProfessor extends Professor{
9      public int office_number;
10     public int get_office_number() {
11         return office_number;
12     }
13 }
```

- CSIEProfessor: two instance variables;  
Professor: one instance variable

## Instance Variables and Inheritance (2/3)

```
1  class Professor{
2      public String name;
3      public String get_name(){ return name;}
4  }
5  class CSIEProfessor extends Professor{
6      public String name; //?!
7      public int office_number;
8      public int get_office_number(){ return office_number;}
9      public String get_this_name(){ return name;}
10 }
11 /* CSIEProfessor HTLin = new CSIEProfessor();
12    Professor HTLin = new CSIEProfessor();
13    Professor HTLin = new Professor(); */
14 /* System.out.println(HTLin.get_name());
15    System.out.println(HTLin.get_this_name());
16    System.out.println(HTLin.name); */
```



- CSIEProfessor: three instance variables;  
Professor: one instance variable
- which name will we get?

## Instance Variables and Inheritance (3/3)

```
1  class Professor{
2      public String p_name;
3      public String get_name(){ return p_name; }
4  }
5  class CSIEProfessor extends Professor{
6      public String c_name;
7      public int office_number;
8      public int get_office_number(){ return office_number;}
9      public String get_this_name(){ return c_name;}
10 }
```

- an (almost) equivalent view of what the compiler sees

## Instance Variables and Inheritance: Key Point

- instance variable binding: determined at compile time
- same “name” can co-exist in a derived class, binding determined by compile-time type