Inheritance

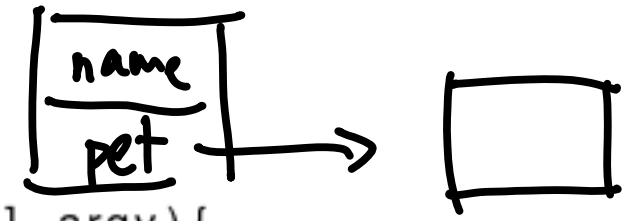
Hsuan-Tien Lin

Department of CSIE, NTU

OOP Class, March 21, 2013

Has-A (1/3)

```
1  class Person{
2    String name; Pet pet;
3    void sing(){
4      System.out.print(name + " has a " + pet.type + ", ");
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 }
```



- 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  }
```

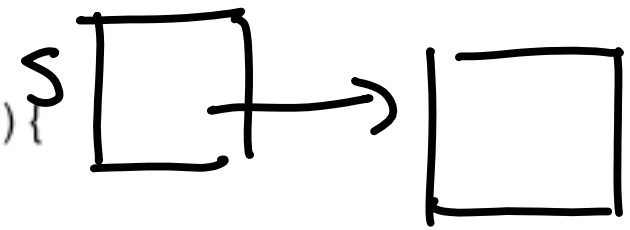
- has-a: a basic mechanism for objects to "connect" with each other
- "He has two friends: George and Mary."

Has-A: Key Point

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: Key Point

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)

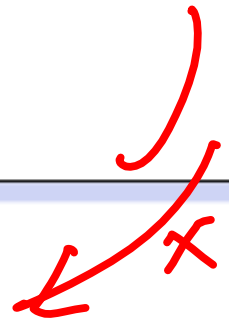
```
1  class CassetePlayer{
2    void play(Cassete c){ }
3  }
4  class CDPlayer extends CassetePlayer{
5    void play(Cassete c){
6      System.out.println("I cannot play Cassete");
7    }
8    void play(CD c){ }
9  }
```

改寫 override

- 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?