

Inheritance

Hsuan-Tien Lin

Department of CSIE, NTU

OOP Class, April 14, 2009

Instance Variables and Inheritance (1/3)

```
1  class Professor{
2      String title;
3      Professor(){ title = "NTU"; }
4      public String get_title(){ return title; }
5  }
6  class CSIEProfessor extends Professor{
7      String title; //what?
8      CSIEProfessor(){ title = "CSIE"; //which title?}
9      public String get_csie_title(){ return title; //which title?}
10 }
11
12 CSIEProfessor HTLin1 = new CSIEProfessor();
13 System.out.println(HTLin1.get_title()); //NTU
14 System.out.println(HTLin1.get_csie_title()); //CSIE
15 Professor HTLin2 = new CSIEProfessor();
16 System.out.println(HTLin2.get_title()); //NTU
17 System.out.println(HTLin2.get_csie_title()); //hahaha
18 Professor HTLin3 = new Professor();
19 System.out.println(HTLin3.get_title()); //NTU
20 System.out.println(HTLin3.get_csie_title()); //hahaha
```

Instance Variables and Inheritance (2/3)

What TYPE-of-reference wants to access its title?

```
1  class Professor{
2      String title; /* Professor (this) */
3      Professor(){ title = "NTU"; } /* Professor (this) */
4      public String get_title(){ return title; }
5      /* Professor (this) */
6  }
7  class CSIEProfessor extends Professor{
8      String title; /* CSIEProfessor (this) */
9      CSIEProfessor(){ title = "CSIE"; } /* CSIEProfessor (this) */
10     public String get_csie_title(){ return title; }
11     /* CSIEProfessor (this) */
12     public String get_super_title(){ return super.title; }
13     /* Professor (super) */
14     public String get_super_title_2(){
15         return ((Professor)this).title;
16     }
17     /* Professor (this as Professor) */
18 }
```

Instance Variables and Inheritance (3/3)

What TYPE-of-reference wants to access its title?

```

1  CSIEProfessor HTLin1 = new CSIEProfessor();
2  System.out.println(HTLin1.title); /* CSIEProfessor (HTLin1) */
3  Professor HTLin2 = new CSIEProfessor();
4  System.out.println(HTLin2.title); /* Professor (HTLin2) */
5  Professor HTLin3 = new Professor();
6  System.out.println(HTLin3.title); /* Professor (HTLin3) */
7  CSIEProfessor HTLin4 = new CSIEProfessor();
8  System.out.println(((Professor)HTLin4).title);
9  /* Professor (HTLin4 as Professor) */

```

exercise. `((CSIEProfessor)HTLin2).title`

- compile-time (static) binding: compiler easily decide intention by declared type
- instance variables: static binding, new variable w/ same name “hides” old one (when using new type)

Class Variables and Inheritance (1/1)

```
1  class Professor{
2      static int count;
3      Professor(){ count++; } //i.e., Professor.count++;
4  }
5  class CSIEProfessor extends Professor{
6      static int count;
7      CSIEProfessor(){
8          super(); count++; //i.e., CSIEProfessor.count++;
9      }
10 }
11 System.out.println(CSIEProfessor.count); //clear intention
12 System.out.println(Professor.count); //clear intention
13 //intention determined by reference TYPE
14 CSIEProfessor HTLin1 = new CSIEProfessor();
15 System.out.println(HTLin1.count); //CSIEProfessor.count
16 Professor HTLin2 = new CSIEProfessor();
17 System.out.println(HTLin2.count); //Professor.count
18 Professor HTLin3 = null;
19 System.out.println(HTLin3.count); //Professor.count
```

- class variables: static binding (as said before)

Class Methods and Inheritance (0/0)

- class methods: exactly the same as class variables
intention determined by directly using class names, or by
reference TYPE

Instance Methods and Inheritance (1/2)

```

1  class Professor{
2      String skill = "tell_jokes";
3      public String get_skill(){ return skill; }
4  }
5  class JaiJaiProfessor extends Professor{
6      public String get_skill(){ return skill + ",_play_on_BBS"; }
7  }
8  }
9  JaiJaiProfessor HTLin1 = new JaiJaiProfessor();
10 System.out.println(HTLin1.get_skill());
11 System.out.println(((Professor)HTLin1).get_skill());
12 Professor HTLin2 = new JaiJaiProfessor();
13 System.out.println(HTLin2.get_skill());
14 System.out.println(((JaiJaiProfessor)HTLin2).get_skill());
15 Professor HTLin3 = new Professor();
16 System.out.println(HTLin3.get_skill());
17 System.out.println(
18     ((JaiJaiProfessor)HTLin3).get_skill()
19 );//ohohoh, why do you want to be JaiJai?

```

(Static) ★ dynamic
 t_j p b t_j p b
 t_j t_j p b
 t_j t_j p b
 t_j p b t_j p b
 t_j t_j

Instance Methods and Inheritance (2/2)

```

1  class Professor{
2      private String skill = "tell_jokes";
3      public String get_skill(){ return skill; }
4      public String get_skill_ohohoh(){ return skill; }
5  }
6  class JaiJaiProfessor extends Professor{
7      public String get_skill_hahaha(){ return skill + ",_play_on_BBS";
8      public String get_skill_ohohoh(){
9          return get_skill_ohohoh() + ",_play_on_BBS";
10         }
11         ↑thrs.
12     public String get_skill(){
13         return super.get_skill() + ",_play_on_BBS";
14     }
15 }

```

- the hahaha version: won't work because skill is private
- the ohohoh version: stack overflow (a.k.a. run out of paper)

Key Point: Type versus Content

- static (compile-time) binding: determined by type
 - instance variables
 - class variables
 - class methods

—inherited class “hides” things from base class

—hidden stuff can be revealed by casting
- dynamic (run-time) binding: determined by content
 - all (non-private) instance methods in Java
 - “virtual” instance methods in C++

—inherited class “overrides” things from base class

—overridden stuff cannot be revealed by casting