

Inheritance

Hsuan-Tien Lin

Department of CSIE, NTU

OOP Class, April 12-13, 2010

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 } X CSIEProfessor Q = new Professor();
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

Instance Methods and Inheritance (1/2)

```
1  class Professor{
2      public void say_hello(){
3          System.out.println("Hello!");
4      }
5  }
6  class CSIEProfessor extends Professor{
7      public void play_BBS(){
8          System.out.println("Fun!");
9      }
10 }
```

- CSIEProfessor: two instance methods;
Professor: one instance method

Instance Methods and Inheritance (2/2)

```
1  class Professor{
2      public void say_hello(){
3          System.out.println("Hello!");
4      }
5  }
6  class CSIEProfessor extends Professor{
7      public void say_hello(){
8          System.out.println("May_the_OOP_course_be_with_you!");
9      }
10 }
11 /* alternatives
12 CSIEProfessor HTLin = new CSIEProfessor();
13 Professor HTLin = new CSIEProfessor();
14 Professor HTLin = new Professor();
15 */
16 /* calls
17 HTLin.say_hello();
18 */
```

P C H O
F -
T F

- which say_hello() will be called?

Instance Methods and Inheritance: Key Point

instance method binding: dynamic, depending on run-time instance types

Reference Assignment and Inheritance (1/4)

```
1  class Student{ int ID; String name; }
2  class GodStudent extends Student{ Award[] president_awards; }
3
4  public class StudentDemo{
5      public static show_student_id(Student s){
6          System.out.println(s.ID);
7      }
8      public static void main(String[] argv){
9          GodStudent seanwu = new GodStudent();
10         show_student_id(seanwu);
11         Student CharlieL = new Student();
12         show_student_id(CharlieL);
13     }
14 }
```

- seanwu refers to an instance of GodStudent (for sure!)
- seanwu refers to an instance of Student as well
- one instance, many coherent types (in argument passing, return value, etc.)

Reference Assignment and Inheritance (2/4)

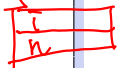


```
1  class Student{ int ID; String name; }
2  class GodStudent extends Student{ Award[] president_awards; }
3
4  public class StudentDemo{
5      public static void main(String[] argv){
6          GodStudent seanwu = new GodStudent();
7          //I want to be a usual student
8          Student usualstudent = seanwu;
9          Student anotherusualstudent = new Student();
10     }
11 }
```

us [123]

SW [123]

us [456]



- if “copying assignment”, a copy of seanwu can be a usual student
- but “reference assignment” in Java, how can seanwu be usual?
- mechanism?

Reference Assignment and Inheritance (3/4)

```
1  class Student{ int ID; String name; }
2  //takes 4 + (8) bytes
3  class GodStudent extends Student{ Award[] president_awards; }
4  //takes 4 + (8) + (8) bytes
5
6  /** excluding some other information, much like
7  class GodStudent{
8      //first 4 + (8) bytes (for Student)
9      int ID;
10     String name;
11     //last (8) bytes (for GodStudent)
12     Award[] president_awards;
13 }//takes 4 + (8) + (8) bytes
14 **/
```

- one possible mechanism for single inheritance (Java!!): shared prefix (virtually)

Reference Assignment and Inheritance (4/4)

```
1  class Student{ int ID; String name; }
2  //class Student{ int 000; reference 004; }
3  class GodStudent extends Student{ Award[] president_awards; }
4  //class GodStudent{ int 000; reference 004; reference 012; }
```

- one possible mechanism: variable name \Rightarrow a single number when class **loads**
- (after instance type check) objects can just safely access memory contents by numbers

a simple run-time mechanism (shared prefix):
ancestor first, descendant last

Note, however, that *the Java virtual machine does not mandate any particular internal structure for objects.*

Constructor and Inheritance (1/3)

```
1  class Student{ int ID; String name;
2      Student(int ID, String name){this.ID = ID; this.name =
        name;}
3  }
4  class AwardStudent extends Student{
5      Award[] awards;
6      AwardStudent(int ID, String name, int nAward){
7          super(ID, name); //means invoke Student(ID, name);
8          awards = new Award[nAward];
9      }
10 }
```

- initialize the ancestor part **first**
- construct AwardStudent \Rightarrow (i.e. calls) construct Student
- thus, the “Student” parts of the memory are initialized first, then the “AwardStudent” part

Constructor and Inheritance (2/3)

```
1  class Student{ int ID; String name;
2      Student(int ID, String name){this.ID = ID; this.name =
        name;}
3  }
4  class AwardStudent extends Student{
5      Award[] awards;
6      AwardStudent(int ID, String name, int nAward){
7          super(ID, name); //means invoke Student(ID, name);
8          //any utility function in Student can be used at this
                stage
9          awards = new Award[nAward];
10     }
11 }
```

- `super` goes first so that there is a valid “Student” object in the very beginning

Constructor and Inheritance (3/3)

java.lang.Object

```
1  class Student /* extends Object */ {
2      int ID; String name;
3      Student(int ID, String name){this.ID = ID; this.name =
4          name;}
5      /*
6      Student(int ID, String name){
7          super(); //like Object();
8          this.ID = ID; this.name = name;
9      }
10     */
11 }
12 class AwardStudent extends Student{
13     Award[] awards;
14     AwardStudent(int ID, String name, int nAward){
15         //?
16     }
17 }
```

- `super()` automatically added if no explicit call in the beginning

A Fallback: Constructor Calls

```
1  class Record{
2      String name; int score;
3      Record(int init_score){score = init_score;}
4      Record() { Record(40);}
5  }
6  public class RecordDemo{
7      public static void main(String[] arg){
8          Record r1 = new Record(60);
9          Record r2 = new Record();
10         System.out.println(r1.score);
11         System.out.println(r2.score);
12     }
13 }
```

- there is a bug above
- one constructor can only call one other constructor (via this or super)

Constructor and Inheritance: Key Point

calls ancestor constructor first
(and thus ancestor forms first)

Private Variables and Inheritance (1/1)

```
1  class Parent{
2      private int hidden_money;
3      public void show_hidden_money_amount() { }
4  }
5  class Child extends Parent{
6      void spend_money() {
7          // (1) can hidden_money be spent here?
8          // (2) can show_hidden_money_amount() be called here?
9      }
10 }
```

- (1) not directly (2) yes
- does Child have a hidden_money slot in the memory?
 - yes, to make show_hidden_money_amount() work!

Private Variables and Inheritance: Key Point

private variables are still “inherited” in memory, but not “visible” to the subclass because of encapsulation

Private Methods and Inheritance (1/1)

```
1  class Parent{
2      private int hidden_money;
3      private void buy_liquor(){ }
4      public void show_hidden_money_amount(){ }
5  }
6  class Child extends Parent{
7      void buy(){
8          //can buy_liquor() be called here?
9      }
10 }
```

- no, because cannot see
- private methods effectively not inherited

Private Methods and Inheritance: Key Point

private methods effectively not inherited because not “visible” to the subclass

More on Access Permissions (1/2)

```
1 package generation.old;
2 class Parent{
3     private int hidden_money;
4     public void show_hidden_money_amount() { }
5     /* default */ void middle_age_issues();
6     /* default */ void cross_gen_issues();
7 }
8 //different file , Child.java
9 package generation.new;
10 class Child extends Parent{
11 }
```

- in Child, can hidden_money be accessed? no.
- can show_hidden_money_amount be accessed? yes.
- can middle_age_issues be accessed? no.
- can cross_gen_issues be accessed? no.
—what if we want “yes”?

More on Access Permissions (2/2)

```
1  package generation.old;
2  public class Parent{
3      private int hidden_money;
4      public void show_hidden_money_amount(){ }
5      /* default */ void middle_age_issues();
6      protected void cross_gen_issues();
7  }
8  //different file , Child.java
9  package generation.young;
10 class Child extends Parent{
11 }
```

- can `cross_gen_issues` be accessed? no.
—what if we want “yes”?
- `protected`: accessible to `Child` (sub-classes) and `Friend` (same-package-classes)

More on Access Permissions: Key Point

`public`: accessible to everyone

`protected`: accessible to sub-classes and same-package-classes

(default): accessible to same-package-classes

`private`: accessible within my class definitions

Permissions and Method Overriding (1/1)

```
1  public class Parent{
2      protected void method(){ }
3  }
4  class Child extends Parent{
5      public void method(){ } //?
6      private void method(){ } //?
7  }
8
9  //bottom line
10 Parent var = new Child();
11 var.method();
```

- need last two lines to work for things same-package as Parent
- need last two lines to reflect overriding (calling Child's)
- Child: same or more open than Parent

Permissions and Method Overriding: Key Points

Child: same or more open than Parent

More on super (1/1)

```
1  class Student{
2      int ID; String name;
3      void study_for_midterm() {
4          System.out.println("I_am_studying_for_midterm.");
5      }
6  }
7  class AwardStudent extends Student{
8      Award[] president_awards;
9      void study_for_midterm() {
10         for(int i = 0; i < 10; i++)
11             super.study_for_midterm();
12     }
13 }
```

- super: much like (ParentName) this (but NOT the same)

More on `super`: Key Point

`this`: a Sub-type reference variable pointing to the object itself
`super`: a Base-type reference variable pointing to the object itself
same reference value, different type
`super.super`: **no**

Inheritance (1/5)

```
1  class Professor{ }
2  class CSIEProfessor extends Professor{ }
3  class EEProfessor extends Professor{ }
4
5  class Demo{
6      public static void main(String[] argv){
7          CSIEProfessor HTLin = new CSIEProfessor();
8          System.out.println(HTLin.getClass());
9          System.out.println(HTLin instanceof CSIEProfessor);
10         System.out.println(HTLin instanceof Professor);
11         System.out.println(HTLin instanceof EEProfessor);
12     }
13 }
```

- HTLin is an instance of CSIEProfessor
- HTLin is an instance of Professor (because CSIEProfessor is a type of Professor)
- HTLin is **not** an instance of EE Professor [the compiler knows]

Inheritance (2/5)

```
1  class Professor{ }
2  class CSIEProfessor extends Professor{ }
3  class EEProfessor extends Professor{ }
4
5  class Demo{
6      public static void main(String[] argv){
7          Professor HTLin = new CSIEProfessor();
8          System.out.println(HTLin.getClass());
9          System.out.println(HTLin instanceof CSIEProfessor);
10         System.out.println(HTLin instanceof Professor);
11         System.out.println(HTLin instanceof EEProfessor);
12     }
13 }
```

T	F	H
多	1	0
多	0	0
0	30	7

- HTLin is an instance of CSIEProfessor
- HTLin is an instance of Professor
- HTLin is **not** an instance of EE Professor [not easily determined at compile-time, but can be checked at run-time]

Inheritance (3/5)

```
1  class Professor{ }
2  class CSIEProfessor extends Professor{ }
3  class EEProfessor extends Professor{ }
4
5  class Demo{
6      public static void main(String[] argv){
7          Professor HTLin = new CSIEProfessor();
8          int score = 100;
9          System.out.println(HTLin.getClass());
10         System.out.println(score.getClass()); hahaha
11         System.out.println(HTLin instanceof java.lang.Object);
12         System.out.println(score instanceof java.lang.Object);
13     }
14 }
```

- every valid object is an instance of java.lang.Object
- primitive type is not an instance of anything [easily determined at compile-time]

Inheritance (4/5)

```
1  class Professor{ }
2  class CSIEProfessor extends Professor{ }
3  class EEProfessor extends Professor{ }
4
5  class Demo{
6      public static void main(String[] argv){
7          Professor[] parr = new CSIEProfessor[3];
8          System.out.println(parr.getClass());
9          System.out.println(parr instanceof Object);
10         System.out.println(parr instanceof Object[]);
11         System.out.println(parr instanceof Professor[]);
12         System.out.println(parr instanceof CSIEProfessor[]);
13         System.out.println(parr instanceof EEProfessor[]);
14     }
15 }
```

	T	F	H
7	28	3	0
9	25	0	4
11	32	0	0
12	25	1	0
13	0	27	0

- every object array is an instance of Object[]
- Object[] is a reference type
- every reference type “extends” Object
- every object array is an instance of Object

Inheritance (5/5)

```
1  class Demo{
2      public static void main(String [] argv){
3          int [] oarr = new int [3];
4          System.out.println(oarr.getClass());
5          System.out.println(oarr instanceof Object);
6          System.out.println(oarr instanceof Object []);
7          System.out.println(oarr instanceof double []);
8          System.out.println(oarr instanceof short []);
9          System.out.println(oarr instanceof int []);
10     }
11 }
```

- a int array is an instance of int[]
- int[] is a reference type
- every reference type “extends” Object
- every object array is an instance of Object

Inheritance: Key Point

- compile-time detectable: only siblings (int sibling of Object, EEPprofessor sibling of CSIEProfessor, etc.)
- run-time detectable: other instances (or null)

```
1 CSIEProfessor SomeOne = new CSIEProfessor();
2 Professor p;
3 p = SomeOne; //note: reference assignment
4 if (p instanceof CSIEProfessor && p != null){
5     CSIEProfessor pCSIE = (CSIEProfessor)p;
6     // ...
7 }
8 /* instanceof is not really used very often;
9    but useful in enhancing understanding. */
```

Java Type Hierarchy

int char double

