Wrap-up and Inheritance

Hsuan-Tien Lin

Deptartment of CSIE, NTU

OOP Class, April 7, 2009

What We Should Have Known: Object as Modules

- NOP: no modules
- POP: procedures as modules
- OOP: objects as modules

What We Should Have Known: Object Blueprints

- declare classes/variables
- instance variables
- instance methods
- static variables
- static methods
- final variables

What We Should Have Known: Object Lifecycle

- the `new` operator
- instance constructor
- garbage collection
- instance finalizer
- static constructor
- Java memory managements

What We Should Have Known: Processing

- primitive types
- “extended” (reference) types
- array types
- `null`
- comparisons (==, !=)

What We Should Have Known: Passing

- primitive type argument/parameter
- primitive type return
- reference type argument/parameter
- reference type return

What We Should Have Known: Method Invocation

- recursive call and frames
- local variable versus instance variable versus static variable
- method overloading

A Journey of Recap by HW3

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.name + ", " + pet.name + ".");
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
(some more in your UML class)

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 **extends** TypeSuperClass (or TypeBaseClass)

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

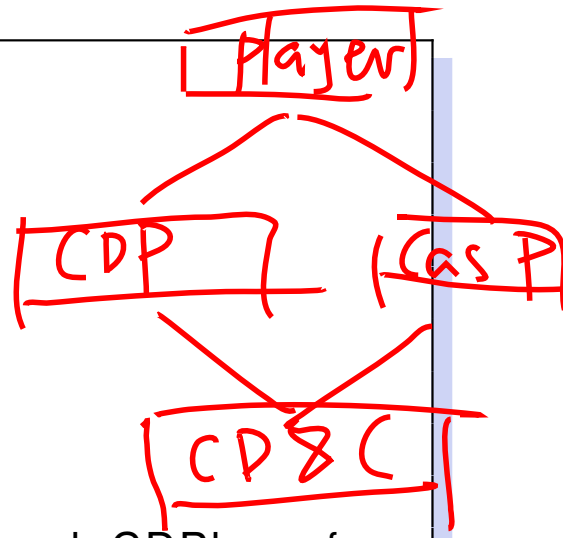
Uses of Is-A (3/4)

```
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  }
```

- 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_number; }
2  class HTLin extends CSIEProfessor{
3    //HTLin is a CSIEProfessor
4    HTLin(){ name = "HTLin"; office_number = 314; }
5  }
6  class CLChen extends CSIEProfessor{
7    //CLChen is a CSIEProfessor
8    String title;
9    CLChen(){
10     name = "CLChen"; office_number = 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_number = 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", 200);
12     YDLyuu.decide_budget();
13   }
14 }
```

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

More on Is-A: Key Point

```
class ViceChair extends Professor
```

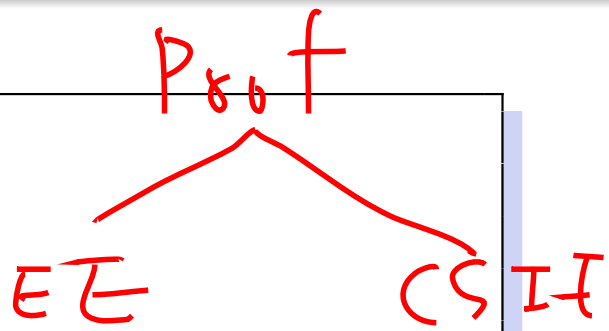
is a kind (type) of

```
Professor HTLin = new Professor()
```

is an instance of

Inheritance (1/6)

```
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 instanceof CSIEProfessor);
9      System.out.println(HTLin instanceof Professor);
10     System.out.println(HTLin instanceof EEProfessor);
11   }
12 }
```



true
true
false

- 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 (2/6)

```
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 EMPTY = null;
8      System.out.println(EMPTY instanceof CSIEProfessor);
9      System.out.println(EMPTY instanceof Professor);
10     System.out.println(EMPTY instanceof EEProfessor);
11   }
12 }
```

false
false

- EMPTY **could be** an instance of CSIEProfessor [actually not at run-time]
- EMPTY **could be** an instance of Professor [actually not at run-time]
- EMPTY is **not** an instance of EE Professor [easily determined at compile-time]

Inheritance (3/6)

```
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 instanceof CSIEProfessor);
9      System.out.println(HTLin instanceof Professor);
10     System.out.println(HTLin instanceof EEProfessor);
11   }
12 }
```

// ok, no complaint
true
true
false

- 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 (4/6)

```
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 instanceof java.lang.Object);
10     System.out.println(score instanceof java.lang.Object);
11   }
12 }
```

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

Inheritance (5/6)

```
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 instanceof Object);
9      System.out.println(parr instanceof Object[]);
10     System.out.println(parr instanceof Professor[]);
11     System.out.println(parr instanceof CSIEProfessor[]);
12     System.out.println(parr instanceof EEProfessor[]);
13   }
14 }
```

true
true
true
true
false

- 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 (6/6)

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

- 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: siblings (int sibling of Object, EEProfessor sibling of CSIEProfessor, etc.)
- run-time detectable: `null` or other instances (a safe check for casting error)

```
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