

# Class Definitions

Hsuan-Tien Lin

Department of CSIE, NTU

OOP Class, March 8-9, 2010

# The Basic of OOP Revisited

- OOP: organized DATA + organized code (ACTION)
  - using **classes** as the basic module
  - action closely coupled with data
    - data + do something

精華

金魚

**DATA**

id	nick
passwd	好友
文章	上站
P幣	故鄉
信 / 函 文 數	IP
email	My Fav
名 單	push/bo
名 單	permission

**ACTIONS**

水桶	game
post	push
del	開版
mark	收入精華
login	加好友
msg v	紅包
轉錄	罰錢
修文	

信 / 函 文 數

時間地點

# Type versus Values

```
1  int a = 3;  
2  int b = a + 2;  
3  int c;
```

- `int` is a type
- `a`, `b`, `c` are variables of the type
- `a` is of value 3  
 `b` is of value 5  
 `c` is of **uncertain** (unassigned) value

one type (`int`), many values (3, 5, ... or uncertain)

# Class versus Instances

```
1 String s1 = "lalala";  
2 String s2 = s1 + s1;  
3 String s3;
```

- `String` is a type
- `s1`, `s2`, `s3` are variables of the type
- `s1` refers to an instance "lalala"  
  `s2` refers to an instance "lalalalalala"  
  `s3` is not assigned to any instances

one class (`String`), many instances ("lalala", "lalalalalala", ...)

# Class versus Instances

```
1 public class POOUser{ //class
2     public String ID; //variable declaration
3     public String name; //variable declaration
4 }
5 //within , say, main
6 POOUser r1 = new POOUser(); //r1 is an instance
7                               //with r1.ID and r1.name
8                               //as its data (variables)
9 POOUser r2 = new POOUser(); //r2 is another instance
```

one class (POOUser), many instances (val of r1, val of r2, ...)

# Type: Defining Memory Interpretation

- primitive type: what the language supports as a basic building block
- extended type:
  - e.g. `String` (as character array in C)
  - e.g. structures in C, classes in Java

abstraction:

don't care (much) about what the bytes contain,  
care about what the **type means**—data- and action-wise

# Extended Type: the Data Side

```
1  public class POOUser{  
2      public String ID;  
3      public String name;  
4      public int onlineCount;  
5  }
```

intend to extract a `String` type memory space, another `String` type memory space and an `int` type memory space from a `POOUser` type variable



# Extended Type: the Action Side

```
1  public class POOUser{
2      public String ID;
3      public String name;
4      public int onlineCount;
5
6      public void setName(String the_name){ name = the_name; }
7      public boolean isFrequent(){ return (onlineCount > 10000); }
8  }
```

intend to let the instance (object) set its name and check whether it can be called “frequent”

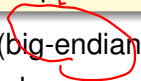
# Eight Java Primitive Types

primitive type: defining direct memory interpretations

- `byte`, `short`, `int`, `long`: 1/2/4/8 byte (big-endian) integers
- `float`, `double`: 4/8 byte floating point numbers
- `boolean`: `true` or `false`
- `char`: 2 byte unicode

all (except `boolean`) very similar to C

李 wiki?



# Many Java Extended Types

class WhateverYouWant

- class ~~2D~~Point <sup>2D</sup>
- class POOUser
- class java.io.PrintStream
- class java.util.Scanner
- class java.lang.String

read the API, **guess**, and write the program you want

App Prog Inter face

# Some Hints on import et al.

```
import java.lang.*;  
import java.io.*;
```

```
1  /* POOUser.java */  
2  public class POOUser{  
3      public java.lang.String ID;  
4      public java.lang.String name;  
5      public int onlineCount;  
6  
7      public void setName(java.lang.String the_name){ name =  
8          the_name; }  
9      public boolean isFrequent(){ return (onlineCount > 10000); }  
10 }  
11 /* POOUserDemo.java */  
12 public class POOUserDemo{  
13     public static void main(String[] args){  
14         POOUser u = new POOUser();  
15         java.io.PrintStream ps = java.lang.System.out;  
16         ps.println(u.isFrequent());  
17     }  
18 }
```

# Primitive versus Extended

- primitive:
  - one single piece of data with literal support (e.g. `true`)
  - no extended actions except basic operations
- extended (classes):
  - one or many pieces of data (instance variables)
    - all instances with the same pieces, but (possibly) with different values
  - extended actions (instance methods)
    - all instances with the same capability, but (possibly) with different behavior

# One Java Extended Type with Native Support

```
class java.lang.String
```

- the same as any extended type you see
- native operation support (e.g. +)
- literals "abc" recognized by the language
- some other special handling

# The Big Picture of Java Programs

```
1  /* POOUser.java */
2  public class POOUser{
3      public String ID;
4      public String name;
5      public int onlineCount;
6
7      public void setName(String the_name){ name = the_name; }
8      public boolean isFrequent(){ return (onlineCount > 10000); }
9  }
10 /* POOUserDemo.java */
11 public class POOUserDemo{
12     public static void main(String [] args){
13         POOUser u1 = new POOUser();
14         u1.onlineCount = 12345;
15         u1.setName("CharlieL");
16         System.out.println(u1.name);
17         System.out.println(u1.isFrequent());
18         POOUser u2 = new POOUser();
19         System.out.println(u2.onlineCount);
20         System.out.println(u2.ID);
21     }
22 }
```

*instance variable*

*undef haha 0/26*  
*undef haha 0*  
*ohohoh*  
*ohohoh*

*↓*  
*null*

# An OO Design of the RandomIndex class

- DATA: a randomly permuted index array of size  $N$
- ACTION: setSize, initializeIndex, permuteIndex, getNext
- see `RandomIndex.java`
- now you can use it for name calling in class, distributing cards in POO games, etc.