

# Introduction to Object Oriented Programming

Hsuan-Tien Lin

Dept. of CSIE, NTU

OOP Class, March 1-2, 2010

# 要做一台投影機，需要哪些零件？

- bulb
- fan
- case
- button
- image processor
- lens
- remote
- elec. board
- screw
- wire
- plastic
- led
- glass

# 要做一个CPU，需要哪些零件？

- IC board
- heat sink
- silicon wafer
- pin
- transistor
- logic gate
- capacitor
- led
- screw
- chips

# 要做一個POO BBS，需要哪些零件？

- machine
- space
- network
- 鄉民，鄉長
- admin
- compiler
- login system
- manage system
- user system
- board, gem, mail, edit, editor
- play, pet
- vote, refer
- connection
- chat, message
- myfav
- 口卡獸，戰文，……

模組化！

# 有好的模組才有好的系統

- easy to debug
- make the goal clear
- divide and conquer
- easy to maintain/manage
- easy to modify/update
- easy to reuse

模組化：節省很多未來的時間和心力！

# 程式輸入員 v.s. 程式設計師

- 輸入員：basic language skill
- 設計師：good design skill + good language skill
  - what's the purpose of the program?
  - what's the specialty of the language?
  - what's the current need of the program?
  - what's the future need of the program?

設計師：

願意用**現在的**專業付出，來節省**未來的**時間和心力！

不要只想寫出  
用後即丟的程式



# Noodle-Oriented Programming

- whatever ingredients you put in, edible noodles are good noodles
- too salty? add more water
- no vegetables? get whatever is in your refrigerator
- **spaghetti code**: a program flow that looks like a bowl of spaghetti, i.e. twisted and tangled

NOP: generate whatever code that works for now

# An Example of NOP

See `OOPLotteryV0.java`.

- 不想讓人看懂，病毒，etc.
- 交作業，趕時間，etc.
- 一直WA 要修錯，patch，批改娘，etc.
- 報復不喜歡的老師/助教，幫不喜歡的人寫程式

# From NOP to Procedure Oriented Programming

- organize the code
- identify the purpose of procedures (what a block of code can do)
- isolate (modularize) the procedures (as individual functions)
- reuse the procedures (by function calls)

You basically learned those in the C class.

# An Example of POP

See `OOPLotteryV1.java`.

# Some Remaining Problems

- the tokenizing program looks messy

```
1 public static String[] getTokens(String str){  
2     return str.split(",");  
3 }
```

–tell a str to split **itself** by ','

- what if we want both department and names?

```
1 NameList[index * 3 + 2], NameList[index * 3 + 0]
```

–can we use **names** instead of calculating the offsets by ourselves?

# Object Oriented Programming 101-1

- group related data together in design

```
1      /* C */
2      typedef struct{
3          char dept[100];
4          char ID[100];
5          char name[100];
6      } Record;
```

```
1      /* Java */
2      class Record{
3          String dept;
4          String ID;
5          String name;
6      }
```

# Object Oriented Programming 101-2

- use the struct/class to generate objects

```
1  /* C */
2  Record r;
3  Record* rp=(Record*) malloc(sizeof(Record));
4  strcpy(r.dept, "CSIE");
5  strcpy(rp->name, "HTLIN");
6  free(rp);
```

```
1  /* Java */
2  Record r = new Record();
3  r.dept = "CSIE";
4  r.name = "HTLIN";
```



- don't "do something on" the object; let the object "do something"

```
1      /* Java */
2      PrintStream ps = System.out;
3      ps.println("CSIE");
4      String s = "a,b,c";
5      tokens = s.split(",");
```

# From POP to Object Oriented Programming

- use a procedure (`strtok`) to manipulate some representation of data (`str`)

```
1  /* C */
2  char* str = "to_be_splitted.";
3  char*_p;
4  p=_strtok(str, " ");
5  while_(p!=_NULL){
6  _printf("%s\n",_p)
7  _p=_strtok(NULL,_" ");
8  }
```

- ask an object (`str`) to perform an action (`split`)

```
1  /* Java */
2  String str = "to_be_splitted.";
3  String []_res=_str.split(" ");
4  for(int_i=0;i<res.length;i++)
5  _System.out.println(res[i]);
```

similar, but possibly easier if you can **think from objects**

# An Example of OOP

See `OOPLotteryV2.java`.

# From Noodle to Procedural to Object

- NOP: spaghetti code + (possibly spaghetti) data
  - You can write NOP with virtually ANY languages
  - Some easier to NOP (e.g. assembly), some harder
- POP: organized CODE + (possibly organized) data
  - using procedures as the basic module
    - maintain, reuse
  - action as separated procedures from data (do on the data)
  - C, Pascal
- OOP: organized DATA + organized code (ACTION)
  - using classes as the basic module
  - action are closely coupled with data (data do something)
  - Java, C++ (?), C#

# From Noodle to Procedural to Object

- OOP: organized DATA + organized code (ACTION)
  - using classes as the basic module
  - action are closely coupled with data (data do something)
  - Java, C++ (?), C#
- You can write virtually any-OP with any language
- OO design: think in the OO way
- OO language: help (force) you to think in the OO way

# An OO Trip with Java `String` (see: Sec 1.3)

```
1  import java.lang.*;
2
3  public class HelloWorld{
4      public static void main(String [] argv){
5          String s = "Hello_World";
6          //imagine: s 'holds' all characters instead of pointing to H
7          //(note: an inaccurate description)
8
9          String t = "Hello_world";
10         System.out.println(s.length());
11         //action for probing obj status
12
13         System.out.println(s.charAt(3));
14         //action for probing obj status
15     }
16 }
```

# More OO Trip with Java String

```
1 import java.lang.*;
2
3 public class HelloWorld{
4     public static void main(String [] argv){
5         String s = "Hello_World";
6         String t = "Hello_world";
7
8         System.out.println(s.equalsIgnoreCase(t));
9         //action concerning one obj and the other
10
11        System.out.println(s.replace('o', 'a'));
12        //action for 'manipulating' the object
13        //(note: an inaccurate description)
14
15        System.out.println(s.concat(t));
16        System.out.println(s.concat("_lala"));
17        System.out.println(s + "_lala");
18        //action for 'manipulating' the object
19        //(note: an inaccurate description)
20    }
21 }
```

# Three Levels of OO

- Object-Oriented Analysis (OOA):  
what the system does
  - from (customer/system) needs to software models
  - learn from your UML class
- Object-Oriented Design (OOD):  
how the system does it
  - from software model to class diagrams
  - (mostly) learn from your UML class; some in this class
- Object-Oriented Programming (OOP):  
how to implement such a system
  - from class diagrams to class implementations
  - learn from this class

this class: called software design but actually **programming**



# Three Levels of OO

- Object-Oriented Analysis (OOA):  
what the system does
- Object-Oriented Design (OOD):  
how the system does it
- Object-Oriented Programming (OOP):  
how to implement such a system

not necessarily separate levels

—**connect the dots** with the UML class by yourself