From Noodle-Oriented Programming$^{(*)}$ to Object-Oriented Programming

Hsuan-Tien Lin

Deptartment of CSIE, NTU

OOP Class, March 3, 2009

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

Applications of Noodle-Oriented Programming

generate whatever code that works for now
課堂腦力激盪：

- CSIE any homework
- 用後即丟
- 有時間壓力
- 報復不喜歡的老師/TA
- 小規模

An Example of NOP

See `OOPLotteryV0.java`.

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

```c
/* C */
typedef struct{
  char dept[100];
  char ID[100];
  char name[100];
} Record;
```

```java
/* Java */
class Record{
  String dept;
  String ID;
  String name;
}
```

Object Oriented Programming 101-2

- use the struct/class to generate objects

```c
/* C */
Record r;
Record* rp=(Record*)malloc(sizeof(Record));
strcpy(r.dept, "CSIE");
strcpy(rp->name, "HTLIN");
free(rp);
```

```java
/* Java */
Record r = new Record();
r.dept = "CSIE";
r.name = "HTLIN";
```

Object Oriented Programming 101-3

- 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 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

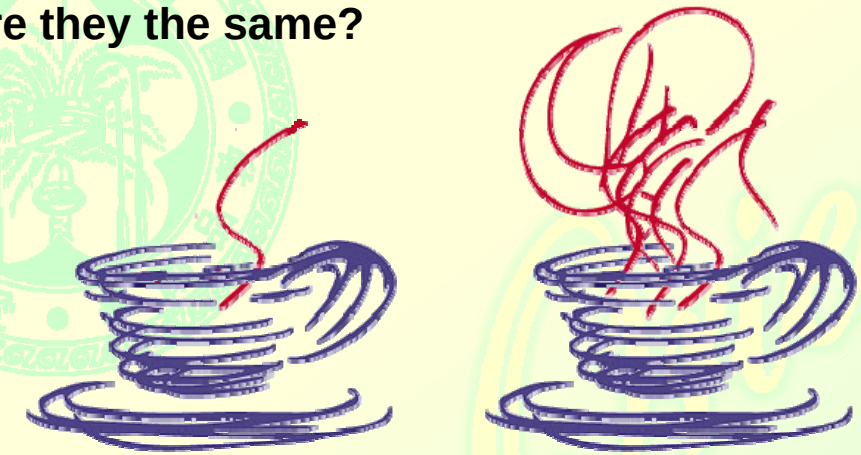
OOP 101

- consider the representation of information (data)
- group related data together in design (class)
- use the class to generate objects (instance)
- let the object “do something” (action)

Class versus Instances (courtesy of Prof. Chuen-Liang Chen)

Class vs. Instance

- Are they the same?



- instance 個體 (object)
  - with different status
  - representation of status (in high-level language): variable
  - instance: set of instance variables
- class 類別
  - it is no way & unnecessary to write program for instances one by one
  - OO programming = class (interface) declarations

Class versus Instances

```
1   class Record{//class
2     String name; //variable declaration
3     String ID; //variable declaration
4     public bool isB86(){ //action
5       return ID.startsWith("B86");
6       //here ID is an instance of the class String
7       //and performs an action (method) startsWith()
8     }
9   }
10
11  Record r1 = new Record(); //r1 is an instance
12                            //with r1.name and r1.ID
13                            //as its data (variables)
14  Record r2 = new Record(); //r2 is another instance
15  Record rarray = new Record[3];
16
17  if (r2.isB86()) {} //r2 performs an action (method)
```