

# Classes and Instances

Hsuan-Tien Lin

Dept. of CSIE, NTU

OOP Class, February 25, 2013

Decompose computation into interactions of “computing parts” called *objects*, each containing its own data (to be manipulated by itself) and own code (to be called by other objects)

- data (variables): status of object
- code (methods): actions of objects

Decompose computation into interactions of “computing parts” called *objects*, each containing its own data (to be manipulated by itself) and own code (to be called by other objects)

- designing object (what variables? what methods?)
- creating “first” object and calling its first action
- creating other objects
- calling other objects
- manipulating object status
- deleting objects

# Big Picture: Java Solution of OOP Needs

- designing object (what variables? what methods?)
  - declaring `class`, the blueprint of object**
- creating “first” object and calling its first action
  - JVM does so, with first action called `main`**
  - (note: not exactly so, will be clarified later)**
- creating other objects
  - with something called `new`**
- calling other objects
  - with grammar like `objectname.methodname(...)`**
- manipulating object status
  - with grammar like `variablename = ...` in the object code**
- deleting objects
  - JVM automatically and autocratically does so, when object “no longer needed” (garbage collection)**

# Big Picture: C++ Solution of OOP Needs

- designing object (what variables? what methods?)
  - declaring `class`, the blueprint of object**
- creating “first” object and calling its first action
  - you need to write code to do so, while C++ only provides a `main` like C, a non-OO program entry**
- creating other objects
  - with something called `new`**
- calling other objects
  - with grammar like `objectname.methodname(...)`**
- manipulating object status
  - with grammar like `variablename = ...` in the object code**
- deleting objects
  - with something called `delete`**

# Big Picture: Objective C Solution of OOP Needs

- designing object (what variables? what methods?)
  - declaring interface (variable and method prototype) and implementation (code), the blueprint of object**
- creating “first” object and calling its first action
  - you need to write code to do so, while ObjC only provides a main like C, a non-OO program entry**
- creating other objects
  - with something like** `[[ClassName alloc] init]`
- calling other objects (sending message to other objects)
  - with grammar like** `[objectname methodname]`
- manipulating object status
  - with grammar like** `variablename = ...` in the object code
- deleting objects
  - modernly, with an automatic mechanism called “automatic reference counting”**

# Java Solution of OOP Needs (1/6)

- designing object (what variables? what methods?)
  - declaring `class`, the blueprint of object

```
1 //Song.java
2 public class Song{
3     private String name;
4     private byte [] data;
5     public void play(){
6         // ...
7     }
8 }
9 //MP3Player.java
10 public class MP3Player{
11     private Song[] songs;
12     private int currentSong;
13     public boolean playSongAtIndex(int index){
14         // ...
15     }
16 }
```

## Java Solution of OOP Needs (2/6)

- creating “first” object and calling its first action
  - JVM does so, with first action called `main`
  - (note: not exactly so, will be clarified later)

```
java MyProgram
```

```
1 //MyProgram.java
2 public class MyProgram{
3     public static void main(String [] argv){
4         // ...
5     }
6 }
```



## Java Solution of OOP Needs (3/6)

- creating other objects  
—with something called `new`

```
1 public class MyProgram{
2     public static void main(String [] argv){
3         MP3Player p1 = new MP3Player ();
4         MP3Player p2 = new MP3Player ();
5         MP3Player p3; //not linked to any objects yet
6     }
7 }
```

- can use one blueprint (class) to create many objects
- class is an extended type, like a C structure  
(Record\* r = (Record\*)malloc(sizeof(Record)); )  
object of type R ≡ instance of class R ≡ object of class R
- a variable of the type links to (can be used to refer to) some object

## Java Solution of OOP Needs (4/6)

- calling other objects  
—with grammar like `objectname.methodname (...)`

```
1 public class MyProgram{  
2     public static void main(String [] argv){  
3         MP3Player p1 = new MP3Player();  
4         p1.playSongAtIndex(5);  
5     }  
6 }
```

- similar to procedure invocation in C, but specifying which instance to be called/messaged

## Java Solution of OOP Needs (5/6)

- manipulating object status

—with grammar like `variablename = ...` within the object code

```
1  public class MP3Player{
2      private Song[] songs;
3      private int currentSong;
4
5      public void next(){
6          currentSong++;
7          // ...
8      }
9  }
```

- the same as variable assignment in C, but (for now) within the “scope” of the object

- deleting objects
  - JVM automatically and autocratically does so, when object “no longer needed” (garbage collection)

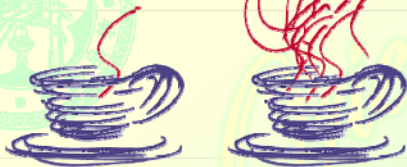
no worries yet, more to be discussed later.

# Class versus Instances (Objects)

```
1 public class Record{ //class
2     private String name; //variable/field declaration
3     private String ID; //variable declaration
4     public boolean 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 //somewhere else
12 Record r1 = new Record(); //r1 is an instance
13                        //with r1.name and r1.ID
14                        //as its data (variables)
15 Record r2 = new Record(); //r2 is another instance
16
17 if (r2.isB86()) { ... } //r2 performs an action (method)
```

## 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

## Not Just Record: An OO Design of RandomIndex

- DATA: a randomly permuted index array of size  $N$
- ACTION: setSize, initializeIndex, permuteIndex, getNext
- see `RandomIndex.java`, `OOPLotteryV3.java`
- Now you can use it for name calling in class, distributing cards in games, etc.