

Classes and Instances (Java)

Hsuan-Tien Lin

Department of CSIE, NTU

OOP Class, March 7, 2013

More on Constructors (3/3)

```
1 class Record{  
2     private int score;  
3     public Record(int init_score){score = init_score;}  
4     public Record(){ Record(40);}  
5 }  
6 public class RecordDemo{  
7     public static void main(String [] arg){  
8         Record r1 = new Record(60);  
9         Record r2 = new Record();  
10    }  
11 }
```

- can **overload**: same name, different parameters
- can call other constructors to help initialize

More on Constructors: Key Point

often better to use self-defined and overloaded constructors to help initialize

[3] Object Lifecycle

Garbage Collection (1/2)

```
1 public class Record{  
2     private int score;  
3 }  
4 public class RecordDemo{  
5     public static void main(String [] arg){  
6         int i; Record r1;  
7         for(i = 0; i < 100; i++){  
8             r1 = new Record();  
9         }  
10    }  
11 }
```

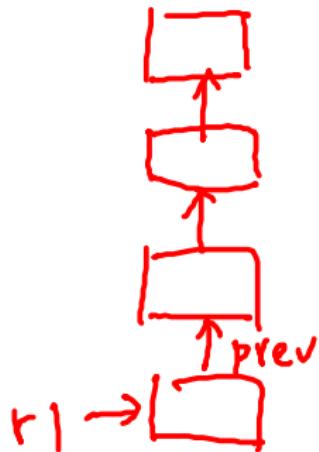


- 100 instances created, only 1 alive after the loop
- the other 99 memory slots: automatically recycled

garbage collection

Garbage Collection (2/2)

```
1 class Record{  
2     private Record prev;  
3     public Record(Record p){ prev = p; }  
4 }  
5 public class RecordDemo{  
6     public static void main(String [] arg){  
7         int i; Record r1 = null;  
8         for(i = 0; i < 100; i++){  
9             Record tmp = new Record(r1);  
10            r1 = tmp;  
11        }  
12    }  
13 }
```



- 100 instances created, all of them alive

Garbage Collection: Key Point

Garbage Collection: when a memory slot becomes an orphan (and) system in need of memory

Finalizer (1/2)

```
1  public class Record{
2      private int score;
3      public Record(){ sys.mem_usage += 10; }
4      public void when_truck_comes(){ sys.mem_usage -= 10; }
5  }
6  public class RecordDemo{
7      public static void main(String [] arg){
8          int i; Record r1;
9          for(i = 0; i < 100; i++){
10              r1 = new Record();
11          }
12      }
13 }
```

- finalizer: something you want to do when truck comes
- calculate memory usage, write something back (say, on BBS), ...

Finalizer (2/2)

```
1 public class Record{  
2     private int score;  
3     public Record(){ sys.mem_usage += 10; }  
4     protected void finalize() throws Throwable{  
5         sys.mem_usage -= 10;  
6         System.out.println("Good_Bye!");  
7     }  
8 }
```

- GC: no guarantee on when the truck comes
- if JVM halts before truck comes, even no finalizer calls
- **use carefully**

Finalizer: Key Point

finalizer:

a mechanism to let the instance say goodbye

Object Lifecycle (1/1)

```
1  public class Record{  
2      private int score;  
3      public Record(int init_score){ score = init_score; }  
4      protected void finalize() throws Throwable{}  
5  }  
6  public class RecordDemo{  
7      public static void main(String[] arg){  
8          Record r; //variable declared  
9          r = new Record(60); //memory allocated (RHS)  
10         //and constructor called  
11         //variable linked (LHS)  
12         r.show_score(); //instance action performed  
13         r = null; //memory slot orphaned  
14         // ....  
15         //finalizer called  
16         //or JVM terminated  
17     }  
18 }
```

Object Lifecycle: Key Point

we control birth, life, death, funeral design, but not the exact funeral time

[4] Back to Class

Static Variables (1/3)

```
1  public class Record{  
2      private int total_rec;  
3      public Record(){  
4          total_rec += 1;  
5      }  
6      public void show_total_rec(){  
7          System.out.println(total_rec);  
8      }  
9  }  
10 public class RecordDemo{  
11     public static void main(String[] arg){  
12         Record r1 = new Record();  
13         r1.show_total_rec();  
14         Record r2 = new Record();  
15         r2.show_total_rec();  
16     }  
17 }
```

r1

A red bracket labeled 'r1' points to a rectangular box containing the letters 'tr'. This represents the state of the program where two separate objects, r1 and r2, both point to the same shared static variable.

r2

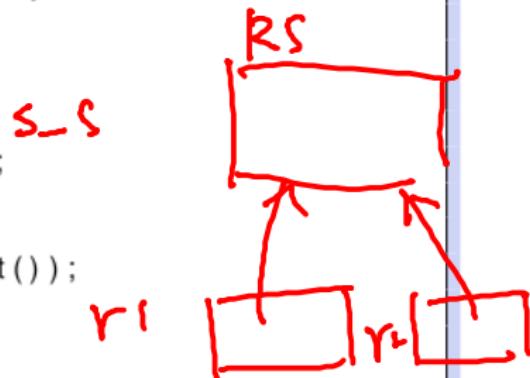
A red bracket labeled 'r2' points to a rectangular box containing the letters 'tv'. This represents the state of the program after the second object r2 has been created and assigned to the static variable.

- no shared space to store the total records

Static Variables (2/3)

```

1  public class RecordShared{
2      private int count;
3      public void increase_count(){ count++; }
4      public int get_count(){ return count; }
5  }
6  class Record{
7      RecordShared shared;
8      public Record(RecordShared s){
9          share = s; shared.increase_count();
10     } d
11     public void show_total_rec(){
12         System.out.println(shared.get_count());
13     }
14 }
15 public class RecordDemo{
16     public static void main(String[] arg){
17         RecordShared shared_space = new RecordShared();
18         Record r1 = new Record(shared_space);
19         r1.show_total_rec();
20         Record r2 = new Record(shared_space);
21         r2.show_total_rec();
22     }
23 }
```



Static Variables (3/3)

```

1  public class Record{
2      private static int total_rec = 0;
3      public Record(){ total_rec++; }
4      public void show_total_rec(){
5          System.out.println(total_rec);
6      }
7  }
8  public class RecordDemo{
9      public static void main(String [] arg){
10         Record r1 = new Record();
11         r1.show_total_rec();
12         Record r2 = new Record();
13         r2.show_total_rec();
14         System.out.println(Record.total_rec);
15     }
16 }
```

class variable

total_rec



- static: shared between all X-type instances
- like a global variable within the scope of the class
- use scarcely