

# More on Object Lifecycle

Hsuan-Tien Lin

Department of CSIE, NTU

OOP Class, March 22-23, 2010

# Garbage Collection (1/2)

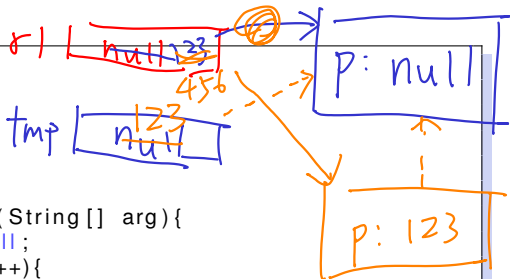
```
1  class Record{
2      static int count = 0;
3      Record(){ count++; }
4  }
5  public class RecordDemo{
6      public static void main(String [] arg){
7          int i; Record r1;
8          for(i = 0; i < 100; i++){
9              r1 = new Record();
10             System.out.println(Record.count);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

The diagram shows a variable `r1` pointing to a `Record` object with value `123`. A red scribble indicates that 99 other `Record` objects were created and then garbage collected. A red arrow points from the `r1` box to a single `Record` object box, and another red arrow points from the scribbled-out area to a larger, empty box representing the memory space for the 99 garbage-collected objects.

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

## Garbage Collection (2/2)

```
1 class Record{
2     static int count = 0;
3     Record prev;
4     Record(){ count++; }
5 }
6 public class RecordDemo{
7     public static void main(String [] arg){
8         int i; Record r1 = null;
9         for(i = 0; i < 100; i++){
10            Record tmp = r1;
11            r1 = new Record();
12            r1.prev = tmp;
13            System.out.println(Record.count);
14        }
15    }
16 }
```



- 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  class Record{
2      static int mem = 0;
3      Record(){ mem += 10; }
4      void when_truck_comes(){ mem -= 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             System.out.println(Record.mem);
12         }
13     }
14 }
```

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

## Finalizer (2/2)

```
1  class Record{
2      static int mem = 0; static int count = 0;
3      int id;
4      Record(){ mem += 10; count++; id = count; }
5      protected void finalize() throws Throwable{
6          System.out.print(id);
7          System.out.println(",_Good_Bye!");
8          mem -= 10;
9      }
10 }
11 public class RecordDemo{
12     public static void main(String [] arg){
13         int i; Record r1 = null;
14         for(i = 0; i < 100; i++){
15             Record tmp = r1; r1 = new Record();
16             System.out.println(Record.mem);
17         }
18     }
19 }
```

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

finalizer:

a mechanism to let the instance say goodbye

# Object Lifecycle (1/1)

```
1  class Record{
2      int score;
3      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; //reference declared
9          Record r2; //reference declared
10         r = new Record(60); //memory allocated (RHS)
11                                 //and constructor called
12                                 //reference assigned (LHS)
13         r2 = r; //reference copied
14         r.score = 3; //instance content accessed
15         r.show_score(); //instance action performed
16         r2 = null; r = null; //memory slot orphaned
17         // ....
18                                 //finalizer called
19                                 //or JVM terminated
20     }
21 }
```



# Object Lifecycle: Key Point

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