Classes and Instances (Java)

Hsuan-Tien Lin

Deptartment of CSIE, NTU

OOP Class, March 7, 2013

More on Constructors (3/3)

```
class Record{
  private int score;
  public Record(int init_score){score = init_score;}
  public Record(){ Record(40);}
}
public class RecordDemo{
  public static void main(String[] arg){
    Record r1 = new Record(60);
    Record r2 = new Record();
  }
}
```

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

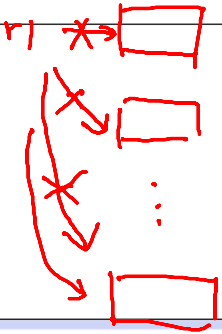
More on Constructors: Key Point

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

[3] Object Lifecycle

Garbage Collection (1/2)

```
public class Record{
  private int score;
}
public class RecordDemo{
  public static void main(String[] arg){
    int i; Record r1;
    for(i = 0; i < 100; i++){
      r1 = new Record();
    }
  }
}
```

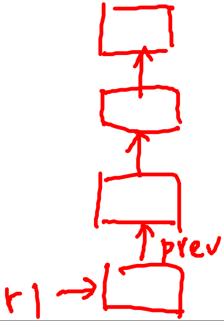


- 100 instances created, only 1 alive after the loop
- the other 99 memory slots: automatically recycled

garbage collection

Garbage Collection (2/2)

```
class Record{
  private Record prev;
  public Record(Record p){ prev = p; }
}
public class RecordDemo{
  public static void main(String[] arg){
    int i; Record r1 = null;
    for(i = 0; i < 100; i++){
      Record tmp = new Record(r1);
      r1 = tmp;
    }
  }
}
```



- 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)

```
public class Record{
  private int score;
  public Record(){ sys.mem_usage += 10; }
  public void when_truck_comes(){ sys.mem_usage -= 10; }
}
public class RecordDemo{
  public static void main(String[] arg){
    int i; Record r1;
    for(i = 0; i < 100; i++){
      r1 = new Record();
    }
  }
}
```

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

Finalizer (2/2)

```java
public class Record{
  private int score;
  public Record(){ sys.mem_usage += 10; }
  protected void finalize() throws Throwable{
    sys.mem_usage -= 10;
    System.out.println("Good␣Bye!");
  }
}
```

- 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)

```
public class Record{
  private int score;
  public Record(int init_score){ score = init_score; }
  protected void finalize() throws Throwable{ }
}
public class RecordDemo{
  public static void main(String[] arg){
    Record r; //variable declared
    r = new Record(60); //memory allocated (RHS)
                        //and constructor called
                        //variable linked (LHS)
    r.show_score();     //instance action performed
    r = null;//memory slot orphaned
    // ....
                        //finalizer called
                        //or JVM terminated
  }
}
```

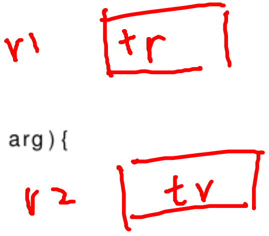
Object Lifecycle: Key Point

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

[4] Back to Class

Static Variables (1/3)

```
public class Record{
  private int total_rec;
  public Record(){
    total_rec += 1;
  }
  public void show_total_rec(){
    System.out.println(total_rec);
  }
}
public class RecordDemo{
  public static void main(String[] arg){
    Record r1 = new Record();
    r1.show_total_rec();
    Record r2 = new Record();
    r2.show_total_rec();
  }
}
```

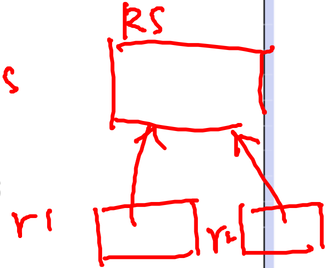


- no shared space to store the total records

Static Variables (2/3)

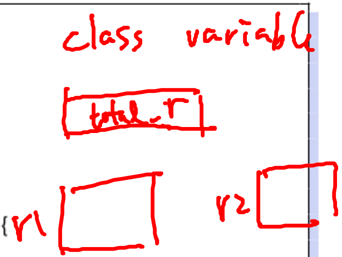
```java
public class RecordShared{
  private int count;
  public void increase_count(){ count++; }
  public int get_count(){ return count; }
}
class Record{
  RecordShared shared;
  public Record(RecordShared s){
    share = s; shared.increase_count();
  }      d
  public void show_total_rec(){
    System.out.println(shared.get_count());
  }
}
public class RecordDemo{
  public static void main(String[] arg){
    RecordShared shared_space = new RecordShared();
    Record r1 = new Record(shared_space);
    r1.show_total_rec();
    Record r2 = new Record(shared_space);
    r2.show_total_rec();
  }
}
```

s_s



Static Variables (3/3)

```
public class Record{
  private static int total_rec = 0;
  public Record(){ total_rec++; }
  public void show_total_rec(){
    System.out.println(total_rec);
  }
}
public class RecordDemo{
  public static void main(String[] arg){
    Record r1 = new Record();
    r1.show_total_rec();
    Record r2 = new Record();
    r2.show_total_rec();
    System.out.println(Record.total_rec);
  }
}
```



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