Summary of Last Class

Hsuan-Tien Lin

Department of CSIE, NTU

OOP Class, March 29-30, 2010

Static Variables

```
1  class Record{
2    public static int count;
3    Record(){
4      count++; //same as Record.count++;
5    }
6  }
```

Static Final Variables

```
1  class myMath{
2    public static final double PI=3.1416;
3  }
```

Static Methods

```
1  class myMath{
2    public static final double PI=3.1416;
3
4    public static double sum(double a, double b){
5      return (a + b);
6    }
7  }
8  class Demo{
9    public static void sub1(){ /* ... */ }
10   public static void sub2(){ /* ... */ }
11   public static void sub3(){ /* ... */ }
12
13   public static void main(String[] argv){
14     Demo.sub1();
15     Demo.sub2();
16     sub3(); //same as Demo.sub3() in Demo
17   }
18 }
```

Object Lifecycle

```
1  class Record{
2    int score;
3    Record(int init_score){ score = init_score; }
4    protected void finalize()
5      throws Throwable{ ModlitwaDziewicy.play(); }
6  }
7  public class RecordDemo{
8    public static void main(String[] arg){
9      Record r; //reference declared
10     Record r2;//reference declared
11     r = new Record(60); //memory allocated (RHS)
12                         //and constructor called
13                         //reference assigned (LHS)
14     r2 = r;             //reference copied
15     r.score = 3;        //instance content accessed
16     r.show_score();     //instance action performed
17     r2 = null; r = null;//memory slot orphaned
18     //....
19                         //finalizer called
20                         //or JVM terminated
21   }
22 }
```

Arrays

```
1  class Demo{
2    public static void main(String[] argv){
3      int[] iarr = new int[3];
4      Record[] rarr = new Record[3];
5      rarr[0] = new Record("HTLin", 80);
6      rarr[1] = null;
7      int[][] iarrarr = {null, {1, 2}, {5, 7, 9}};
8      System.out.println(iarrarr[1].length);
9    }
10 }
```

A Big Crowded Picture of Java Basics

- **strongly typed**: every variable must have a type
- primitive types: the "value" in the small box directly stores the info
  —everything related is done by **primitive value copying and manipulation**
- reference (extended) types: the "value" in the small box stores the reference (to a typed instance), including
  - `java.lang.String`
  - **self-defined types** like `class Record`
  - **arrays**

  —everything related is done by **reference value copying (and limited manipulation)**
- JVM memory
  - a stack that holds local variables (method call frames)
  - a **heap** that holds instances
  - a constant pool that holds constant instances (like constant Strings)
  - somewhere that holds the class (codes and static variables)
    —HEAP: 2nd one, or the last three (automatic GC on **heap**)