CS202 Java Object Oriented Programming
Common Utility Classes and Methods

Chengyu Sun
California State University, Los Angeles

System.out

System.out.println(3 );
System.out.println(3.5 );
System.out.println("Hello");
System.out.println(3*4 );
System.out.println(3+4*5 );
System.out.println("sum of 1+1 IS " + (1+1) ); // coercion at work
System.out.println(3+4*5 );
System.out.println("%n"); // \n is the newline character
System.out.println("\t"); // \t is the tab character
System.out.println("\n\t\n\t\n");

◆ System.out is easy to use
◆ System.in is not nearly so

ConsoleReader

◆ A wrapper of System.in
◆ Provides three methods:
  - readLine()
  - readInt()
  - readDouble()
◆ Available from the class homepage

ConsoleReader Usage – Code

/**
 * Usage test. read an integer and a double, and output
 * their sum.
 */
public static void main( String args[] )
{
  ConsoleReader in = new ConsoleReader();
  int a = in.readInt();
  double b = in.readDouble();
  System.out.println( a + " + " + b + " = " + (a+b) );
}

Math

◆ Common constants and functions for mathematic calculation
◆ Fields
  - 𝜋 and 𝜀
◆ Methods
  - random, abs, min, max
  - sin, cos, tan, asin, acos, atan
  - pow, sqrt, log, exp
  - ceil, floor, round
◆ All members are static

ConsoleReader Usage – Files

◆ Download ConsoleReader.java from the class homepage
◆ Copy it to the directory where your programs are
◆ Compile and run your programs as before
◆ NOTE: If you use Netbeans, and your program has a line says "package xxx.xx;", you need to add the same line to the beginning of ConsoleReader.java
Usage of Math Constants and Functions

double radius = 10.0; // radius of a circle
double area = Math.PI * radius * radius; // area of a circle

double a = 10.7;
long fa = (long) a; // truncation
long ia = Math.round(a); // rounding

double x1 = y1 = 1.0;
double x2 = y2 = 2.0;
double distance; // distance between (x1,y1) and (x2,y2)
distance = Math.sqrt((x1-x2)*(x1-x2) + (y1-y2)*(y1-y2));

Random Number Generation

◆ Extremely important for scientific experiments and simulations
◆ Very useful for software testing and profiling
◆ And a good way to populate an array

Usage of Math.random()

◆ Returns a random double value in [0,1)
◆ Example: populate an array of size 10 with random integers in the range [23,35]
  ● Or in the range [m,n]??

   int a[] = new int[10];
   for( int i=0 ; i < a.length ; ++i )
     a[i] = ??

Format Numerical Output

◆ DecimalFormat class
  ● Constructor
    ● DecimalFormat( String pattern )
  ● String format( double num )
  ● String format( long num )
◆ Symbols for pattern
  ● 0, #, -, E, %, $

Usage of DecimalFormat

import java.text.DecimalFormat;
...

double a = 1000.355;
double b = 0.765;
int c = 7354;

DecimalFormat f1 = new DecimalFormat("##.##");
DecimalFormat f2 = new DecimalFormat("00.00");

System.out.println( f1.format(a) ); // be ware of the
System.out.println( f1.format(b) ); // rounding behavior
System.out.println( f2.format(c) );

◆ Exercise: ":##.##", "0.##%", "0.000E0"

Timing

◆ The best way to appreciate a good algorithm is to see how fast it runs
◆ And time it

◆ Exercise: compare the speed of sequential search and binary search
System.currentTimeMillis

- Returns the current time in milliseconds
- We don't care about current time
- We do care about elapsed time
- NOTE: cTM is not very accurate

```java
long startTime = System.currentTimeMillis();
/* some slow code here */
long endTime = System.currentTimeMillis();
System.out.println( startTime - endTime );
```

Arrays Are Not Enough

- The Good
  - Easy to use
  - Space efficient
  - Constant time to access an array element
- The Bad
  - Cannot dynamically add or remove elements

Collection Framework For Dummies

Choose From Collections

- **Set**
  - Elements are objects
  - No duplicates
- **List**
  - Elements are objects
  - Allow duplicates
- **Map**
  - Elements are object pairs <key,value>
  - No duplicate keys

Benefits and Limitations of Collection Classes

- **Benefits**
  - Resizable
  - Elements can be of any class type
- **Limitation**
  - Cannot hold values of primitive types
  - Element access via an Object reference

ArrayList

- Resizable array of Object
- Important methods
  - `ArrayList()
  - add(Object o), remove(int index), get(int index)
  - contains(Object o), indexOf(Object o)
  - size()
  - isEmpty(), clear()
**ArrayList Example**

```java
ArrayList a = new ArrayList();
    a.add( new Integer.valueOf(st.nextToken()) );
...
int index = -1;
for( int i=0; i < a.size(); ++i )
    if( ? )
        { 
            index = i;
            break;
        }
System.out.println( "found at index " + index );
```

**HashMap**

- **Resizable array of Object pairs**
- **Important methods**
  - HashMap()
  - put(Object key, Object value)
  - get(Object key)
  - containsKey(Object key), containsValue(Object value)
  - size(), clear()
  - keySet()

**Iterator and Enumeration**

- **Iterator**
  - hasNext()
  - next()
- **Enumeration**
  - hasMoreElements()
  - nextElement()

- Sequential access of elements
- Iterator is preferred

**HashMap Example**

**Lab 4, Ex.2**