UCLA PIC 20A
Java Programming

- **Instructor:** Ivo Dinov,
  Asst. Prof. In Statistics, Neurology and Program in Computing
- **Teaching Assistant:** Yon Seo Kim, PIC

University of California, Los Angeles, Summer 2002

http://www.stat.ucla.edu/~dinov/

Chapter 4 – Simple Data Objects

- The Life Cycle of an Object
- Characters and Strings
- "Numbers"
- Arrays

Creating Objects

- **A class provides the blueprint for objects; we create objects from a classes.**
- **Ex. CreateObjectDemo.java**

```
Point origin_one =new Point(23,94);
Rectangle rect_one =new Rectangle(origin_one,100,200);
Rectangle rect_two =new Rectangle(50,100);
```

- **Declaration: associate a name with a type.**
- **Instantiation: new keyword is an operator that creates the new object (allocates space for it).**
- **Initialization: The new operator is followed by a call to a constructor.**

Creating Objects - constructors

```
public class Rectangle {
        public int width =0;
        public int height =0;
        public Point origin;
                //      2  constructors
        public Rectangle(){
                origin =new Point(0,0);
        }
        public Rectangle(Point p){
                origin =p;
        }
}
Rectangle rect_one = new Rectangle(new Point(100,200));
```


Using Objects

- Once you've created an object, you can use it, change its state, or have it perform some action.
  - Manipulate or inspect its variables

    objectReference.variableName

    System.out.println("Width of rect_one:"+**rect_one.width**);
    System.out.println("Height of rect_one:"+**rect_one.height**);

    Note: The direct manipulation of an object's variables by other objects and classes is discouraged because it's possible to set the variables to values that don't make sense. One should use **set** & **get** interface methods …

  - Call its methods

    objectReference .methodName (argumentList);

    rect_one.area(); rect_two.move(40,72);

Cleaning up Objects – memory management

- C/C++, and other languages, require that you keep track of all the objects you create and that you explicitly destroy them when they are no longer needed. Tedious and error prone approach.
- Java runtime environment deletes objects when it determines that they are no longer being used. This process is called ***garbage collection***. An object is eligible for garbage collection when there are no more references to that object.
- Variable references are usually dropped when the variable goes out of scope. Or, you can explicitly drop an object reference by **variable = null** .
- The garbage collector does its job automatically, although, in some situations, you may want to run the garbage collection explicitly by **System.gc();**

Cleaning up Objects – finalize()

- Before an object gets garbage-collected, the garbage collector gives the object an opportunity to clean up after itself through a call to the object's **finalize()** method.
- A programmer may or may not want to implement finalize()
- A default finalize() method is a member of the Object class, which is the top of the Java platform's class hierarchy and a superclass of all classes.
- If you override finalize, your implementation of the method should call **super.finalize()** as the last thing it does.

Characters & Strings in Java

- Java contains 3 classes that deal with character data:
- **Character** —A class whose instances can hold a single character value. This class also defines handy methods that can manipulate or inspect single-character data.
- **String** —A class for working with immutable (unchanging) data composed of multiple characters.
- **StringBuffer** —A class for storing and manipulating mutable data composed of multiple characters.
  - http://java.sun.com/j2se/1.3/docs/api/java/lang/Character.html
  - http://java.sun.com/j2se/1.3/docs/api/java/lang/String.html
  - http://java.sun.com/j2se/1.3/docs/api/java/lang/StringBuffer.html

Characters

```
Character a =new Character('a ');
Character a2 =new Character('a ');
Character b =new Character('b ');

int difference =a.compareTo(b);
if (difference == 0)
        System.out.println("a is equal to b.");
else if (difference < 0)
        System.out.println("a is less than b.");
else if (difference > 0)
        System.out.println("a is greater than b.");
```


Characters – some methods …

Character(char)—Unique Character class constructor
compareTo(Character) —An instance method
equals(Object) — An instance method
toString() — An instance method
charValue() — An instance method
isUpperCase(char) — A class method

String and StringBuffered

- String and StringBuffer store and manipulate strings—character data consisting of more than one character.
- The String class provides for strings whose value will not change.
- The StringBuffer class provides for strings that will be modified; Typically we use string buffers for constructing character data dynamically

String and StringBuffer

```
public class StringsDemo {
    public static void main(String [] args)){
        String palindrome ="Dot saw I was Tod";
        int len =palindrome.length();
        StringBuffer dest =new StringBuffer(len);
        for (int i =(len -1) ;  i  >= 0;  i--)
                dest.append(palindrome.charAt(i));
        palindrome = new String( dest.toString() );
        System.out.println(dest.toString());
    }
}
```

String and StringBuffered - constructors

String()
String(byte [])
String(byte [],int,int)
String(byte [],int,int,String)
String(byte [],String)
String(char [])
String(char [],int,int)
String(String)
String(StringBuffer)

StringBuffer()
StringBuffer(int)
StringBuffer(String)

String and StringBuffered – substrings

String substring (int)
String substring (int, int)

- Returns a new string that is a substring of this string or string buffer.
- The first integer argument specifies the index of the first character.
- The second integer argument is the index of the last character -1 .
- The length of the substring is therefore the first int minus the second int .

String class – other methods

int indexOf (int)
int lastIndexOf (int)
int indexOf (int,int)
int lastIndexOf (int,int)
int indexOf (String)
int lastIndexOf (String)
int indexOf (String,int)
int lastIndexOf (String,int)
Ex: **FilenameDemo.java**

String class – other methods

boolean endsWith (String)
boolean startsWith (String)
boolean startsWith (String,int)
int compareTo (String)
int compareTo (Object)
int compareToIgnoreCase (String)
boolean equals (Object)
boolean equalsIgnoreCase (String)
boolean regionMatches (int, String, int, int)
boolean regionMatches (boolean, int, String, int, int)

String manipulation

String concat(String) - Concatenates the String argument to the end of this string.
String replace(char,char) - Replaces all occurrences of the character specified as the first argument with the character specified as the second argument.
String trim() - Removes white space from both ends
String toLowerCase()
String toUpperCase()

FilenameDemo.java

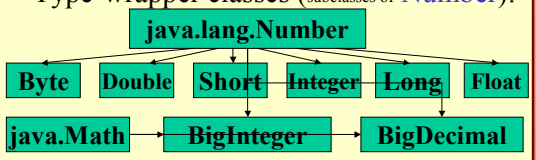
```
public class FilenameDemo {
public static void main(String[] args)
{  Filename myHomePage = new
     Filename("/home/mem/index.html",
        '/', '.');
     System.out.println("Extension = "
        + myHomePage.extension());
     System.out.println("Filename = "
        + myHomePage.filename());
      System.out.println("Path = "
        + myHomePage.path());
   } }
```

Filename.java

```
public class Filename {
    private String fullPath;
    private char pathSeparator, extensionSeparator;
    public Filename(String str, char sep, char ext) {
        fullPath = str;
        pathSeparator = sep;
        extensionSeparator = ext;  }
    public String extension() {
        int dot = fullPath.lastIndexOf(extensionSeparator);
        return fullPath.substring(dot + 1);}
    public String filename() {
        int dot = fullPath.lastIndexOf(extensionSeparator);
        int sep = fullPath.lastIndexOf(pathSeparator);
        return fullPath.substring(sep + 1, dot);
    }
    public String path() {
        int sep = fullPath.lastIndexOf(pathSeparator);
        return fullPath.substring(0, sep); }
```


Numbers

- *Type-wrapper classes* vs. **primitive data types.**
- Primitive data types: byte, int, short, long, float, double, char, string.
- Type-wrapper classes (subclasses of Number):



Numbers

- *Type-wrapper classes* vs. **primitive data types.**
- You can store a value of primitive type in a type-wrapper object whenever an object is required. Ex., the Vector class holds only objects. if you want to put numbers in a vector, you wrap them in a type-wrapper object.
- The classes define useful variables, such as MIN_VALUE and MAX_VALUE. Also provide useful methods for converting values to other types.
- The classes are used in **reflection**, part of the Java platform that allows programs to gather information about any object or class in the Java VM.
- BigInteger and BigDecimal extend the primitive data types and allow for arbitrary-precision.

Numbers – common instance methods

byte **byteValue()**
short **shortValue()**
int **intValue()**
long **longValue()**
float **floatValue()**
double **doubleValue()**

- Convert the value of this number object to the primitive data types of byte , short , int , long , float , and double .

int **compareTo(Integer)**
int **compareTo(Object)**

- Compare this number object to the argument.

Converting Strings to Numbers

```
public class ValueOfDemo {
    public static void main(String [] args)){
        if (args.length == 2){  // convert strings to numbers
            float a = Float.valueOf(args [0]).floatValue();
            float b = Float.valueOf(args [1]).floatValue();
            System.out.println("a +b ="+(a +b));
            System.out.println("a -b ="+(a -b));
            System.out.println("a *b ="+(a *b));
            System.out.println("a /b ="+(a /b));
            System.out.println("a %b ="+(a %b));
        }  else  {
            System.out.println("Enter TWO NUMBERS!");
}   }   }
```


Conversely – Converting Numbers to Strings

```
public class ToStringDemo       {
    public static void main(String [] args)){
        String s =Double.toString(858.48);
        int dot =s.indexOf('.');
        System.out.println(s.substring(0,dot).length()
                +"digits before decimal point.");
        System.out.println(s.substring(dot+1).length()
        +"digits after decimal point.");
    }
}
```

3 digits before decimal point.
2 digits after decimal point.

Formatting Numbers –look it up online …

java.text.NumberFormat
java.text.DecimalFormat

1.23 vs. 1,23

- **NumberFormat** numberFormatter = NumberFormat.**getNumberInstance**(Locale.FRANCE);
  NumberFormat currencyFormatter = NumberFormat.**getCurrencyInstance**();
  see Java Internationalization …
- **DecimalFormat** myFormatter =new DecimalFormat(pattern);
  String output =myFormatter.format(value);

Basic Math functions

```
public class BasicMathDemo {
  public static void main(String [] args)){
    double aNumber =-191.635;
   System.out.println("|"+aNumber +"|= "
                +Math.abs(aNumber));
    System.out.println("Ceiling("+aNumber +")= "
                +Math.ceil(aNumber));
    System.out.println("Floor("+aNumber +")= "
                +Math.floor(aNumber));
    System.out.println("Rint("+aNumber +")= "
                +Math.rint(aNumber));
}  }
```

```
|-191.635| = 191.635
Ceiling(-191.635)=-191
Floor(-191.635)= -192
Rint(-191.635)= -192
```


More Math functions

double **exp** (double) // e^arg
double **log** (double) // ln(arg)
double **pow** (double,double) // arg1^ arg2
double **sqrt** (double) // Sqrt(arg)
double **min** (double, double)
double **max** (double, double)
double **sin** (double)
…
double **atan** (double)
0.0 <=Math.**random**()<1.0

Example – ComplexNumber.java

```
public class ComplexNumber {
 public ComplexNumber  (double real, double imaginary)
     { this.x = real; this.y = imaginary; }
     /** * An accessor method. Returns the real part of the complex
     number. Note that there is no setReal() method to set the real
     part. This means that the ComplexNumber class is "immutable". **/
public double real() { return x; }
 /** An accessor method. Returns the imaginary part of the
     complex number */
public double imaginary() { return y; }
     /** Compute the magnitude of a complex number */
public double magnitude() { return Math.sqrt(x*x + y*y); }
     /** * This method converts a ComplexNumber to a string. This
     is a method of Object that we override so that complex numbers
     can be meaningfully converted to strings, and so they can
     conveniently be printed out with System.out.println() and related methods **/
```


Example – ComplexNumber.java (cont.)

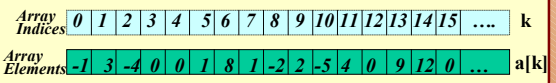
```
public String toString() { return "{" + x + "," + y + "}"; }
     /** * This is a static class method. It takes two complex numbers, adds them,
     and returns the result as a third number. Because it is static, there is no
     "current instance" or "this" object. Use it like this: ComplexNumber c =
     ComplexNumber.add(a, b); **/
public static ComplexNumber add(ComplexNumber a, ComplexNumber b)
     { return new ComplexNumber(a.x + b.x, a.y + b.y); }
     /** * This is a non-static instance method by the same name. It adds the
     specified complex number to the current complex number. Use it like this:
     ComplexNumber c = a.add(b); **/
public ComplexNumber add(ComplexNumber a)
     {  return new ComplexNumber(this.x + a.x, this.y+a.y); }
          /** A static class method to multiply complex numbers */
public static ComplexNumber multiply(ComplexNumber a, ComplexNumber b)
     { return new ComplexNumber(a.x*b.x - a.y*b.y, a.x*b.y + a.y*b.x);  }
          /** An instance method to multiply complex numbers */
public ComplexNumber multiply(ComplexNumber a)
     {  return new ComplexNumber(x*a.x - y*a.y, x*a.y + y*a.x); }
}
```


Example – Randomizer.java

```
public class Randomizer {
      // All "static final" fields are constants.
     static final int m = 233280, a = 9301, c = 49297;  /*The state variable
     maintained by each Randomizer instance */
     int seed = 1;  */
     /** * The constructor for the Randomizer() class. It must be passed some
     arbitrary initial value or "seed" for its pseudo-randomness. **/
 public Randomizer(int seed) { this.seed = seed; }
     /** * This method computes a pseudo-random number between 0 and 1
     using a very simple algorithm. Math.random() and java.util.Random are
     actually a lot better at computing randomness. **/
public float randomFloat()
     {  seed = (seed * a + c) % m; return (float) Math.abs((float)seed/(float)m); }
     /** Computes a pseudo-random integer between 0 and specified maximum.*/
public int randomInt(int max) { return Math.round(max * randomFloat()); }
     /** This nested class is a simple test program: it prints 10 random ints. Note
     how the Randomizer object is seeded using the current time. */
   public static class Test {
     public static void main(String[] args)
     {    Randomizer r = new Randomizer((int)new java.util.Date().getTime());
          for(int i = 0; i < 10; i++) System.out.println(r.randomInt(100)); }
} }
```

Arrays

- An **array** is a structure that holds multiple values of the same type. The length of an array is established when the array is created (at runtime). After creation, an array is a fixed-length structure
- An ***array element*** is one of the values within an array and is accessed by its position within the array.
- If you want to store data of different types in a single structure or if you need a structure whose size can change dynamically, e.g., **Vector**.



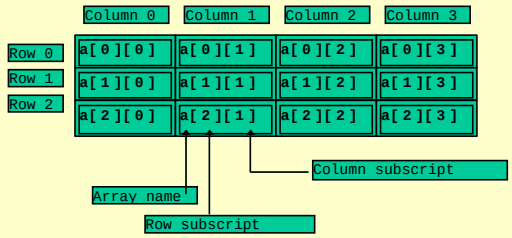
Arrays – example

```
public class ArrayDemo {
    public static void main(String [] args)) { // array of ints
        int [] anArray;              // declaration
        anArray =new int [10];  // create an array of ints
                //assign a value to each array element
        for (int i =0; i < anArray.length; i++)  {
                anArray [i] = 2*i;          // initialize array
                System.out.print(anArray [i] +"");
        }
        System.out.println();
} }
```

Output: 0 2 4 6 8 10 12 14 16 18

Multidimensional Arrays – arrays of arrays

String [][][] cartoons // sub-arrays may have different sizes!



Multidimensional Arrays – Example

```
int [][] aMatrix ==new int [4 ][]; // subarray.length unspecified
for (int i =0;i <aMatrix.length;i++) { //populate matrix
    aMatrix [i ] ==new int [5 ];     //create sub-array
    for (int j =0;j <aMatrix [i ].length;j++){
            aMatrix [i ][j ] =i +j;
    }
}
for (int i =0; i < aMatrix.length;  i++)  { // print matrix
    for (int j =0;j <aMatrix [i ].length;j++){
        System.out.print(aMatrix [i ][j ] ++"");
    }
    System.out.println();
}
```


Array Copying

The System's **arraycopy** method to efficiently copy data from one array into another.

public static void arraycopy(Object *source* ,int *srcInd*, Object *dest* int *desiInd* ,int *length*)

```
public class ArrayCopyDemo {
    public static void main(String [] args)){
        char [] copyFrom ==new char [] {'d', 'e', 'c', 'a', 'f', 'f', 'e',
                                        'i', 'n', 'a', 't', 'e', 'd'};
        char [] copyTo ==new char [7 ];
        System.arraycopy(copyFrom, 2, copyTo, 0, 7);
        System.out.println(new String(copyTo));
} }
```

Output: caffein

Example: Sort_1D_Array.java

```
public class Sort_1D_Array {
    /** A very simple sorting algorithm NOT very efficient **/
public static void sort(double[] nums) {
    // Loop through each element of the array, sorting as we go.
    // Each time through, find the smallest remaining element,
    // and move it  to the first unsorted position in the array.
    for(int i = 0; i < nums.length; i++)          {          int min = i;
        // holds the index of the smallest element
        // find the smallest one between i and the end of the array
        for(int j = i ; j < nums.length; j++)
                if (nums[j] < nums[min]) min = j;
                // Now swap the smallest one with element i.
                // This leaves all elements between 0 and i sorted.
    double tmp = nums[i]; nums[i] = nums[min]; nums[min] = tmp;
} }
```

Example: Sort_1D_Array.java

```
/* This is a simple test program for the algorithm above */

public static void main(String[] args) { double[]
    nums = new double[10];
        // Create an array to hold numbers
    for(int i = 0; i < nums.length; i++) // Get random numbers
                nums[i] = Math.random() * 100;
    sort(nums);       // Sort them
    for(int i = 0; i < nums.length; i++)           // Print them
                System.out.println(nums[i]);
}
}  // End:: Sort_1D_Array class
```