Data Structures and Algorithms

Prof. Nadeem Abdul Hamid
CSC 220 - Fall 2005
Lecture Unit 1 - Java Review

Introduction: Data Structures

- **What is a data structure?**
  - Way of storing data in computer so can be used efficiently
  - Set of operations that access the data in prescribed ways

Choosing Data Structures

- **Choice of data structures affects**
  - Performance
  - Simplicity
  - Readability

  of computer programs
- Proper use of data structures necessary for large programs

What is the Study of Data Structures?

- **'Program'**: computer language algorithm for solving a problem
- To solve larger problems, need to build on solutions to smaller problems - reuse solutions, etc.
- **Data structures**: study of how to abstract solutions to problems of storing data so that we can easily reuse them

Course Mechanics

- **Syllabus, lectures notes, assignments, etc. on web page**
  - *http://fsweb.berry.edu/academic/mans/nhamid/classes/cs220/05fall*
- **Class meetings**
  - Lectures: Mon/Wed/Fri, 9-9:50AM, SCI 233
  - Labs: Tues, 12:30–2:30PM, SCI 233
- **Contact**
  - Office: SCI 354B — Phone: 368-5632
  - Email: *nadeem@acm.org*
- **Office Hours**
  - Mon — 11AM–12:30PM
  - Tue — 11AM–12:30PM
  - Wed — 11AM–12:30PM and 2–4PM
  - Thu — 10AM–12:30PM and 2-3PM
  - (or by appointment…)

Assignments

- Weekly lab/homeworks
  - Due on Mondays
- Programming Projects
- **DON'T WAIT UNTIL DAY/NIGHT BEFORE TO START WORKING ON ASSIGNMENTS**
  - No late work accepted, without formal excuse/prior arrangement
  - You will NOT be able to complete the programming assignments in one night
- Send email if you have a problem (attached relevant files and say where you're stuck)

Programming Assignments

- Completed programs must 'work'!!!
  - Compile and run
- If you leave programming assignments to the last minute, you will run a major risk of having incomplete work

7

Materials and Resources

- Textbook:
  - *Objects, Abstraction, Data Structures and Design Using Java*, Elliot B. Koffman and Paul A.T. Wolfgang
- Online course website: Check regularly
- Software (in computer lab SCI 228/233)
  - Java 5.0 (JDK): *http://java.sun.com/j2se/1.5.0/download.jsp*
    - Compiler; runtime system
  - Eclipse: *http://www.eclipse.org*
    - integrated development environment
- Using computers during class

8

Grading and Evaluation

- Class participation and attendance (10%)
- Lab participation and attendance (10%)
- Assignments/Projects (50%)
- Exams (30%) *Tentative dates:*
  - Midterm exam: Friday, October 7, 2005
  - Final exam: Thursday, December 8, 2005 (8 - 10AM)
- Policies (see syllabus)
  - Attendance
  - Academic integrity
  - Late work
  - Disabilities

9

What is Java?

- JVM - virtual bytecodes, platform independence, security, Java vs. JVML
  - javac -- compiler
  - java -- virtual machine

10

Java Web Pages

- Download from:
  - http://java.sun.com/j2se/1.5.0/download.jsp
- API documentation
  - http://java.sun.com/j2se/1.5.0/docs/api/index.html
- The Java Tutorial
  - http://java.sun.com/docs/books/tutorial/

11

Object-Oriented Programming

- is …?
- Classes vs. objects
- Fields (attributes)
- Methods (operations)

12

Hello World!

```
import javax.swing.*;

public class HelloWorld {
  public static void main( String[] args ) {
    String name = JOptionPane.showInputDialog( "Enter your name" );
    JOptionPane.showMessageDialog( null, "Hello " + name
                                   + ", welcome to Java!" );
    System.exit( 0 );
  }
}
```

- Java API - Swing, AWT, util, …
- `import` statement
- `main` method

Java Data Types

- Java distinguishes between **primitive types** (numbers, characters) and **objects**
- Values of primitive types stored directly in variables
- Objects are manipulated through **reference variables**, which 'point to' (store address in memory of) an object

Primitive Data Types

TABLE A.1
Java Primitive Data Types in Increasing Order of Range

Data Type	Range of Values
`byte`	–128 through 127
`short`	–32,768 through 32,767
`int`	–2,147,483,648 through 2,147,483,647
`long`	–9,223,372,036,854,775,808 through 9,223,372,036,854,775,807
`float`	Approximately $\pm 10^{-38}$ through $\pm 10^{38}$ and 0 with 6 digits precision
`double`	Approximately $\pm 10^{-308}$ through $\pm 10^{308}$ and 0 with 15 digits precision
`char`	The Unicode character set
`boolean`	`true`, `false`

Unicode Character Set

TABLE A.2
The First 128 Unicode Symbols

	000	001	002	003	004	005	006	007
0	Null		Space	0	@	P	`	p
1			!	1	A	Q	a	q
2			"	2	B	R	b	r
3			#	3	C	S	c	s
4			$	4	D	T	d	t
5			%	5	E	U	e	u
6			&	6	F	V	f	v
7	Bell		'	7	G	W	g	w
8	Backspace		(	8	H	X	h	x
9	Tab		)	9	I	Y	I	y
A	Line feed		*	:	J	Z	j	z
B		Escape	+	;	K	[	k	{
C	Form feed		,	<	L	\	l	|
D	Return		-	=	M	]	m	}
E			.	>	N	^	n	~
F			/	?	O	_	o	delete

Primitive Variables and Constants

- Declaring and initializing variables
- Constants
- Identifier conventions

Operators

TABLE A.3
Operator Precedence

Rank	Operator	Operation	Associativity
1	[]	Array subscript	Left
	()	Method call	
	.	Member access	
2	++	Pre- or postfix increment	Right
	--	Pre- or postfix decrement	
	+ -	Unary plus or minus	
	!	Complement	
	~	Bitwise complement	
	(*type*)	Type cast	
	new	Object creation	
3	*, /, %	Multiply, divide, remainder	Left
4	+	Addition or string concatenation	Left
	-	Subtraction	

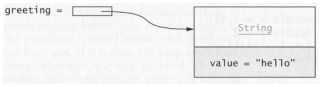
Operators (cont.)

5	<<	Signed bit shift left	Left
	>>	Signed bit shift right	
	>>>	Unsigned bit shift right	
6	<, <=	Less than, less than or equal	Left
	>, >=	Greater than, greater than or equal	
	instanceof	Reference test	
7	==	Not equal to	Left
	!=	Equal to	
8	&	Bitwise and	Left
9	^	Bitwise exclusive or	Left
10	|	Bitwise or	Left
11	&&	Logical and	Left
12	||	Logical or	Left
13	?:	Conditional	Left
14	=	Assignment	Right
	*=, /=, %=, +=, -=, <<=, >>=, >>>=, &=, |=	Compound assignment	

Object Variables

- Store references to objects

```
String greeting = "hello";
String welcome = greeting;
String hello = new String( welcome );
```



Control Statements

- Determine flow of execution through a program
  - Sequence
  - Selection -- if…else / switch
  - Repetition -- while / for / do…while

If/else and Switch Statements

```
if ( operator == '+' ) {
  result = x + y;
  addOp++;
}
else if ( operator == '-' ) {
  result = x - y;
  subtractOp++;
}
else if ( operator == '*' ) {
  result = x * y;
  multiplyOp++;
}
else if ( operator == '/' ) {
  result = x / y;
  divideOp++;
}
```

```
switch ( operator ) {
  case '+':
    result = x + y;
    addOp++;
    break;
  case '-':
    result = x - y;
    subtractOp++;
    break;
  case '*':
    result = x * y;
    multiplyOp++;
    break;
  case '/':
    result = x / y;
    divideOp++;
    break;
  default:
    // do nothing...
}
```


Defining Classes

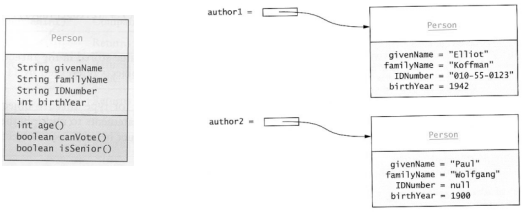
- A Java program is a collection of interacting objects, defined by their classes
- Example: Person class
  - Objects of class Person store data:
    - Given name
    - Family name
    - ID number
    - DOB

Person Object Operations

- Calculate age
- Determine if old enough to vote
- Determine if senior citizen
- Get values of Person object data fields
- Set values of Person object data fields

UML Diagrams

- UML = Unified Modeling Language ™
  - Industry standard for documenting class relationships



Person Class Implementation

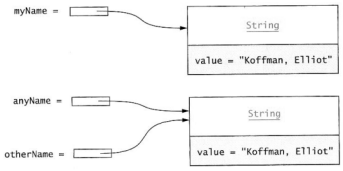
- Private data fields ('instance variables')
- Public methods
- Constants
- Constructors
- Accessor and mutator (modifier) methods
- Use of `this` keyword
- Methods `toString`, `equals`

Comparing Objects

```
String myName = "Elliot Koffman";
myName = myName.substring(7) + ", "
         + myName.substring(0,6);

String anyName = new String( myName );

String otherName = anyName;
```



TestPerson Program

```
/** TestPerson is an application that tests class Person. */
public class TestPerson {
  public static void main( String[] args ) {
    Person p1 = new Person( "Sam", "Jones", "1234", 1930 );
    Person p2 = new Person( "Jane", "Jones", "5678", 1990 );
    System.out.println( "Age of " + p1.getGivenName() +
                        " is " + p1.age( 2005 ) );
    if ( p1.isSenior( 2005 ) )
      System.out.println( p1.getGivenName() +
                          " can ride the subway for free" );
    else
      System.out.println( p1.getGivenName() +
                          " must pay to ride the subway" );

    System.out.println( "Age of " + p2.getGivenName() +
                        " is " + p2.age( 2005 ) );
    if ( p2.canVote( 2005 ) )
      System.out.println( p2.getGivenName() + " can vote");
    else
      System.out.println( p2.getGivenName() + " can't vote");

  }
}
```


Class Components (UML)

- String objects are components of a Person object

FIGURE A.10
UML Diagram Showing That `String` Objects Are Components of Class `Person`

Javadoc

- Uses standard form of writing comments to generate HTML
- Focuses on text within /** and */
- Javadoc tags:

TABLE A.12
Javadoc Tags

Javadoc Tag and Example of Use	Purpose
`@author` *Koffman and Wolfgang*	Identifies the class author.
`@param first` *The given name*	Identifies a method parameter.
`@return` *The person's age*	Identifies a method return value.

Math Class

TABLE A.5
Class Math Methods

Method	Behavior
`static` *numeric* `abs(`*numeric*`)`	Returns the absolute value of its *numeric* argument (the result type is the same as the argument type).
`static double ceil(double)`	Returns the smallest whole number that is not less than its argument.
`static double cos(double)`	Returns the trigonometric cosine of its argument (an angle in radians).
`static double exp(double)`	Returns the exponential number *e* (i.e., 2.718 ...) raised to the power of its argument.
`static double floor(double)`	Returns the largest whole number that is not greater than its argument.
`static double log(double)`	Returns the natural logarithm of its argument.
`static` *numeric* `max(`*numeric*`,` *numeric*`)`	Returns the larger of its *numeric* arguments (the result type is the same as the argument types).
`static` *numeric* `min(`*numeric*`,` *numeric*`)`	Returns the smaller of its *numeric* arguments (the result type is the same as the argument type).
`static double pow(double, double)`	Returns the value of the first argument raised to the power of the second argument.
`static double random()`	Returns a random number greater than or equal to 0.0 and less than 1.0.
`static double rint(double)`	Returns the closest whole number to its argument.
`static long round(double)`	Returns the closest `long` to its argument.
`static int round(float)`	Returns the closest `int` to its argument.
`static double sin(double)`	Returns the trigonometric sine of its argument (an angle in radians).
`static double sqrt(double)`	Returns the square root of its argument.
`static double tan(double)`	Returns the trigonometric tangent of its argument (an angle in radians).
`static double toDegrees(double)`	Converts its argument (in radians) to degrees.
`static double toRadians(double)`	Converts its argument (in degrees) to radians.

String Class

TABLE A.7
`String` Methods in `java.lang.String`

Method	Behavior
`char charAt(int pos)`	Returns the character at position `pos`.
`int compareTo(String)`	Returns a negative integer if this string's contents precede the argument string's contents in the dictionary; returns 0 if this string and the argument string have the same contents; returns a positive integer if this string's contents follow those of the argument string. This comparison is case sensitive.
`int compareToIgnoreCase(String)`	Returns a negative, zero, or positive integer according to whether this string's contents precede, match, or follow the argument string's contents in the dictionary, ignoring case.
`boolean equals(Object)`	Returns `true` if this string's contents is the same as its argument string's contents.
`boolean equalsIgnoreCase(String)`	Returns `true` if this string's contents is the same as the argument string's contents, ignoring case.
`int indexOf(char)` `int indexOf(String)`	Returns the index within this string of the first occurrence of its character or string argument, or `-1` if the argument is not found.
`int indexOf(char, int index)` `int indexOf(String, int index)`	Returns the index within this string of the first occurrence of its first character or string argument, starting at the specified `index`.
`int lastIndexOf(char)` `int lastIndexOf(String)`	Returns the index within this string of the rightmost occurrence of its character or string argument.
`int lastIndexOf(char, int index)` `int lastIndexOf(String, int index)`	Returns the index within this string of the last occurrence of its first character or string argument, searching backward stopping at the specified `index`.
`int length()`	Returns the length of this string.
`String replace(char oldChar, char newChar)`	Returns a new string resulting from replacing all occurrences of `oldChar` in this string with `newChar`.
`String substring(int start)`	Returns a new string that is a substring of this string, starting at position `start` and going to the end of the string.
`String substring(int start, int end)`	Returns a new string that is a substring of this string, starting with the character at position `start` and ending with the character at position `end - 1`.
`String toLowerCase()`	Returns a new string in which all of the letters in this string are converted to lowercase.
`String toUpperCase()`	Returns a new string in which all of the letters in this string are converted to uppercase.
`String trim()`	Returns a new string in which all the white space is removed from both ends of this string.

32

StringBuffer Class

TABLE A.8
`StringBuffer` Methods in `java.lang.StringBuffer`

Method	Behavior
`void StringBuffer append(`*anyType*`)`	Appends the string representation of the argument to this `StringBuffer`. The argument can be of any data type.
`int capacity()`	Returns the current capacity of this `StringBuffer`.
`void StringBuffer delete(int start, int end)`	Removes the characters in a substring of this `StringBuffer`, starting at position `start` and ending with the character at position `end - 1`.
`void StringBuffer insert(int offset,` *anyType* `data)`	Inserts the argument `data` (any data type) into this `StringBuffer` at position `offset`, shifting the characters that started at `offset` to the right.
`int length()`	Returns the length (character count) of this `StringBuffer`.
`StringBuffer replace(int start, int end, String str)`	Replaces the characters in a substring of this `StringBuffer` (from position `start` through position `end - 1`) with characters in the argument `str`. Returns this `StringBuffer`.
`String substring(int start)`	Returns a new string containing the substring that begins at the specified index `start` and extends to the end of this `StringBuffer`.
`String substring(int start, int end)`	Return a new string containing the substring in this `StringBuffer` from position `start` through position `end - 1`.
`String toString()`	Returns a new string that contains the same characters as this `StringBuffer` object.

33

StringTokenizer Class

TABLE A.9
`StringTokenizer` Methods in `java.util.StringTokenizer`

Method	Behavior
`StringTokenizer(String str)`	Constructs a new `StringTokenizer` object for the string specified by `str`. The delimiters are "whitespace" characters (space, newline, tab, and so on).
`StringTokenizer(String str, String delim)`	Constructs a new `StringTokenizer` object for the string specified by `str`. The delimiters are the characters specified in `delim`.
`boolean hasMoreTokens()`	Returns `true` if this tokenizer's string has more tokens; otherwise, returns `false`.
`String nextToken()`	Returns the next token of this tokenizer's string if there is one; otherwise, a run-time error will occur.

34

Using StringTokenizer

```
import java.util.StringTokenizer;

public class TestTokenizer {
  public static void main( String[] args ) {
    String personData = "Doe, John 5/15/65";
    StringTokenizer sT = new StringTokenizer( personData, " ,/" );

    String familyName = sT.nextToken(); // stores "Doe"
    String givenName = sT.nextToken();  // stores "John"
    String month = sT.nextToken();      // stores "5"
    String day = sT.nextToken();        // stores "15"
    String year = sT.nextToken();       // stores "65"

    String sentence = "This is a set of seven tokens";
    StringTokenizer getWords = new StringTokenizer( sentence );

    while ( getWords.hasMoreTokens() )
      System.out.println( getWords.nextToken() );
  }
}
```

35

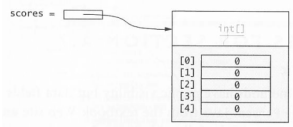
Arrays

- Java arrays are also objects (with some special syntax provided)
  - Indexed using subscript notation
    - arrayName[subscript]

36

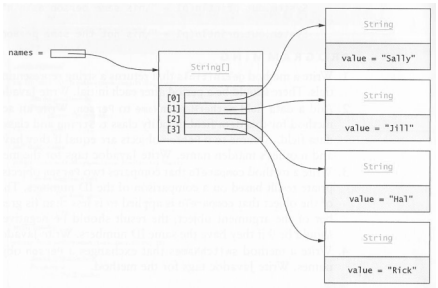
Array Example A.14

```
int[] scores = new int[5];   // an array with 5 type int values
```



Array Example A.15

```
String[] names = { "Sally", "Jill", "Hal", "Rick" };
```



Array Example A.16

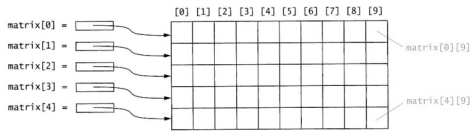
```
// Declare people as type Person[]
Person[] people;
// Define n in some way.
int n = ...
// Allocate storage for the array
people = new Person[n];

...

people[0] = new Person("Elliot", "Koffman", "010-055-0123", 1942);
people[1] = new Person("Paul", "Wolfgang", "015-023-4567", 1945);
```

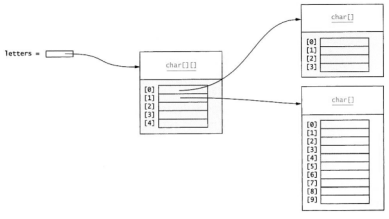

Array Example A.18

```
double[][] matrix = new double[5][10];
```

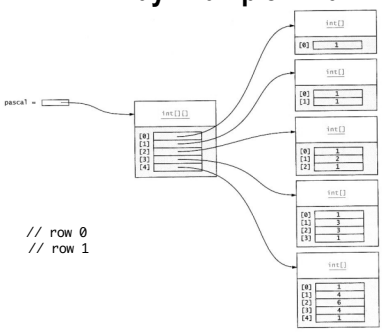


Array Example A.19

```
char[][] letters = new char[5][];
letters[0] = new char[4];
letters[1] = new char[10];
...
```



Array Example A.20




```
int[][] pascal = {
        {1},          // row 0
      {1, 1},         // row 1
    {1, 2, 1},
  {1, 3, 3, 1},
{1, 4, 6, 4, 1} };
```

Array Size

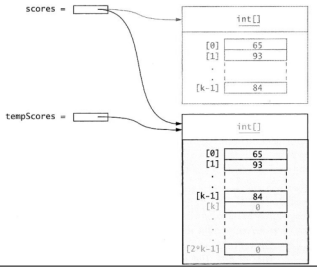
- An array has a **length** data field that stores its size
- Array size cannot be changed
- But can use **System.arraycopy** method to copy data into a bigger array

System.arraycopy(*source*, *sourcePos*, *dest*, *destPos*, *numElems*)

Using arraycopy()

```
int[] scores = new int[k];
...

int[] tempScores = new int[2 * scores.length];
System.arraycopy( scores, 0, tempScores, 0, scores.length );
scores = tempScores;
```



Pascal's Triangle

- Write a PascalTriangle class
  - Constructor should take integer argument of number of rows to build up
  - Provide a toString() method to format the triangle as a string suitable for printing out
- Generating the triangle:
  - pascal[i+1][j] = pascal[i][j-1] + pascal[i][j]
  - First and last elements in each row are 1

Keyboard Input

- System.in – object corresponding to keyboard input stream
  - Very primitive - reads byte at a time
- For more convenient user input, use the Scanner class (new to Java 5.0)

```
Scanner in = new Scanner(System.in);
System.out.print("Enter quantity: ");
int quantity = in.nextInt();
```

'Input prompt'

Scanner Methods

- nextInt()
- nextDouble()
- nextWord()
  - Returns the next word input as a String object
  - End of the word is indicated by *whitespace*: space/end of line/tab
- nextLine()
  - Returns next entire line of input as a String

Input from a Dialog Box

- If not using Scanner (Java version prior to 5.0), easy way to get user input is create pop-up window

```
import javax.swing.JOptionPane;

public class Test {
    public static void main(String[] args) {

        String input = JOptionPane.showInputDialog( "Enter price:" );
        double price = Double.parseDouble( input );
        System.out.println( "You entered: " + price );

        System.exit(0);
    }
}
```

Needed to force program to exit

Formatted Output

```
double total = 3.50;
final double TAX_RATE = 8.5;  // Tax rate in percent
double tax = total * TAX_RATE / 100; // tax is 0.2975
System.out.println( "Total: " + total );
System.out.println( "Tax:   " + tax );
```

Output:

```
Total: 3.5
Tax:   0.2975
```

```
System.out.printf( "Total: %5.2f%n", total );
System.out.printf( "Tax:   %5.2f%n", tax );
```

Output:

```
Total: 3.50
Tax:   0.30
```


Using the `printf` Method

```
System.out.printf( "Total: %5.2f%n", total );
```

Format string

Format specifiers

Other parameters - values filled into corresponding fields of the format string

Format Specifiers

Basic format code: %f

Format type

- d — decimal integer
- x — hexadecimal integer
- o — octal integer
- f — fixed floating-point
- e — exponential f.p.
- g — general f.p.
  - (uses shorter of e/f)
- s — string
- n — platform-independent line end

Format code options: %5.2f

Width *- the number of spaces in which to fit the value (adds blank spaces if necessary)*

Precision *- the number of digits after decimal point*

Format Flags

- Immediately follow the % character
  - – (hyphen) — left justification
  - 0 (zero) — show leading zeroes (in numbers)
  - + (plus) — show plus sign for positive numbers
  - (— enclose negative numbers in parentheses
  - , (comma) — show decimal separators
  - ^ — convert letters to uppercase

String `format` Method

- `printf` is a method of the `PrintStream` class
  - `System.out` is a `PrintStream` object
- The `String` class has a (static) `format` method similar to `printf`
  - Returns a string instead of producing output

```
String message = String.format( "Total:%5.2f", total );
```

  - sets message to the value "Total: 3.50"

Other Dialog Boxes

- Display simple message window using

```
JOptionPane.showMessageDialog(null, "Hello");
```

- Display window of button choices using **JOptionPane.showOptionDialog** method

JOptionPane.showOptionDialog

```
String[] choices = { "insert", "delete", "add", "display" };
int sel = JOptionPane.showOptionDialog( null,
                                        "Select an operation",
                                        "Operation menu",
                                        JOptionPane.YES_NO_CANCEL_OPTION,
                                        JOptionPane.QUESTION_MESSAGE,
                                        null,
                                        choices,
                                        choices[0] );
```


File Input/Output

- Two ways of storing data in files
  - *Text* format – human readable sequence of characters
    - Convenient for humans
  - *Binary* format – bytes of data
    - More compact and efficient
- We will use
  - `Scanner` class to read input from text files
  - `PrintWriter` class to write output to text files

Reading Text File

- First construct **`FileReader`** object with the name of the input file
- Then use it to construct a `Scanner` object
- Use the `Scanner` object for input just as if it was keyboard input
  - Use `next`, `nextLine`, `nextInt`, `nextDouble` methods

```
FileReader reader = new FileReader( "input.txt" );
Scanner in = new Scanner( reader );
```

- After done reading input, call the **`close`** method on the `FileReader` object

Writing Text File

- Construct a `PrintWriter` object with the name of the output file
  - Use `print`, `println`, `printf` methods

```
PrintWriter out = new PrintWriter( "output.txt" );
```

- Close the file when done
  - Otherwise not all output may be written to the file

```
out.close();
```


Skeleton Code for File Input/Output

```
// import necessary classes
import java.io.IOException;
import java.io.PrintWriter;
import java.io.FileReader;
import java.util.Scanner;

public class . . . {

   public . . . { // method
      . . .
      try {
         // Do file input/output stuff here
         // . . .

      } catch ( IOException exc ) {
          System.out.println( "Error processing file: " + exc );
      }
      . . .
   }
}
```

LineNumberer.java

Passing Arguments to main()

- Command line arguments are put into the `args` array (parameter of `main`)
- Example:

```
java FileTest indata.txt output.txt
```