9 TEXT-BASED APPLICATIONS

Objectives

- Write a program that uses command-line arguments and system properties
- Write a program that reads from *standard input*
- Write a program that can create, read, and write files
- Write a program that uses sets and lists
- Write a program to iterate over a collection

Command-Line Arguments

- Any Java technology applications can use command-line arguments
- These string arguments are placed on the command lien to launch the Java interpreter, after the class name:

```
java TestArgs arg1 arg2 "another arg"
```

- Each command-line arguments is placed in the args array that is passed to the static `main` method:

```
public static void main(String[] args)
```



```
public class TestArgs {
   public static void main(String[] args) {
      for ( int i=0 ; i<args.length ; i++ ) {
          System.out.println("args[" + i +
                             "] is '" +
                             args[i] + "'") ;
      }
  }
}
```

java TestArgs arg1 arg2 "another arg"

System Properties

- System properties is a feature that replaces the concept of *environment variables* (which is platform-specific)
- The `System.getProperties` method returns a `Proeprties` object
  - System.getProperties()
  - System.getProperties(String)
  - System.getProperties(String,String)
- The `getProperty` method returns a `String` representing the value of the named property.
- Use the –D option to include a new property.

The Properties Class

- The `Properties` class implements a mapping of names to values (a `String` to `String` map).
- The `propertyNames` method returns an `Enumeration` of all property names.
- The `getProperty` method returns a `String` representing the value of the named property.
- You can also read and write a properties collection into a file using `load` and `store`.

```
import java.util.Properties ;
import java.util.Enumeration ;
public class TestProperties {
   public static void main(String[] args) {
    Properties props = System.getProperties() ;
    Enumeration prop_names = props.propertyNames();
    while( prop_names.hasMoreElements() ){
      String prop_name=(String)prop_names.nextElement() ;
      String property = props.getProperty(prop_name) ;
      System.out.println("property '" + prop_name +
                         "' is '" + property + "'") ;
    }
  }
}
```



```
property 'java.runtime.name' is 'Java(TM) 2 Runtime Environment, Standard Edition'
property 'sun.boot.library.path' is 'F:\jdk1.3\jre\bin'
property 'java.vm.version' is '1.3.0-C'
property 'java.vm.vendor' is 'Sun Microsystems Inc.'
property 'java.vendor.url' is 'http://java.sun.com/'
property 'path.separator' is ';'
property 'java.vm.name' is 'Java HotSpot(TM) Client VM'
property 'file.encoding.pkg' is 'sun.io'
property 'java.vm.specification.name' is 'Java Virtual Machine Specification'
property 'user.dir' is 'C:\'
property 'java.runtime.version' is '1.3.0-C'
property 'java.awt.graphicsenv' is 'sun.awt.Win32GraphicsEnvironment'
property 'os.arch' is 'x86'
property 'java.io.tmpdir' is 'C:\DOCUME~1\kurt\LOCALS~1\Temp\'
property 'line.separator' is ' '
property 'java.vm.specification.vendor' is 'Sun Microsystems Inc.'
property 'java.awt.fonts' is ''
property 'os.name' is 'Windows 2000'
```



```
String osName= (System.getProperties()).getProperty("os.name");

switch ( osName.hashCode() ) {
  case -113889189 :
       System.out.println("Program has not been tested on this os!") ;
   break;
  default:
       System.out.println("Ok.") ;
}
```


Console I/O

- `System.out` allows you to write to "standard output".
  - It is an object of type `PrintStream`.
- `System.in` allows you to read from "standard input".
  - It is an object of type `InputStream`.
- `System.err` allows you to write to "standard error"
  - It is an object of type `PrintStream`.

Writing to Standard Output

- The `println` methods print the argument and a newline '\n'.
- The `print` methods print the argument without a newline
- The `print` and `println` methods are overloaded for most primitive types (`boolean`, `char`, `int`, `long`, `float`, and `double`) and for `char[]`, `Object`, and `String`.
- The `print(Object)` and `println(Object)` methods call the `toString()` method on the argument.

Reading From Standard Input

```
import java.io.* ;
public class KeyboardInput {
   public static void main(String[] args) throws IOException {
     String s                                              ;
     InputStreamReader ir = new InputStreamReader(System.in) ;
     BufferedReader in = new BufferedReader(ir)            ;
     System.out.println("Enter an integer : ")  ;
     try{

       while( (s = in.readLine()) != null )  )
            System.out.println("Read: "+ s) ;

        in.close();
     }
     catch( IOException e) {
          e.printStackTrace();
     }
   }
}
```

Files and File I/O

- The `java.io` package
- Creating `File` Objects
- Manipulating `File` Objects
- Reading and writing to file streams

Creating a New File Object

- `File myFile ;`
- `myFile = new File("myfile.txt") ;`
- `myFile = new File("MyDocs","myfile.txt") ;`
- Directories are treated just like files in Java; the File class supports methods for retrieving an array of files in the directory
- ```
  File myDir = new File("MyDocs") ;
  myFile = new File(myDir,"myfile.txt") ;
  ```

The class File defines platform-independent methods for manipulating a file maintained by a native file system. However, it does not allow you to access the contents of the file.

File Tests and Utilities

- `File` names :
  - String getName()
  - String getPath()
  - String getAbsolutePath()
  - String goParent()
  - boolean renameTo(File newName)
- `File` tests :
  - boolean exists()
  - boolean canWrite()
  - boolean canRead()
  - boolean isFile()
  - boolean isDirectory()

- `File` information
  - long lastModified()
  - long length()
  - boolean delete()
- `File` utilities :
  - boolean mkdir()
  - String[] list()

File Stream I/O

- File input:
  - Use the `FileReader`. class to read characters
  - Use the `BufferedReader` class to use the `readLine` method
- File output:
  - Use the `FileWriter` class to write characters
  - Use the `PrintWriter` class to use the `print` and `println` methods


```
import java.io.* ;
public class ReadFile {
    public static void main(String[] args) {
      File file = new File(args[0])                                    ;
      try {
         BufferedReader in = new BufferedReader(new FileReader(file)) ;
         String s ;
         s = in.readLine() ;
         while( s != null ) {
               System.out.println("Read: " + s ) ;
               s = in.readLine() ;
         }
         in.close() ;
      } catch ( FileNotFoundException e1) {
          System.err.println("File not found: " + file);
      } catch ( IOException e2 ) {
          e2.printStackTrace() ;
      }
    }
}
```
```

```
import java.io.* ;
public class WriteFile {
    public static void main(String[] args) {
      File file = new File(args[0])                                    ;
      try {
         BufferedReader in = new
                        BufferedReader(new InputStreamReader(System.in)) ;
         PrintWriter out = new PrintWriter(new FileWriter(file)) ;
         String s ;
         System.out.print("Enter file text.   ") ;
         System.out.println("[Type ctrl-d (or ctrl-z) to stop.]") ;
         while( (s=in.readLine()) != null ) {
               out.println(s) ;
         }
         in.close() ;
         out.close();
      } catch ( IOException e) {
         e.printStackTrace() ;
      }
    }
}
```


The Math Class

- The `Math` class contains a group of static math functions
  - Truncation: `ceil`, `floor`, and `round`
  - Variations on max, min, and abs
  - Trigonometry: `sin`, `cos`, `tan`, `asin`, `atan`, `toDegrees`, and `toRadians`
  - Logarithms: `log` and `exp`
  - Others: `sqrt`, `pow`, and `random`
  - Canstants: `PI` and `E`

The String Class

- `String` objects are immutable sequences of Unicode characters.
- Operations that create new strings: `concat`, `replace`, `substring`, `toLowerCase`, `toUpperCase`, and `trim`
- Search operations: `endsWith`, `startWith`, `indexOf`, and `lastIndexOf`.
- Comparisons: `equals`, `equalsIgnoreCase`, and `compareTo`
- Others: `charAt`, and `length`

Methods That Create New Strings

- `String concat(String s)` – returns a new string consisting of `this` string followed by the s string
- `String replace(char old, char new)` – returns a new string that is copy of `this` string with the `new` string replacing all occurences of the `old` string
- `String substring(int start,int end)` – returns a portion of `this` string starting at the `start` index and ending at `end`.
- `String toLowerCase()` – returns a new string consisting of `this` string converted to lowercase
- `String toUpperCase()` – returns a new string consisting of `this` string converted to uppercase

Search Methods

- `boolean endsWith(String s)` – returns `true` if `this` string ends with s
- `boolean startsWith(String s)` – returns `true` if `this` string starts with s
- `int indexOf(String s)` – returns index within `this` string that starts with s
- `int indexOf(int ch)` – returns index within `this` string of the first occurence of the character ch
- `int indexOf(String s, int offset)` – returns index within `this` string that matches with s starting at `offset`
- `int indexOf(int ch, int offset)` – returns index within `this` string of the first occurence of the character ch starting at `offset`

Comparison Methods

- `boolean equals(String s)` – returns `true` if `this` string is equal to string s
- `boolean equalsIgnoreCase(String s)` – returns `true` if `this` string is equal to (ignoring case) the string s
- `int compareTo(String s)` – performs a lexical comparison between `this` string and s; returns a negative `int` if `this` string is less than s, a positive `int` if `this` string is greater than s, or 0 if the two strings are equal

The StringBuffer Class

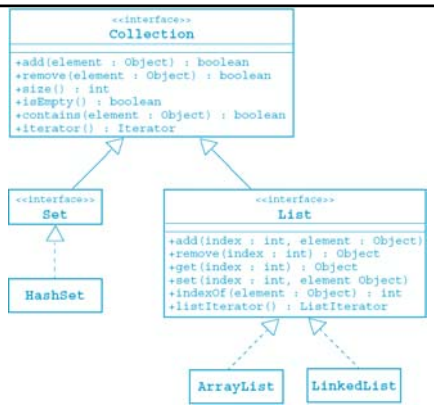
- `StringBuffer` objects are mutable sequences of Unicode characters
- Constructors:
  - `StringBuffer()` – creates an empty buffer
  - `StringBuffer(int capacity)` – creates an empty buffer with a specified initial capacity
  - `StringBuffer(String initialString)` – creates a buffer that initially contains the specified string
- Modification operations: `append`, `insert`, `reverse`, `setCharAt`, and `setLength`

Modifications Methods

- `StringBuffer append(String s)` – Modifies `this` string buffer by appending the s string onto the end of the buffer
- `StringBuffer insert(int offset, String s)` – Modifies `this` string buffer by inserting the s string into the buffer at the specified offset location.
- `StringBuffer reverse()` – Reverses the order of the string buffer
- `void setCharAt(int index, char ch)` – Modifies `this` string buffer by changing the character at the location specified by `index` to the specified character, `ch`.
- `void setLength(int newLength)`

The Collections API

- A *collection* is a single object representing a group of objects known as its elements
- Collection API contains interfaces that group objects as a:
  - `Collection` – A group of objects called elements; any specific ordering and allowance of duplicates is specified by each implementation
  - `Set` – An unordered collection; no duplicates are permitted
  - `List` – An ordered collection; duplicates are permitted



Set Example

```
import java.util.* ;
public class SetExample {
   public static void main(String[] args) {
      Set set = new HashSet();
      set.add("one") ;
      set.add("second") ;
      set.add("3rd") ;
      set.add(new Integer(4)) ;
      set.add(new Float(5.0F)) ;
      set.add("second") ;
      set.add(new Integer(4)) ;
      System.out.println(set) ;
   }
}
```


List Example

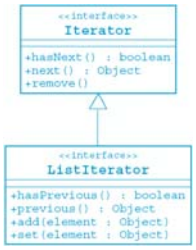
```
import java.util.* ;
public class ListExample {
   public static void main(String[] args) {
      List list = new ArrayList();
      list.add("one") ;
      list.add("second") ;
      list.add("3rd") ;
      list.add(new Integer(4)) ;
      list.add(new Float(5.0F)) ;
      list.add("second") ;
      list.add(new Integer(4)) ;
      System.out.println(list) ;
   }
}
```

Iterators

- Iteration is the process of retrieving every element in a collection
- An `Iterator` of a `Set` is unordered
- A `ListIterator` of a `List` can be scanned forwards (using the `next` method) or backwards (using the `previous` method):

```
List list = new ArrayList() ;
Iterator elements = list.iterator();
while( elements.hasNext() )
   System.out.println( elements.next() )
```


The Iterator Interface Hierarchy



Collections in JDK 1.1

- `Vector` implements the `List` interface
- `Stack` is a subclass of `Vector` and supports the `push`, `pop`, and `peek` methods
- `Hashtable` implements the `Map` interface
- `Enumeration` is a variation on the `Iterator` interface:
  - An enumeration is returned by the `elements` method in `Vector`, `Stack`, and `Hashtable`