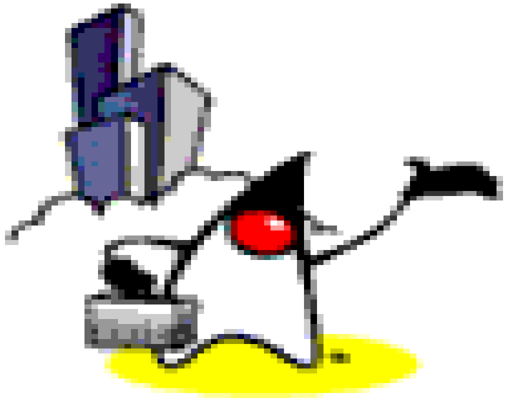
java.lang.* Classes

Sang Shin
www.JPassion.com
"Learn with JPassion!"

Topics

- *String* Class
- *StringBuffer* Class
- *Wrapper* Classes
- *Math* Class
- *System* Class
- *Process* and the *Runtime* Class
- *ProcessBuilder* class



String Class

The *String* Class

- Definition:
  - Represents combinations of character literals
  - Using Java, strings can be represented using:
    - Array of characters
    - The *String* class

- Constructors of String class
  - 11 constructors

The *String* Class: Constructors

```
class StringConstructorsDemo {
   public static void main(String args[]) {
      String s1 = new String();        //empty string
      char chars[] = { 'h', 'e', 'l', 'l', 'o'};
      String s2 = new String(chars);   //s2="hello";
      byte bytes[] = { 'w', 'o', 'r', 'l', 'd' };
      String s3 = new String(bytes);   //s3="world"
      String s4 = new String(chars, 1, 3);
      String s5 = new String(s2);
      String s6 = s2;
//continued
```

The *String* Class: Constructors

```
        System.out.println(s1);
        System.out.println(s2);
        System.out.println(s3);
        System.out.println(s4);
        System.out.println(s5);
        System.out.println(s6);
    }
}
```

The *String* Class: Methods

String Methods
`public char charAt(int index)`
Returns the character located in the specified *index*.
`public int compareTo(String anotherString)`
Compares this string with the specified parameter. Returns a negative value if this string comes lexicographically before the other string, 0 if both of the strings have the same value and a postive value if this string comes after the other string lexicographically.
`public int compareToIgnoreCase(String str)`
Like compareTo but ignores the case used in this string and the specified string.
`public boolean equals(Object anObject)`
Returns true if this string has the same sequence of characters as that of the *Object* specified, which should be a *String* object. Otherwise, if the specified parameter is not a *String* object or if it doesn't match the sequence of symbols in this string, the method will return false.
`public boolean equalsIgnoreCase(String anotherString)`
Like equals but ignores the case used in this string and the specified string.
`public void getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)`
Gets the characters from this string starting at the *srcBegin* index up to the *srcEnd* index and copies these characters to the *dst* array starting at the *dstBegin* index.

The *String* Class: Methods

String Methods
`public int length()`
Returns the length of this string.
`public String replace(char oldChar, char newChar)`
Returns the string wherein all occurrences of the *oldChar* in this string is replaced with *newChar*.
`public String substring(int beginIndex, int endIndex)`
Returns the substring of this string starting at the specified *beginIndex* index up to the *endIndex* index.
`public char[] toCharArray()`
Returns the character array equivalent of this string.
`public String trim()`
Returns a modified copy of this string wherein the leading and trailing white space are removed.
`public static String valueOf(-)`
Takes in a simple data type such as boolean, integer or character, or it takes in an object as a parameter and returns the *String* equivalent of the specified parameter.

The *String* Class: Example

```
class StringDemo {
   public static void main(String args[]) {
      String name = "Jonathan";
      System.out.println("name: " + name);
      System.out.println("3rd character of name: " +
                         name.charAt(2));
      /* character that first appears alphabetically
         has lower unicode value */
      System.out.println("Jonathan compared to Solomon: "
                         + name.compareTo("Solomon"));
      System.out.println("Solomon compared to Jonathan: "
                       + "Solomon".compareTo("Jonathan"));
//continued...
```

The *String* Class: Example

```
        /* 'J' has lower unicode value compared to 'j' */
        System.out.println("Jonathan compared to jonathan: " +
                                      name.compareTo("jonathan"));
        System.out.println("Jonathan compared to jonathan
     (ignore case): " + name.compareToIgnoreCase("jonathan"));
        System.out.println("Is Jonathan equal to Jonathan? " +
                                      name.equals("Jonathan"));
        System.out.println("Is Jonathan equal to jonathan? " +
                                      name.equals("jonathan"));
        System.out.println("Is Jonathan equal to jonathan
        (ignore case)? " + name.equalsIgnoreCase("jonathan"));
 //continued...
```

The *String* Class: Example

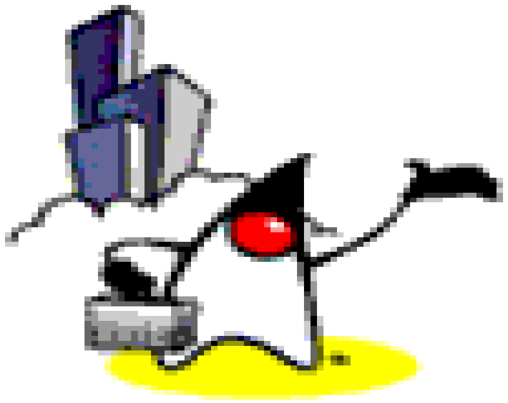
```java
     char charArr[] = "Hi XX".toCharArray();
     /* Need to add 1 to the endSrc index of getChars */
     "Jonathan".getChars(0, 2, charArr, 3);
     System.out.print("getChars method: ");
     System.out.println(charArr);
     System.out.println("Length of name: " +
                         name.length());
     System.out.println("Replace a's with e's in name: " +
                         name.replace('a', 'e'));
      // The substring begins at the specified beginIndex and
      // extends to the character at index endIndex - 1
     System.out.println("A substring of name: " +
                         name.substring(0, 2)); // Jo
//continued...
```

The *String* Class: Example

```
      System.out.println("Trim \"  a b c d e f   \": \"" +
                        "  a b c d e f   ".trim() + "\"");
      System.out.println("String representation of boolean
               expression 10>10: " + String.valueOf(10>10));
      /* toString method is implicitly called in the println
         method*/
      System.out.println("String representation of boolean
                  expression 10<10: " + (10<10));
      /* Note there's no change in the String object name
         even after applying all these methods because
         String is a final class. */
      System.out.println("name: " + name);
   }
}
```

Demo:

StringConstructorsDemo,
StringDemo,
StringObjectComparison,
StringLexComparison
1010_javase_lang.zip



StringBuffer Class

The *StringBuffer* Class

- Problem with *String* objects:
  - Once created, can no longer be modified (It is a *final* class)
    - The length and content of the String object cannot be changed
- A *StringBuffer* object
  - Similar to a String object
  - But, mutable or can be modified
    - Unlike *String* in this aspect
    - Length and content may be changed through some method calls

The *StringBuffer* Class: Methods

StringBuffer Methods
`public int capacity()`
Returns the current capacity of this *StringBuffer* object.
`public StringBuffer append(-)`
Appends the string representation of the argument to this *StringBuffer* object. Takes in a single parameter which may be of these data types: *boolean, char, char [], double, float, int, long, Object, String and StringBuffer*. Still has another overloaded version.
`public char charAt(int index)`
Returns the character located in the specified *index*.
`public void getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)`
Gets the characters from this object starting at the *srcBegin* index up to the *srcEnd* index and copies these characters to the *dst* array starting at the *dstBegin* index.
`public StringBuffer delete(int start, int end)`
Deletes the characters within the specified range.
`public StringBuffer insert(int offset, -)`
Inserts the string representation of the second argument at the specified offset. An overloaded method. Possible data types for the second argument: *boolean, char, char [], double, float, int, long, Object and String*. Still has another overloaded version.

The *StringBuffer* Class: Example

```
class StringBufferDemo {
   public static void main(String args[]) {
      StringBuffer sb = new StringBuffer("Jonathan");
      System.out.println("sb = " + sb);
      /* initial capacity is 16 */
      System.out.println("capacity of sb: "+sb.capacity());
      System.out.println("append \'O\' to sb: " +
                                    sb.append('O'));
      System.out.println("sb = " + sb);
      System.out.println("3rd character of sb: " +
                                    sb.charAt(2)); // n
//continued...
```

The *StringBuffer* Class: Example

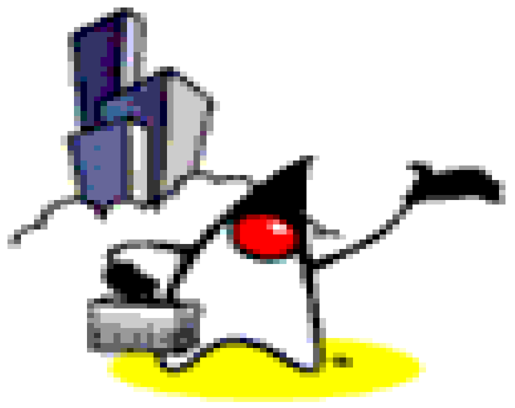
```
        char charArr[] = "Hi XX".toCharArray();
        /* Need to add 1 to the endSrc index of getChars */
        sb.getChars(0, 2, charArr, 3);
        System.out.print("getChars method: ");
        System.out.println(charArr);
        System.out.println("Insert \'jo\' at the 3rd cell: "
                                    + sb.insert(2, "jo"));
        System.out.println("Delete \'jo\' at the 3rd cell: "
                                    + sb.delete(2,4));
        System.out.println("length of sb: " + sb.length());
//continued
```

The *StringBuffer* Class: Example

```
        System.out.println("replace: " +
                            sb.replace(3, 9, " Ong"));
        /* Need to add 1 to the endIndex parameter of
           substring*/
        System.out.println("substring (1st two characters): "
                            + sb.substring(0, 3)); // Jon
        System.out.println("implicit toString(): " + sb);
    }
}
```

Demo:

StringBufferDemo
1010_javase_lang.zip



Wrapper Classes

The Wrapper Classes

- Some Facts:
  - Primitive data types are not objects
    - Cannot access methods of the *Object* class
  - Only actual objects can access methods of the *Object* class
  - Why wrapper classes?
    - Need an object representation for the primitive type variables to use Java built-in methods
- Definition:
  - Object representations of primitive types

The Wrapper Classes

- Boolean (Wrapper class), boolean (Primitive)
- Integer (Wrapper class), int (Primitive)
- Long (Wrapper class), long (Primitive)
- Double (Wrapper class), double (Primitive)

The Wrapper Classes: Boolean Example

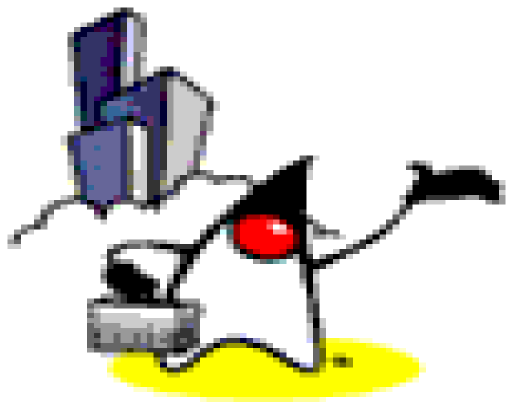
```
class BooleanWrapper {
   public static void main(String args[]) {
      boolean booleanVar = 1>2;
      Boolean booleanObj = new Boolean("True");
      Boolean booleanObj2 = new Boolean(booleanVar);
      System.out.println("booleanVar = " + booleanVar);
      System.out.println("booleanObj = " + booleanObj);
      System.out.println("booleanObj2 = " +
                                   booleanObj2);
//continued...
```

The Wrapper Classes: Boolean Example

```
        System.out.println("compare 2 wrapper objects: "
                           + booleanObj.equals(booleanObj2));
        /* object to primitive */
        booleanVar = booleanObj.booleanValue();
        System.out.println("booleanVar = " + booleanVar);
    }
}
```

Demo:

WrapperClassExamples
1010_javase_lang.zip



Math Class

The *Math* Class: Example

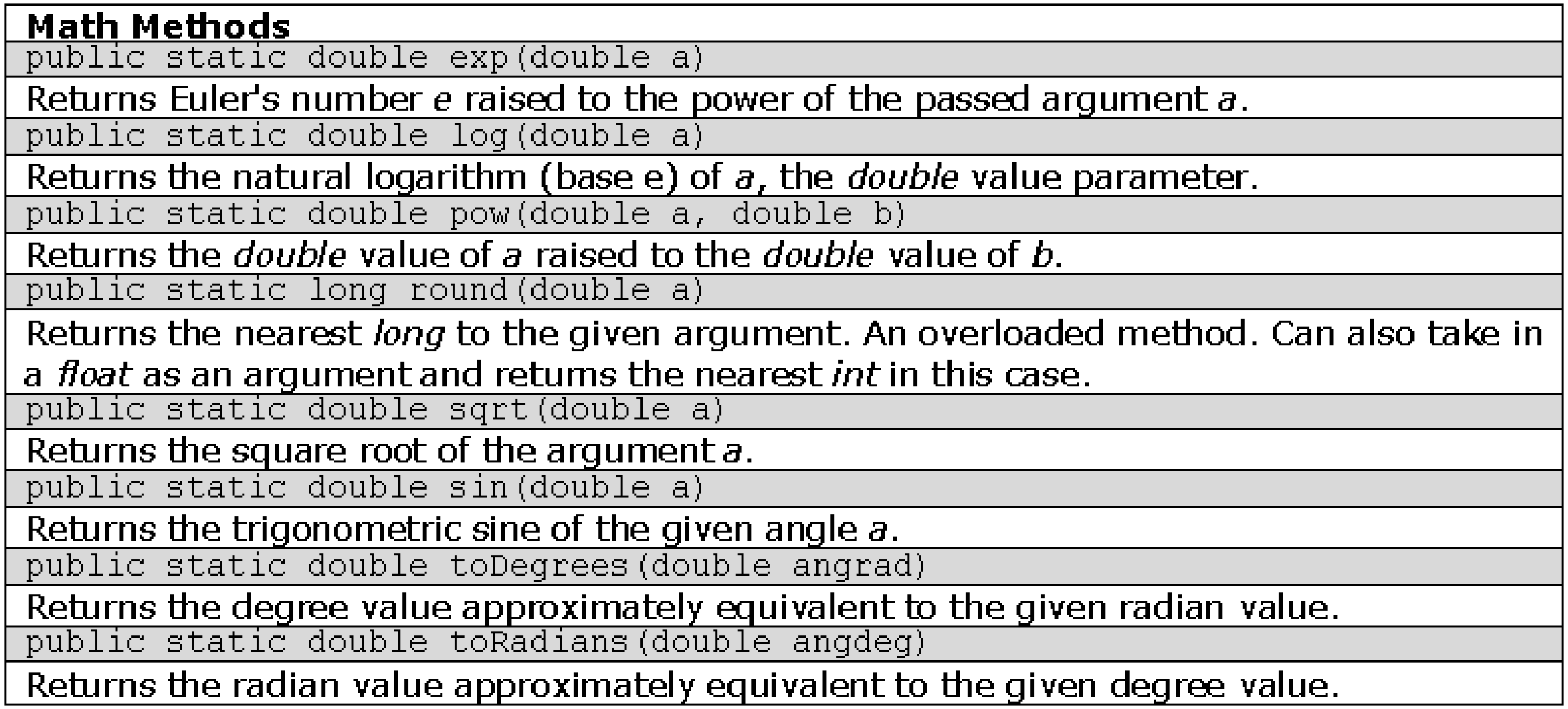
```
        System.out.println("rounded off value of pi: " +
                                    Math.round(Math.PI));
        System.out.println("square root of 5 = " +
                                    Math.sqrt(5));
        System.out.println("10 radian = " +
                          Math.toDegrees(10) + " degrees");
        System.out.println("sin(90): " +
                          Math.sin(Math.toRadians(90)));
    }
}
```

The *Math* Class

- Provides predefined constants and methods for performing different mathematical operations
- Methods: Note that they are all static methods

Math Methods
`public static double abs(double a)`
Returns the positive value of the parameter. An overloaded method. Can also take in a float or an integer or a long integer as a parameter, in which case the return type is either a float or an integer or a long integer, respectively.
`public static double random()`
Returns a random postive value greater than or equal to 0.0 but less than 1.0.
`public static double max(double a, double b)`
Returns the larger value between two *double* values, *a* and *b*. An overloaded method. Can also take in float or integer or long integer values as parameters, in which case the return type is either a float or an integer or a long integer, respectively.
`public static double min(double a, double b)`
Returns the smaller value between two *double* values, *a* and *b*. An overloaded method. Can also take in float or integer or long integer values as parameters, in which case the return type is either a float or an integer or a long integer, respectively.
`public static double ceil(double a)`
Returns the smallest integer that is greater than or equal to the specified parameter *a*.
`public static double floor(double a)`
Returns the largest integer that is lesser than or equal to the specified parameter *a*.

The *Math* Class: Methods



Math Methods
`public static double exp(double a)`
Returns Euler's number *e* raised to the power of the passed argument *a*.
`public static double log(double a)`
Returns the natural logarithm (base e) of *a*, the *double* value parameter.
`public static double pow(double a, double b)`
Returns the *double* value of *a* raised to the *double* value of *b*.
`public static long round(double a)`
Returns the nearest *long* to the given argument. An overloaded method. Can also take in a *float* as an argument and returns the nearest *int* in this case.
`public static double sqrt(double a)`
Returns the square root of the argument *a*.
`public static double sin(double a)`
Returns the trigonometric sine of the given angle *a*.
`public static double toDegrees(double angrad)`
Returns the degree value approximately equivalent to the given radian value.
`public static double toRadians(double angdeg)`
Returns the radian value approximately equivalent to the given degree value.

The *Math* Class: Example

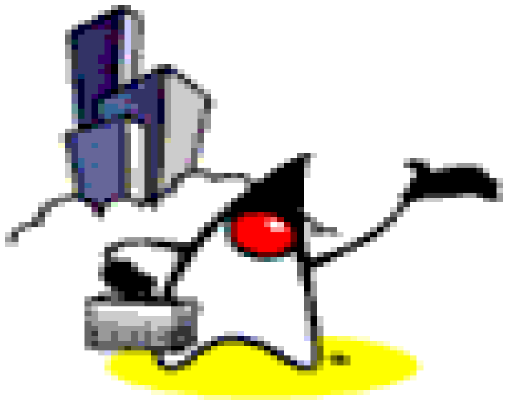
```
class MathDemo {
   public static void main(St ring args[]) {
      System.out.println("absolute value of -5: " +
                                  Math.abs(-5));
      System.out.println("absolute value of 5: " +
                                  Math.abs(5));
      System.out.println("random number(max is 10): " +
                                     Math.random()*10);
      System.out.println("max of 3.5 and 1.2: " +
                                    Math.max(3.5,1.2));
      System.out.println("min of 3.5 and 1.2: " +
                                    Math.min(3.5,1.2));
//continued...
```

The *Math* Class: Example

```
        System.out.println("ceiling of 3.5: " +
                                   Math.ceil(3.5));  // 4.0
        System.out.println("floor of 3.5: " +
                                   Math.floor(3.5)); // 3.0
        System.out.println("e raised to 1: " +
                                   Math.exp(1));
        System.out.println("log 10: " +
                                   Math.log(10));
        System.out.println("10 raised to 3: " +
                                   Math.pow(10,3));
//continued...
```

Demo:

MathDemo
1010_javase_lang.zip



System Class

The *System* Class

- Provides many useful fields and methods
  - Standard input
  - Standard output
  - Utility method for fast copying of a part of an array

The *System* Class: Methods

System Methods
`public static void arraycopy(Object src, int srcPos, Object dest, int destPos, int length)`
Copies *length* items from the source array *src* starting at *srcPos* to *dest* starting at index *destPos*. Faster than manually programming the code for this yourself.
`public static long currentTimeMillis()`
Returns the difference between the current time and January 1, 1970 UTC. Time returned is measured in milliseconds.
`public static void exit(int status)`
Kills the Java Virtual Machine (JVM) running currently. A non-zero value for status by convention indicates an abnormal exit.
`public static void gc()`
Runs the garbage collector, which reclaims unused memory space for recycling.
`public static void setIn(InputStream in)`
Changes the stream associated with *System.in*, which by default refers to the keyboard.
`public static void setOut(PrintStream out)`
Changes the stream associated with *System.out*, which by default refers to the console.

The *System* Class: Example

```

        startTime = System.currentTimeMillis();
        for (int i = 0; i < arr1.length; i++) {
           arr2[i] = arr1[i];
        }
        endTime = System.currentTimeMillis();
        System.out.println("Time for manual copy: " +
                                (endTime-startTime) + " ms.");
//continued...
```

The *System* Class: Example

```

        startTime = System.currentTimeMillis();
        System.arraycopy(arr1, 0, arr2, 0, arr1.length);
        endTime = System.currentTimeMillis();
        System.out.println("Time for manual copy: " +
                          (endTime-startTime) + " ms.");
        System.gc();   //force garbage collector to work
        System.setIn(new FileInputStream("temp.txt"));
        System.exit(0);
    }
}
```

System Properties

java.version	Java Runtime Environment version
java.vendor	Java Runtime Environment vendor
java.vendor.url	Java vendor URL
java.home	Java installation directory
java.vm.specification.version	Java Virtual Machine specification version
java.vm.specification.vendor	Java Virtual Machine specification vendor
java.vm.specification.name	Java Virtual Machine specification name
java.vm.version	Java Virtual Machine implementation version
java.vm.vendor	Java Virtual Machine implementation vendor
java.vm.name	Java Virtual Machine implementation name
java.specification.version	Java Runtime Environment specification version
java.specification.vendor	Java Runtime Environment specification vendor
java.specification.name	Java Runtime Environment specification name

System Properties

java.class.version	Java class format version number
java.class.path	Java class path
java.library.path	List of paths to search when loading libraries
java.io.tmpdir	Default temp file path
java.compiler	Name of JIT compiler to use
java.ext.dirs	Path of extension directory or directories
os.name	Operating system name
os.arch	Operating system architecture
os.version	Operating system version
file.separator	File separator ("/" on UNIX)
path.separator	Path separator (":" on UNIX)
line.separator	Line separator ("\n" on UNIX)
user.name	User's account name
user.home	User's home directory
user.dir	User's current working directory

Example: Display System Properties

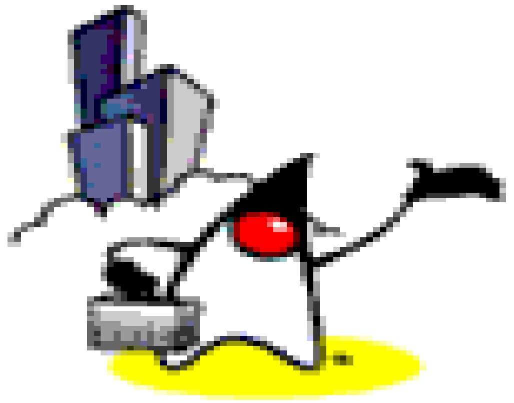
```
public static void main(String[] args) {
    Properties p1 = System.getProperties();
    p1.list(System.out);
}
```

Example: Display System Properties

```
java.runtime.name=Java(TM) 2 Runtime Environment, Stand...
sun.boot.library.path=C:\Program Files\Java\jdk1.5.0_06\jre...
java.vm.version=1.5.0_06-b05
java.vm.vendor=Sun Microsystems Inc.
java.vendor.url=http://java.sun.com/
path.separator=;
java.vm.name=Java HotSpot(TM) Client VM
file.encoding.pkg=sun.io
user.country=US
sun.os.patch.level=Service Pack 2
java.vm.specification.name=Java Virtual Machine Specification
user.dir=C:\handson2\development\javalang\samp...
java.runtime.version=1.5.0_06-b05
java.awt.graphicsenv=sun.awt.Win32GraphicsEnvironment
java.endorsed.dirs=C:\Program Files\Java\jdk1.5.0_06\jre...
...
```

Demo:

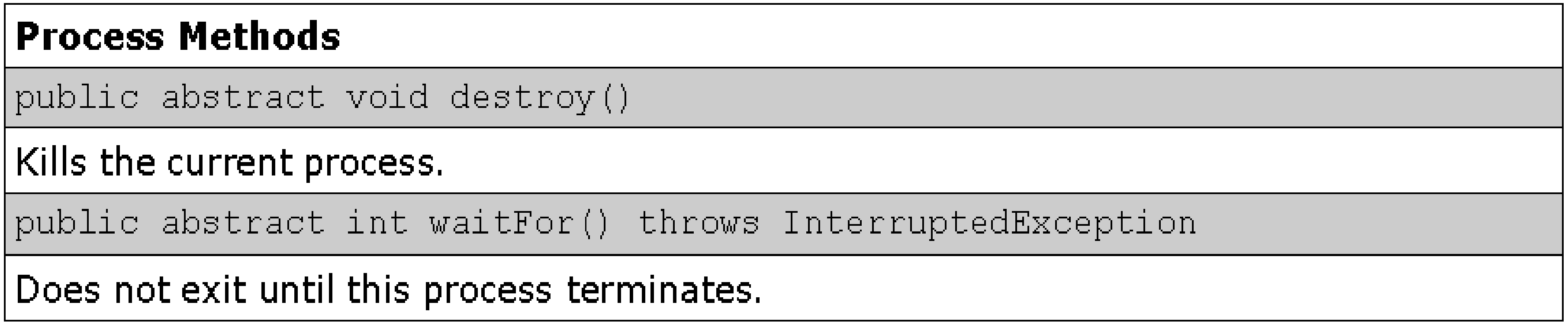
SystemClass, SystemClass2, SetSystemProperties 1010_javase_lang.zip



Process & Runtime Classes

The *Process* Class

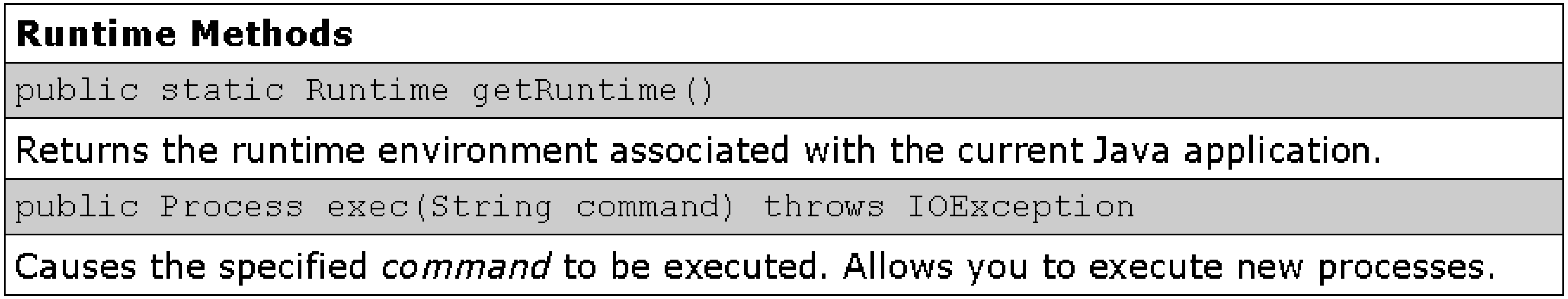
- Definition:
  - Provides methods for manipulating processes
    - Killing the process
    - Running the process
    - Checking the status of the process
  - Represents running programs
- Methods:



Process Methods
`public abstract void destroy()`
Kills the current process.
`public abstract int waitFor() throws InterruptedException`
Does not exit until this process terminates.

The *Runtime* Class

- Represents the runtime environment
- Has two important methods:



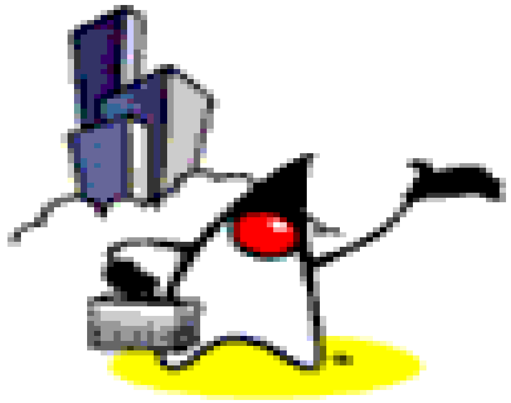
Runtime Methods
`public static Runtime getRuntime()`
Returns the runtime environment associated with the current Java application.
`public Process exec(String command) throws IOException`
Causes the specified *command* to be executed. Allows you to execute new processes.

The *Process* and *Runtime* Class: Example

```
class RuntimeDemo {
   public static void main(String args[]) {
      Runtime rt = Runtime.getRuntime();
      Process proc;
      try {
         proc = rt.exec("regedit");
         proc.waitFor();   //try removing this line
      } catch (Exception e) {
         System.out.println("regedit is an unknown
                                    command.");
      }
   }
}
```

Demo:

Runtime_exec,
Runtime_Memory,
Runtime_getRuntime
1010_javase_lang.zip



ProcessBuilder Class

ProcessBuilder Class

- Before JDK 5.0, the only way to fork off a process and execute it local to the user runtime was to use the exec() method of the *java.lang.Runtime* class.
- JDK 5.0 adds a new way of executing a command in a separate process, through a class called *ProcessBuilder*

Why ProcessBuilder Class?

- Runtime.exec approach doesn't necessarily make it easy to customize and invoke subprocesses.
- The new *ProcessBuilder* class simplifies things. Through various methods in the class, you can easily modify the environment variables for a process and start the process.

Streams

- All its standard io (i.e. stdin, stdout, stderr) operations will be redirected to the parent process through three streams (getOutputStream(), getInputStream(), getErrorStream()).
- The parent process uses these streams to feed input to and get output from the subprocess.

Example: Runtime.exec

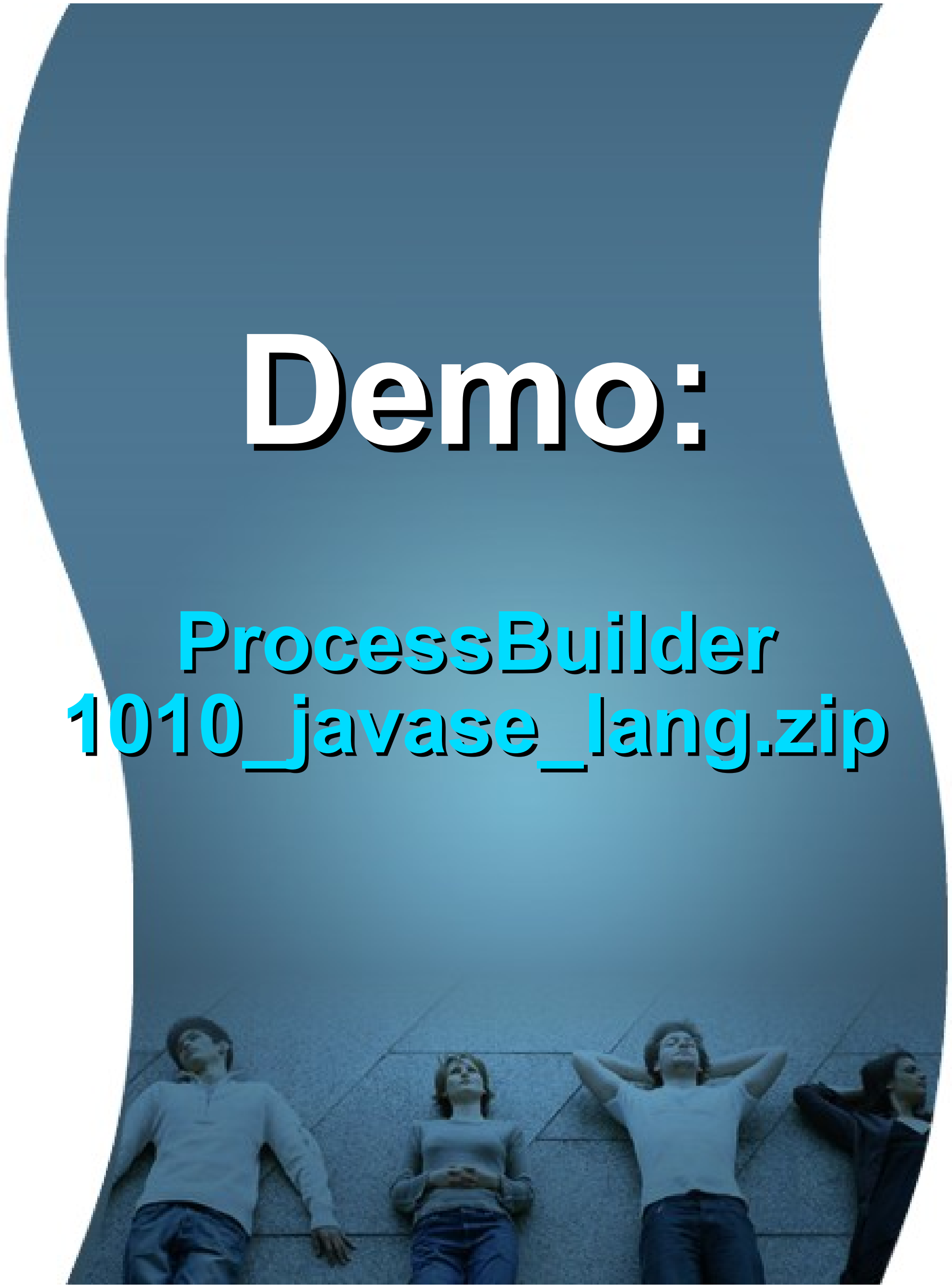
```
String[] commandArray = {"javap", "-private",
    "java.lang.String"};
String[] environment = {"path=;",
    "path=C:\\Java\\jdk1.6.0_18\\bin;"};

Runtime runtime = Runtime.getRuntime();
Process process = runtime.exec(commandArray, environment);
```

Example: ProcessBuilder Class

```
ProcessBuilder processBuilder = new ProcessBuilder("javap", "-private",
        "java.lang.String");

// The options for manipulating the environment include adding environment
// variables with the put() method, and removing them with the remove()
// method.
Map<String, String> environment = processBuilder.environment();

environment.put("path", ";"); // Clearing the path variable;
environment.put("path", "C:\\Java\\jdk1.6.0_18\\bin;");

// If you want the process to start in a different directory, you don't
// pass a File in as a command line argument. Instead, you set the process
// processBuilder's working directory by passing the File to the directory()
// method:
processBuilder.directory(new File("/tmp"));

// With ProcessBuilder, you call start() to execute the command. Prior
// to calling start(), you can manipulate how the Process will be created.
Process process = processBuilder.start();
```

Example: ProcessBuilder Class

```
// Capture Output through process.getOutputStream()
InputStreamReader tempReader = new InputStreamReader(
            new BufferedInputStream(process.getOutputStream()));

// Capture error through process.getErrorStream()
InputStreamReader tempReader = new InputStreamReader(
            new BufferedInputStream(process.getErrorStream()));
```



Demo:

ProcessBuilder
1010_javase_lang.zip

Thank you!

Check JPassion.com Codecamps!
http://www.javapassion.com/codecamps
“Learn with JPassion!”