

# Strings

# Is string an array of char?

- NO! String is implement as CLASS OBJECT
- String class vs. StringBuffer class

# Java syntax for String

## ■ Declaration

- `String name;`

## ■ Allocation

- `name = new String("John Doe");` // just like any other objects
- `name = "John Doe";` // special case
- `name = new String( ... );` // a lot of other ways to construct a string

## ■ Using String

# String Methods ...

- Length (not the same as array)
- Character extraction
- Concatenation
- Substring extraction
- Equality comparison
- Conversion to different case
- Notes on String as method argument

# String Methods ... Sample

- **Length** –Returns the length of the string.
  - `int nameLength = fullName.length();`
- **Character extraction** - Extracts a single character from a string.
  - `char fourthLetter = fullName.charAt( 3 );`
- **Conversion to different case** - Converts all the characters to either upper or lower case.
  - `anotherName = anotherName.toUpperCase();`

# String Methods ... Sample

- **Equality comparison** –Compares two strings for equality.
  - `String anotherName = "Joan of Arc";`
  - `if ( anotherName.equals( fullName )...`
- **Equality comparison ignoring case** –Compares two strings for equality while ignoring differences in case.
  - `if ( name.equalsIgnoresCase( "Paul Bunyon" )...`
- **Lexicographic comparison** –Compare the lexicographic ordering of two strings. Returns  $<0$  if name before otherName,  $=0$  if identical,  $>0$  if name after otherName.
  - `if ( name.compareTo( otherName ) < 0 )...`

# String Methods Sample

## ■ Concatenation

- `String firstName = "Joan";`
- `String lastName = "of Arc";`
- `String fullName = firstName + lastName;`

## ■ Substring extraction

- Extract a substring from a string. Characters within strings are numbered starting with zero, so the position of the first character is 0. The extraction takes the position of the first character to extract up to but not including the second position given:
  - `String alphabet = "abcdefghijklmnopqrstuvwxy";`
  - `String initialLetters = alphabet.substring(0, 3); //abc`
  - `String finalLetters = alphabet.substring(23, 25); //xy`

# String is a reference

- **A String variable is a reference to a string.** A String variable might refer to different strings at different points in the program:
  - `String studentName = "Jane Doe";`
  - `String graduateName = "";`
  - `graduateName = studentName;`
  - `studentName = "John Smith";`
  - `graduateName = "Paul Jones";`

# String

- **Strings are not a primitive type** in Java. Strings are a class type defined in the `java.lang` package. String constants are expressed using quotation marks and *are* a built-in part of the language.
- **Strings are immutable.** Individual characters in a string cannot be changed. A new string can be created from an existing string through concatenation, substring or uppercase, but the original string remains unchanged.
- If a variable of a primitive type is concatenated with a String, the result is a String. This is common in input/output operations:
  - `System.out.println( "Integer value: " + anInt );`

# Immutable, so what?

- The String methods do not alter the String after construction. Methods like toUpperCase and substring return a new string!
- Strings can be allocated with exactly enough character locations at construction. They **do not need to be “expandable.”**
- If String contents must be changed, a new String reference must be constructed. This may be inefficient for certain kinds of problems!
- Strings can be safely **shared** since the contents cannot be changed. Accessor and mutator methods may accept or return String references without violating privacy.

# Java syntax for StringBuffer

- Declaration

- `StringBuffer buffer;`

- Allocation

- `buffer = new StringBuffer("John Doe"); // just like any other objects`

- `buffer = new StringBuffer(); // empty buffer`

- `buffer = new StringBuffer(anotherStringOrStringBuffer);`

# StringBuffer

- There are a set of operations to work with StringBuffer:
  - Get the length of a buffer
  - Examine individual character
  - Set an individual character
  - Convert to string
  - Append string or string buffer to end of buffer
  - Insert a substring
  - Delete a substring

# StringBuffer methods

- **Length** –Returns the length of the StringBuffer.
  - `int nameLength = fullName.length();`
- **Examine character** –Examine an individual character in the StringBuffer
  - `if ( anotherName.charAt( 0 ) == 'A' )...`
- **Set a character** –Change an individual character in the StringBuffer
  - `anotherName.setCharAt( 0, 'B' );`
- **Convert to string** –Convert a StringBuffer to a string
  - `aString = anotherName.toString();`

# StringBuffer methods

- **Append** –Appends to the end of the StringBuffer.
  - `anotherName.append( “, esq.” );`
  - `output.append( 12 ); // appends the string!`
  - `output.append( numPeanuts );`
  - `output.append( isRaining ); // boolean string...`
  - StringBuffer append can be used to implement String concatenation
- **Insert** –Insert a substring into a StringBuffer
  - `name.insert(7, “M.” );`

# StringBuffer methods

- **Set a minimum starting capacity.** –Allocates a minimum capacity for the StringBuffer.
  - `StringBuffer inputLine = new StringBuffer( "" );`
  - `inputLine.ensureCapacity( 80 );`
  - Use if the anticipated actual size of the StringBuffer will be larger than the default 16.
- **Set a specified length.** –*Force* the string to a specific length (truncates or pads with null).
  - `StringBuffer aShortName = new StringBuffer( "Joe" );`
  - `StringBuffer aLongName = new StringBuffer( "Milton Freewater" );`
  - `aShortName.setLength( 6 ); aLongName.setLength( 6 );`
  - `System.out.println( "Long: " + aLongName ); System.out.println( "Short: " + aShortName.length() + " " + aShortName );`

# StringTokenizer

- The StringTokenizer is useful for parsing statements into individual tokens, as in this sample:
  - `import java.util.*;`
  - `StringTokenizer textTokens =  
new StringTokenizer( someString );`
  - `String token;`
  - `int tokenCount = 0;`
  - `int lengthCount = 0;`
  - `while ( textTokens.hasMoreTokens() ) {`
    - `token = textTokens.nextToken();`
    - `++tokenCount;`
    - `lengthCount += token.length();`
  - `}`
  - `System.out.println( "" + lengthCount + " " + tokenCount );`

# Conversion of String and other primitive data type

- Using Data Type Wrapper classes
  - Boolean
  - Byte
  - Short
  - Character
  - Integer
  - Long
  - Float
  - Double

# Common methods in wrapper classes

- `String toString()`
- `boolean equals(Object obj)`
- `static <data_Type>Value()` or `valueOf(String s)`
  - `Integer i = new Integer(17);`
  - `float f = i.floatValue();`
  - `String s = i.toString();`