Characters and Strings

James Brucker

Characters

- Java uses Unicode to represent characters.

- **`char`** is 2-bytes.

- Java *does* *<u>not</u>* use Unicode for reading/writing to files unless you tell it to.

```
char c = 'A';
char at = '@';
char gai = 'ก';   // Thai letter
char z = 'a'+25;  // 'z', increment using int
int  n = 'A';     // n = 65
```

Useful Character Methods

c is a char:

boolean Character.**isLetter**(c)

boolean Character.**isDigit**(c)

boolean Character.**isLetterOrDigit**(c)

boolean Character.**isLowerCase**(c)

char   Character.**toLowerCase**(c)

char   Character.**toUpperCase**(c)

boolean Character.**isWhitespace**(c)

int   Character.**getNumericValue**('3') --> 3

Useful Character Methods

```
String message = "I am trapped in a computer";
char [ ] c = message.toCharArray( );
// c[0]='I', c[1]=' ', c[2]='a', c[3]='m' ...
// count the letters and digits
int letters = 0;
int digits = 0
for(int k=0; k < c.length; k++ ) {
    /* c[k] is letter */
    if ( Character.isLetter(c[k]) ) letters++;
    /* c[k] is a digit 0 ... 9 */
    else if ( Character.isDigit(c[k]) ) digits++;
    else /* something else */;
}
```

Useful String Methods

Useful for Many Text Processing Applications

including the Caesar Cipher problem

Useful String Methods

string.indexOf(char) returns the position of char in the String.

-1 if char is not found in string.

```
String ALPHABET = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
char c = 'G';
int n = ALPHABET.indexOf( c ); // = 7
```

string.length( ) returns the length of the String.

```
String ALPHABET = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
int n = ALPHABET.length( ); // = 26
```

string.charAt(k) returns the character at position k (starts at k = 0)

```
String ALPHABET = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
ALPHABET.charAt( 3 ); // = 'd'
```

Useful String Comparisons

a.equalsIgnoreCase(b) returns true if a and b have same value, ignoring case of letters.

```
String answer = console.next();

if (answer.equalsIgnoreCase("yes")) ...
```

a.compareTo(b) lexical comparison of strings

```
if ("cat".compareTo("dog") < 0)
    print("cat comes before dog");
```

a.compareToIgnoreCase(b) like compareTo ignoring case of letters

```
if ("cat".compareToIgnoreCase("DOG") < 0)
    print("cat comes before DOG");
```

String toCharArray

Problem:

You want to process each character in a string.

Solution:

convert String to a character array

process each character

make a new string from the result

```java
String message = "I am trapped in a computer";
// convert String to array of char
char [] c = message.toCharArray( );
// process each character:
for(k=0; k< c.length; k++) c[k] = ...;
// put the result in a new String
String result = new String( c );
```

How to Create a Formatted String

☐ You can format output using:

`System.out.printf(` "*format string*", arg1, arg2, ...);

☐ Example:

`System.out.printf("p = (%5.2f,%5.2f)\n", x, y);`

for x = 12.345, y = 11.519, this prints:

`p = (12.34,11.52)`

☐ System.out.printf( ) is an alias for System.out.format( ) which uses the java.util.Formatter class to format output.

☐ See Javadoc for Formatter for complete list of format codes and examples.

Using Formatter Objects

☐ Suppose we have a Point class for 2-dimensional points. We want toString( ) to return a nicely formatted "(x,y)" for the point's coordinates.

```java
private double x, y;   // coordinates of the point

Formatter result = new Formatter( );
// use the format( ) method like printf( )
result.format("(%.2f, %.2f)", x, y );
// convert result to a String and return it
String s = result.out( ).toString( );
```

```java
// Easier

String s = String.format("(%.2f, %.2f)", x, y);
```

Formatter Methods

`format("`*`format string`*`", obj1, ...)`

format the objects using the format string and *append* to the format object's *Appendable* attribute.

`out( )`

return the contents of the Formatter object as an *Appendable* object.

`out( ).toString( )`

convert the contents of the Formatter object to a String.

Appendable is a Java 1.5 interface for character sequences that can be appended to. Classes the implement *Appendable* are StringBuffer, StringBuilder, CharBuffer, PrintStream, StringWriter, ...

The **String** class is **not** *Appendable* ! *(Strings are <u>immutable</u>.)*

String Types

There is more than one way to store a String.

If you want to append to or modify a String, use a StringBuilder or StringBuffer.

String is Immutable

You can't change a String after it is created.

```
String s = "hello";

s = s + " there";

        // "+" creates a new string.  Old string is now garbage.

s = s + " class";

        // "+" creates another new String. Old string is garbage.

s = s.toLowerCase( );

        // creates yet another new String
```

Lesson: using "+" to build strings is **slow** and **inefficient**.

Can be an issue in web apps that create HTML as strings.

StringBuffer is a mutable String

- StringBuffer and StringBuilder are two classes that "build" strings.
- You can modify and append to them.

```
StringBuilder sb = new StringBuilder( );

sb.append("hello");   // append to same buffer

sb.append(" there"); // doesn't create new objects

sb.append(" class");

// now we are done. Convert to a String for output or return

String s = sb.toString;
```

Exercise: compare run times

1. Choose a text file of size 10KB - 100KB.

2. Open the file as a FileInputStream.

3. Time how long it takes to read file into a String:

   a) Read the file 1 byte at a time using inputStream.read( )

   b) Append each byte to a String

4. Display the time required and the length of String.

Exercise: part 2

1. Repeat previous steps using StringBuilder instead of String.

2. Append each byte using sb.append( )

StringBuffer or StringBuilder?

- StringBuilder and StringBuffer are nearly the same.

- StringBuffer is *thread safe*, StringBuilder is not.

- Being "thread safe" makes StringBuffer *slightly slower*.

- For a single threaded application, prefer StringBuilder.