CS415

INTRODUCTION TO COMPUTER SCIENCE

FALL 2018

14 STRINGS AND TEXT I/O

CHAPTER 17

PREVIEW

- Strings
  - Concatenation
  - String methods
- Text I/O

STRINGS

- Most programming languages provide a way of representing text.
- Basically text is anything that a user can enter from the keyboard.
- Java's String class represents text.

THE COMMAND LINE

- As we have seen, the command line is available in the main method as an array of Strings.
- And, `System.out.println( String s )` will display its String argument on the system console.

```
public static void main( String [ ] args )
{
      if( args.length > 0 )
          System.out.println( args[ 0 ] );
```

STRING LITERALS

- String literals are written in a Java program as a sequence of characters delimited by double quotes. They are implemented as instances of String.

```
System.out.println( "Go Cats!" );

"move to 20 30"    "7"    "!"
```

- Assigning String literals to String variables.

```
private final String emptyString = "";

String blank;
blank = " ";

String message;
message = "Greetings Earthling";
System.out.println( message );
```


ESCAPE CHARACTERS

- Escape characters are used inside String literals to allow print formatting and to prevent certain characters from causing interpretation errors.

```
System.out.println( "Go\nCats!" );
```

```
Go
Cats!
```

```
System.out.println( "Tab\tMe" );
```

```
tab    Me
```

```
System.out.println( "We call \'\\\' a \"backslash\"");
```

```
We call '\' a "backslash"
```

Escape Character	Meaning
\n	newline
\t	tab
\"	double quote
\'	single quote
\\	backslash

STRING CONCATENATION

- When the + operator is used between two Strings the expression will evaluate to the concatenation of the two Strings.

```
String cheer = "Go " + "Cats! ";

String rahRah = cheer + "\n" + cheer;

String rallyCheer = "";
for( int i = 0; i < 20; i++ )
    rallyCheer += cheer;
System.out.println( rallyCheer );
```

```
Go Cats! Go Cats! Go Cats! Go Cats! Go Cats! Go Cats! Go Cats! Go
```

STRING CONCATENATION

- When the + operator is used with a String and a value of another type, the value is converted to a String.

```
float weight = 12.2;
String display;
display = weight + " grams";
System.out.println( display );
```

```
12.2 grams
```

- Class types are converted using their `toString()` method (inherited from Object or overridden.)

```
String str = "" + java.awt.Color.RED;
System.out.println( str );
```

```
java.awt.Color[r=255,g=0,b=0]
```

TOSTRING

- The method `System.out.println( ... )` accepts a value of any type and converts it to a String.

```
System.out.println( 17 );
System.out.println( myRover );
```

- For your own classes, it's very useful (especially for debugging) to write a `toString()` method; this will automatically be called whenever an object needs to be coerced to a String (as in a System.out arg)

```
public String toString( )
{
   String str = "Rover @ " + getLocation();
   return str;
}
```


STRING METHODS

- Length

```
String title = "CS 415";
System.out.println( "Length: " + title.length());
```

```
Length: 6
```

```
int n = "ab c ".length( );     //  n  is 5
```

- Creating new strings from old

```
String a = " xy z ", b;
// trim: remove leading and trailing blanks
b = a.trim();                  //  b is "xy z"
// toUpperCase: make all upper case; also toLowerCase()
b = a.toUpperCase();           //  b is " XY Z "
// replace: substitute matching substrings with another
b = a.replace( "z", "abc" );   //  b is " xy abc "
```


STRING COMPARISON METHODS

- boolean equals(String)
  are values of two strings equal?
- boolean equalsIgnoreCase(String)
  are the values the same except for case differences
- int compareTo(String)
  is one string less than, equal to, or greater than another
- boolean startsWith(String)
  does a string start with the specified string

STRING EQUALS METHODS

- equals and equalsIgnoreCase
  - alphabetic string comparison
  - Examples

```
String str = "abc";

str.equals( "ABC".toLowerCase() )   // true

str.equals( "aBc" )          // false

str.equalsIgnoreCase( "aBc" )        // true
```

== OPERATOR

- true only if 2 operands reference the same object

```
String s1, s2, s3;
s1 = "abc";
s2 = "abc"
s3 = s1;
s1 == s2     // false
s1 == s3     // true
```

- Usually,(always?) don't want ==, but the equals method

STRING COMPARETO METHOD

- int compareTo(String)

returns <0 if object value is < argument value
returns 0 if object value is the same as argument value
returns >0 if object value is > argument value

```
String s1, s2;
. . . // s1 and s2 acquire some values

int diff = s1.compareTo( s2 );

if ( diff < 0 )
    //  s1 < s2
else if ( diff == 0 )
    // s1 and s2 have the same value
else
    // s1 > s2
```


STRING STARTSWITH METHOD

- boolean startsWith(String)
  - true if the argument string matches the start of the object string

```
String s1 = "abcdef" ;
s1.startsWith( "a" )              // true
s1.startsWith( "abc" )           // true
s1.startsWith( "abcdefg" )    // false
```


SUBSTRING METHODS

- Substrings can be extracted from Strings. As in arrays, indexing begins at 0.

```
String str = "abcdefghij";

// substring starting at 3 to the end of the string
System.out.println( str.substring( 3 ) );
```

defghij

```
// substring starting at 3 ending just before 7
System.out.println( str.substring( 3, 7 ) );
```

defg

```
System.out.println( str.substring( 3, 11 ) );
```

StringIndexOutOfBoundsException:

String search methods

- indexOf and lastIndexOf: return index of substring in a string, if it is there

```
String str = "abracadabra";
str.indexOf( "ra" )          //  2
str.indexOf( "ra", 3 )       //  9  ( first occurrence
                             //    on or after index 3 )
str.lastIndexOf( "ab" )      //  7

str.indexOf( "abc" )         // -1, "abc" is not in str
```

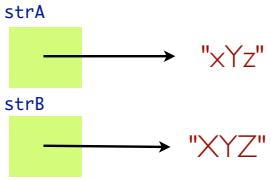
- contains: does the argument exist in the object

```
str.contains( "dabra" )      // true
```


STRINGS ARE "READ ONLY"

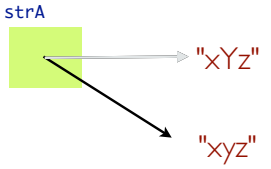
- Strings are immutable, once they are created they cannot be changed.
- String methods do not *change* a String, but some produce a new String.

```
String strA = "xYz", strB;
strB = strA.toUpperCase();
```



- Of course, a variable reference to a String can be changed:

```
strA = strA.toLowerCase( );
```



WRITING STRING METHODS

- Read the Java String API carefully to see the full list of String Methods.
- You may want to write your own String methods
  - Reverse a string:

```
public static String reverse ( String text )
{
    int length = text.length( );
    String reverse = "";

    for ( int i = length; i > 0; i-- )
         reverse += text.substring( i - 1, i );

    return reverse;
}
```


PRIMITIVE TYPE CHAR

- Java has a primitive type char to represent single characters.

```
char ch;
```

- A char literal is written with **single** quotes

```
ch = 'a';
char newLine = '\n';
char space = ' ';
```

- Two char values can be compared with the relational and equality operators

```
if( ch == 'a' )
    ....
```

STRINGS AND CHAR

- A char can be added to a String:

```
String str = "Hello"
char bang = '!';

str = str + bang;
```

- A char is returned by the String method charAt(int i):

```
char initial = str.charAt( 0 );
```

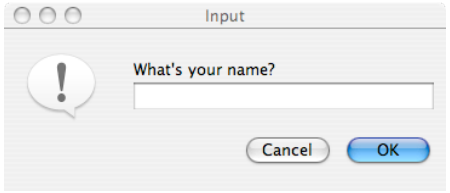
- A char can be converted to a String of length 1 using Character.toString(char ch):

```
char ch = 'J';
String str =   Character.toString(  ch  );
```

- For simple operations it might be easier to deal with a char, for more complicated processing it might be better to convert it to a String and use the rich collection of String methods.

DIALOG BOX TEXT INPUT

- InputDialog.




```
import javax.swing.*;


String name;
name = JOptionPane.showInputDialog( null, "What's your name?" );
```


DIALOG BOX TEXT OUTPUT

- MessageDialog.


```
import javax.swing.*;

String name;
name = JOptionPane.showInputDialog( null, "What's your name?" );

JOptionPane.showMessageDialog( null, "Hi, " + name );
```

CONSOLE TEXT I/O

- *System.out* is an object of type *PrintStream* representing the system's "standard output stream" (the monitor).

```
System.out.println(  " Hi!" );
```

- *System.in* is an object of type *InputStream* representing the system's "standard input stream" (the keyboard).
- We don't use *System.in* directly, we use other classes to make it easier.

SCANNER

```
import java.util.Scanner;
```

- Create a *Scanner* object based on the standard input stream:

```
Scanner keyBoard = new Scanner(System.in);
```

- Read and echo the next line from the input:

```
String input;

input = keyBoard.nextLine( );

System.out.println( input );
```


MORE SCANNER

- A *Scanner* can also read (and convert) numbers:
  Scanner works with "tokens"
  A token is a sequence of non-blank characters ending in "white space" (blank, tab, end-of-line)
  - If the next token in the input stream is a valid integer, read it, convert it to an *int*, and return it

```
Scanner keyBoard = new Scanner(System.in);
int anInt;
if ( keyBoard.hasNextInt() )
   anInt = keyBoard.nextInt();
```


SCANNER FROM A STRING

- A scanner can also be created from a string:

```
String data = " abc\n  def\n  17 23.4   end\n";

Scanner dataScanner = new Scanner( data );
System.out.println( dataScanner.nextLine() );
System.out.println( dataScanner.nextLine() );

if ( dataScanner.hasNextInt() )
   System.out.println( "Got int: " +
                       dataScanner.nextInt() );
if ( dataScanner.hasNextFloat() )
   System.out.println( "Got float: " +
                       dataScanner.nextFloat() );
```

```
 abc
  def
Got int: 17
Got float: 23.4
```


SCANNER METHODS

Method	Returns
hasNextLine()	true if there is a next line
nextLine()	next line
hasNext()	true if there is a next token
next()	next space delimited token
hasNextInt()	true if the next token is an int
nextInt()	next token converted to an int
nextDouble()	etc...

USING SCANNER METHODS

```
// read and print one input token per line
//    until user types "quit"

Scanner s = null;
String input;
s = new Scanner( System.in );

while ( s.hasNext() )   // is there another token?
{
   input = s.next();    // read the token
   System.out.println( input );  // print it
   if ( input.equalsIgnoreCase( "quit" ))
       break;
}
```


PARSING A COMMAND WITH SCANNER

```
run 25 40
quit
```

```
public static void main( String[] args )
{
   Scanner cmdScanner = new Scanner( System.in );

   String  parameters;                  // stores command parameters
   String command = cmdScanner.next();  // get first command
   while ( ! command.equalsIgnoreCase( "quit" ) )
   {
      parameters = cmdScanner.nextLine( );   // get the parameters
      process( command,  parameters );       // process command
      command = cmdScanner.next();           // get next command
   }
}
```


PROCESS METHOD

```
run 25 40
turn 45
quit
```

```
public static void process( String command, String parameters )
{
   if ( command.equalsIgnoreCase( "run" ) )
      handleRun( parameters );

   else if ( command.equalsIgnoreCase( "turn" ) )
      handleTurn( parameters );

   // add other handlers here
   else
      System.out.println( "UNKNOWN COMMAND "
                          + command + " " + parameters );
}
```


HANDLERUN METHOD

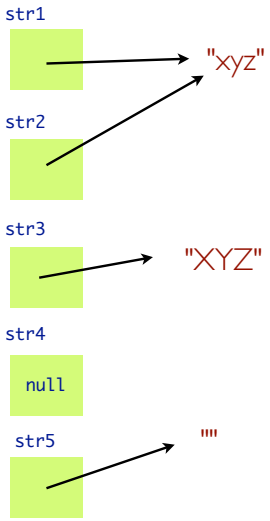
```
run 25 40
```

```
public static void handleRun ( String str )
{  // create a Scanner from the string
   Scanner params = new Scanner ( str );
   int x, y;
   if( params.hasNextInt() )      // try to get the first int
   {
      x = params.nextInt();
      if( params.hasNextInt() )   // try to get the second int
      {
          y = params.nextInt();
          System.out.println( "Ready to RUN! " + x + " " + y );
          run( x, y );
      }
      else
         System.out.println( "One int missing on RUN command!" );
    }
    else
       System.out.println( "RUN command need 2 int parameters!" );
}
```

STRINGS IN MEMORY

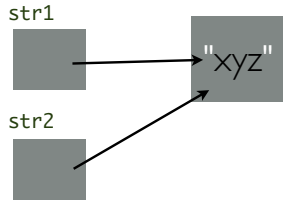
```
String str1 = "xyz";
String str2 = "xyz";
String str3;
String str4;
String str5 = "";

str3 = str2.toUpperCase();
```



STRING EQUALITY

```
String str1 = "xyz";
String str2 = "xyz";
```



- Note that in some situations Java will optimize by keeping only one instance of the string "xyz".
- This means that checking for String equality using == may sometimes seem to work:
  str1 == str2 // wrong, but sometimes works)
- To compare Strings for equality you should always use the equals method
  str1.equals(str2) // always works