Displaying Text with `System.out.println`:

- Debugging programs often requires knowing what the values of some variables are during run-time.
- Explains performance issues (e.g., “Why is the window in the wrong place when I draw it in my program?”)
- Can be complex and error-prone to calculate by hand.

One possibility: have the program print out the values as it runs, so you can see what it is doing.

One simple way to do this is with a “print” or “print-line” command (these each do the same thing, but the second one adds a line-break at the end of what it prints out):

```
System.out.print( "Hello" );
System.out.println( "Hello" );
```

Characters:

- The Java `char` primitive type:
  - Represents a single character, and is given values using *single quotes*:
    ```
    char ch1 = 'a';
    char ch2 = '8';
    char ch3 = '#';
    ```
  - Has increment/decrement operators, just like an integer:
    ```
    char ch = 'a';
    ch++;
    System.out.println( ch ); // gives output: b
    ```

- Other conventions listed in instructions:
  - Do not zip just the Java file.
  - Make sure you submit the correct files.
  - Zip the `src` folder and submit (NOT the entire project folder).
Converting between char and int

- The char type is stored using 16 bits (2 bytes) of memory
- Will automatically widen to any larger numerical types (int, long, float, double)
- Can be cast down to generate a char from an int, etc.
- Note: casting up & down with char can be surprising until you learn the basic character/number equivalencies!

```java
char ch = '2';
int i = ch;
System.out.println( i ); // output: 50
ch = (char) i;
System.out.println( ch ); // output: '2'
i = 99;
ch = (char) 99;
System.out.println( ch ); // output: 'c'
```

- One actually handy use of char arithmetic and int conversion is when we want to compute the n-th letter in the alphabet sequence (which comes in handy surprisingly often!)

```java
char letA = 'a';
char let13 = (char) ( letA + 12 );
```

- We can use this fact, plus the auto-coercion of the smaller type (char) to the larger type (int), to “slide” up and down in the order
- Same trick works for upper-case letters A–Z, and digits 0–9

```java
char letter13 = (char) ( letA + 12 );
char capital26 = (char) ( 'A' + 25 ); // 'Z'
char digit4 = (char) ( '0' + 3 ); // '3'
```

- Basic concatenation (+) operation: takes any values, converts to String objects if needed, and chains them together:

```java
String s = "Hello" + "There!";        // "HelloThere!
String s2 = "Hello" + " " + "There!"; // "Hello There!
System.out.println( "x = " + 100 );   // "x = 100"
```

- A special “basic” class that is fundamental to the language
- Has some behavior that is similar to primitive types
- Can be specified as a sequence of characters in double quotes
- Use of new and constructor is optional and not required

```java
String s = "Hello";
String s2 = new String( "Hello There!" );
String s3 = "Hello There!\n";
```
Strings are **Not** Primitive Types

- **String is a reference type (class) in Java**
  - There is no **direct conversion** between String and any primitive value
  ```java
  String s = "3";
  String s2 = 3;
  String s3 = '3';
  int number = "3";
  ```
  - **Basic concatenation (+) operation can force the conversion of a primitive to a String**:
  ```java
  int number = 3;
  String s1 = "Number is " + number;  // "Number is 3"
  String s2 = "" + number;            // "3"
  ```

Strings are Immutable

- **In Java, once created, a String does not change**
- Thus code like this will run fine, but will **not** change the content of the String in any way
  ```java
  String name = "David";
  name.toUpperCase();
  System.out.println( name );  // David
  ```
- If we want to change the String, we must **replace it** with a new one, using an **assignment**:
  ```java
  String name = "David";
  name = name.toUpperCase();
  System.out.println( name );  // DAVID
  ```

Converting Other Types to String

- **Basic concatenation (+) operation can force the conversion of other values to a String**:
  ```java
  int number = 3;
  String s1 = "Number is " + number;  // "Number is 3"
  String s2 = "" + number;            // "3"
  ```
- Be careful: where you place quotation marks matters!
  - Anything in quotes is reproduced **exactly as typed**
  ```java
  int number = 3;
  String s1 = "" + number;  // "3"
  String s2 = "" + "number";  // "number"
  ```

Methods in the String Class

- **Strings are objects with many methods**
- **Outputs character at input position**
  - `first char at: 0`
  - `last char at: length() - 1`
- **Outputs length of String**
- **Outputs sub-part of String, starting from input position, going all the way to end**
- **Outputs subpart of String, starting from first input position, going to second - 1**
- **Output lower/upper case version (Note: do not change original)**
  ```java
  String toLowerCase();
  String toUpperCase();
  ```

Full list online at:
`http://docs.oracle.com/javase/8/docs/api/java/lang/String.html`
**Methods in the String Class**

```java
String first, last;
first = "Ronald";
last = "McDonald";
System.out.print(first.charAt(1)); // o
System.out.print(last.length()); // 8
System.out.print(last.substring(2)); // Donald
System.out.print(first.substring(0, 3)); // Ron

first = first.toLowerCase();
last = last.toUpperCase();
System.out.println(first + " " + last); // ronald MCDONALD
```

---

**Using `System.out.println()`**

- This "print-line" method writes characters to the default output location (usually a terminal screen).
- In Eclipse, text will usually appear at the bottom of the editing area, but in other contexts, when the program runs, the text may appear somewhere else entirely.
- It will automatically convert most primitive types to a printable representation of their value:
  ```java
  System.out.println( 100 );
  System.out.println( 38.7 );
  ```
- The command also has many uses in writing simple programs that give output back to the user:
  ```java
  System.out.println( 3 * 4 + 5 / 6 – 3 * 2 );
  ```

---

**Elementary Debugging with `println()`**

- When you run into difficulties in a program, it is useful to know the values of some of your variables.
- If you are having trouble determining why a graphical object is not where you want it, for example, you might print out its location variables after some calculation:
  ```java
  int windowWidth = 500;
  int windowHeight = 400;
  win.setSize( windowWidth, windowHeight );
  int ovalSize = 100;
  int ovalLoc = windowWidth / 2 – ovalSize;
  System.out.println( ovalLoc );
  ```

---

**Code Examples: Debugging**

Let's debug some actual Java code examples...
This Week & Next:

- **Meetings next week:**
  - Wednesday: in the CS Lab (16 Wing)
  - Monday/Tuesday/Friday: regular classroom
- **Program 03:** due 11:59 PM, Tuesday 10 March
- **Reading:** Ch. 5 due Monday 09 March by **Noon**
- **Office Hours:** Wing 212
  - Monday/Wednesday/Friday: 11:00 AM–12:00 PM
  - Tuesday: 3:15 PM–4:15 PM
- **Lab and Tutor Hours:** On my website