

## An Introduction

- I. *Identifiers* - the name associated with a particular computer function or data object such as a variable, or a constant;

Syntax rules:

- 1) Can only consist of letters (a-z, A-Z), digits (0-9), or underscore characters ( \_ )  
Note: No periods nor blank (white space) are allowed!
- 2) Must start with either a letter (a-z, A-Z) or an underscore character ( \_ )  
Note: You should avoid starting with an underscore as internal commands in the computer use this format and errors can then occur.
- 3) Maximum length is determined by the computer system
- 4) Reserved words - those that have a special meaning in Java cannot be used
- 5) Identifiers are case-sensitive meaning that uppercase letters are different than lower case letters.  
ex. MrG mrg MRG mrG are all 4 different, distinct identifiers that cannot be interchanged

Proper identifiers: x, x1, X1, x\_1, \_x1, XX, x1X1, X12\_y13, sum, rateOne, PI

Improper identifiers: void, 1x, five\$, hi there, int, Mr.G., 56, PROG.CPP

II. Data Types - a specific set of data values along with a set of operations on those values

Note: Every variable (identifier) in a Java program must be declared (see the next section for the proper syntax for how to do that). When declaring the variable you are telling the computer what it represents by the kind of data you will be storing in it.

We will only be concerned at this time with a few of these:

### 1) Primitive Data Types

#### A) Integer Types

1) **int** (range: -2,147,483,648 to 2,147,483,647 )

2) **long** - also called **long int**

(range: -9223,372,036,854,775.808 to 9223,372,036,854,775.807)

ex. 0, 1, -5, 15000 (note: no comma allowed)

(Note: There are two other integer types, that we will not use, called **short** and **byte**.)

#### B) Floating Point Types

1) **float** (approximate range:  $-3.4 \times 10^{-38}$  to  $3.4 \times 10^{38}$  )

2) **double** (approximate range:  $-1.8 \times 10^{-308}$  to  $1.8 \times 10^{308}$  with 15 digit precision)

*precision* - The maximum number of significant digits a computer is capable of representing.

ex.1) 1234.567890 e 7 has ten significant figures and our computer can have a precision of 6 if this is a **float** type which means that the number stored in the computer would be 1234.56 e 7

### a) Floating Point Arithmetic Errors

of

- 1) *representational error* - The arithmetic error that occurs when the precision of the true result of an arithmetic operation is greater than the precision of the machine.

ex.1) Suppose this problem has six digits of precision.

What would be the result of:  $500060.000 + 19.9$  ?      Answer: 500070

Note: cancellation error - A form of representational error that occurs when numbers of widely differing sizes are added or subtracted like in the example above.

ex.2a) Should you use: `if (x == 5)`      // *x is an **int** type*

ex.2b) Should you use: `if (x == 5.23)`      // *x is a **float** type*

use: `if (Math.abs(x - 5.23) < .000001)` // *x we think it is the number 5.23 but it might really be something like 5.230000000001*

(Note: There are objects, that we will not use, that allow any size of number to be used called BigInteger and BigDecimal.)

ex) .3, -127.54, -0.4, 1.5e3, -4.5678e-2, 18.0

(Note: The **.0** must be there or it will be considered an integer type.)

### C) Other Data Types

- 1) **char** (short for character) - consists of a single character from the keyboard designated by enclosing it in single quotes

(Note: This is a not nonnumeric type in Java but it is in C++.)

ex.) 'a', 'A', '&', ' ' (a blank space - whitespace), '5'

- 2) **boolean** - consists the word *true* or *false*

Note: More in Chapter 5 on **boolean**

Note: There is no **boolean** type in C++.)

III. Variable Declarations - a statement that associates an identifier with a data type (so that the programmer can refer to that item later in the program by that name)

A) Declaring a single variable:

ex.1) **double** rate ;

ex.2) **int** count ;

Note: These are computer statements and therefore must end in a semicolon.

Note: Sometimes it is easier to read these lines from right to left.

Note: Every variable must be declared before that variable can be used.

You can declare the variable anywhere in the program preceding its use.

By convention we usually declare the variables:

a) following **public static void main (String[ ] args)**

right after the starting brace { (my preference) or

b) immediately preceding the usage of the variable

B) Declaring multiple variables in a single line - separate them by a comma:

ex.1) **int** time, hours ;

ex.2) **double** pay, wages2, myPay ;

Note: To be declared in the same line all the variables must be of the same type.

IV. Assignment Statement - a statement that stores the value of an expression into a variable

A) Syntax: Variable = expression ;

ex.1) **int** number ; // the declaration statement  
number = 10 ;

ex.2) **double** payRate ; // the declaration statement  
payRate = 5.55 ;

ex.3) **char** yesNo ; // the declaration statement  
yesNo = 'Y' ;

ex.4) **boolean** repeat ; // the declaration statement  
repeat = false ;

B) Semantic (meaning): The value or expression on the right is evaluated and then this result is assigned or stored in the computer's memory (like a post office box) with the variable on the left side as the memory location's name.

Note: You should read assignment statements from right to left and think of the = as an assignment not as an equal representing equality. See V. for why.

## AP Programming - Chapter 3 Lecture

V. Operators - symbols that perform a predefined function on the data with which it is used

A) Assignment operator ( = ): (Note: see page 69 in the blue book)

B) Arithmetic *Binary* Operators:

<u>Operation</u>	<u>Algebra</u>	<u>Computer</u>
Addition	+	+
Subtraction	-	-
Multiplication	X, ·, ( ), xy, 2x	*
Division	$2\sqrt{6}$ , $\div$ , $\frac{x}{y}$	/ (see below for further explanation) or % (see below for further explanation)

1) / (division)

a) The forward slash / for division when dividing two integers just gives only the integer quotient for the result

ex.1)  $8 / 2$  is 4 but

ex.2)  $7 / 2$  is 3 the decimal part of the answer is not given

b) When dividing two floating point decimals the result is a floating point decimal.

ex.1)  $7.0 / 2.0$  is 3.5

2) % (called the *modulus*) - the % symbol is used only for integer division and the result is the remainder

ex.1)  $7 \% 2$  is 1

ex.2)  $3 \% 5$  is 3

ex.3)  $5 \% 5$  is 0

ex.4)  $6.0 / 3.0$  is 2.0

ex.5)  $2 / 5$  is 0

More assignment statements (assume all variables have already been declared):

ex.1) `number = 2 + 5 ;`

ex.2) `overtimePay = 1.5 * pay ;` // What would happen if the \* were left out?

ex.3) `salary = rate * hours + overtimePay ;`

☆ex.4) `count = count + 1 ;`

☆ex.5) `down = down - 1 ;`

Note: You can only have a single variable on the left side of the = sign.

Invalid assignment statements:

ex.1) `x + 5 = y ;`

ex.2) `count + 1 = count ;`

ex.3) `y = 2x ;` // Why invalid?

### V. Operators - continued

#### C) *Unary* Arithmetic Operators (ones that take just one operand):

1) simple ones: + and -

ex.1) -7

ex.2) +7 (Note: The + as a unary operator is rarely used since if there is no sign then the number is automatically positive.)

2) a) *increment* and *decrement* operators: ++ and --

ex.1a)    number = 10;  
          number = number + 1;

or

ex.1b)    number = 10;  
          number++;

or

ex.1c)    number = 10;  
          ++number;

ex.2a)    number = 10;  
          number = number - 1;

or

ex.2b)    number = 10;  
          number--;

or

ex.2c)    number = 10;  
          --number;

++x (preincrement), x++ (postincrement),  
--y (predecrement), y-- (postdecrement)

ex.1) x = 10;  
      y = ++x;    // x now has the value of 11 and y now has the value of 11

ex.2) x = 10;  
      y = x++;    // x now has the value of 11 and y now has the value of 10

ex.3) x = 10;  
      x = x++;    // x has the value of 10 at the end of execution!  
                  // the right hand side ++ value of 11 is never stored in x

## AP Programming - Chapter 3 Lecture

D) Additional Java Combined Assignment Operators: +=, -=, \*=, /=

ex.1) `y -= 3` // is the same as `y = y - 3`

ex.2) `y /= 3` // is the same as `y = y / 3`

ex.3) `y += 3` // is the same as `y = y + 3`

ex.4) `y *= 3 + x` // is the same as `y = y * (3 + x)`

Note 1: `y =/ 3` // is illegal

Note 2: `y =+ 3` // is legal Why?

VI. Named Constants (also called a declared or symbolic constant.) -  
a location in memory, referenced by an identifier, where the data value that is stored there cannot be changed by a different assignment statement

Syntax: **final** DataType Identifier = LiteralValue;

(Note: This statement must both declare and assign a value to a variable.)

ex.1) **final int** MAXNUMBER = 3;

ex.2) **final char** BLANK = ' ' ; (Note: This represents a blank.)

ex.3) **final double** PI = 3.14;

ex.4) **final double** PI;  
PI = 3.14; // legal in Java – not in C++;

ex.5) **final double** PI = 3.14;  
PI = 3.2415;

Note: By convention the constant identifiers use all CAPITAL LETTERS and variable identifiers use lower case letters except caps are used for runTogether variables.

Note: The constant statement can appear either before the `main( )` heading or somewhere after the starting brace `{`, before the constant is used in the program.

**final** variables can also have the modifiers `public`, `private`, and `static` in front of them.

ex.1) **public static final int** MAXNUMBER = 3;

Note: When a variable is declared it can only be used in that block of code  
Static means that it can be used anywhere through out the class.

Note: Public means that other class methods can have access to the **final** variable.

Note: Private means that only that classes methods can have access to the **final** variable.

### VII. Initializing Variables - giving a value to a variable at the time that it is declared

Note: This is the same as an assignment statement.

A) Observe the following program portion:

```
{
  int firstNum, secondNum, firstAnswer, secondAnswer;    // variable declaration
  firstNum = 10;                                         // variable initialization
  firstAnswer = firstNum;                               // assigning an initialized value to a variable
  secondAnswer = secondNum;                             // oops
}
```

1) What is the value of firstAnswer?

2) What is the value of secondAnswer ?

A variable has no meaningful value until a program gives it one. If no value is given then the variable has a "garbage value" - a random number left in the memory or a number from a previous program which has no meaning and usefulness in the current program. GIGO (Garbage In Garbage Out)

B) Initializing Variables in Declarations (a shortcut)

```
ex.1a) double number ;           // the declaration statement
       number = 10.5 ;          // the initialization statement
```

or

```
ex.1b) double number = 10.5 ; /* the declaration and initialization
                               combined together */
```

```
ex.2a) int x, y, z ;             // the declaration statement
       x = 3 ;                   // initialization statement
       y = 4 ;                   // initialization statement
       z = 5 ;                   // initialization statement
```

or

```
ex.2b) int x = 3, y = 4, z = 5 ; // the declaration and initialization combined together
```

C) Multiple Assignments or Initializing of Variables with the Same Value  
(a shortcut)

```
ex.1a) int x, y, z ;             // the declaration statement
       x = y = z = 3 ;          // assignment/initialization statement
```

### VIII. Precedence Rules - order of operations

Introduction example:  $4 + 2 \times 3 = ?$

- A) Parenthesis,  
Division or Multiplication (from left to right),  
Addition or Subtraction (from left to right)

ex.1)  $2 - 5 + 1$

ex.2)  $4 / 2 * 2$

ex.3)  $3 + 6 / 3 * 2 - 1$

ex.4)  $(3 + 6) / 3 * 2 - 1 + 2$

- B) Convert each of the following mathematical expressions to a C++ arithmetical expression. Use parenthesis only where needed.

ex.1)  $y = 5 \div a$  (integer quotient)

ex.2)  $x = B + 2AC$

ex.3)  $x = 2(m + n)$

ex.4)  $x = \frac{3 - b}{r + 4}$  (floating point result)

### IX. Data Conversion with Mixed Data Types

Recall:

- 1) If  $x$  is an integer type, then after the statement  $x = 5;$   $x$  has the value of **5** stored in its memory location.
- 2) If  $x$  is a double or float type, then after the statement  $x = 5.0;$   $x$  has the value of **5.0** stored in its memory location.

Note: You can only store an integer in an integer variable and a floating point decimal in a float or double. But what if we tried to store an integer in a double or a decimal in an integer variable??

A) Assignment conversion (or also called widening, promotion, or type coercion) - the automatic conversion of a data type to another type due to mixing types in an expression.

1) Storing an integer in a decimal type:

ex.1) If  $x$  is a double type, then after the statement  $x = 5;$   $x$  has the value of **5.0** stored in its memory location.

\*When an integer is assigned to a decimal (double or float type) variable the integer is automatically converted to a number with a decimal point and a zero after it!

2) Storing a decimal in an integer:

ex.2) If  $x$  is an integer type, then the statement  $x = 5.9;$  is illegal in Java (but not in C++)!

\*A decimal (**double** or **float** type) can not directly be assigned to an integer variable. (Note: See the next section on typecasting. )

B) Type Casting - the explicit conversion of a value from one data type to another, by using a typecast operator which is really just the name of the desired data type in parenthesis

ex.1) **int** x;  
**double** y = 5.9;  
x = y; // illegal !

but you can force the value of y into x by casting it:

```
x = (int) y; // *see below
```

or

```
x = (int) (y); // *see below
```

but not

```
x = int (y);
```

\*\*But the value in x is not 5.9 but 5. \*When a decimal (**double** or **float** type) is assigned to an integer variable by type casting the decimal part is truncated (dropped off)!

Mixing data types can cause problems in a program.

```
ex.1) double x;  
      int a = 30, b = 40;  
      x = a / b;
```

What is x after the last line? Answer: 0.0 !!!!!

We can fix this result a couple of different ways:

```
ex.1) double x;  
      int a = 30, b = 40;  
      x = (double)a/b;
```

or

```
x = a/(double)b;
```

or

```
x = (double)a/(double)b;  
(a and/or b could also be in parenthesis but are usually not  
in any of the above)
```

But NOT

```
x = (double)(a/b); Why?
```

Because, the integers are divided first resulting in an integer result and when the result is converted the decimal part is truncated again to 0.0 as a final value in x.

C) Arithmetic Promotion - the explicit conversion of a value from one data type to another due to an arithmetic operation involving mixed data types.

1) In a calculation involving a mixture of integers and decimals the integers are temporarily converted to a decimal and the answer is a decimal.

```
ex.1) y = 2;  
      x = y + 3.5;
```

- a) What is x if x is an integer type? Answer: illegal statement
- b) What is x if x is a double or float type? Answer: 5.5
- c) What is y after the last line? Answer: 2 (not 2.0)

## AP Programming - Chapter 3 Lecture

X. Math methods - predefined (or prewritten) methods that are in the Math class that is included in every Java program because the Math class is found in the **java.lang** package which is automatically imported into all Java programs

A) Basic Methods to Know: (Note: **random** will covered in two pages)

<u>Name</u>	<u>Description</u>	<u>Type of argument</u>	<u>Type of value returned</u>	<u>Example</u>	<u>Result</u>
<b>sqrt</b>	square root	double	double	Math.sqrt(4.0)	2.0
		int	double	Math.sqrt(9)	3.0
		int/double	none	Math.sqrt(-9.0)	error
<b>pow</b>	powers	double	double	Math.pow(2.0,3.0)	8.0
		int	double	Math.pow(5,2)	25.0
<b>abs</b>	absolute value	int	int	Math.abs(-5)	5
		int	int	Math.abs(5)	5
		double	double	Math.abs(-5.3)	5.3
		double	double	Math.abs(5.3)	5.3
<b>ceil</b>	ceiling (round up)	double	double	Math.ceil(5.2)	6.0
		double	double	Math.ceil(5.9)	6.0
		int	double	Math.ceil(5)	5.0
<b>floor</b>	floor (round down)	double	double	Math.floor(5.2)	5.0
		double	double	Math.floor(5.9)	5.0
		int	double	Math.floor(5)	5.0
<b>round</b>	round to integer	double	double	Math.round(5.2)	5.0
		double	double	Math.round(5.9)	6.0
		double	double	Math.round(-5.9)	-6.0
		int	double	Math.round(5)	5.0

(Note: All double's could also be floats.)

\*Note: Math class is a **static** class therefore objects can not be created for it. The methods in Math class are **static** methods. We saw in Unit 1 that to use a method it needs to be accessed by connecting it to an object by the dot operator. Static methods are accessed by connecting them to the name of the static class by the dot operator so no object is needed (nor could one even be created). ex.) Math.sqrt(x);

### X. Math methods - continued

- 1) Determine the value of the following Java arithmetic expressions.  
(Note: Be careful of the answer's data type.)

ex.1a) `Math.sqrt(25.0)`

ex.1b) `Math.sqrt(25)`

ex.2a) `Math.pow(2.0, 5.0)`

ex.2b) `Math.pow(2, 5)`

ex.3a) `Math.abs(5)`

ex.3b) `Math.abs(-5)`

ex.4a) `Math.abs(3.9)`

ex.4b) `Math.abs(-3.9)`

ex.5a) `Math.ceil(5.1)`

ex.5b) `Math.ceil(5.8)`

ex.6a) `Math.floor(5.1)`

ex.6b) `Math.floor(5.9)`

ex.7a) `Math.round(5.5)`

ex.7b) `Math.round(-5.9)`

ex.8) `Math.sqrt(Math.abs(-36))`

ex.9) `Math.abs(Math.sqrt(Math.pow(3,2)))`

\*Note: `ceil( )`, `floor( )`, and `round( )` are not an AP topic.

- 2) To round `x` to the nearest integer you can do a typecasting: `(int)(x + 0.5)`

To round `x` to the nearest tenth you can do a typecasting:  
`((int)(10*x + 0.5))/10.0`

\*You can extend this to any number of decimal places.

ex.1) round 123.4567 to three decimal places and  
store in the decimal variable `y`

Answer: `y = ((int)(1000*123.4567 + 0.5))/1000.0;`

- 2) Convert the following mathematical expressions to a Java arithmetical expression

ex.) 
$$\frac{\sqrt{x^5 + y^7}}{2 - d}$$

- 3) Convert the following Java code into an algebraic expression.

ex.)  $x = (\text{Math.abs}(\text{Math.pow}(b,2) - 4.0 * a * c)) / (3 * a)$

- B) **random** method (Note: Our book does not cover this one and it is an AP topic.)

`Math.random( )` produces a random decimal number between 0 and 1 exclusive. To get a random number between two numbers *inclusive* called **small** and **big** use the following formula:

**`(int)(Math.random( )*(big - small + 1) + small)`**

- ex.1) If we want to get a random number between 5 and 10 inclusive then do:  
Note 1: **big** is the larger number of 10 in this example  
Note 2: **small** is the smaller number of 5 in this example  
Note 3: the formula needs a typecasting of **int**

Answer: **`(int)(Math.random( )*(10 - 5 + 1) + 5)`** or **`(int)(Math.random( )*(6) + 5)`**

- ex.2) write a line that will store in `x` a random number between 15 and 25 inclusive

Answer: `x = (int)(Math.random( )*(11) + 15);`

- ex.3) **`(int)(Math.random( )*(100) + 10)`** will produce a random number between what two numbers?

Answer: between 10 and 109 inclusive (work:  $100 = \mathbf{big} - 10 + 1$ )

(Note: There is a `Random` class with its own methods that will be discussed in Chapter 6.)

## AP Programming - Chapter 3 Lecture

### XI. Binary, Decimal, Octal, Hexidecimal Number Systems

Introduction:

ex. 345    3 hundreds + 4 tens + 5 ones (units)  
           $10^2$          $10^1$          $10^0 = 1$

What is the largest single digit in the decimal (base 10) system?

How many digits are there in the decimal system?

A) Binary Numbers:

Binary Number System - base 2 system consisting of two digits 0 and 1

Place value in base 2:

_____	_____	_____	_____	_____	_____	_____	_____
$2^7$	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$
128	64	32	16	8	4	2	1

1) Write the binary (base 2) numbers below as decimal (base 10) numbers

1

10

101

1100

1010

11001100

10110011

111111111

## AP Programming - Chapter 3 Lecture

_____	_____	_____	_____	_____	_____	_____	_____
128	64	32	16	8	4	2	1

2) Write the decimal numbers below as binary numbers

1

2

3

4

5

8

10

17

31

50

102

210

248

### B) Octal Numbers:

Octal Number System - base 8 system consisting of digits 0-7

Place value in base 8:

_____	_____	_____	_____
$8^3$	$8^2$	$8^1$	$8^0$
512	64	8	1

- 1) Write the octal number below as decimal (base 10) number  
2583
- 2) Write the decimal number below as an octal number  
248

### B) Hexadecimal Numbers:

Hexadecimal Number System - base 16 system consisting of digits  
0-9, A, B, C, D, E, F where A has a value of 10 ... and F has a value of 15

Place value in base 16:

_____	_____	_____
$16^2$	$16^1$	$16^0$
256	16	1

- 1) Write the hexadecimal number below as decimal (base 10) number  
D5
- 2) Write the decimal number below as a hexadecimal number  
248

XII. Strings - Strings are one of the most commonly used data types. A string is a sequence of characters enclosed within double quotes. A string is an object.

A) String Concatenation - Concatenation combines two strings together into one longer string. The + operator concatenates two strings. If one expression to the left or right of the + is a string then the other is automatically converted to a string and the two parts are concatenated.

ex.1)

```
String s = "The number of ";  
s += "Dalmatians is: "; // two strings concatenated
```

s is now the string: The number of Dalmatians is:

```
s += 101; // 101 is automatically converted  
System.out.println(s);
```

Output: The number of Dalmatians is: 101

ex.2)

```
System.out.print(5 + 2 + "A");
```

output is: 7A

```
System.out.print("A" + (5 + 2));
```

output is: A7

```
System.out.print("A" + 5 + 2);
```

output is: A52

B) String Positioning Numbers

0	1	2	3	4	5	6	7	8	9	0	11	12
H	E	L	L	O	,		W	O	R	L	D	!

Strings arrays, and other objects start counting at zero not 1. This is due to how the first programming languages stored and accessed this type of data and Java continued with this numbering method.

\*Note: The length of the above string is 13 not 12.

### C) String methods

The string class has many methods, you are responsible on the AP test for these:

`compareTo(Object other)` - Compares two strings lexicographically:  
returns an integer value  $< 0$  (usually  $-1$ ) if this string is less than other  
returns the value  $= 0$  if this string is equal to other  
returns an integer  $> 0$  (usually  $1$ ) if this string is greater than other

`equals(Object other)` - Compares this string to a specified object.  
The result is *true* if and only if the other is not null and is a String object that represents the same sequence of characters as this object.

`length( )` - returns the length of the string, including spaces, as an integer

`substring( int from, int to)` - returns a new string that is a substring of this one beginning at *from* and ending at *to - 1*

`substring(int from)` - returns a new string that is a substring of this one beginning at *from* and ending at *length - 1*

`indexOf(String str)` - If *str* occurs as a substring within this object, then the index of the first character of the first such substring is returned; if it does not occur as a substring,  $-1$  is returned (Note: `indexOf( )` is an AP topic but hardly discussed in our text.)

Not on the AP test and not in our book but ones to learn anyway:

`toLowerCase(String str)` - returns the string as all lower case letters

`toUpperCase(String str)` - returns the string as all upper case letters

`concat(String str)` - makes one longer string from the two

```
ex.1) string1 = "WHEATON";  
      string2 = "warrenville";  
      string3 = "Wheaton Warrenville South";
```

```
string1.toLowerCase( )           result: wheaton
```

```
string2.toUpperCase( )           result: WARRENVILLE
```

```
string3.length( )                result: 25
```

```
string1.concat(string 2)         result: WHEATONwarrenville
```

Note: The String methods `compareTo( )` and `equals( )` will be discussed in Chapter 5.

ex.2)

```
1 public class StringTest
2 {
3     public static void main(String[] args)
4     {
5         String greeting = "Hello";
6         String bookName = "Computing Concepts with Java Essentials";
7         System.out.println(greeting);
8         System.out.println(bookName);
9
10        String subBook = bookName.substring(10,18);
11        System.out.println(subBook);
12
13        System.out.println(greeting.length( ));
14        System.out.println(greeting.substring(3));
15
16        int position = bookName.indexOf("with");
17        System.out.println(position);
18
19        greeting = greeting + " there!";
20        System.out.println(greeting);
21    }
22 }
```

Lines 5 and 6 create string objects and assign values to these string objects.

Line 7 prints *Hello*

Line 8 prints *Computing Concepts with Java Essentials*

Line 10 creates a new string containing the characters in positions 10 to 17 inclusive.

The first character in a string is in position 0.

Line 11 prints *Concepts*

Line 13 prints *5*

Line 14 prints *lo*

This prints the substring from position 3 to the end of the string.

Line 16 assigns to the integer variable position the index of the first occurrence of *with* in the String *Computing Concepts with Java Essentials*

Line 17 prints *19*

Line 19 creates a new String with the value obtained from concatenating *Hello* with *there!* Strings are immutable objects. This means that a String object is never changed. A new string object is created when an assignment such as the one in line 19 is executed.

Line 20 prints *Hello there!*



C) Immutable objects - value can not be changed by an assignment nor method  
Strings are objects which mean they really contain a memory address that points to the actual string but they are immutable as discussed again in chapter 7.0

```
ex.1) String x = "Hi";  
String y = x;  
x = "Bye";
```

What value is in y?

Answer: "Hi"

D) The null reference vs. the empty string

```
ex.1) String x  
String y = "";
```

x is a null reference sometimes called a null string  
(it does not point - refer - to any string)

y is the empty string, "" that has a length of 0

e

### XIV. Input Dialog Screen

The following code shows what is needed if you want to have an interactive program with user input using an Input Dialog Screen using JOptionPane

```
import javax.swing.JOptionPane; //javax was written after java
public class Ex1
{
    public static void main (String[] args)
    {
        String name1;
        name1 = JOptionPane.showInputDialog(null,"Type your full name then <OK>");
        System.out.println("Your name is: " + name1);
    }
}
```

Note: **int** num = Integer.parseInt(input); is needed if the input is an integer

### XV. Console Input - Summary for Reading from System.in

The system console uses the System.in object to obtain user input. The System.in object provides methods in its interface to read a set of bytes from the keyboard. The programmer, however, cannot use these methods directly (unlike System.out).

Before the system console can use the data from the keyboard, the data must be translated from bytes provided by the keyboard into Unicode characters. The characters then must be formatted to allow the user to read a line of data from the keyboard. This is done by routing the data from the keyboard through two objects.

The first object is an InputStreamReader object that translates the bytes produced by the keyboard into Unicode. The second object is a BufferedReader object that translates the Unicode characters into a line of text. Here is the code to create an object that the programmer can use to read lines of data from the keyboard using the System.in object:

```
InputStreamReader instream;  
BufferedReader keyboard;  
inStream = new InputStreamReader(System.in);  
keyboard = new BufferedReader(inStream);
```

The first line declares the variable inStream to be a reference to an object of the InputStreamReader class. The second line declares the variable keyboard to be a reference to an object of the BufferedReader class. The third line creates the InputStreamReader object using the new keyword and assigns a reference to the object to inStream. The last line creates the BufferedReader object and assigns a reference to the object to variable keyboard.

To use the InputStreamReader and BufferedReader classes, the programmer must include import statements at the top of the program. These import statements inform the Java compiler where the InputStreamReader and BufferedReader classes are located. In this case, the two classes are located in a group of classes (called a package) called java.io. This package contains classes to perform input and output to and from files. The statements placed in the import section at the top of the program are:

```
import Java.io.InputStreamReader;  
import Java.io.BufferedReader;
```

The program uses the `BufferedReader` object to read input from the system console.

The `BufferedReader` class has several methods in its interface. The relevant one here is a function-type method called `readLine()`. This method takes no arguments and returns a line of input typed by the user at the system console. If there is no line of input waiting, the program waits until the user has entered a line of input and then returns a string object that contains the line that the user entered (without the Enter that the user pressed at the end of the line).

```
String LineFromUser = keyboard.readLine();
```

There is one last element you must deal with. Although it is unlikely, the `readLine()` method can fail. Java requires that you at least acknowledge the possibility of failure. This is done by adding the line

**import** `Java.io.IOException;` in the import section of the program,  
and adding the clause

**throws** `IOException` to the declaration of **public static void** `main(String[] args)`  
to make

**public static void** `main(String[] args)` **throws** `IOException`

This is what will produce the built-in error message in to the Java program.

```
import java.io.InputStreamReader;
```

```
import java.io.BufferedReader;
```

```
import java.io.IOException;
```

```
public class ex1
```

```
{  
    public static void main (String[] args) throws IOException  
    {  
        InputStreamReader inStream;  
        BufferedReader keyboard;  
        inStream = new InputStreamReader(System.in);  
        keyboard = new BufferedReader(inStream);  
        String input;  
        int num;  
        System.out.println("Type in an integer then <Enter>");  
        input = keyboard.readLine();  
        num = Integer.parseInt(input);  
        System.out.println("Your input now as an integer is: " + num);  
    }  
}
```

OR a shortcut:

```
BufferedReader keyboard = new BufferedReader(new InputStreamReader(System.in));
```

### XVI. Wrapper Classes

The line `num = Integer.parseInt(input);` uses a method from the class `Integer` to convert the string `input` into a useable integer.

The classes `Double` and `Integer` and `Float` are called wrapper classes and allow numbers to act as objects which is useful if you need to use a method that works on objects such as sorting. When inputting from the keyboard (console) everything is input as a string and if it is actually a number or a character then it must be converted by one of the following

**int** `ex1 = Integer.parseInt(input);`

**double** `ex2 = Double.parseDouble(input);`

**float** `ex3 = Float.parseFloat(input);`

**char** `ex4 = Character.parseChar(input)`