Computer Science I

Chapter 2 Notes

The print and println methods

- Both the print and println methods are used to output information to the screen.
- Each piece of data that we send to a method is called a ***parameter***.
- print and println require <u>one</u> String parameter.
- System.out.println("some string");

  This method will print the string then advance to the next line.
- System.out.print("some string");

  This method will print the string but will not advance to the next line.

Strings

- A string of characters can be represented as a *string literal* by putting double quotes around the text:
- Examples:

```
"This is a string literal."
"123 Main Street"
"X"
```

- Every string is an object in Java, defined by the `String` class.
- Every string literal represents a `String` object.

String Concatenation

- The string concatenation operator (+) is used to append one string to the end of another

```
"Peanut butter " + "and jelly"
```

- It can also be used to append a number to a string.

```
"The answer is " + 37 + "."
```

What actually happens is that Java automatically converts the 37 into a String "37", then concatenates.

String Concatenation

- The + operator is also used for arithmetic addition.
- The function that it performs depends on the type of the information on which it operates.
- If both operands are strings, or if one is a string and one is a number, it performs string concatenation.
- If both operands are numeric, it adds them.
- The + operator is evaluated left to right, but parentheses can be used to change the order of operations.

Escape Sequences

- What if we wanted to print a the quote character?
- The following line would cause a compiler error because it would interpret the second quote as the end of the string

```
System.out.println ("I said "Hello" to you.");
                                ^^error
```

- An *escape sequence* is a series of characters that represents a special character.
- An escape sequence begins with a backslash character (\).

```
System.out.println ("I said \"Hello\" to you.");
```

Escape Sequences

- Some Java escape sequences:

Escape Sequence	Meaning
\b	backspace
\t	tab
\n	newline
\r	carriage return
\"	double quote
\'	single quote
\\	backslash

Variables

- A *variable* is a name for a location in memory.
- A variable must be *declared* by specifying the variable's name and the type of information that it will hold.


```
int total;
int count, temp, result;
```

Multiple variables can be created in one declaration

Variable Initialization

- A variable can be given an initial value in the declaration.

```
int sum = 0;
int base = 32, max = 149;
```

The term "initialization" is used to mean assigning a starting value to a variable.

Assignment

- An *assignment statement* changes the value of a variable.
- The assignment operator is the = sign.

```
total = 55;
```

The variable name goes on the left.

The expression on the right is evaluated and the result is stored in the variable on the left.

The value that was previously in `total` is overwritten.

Primitive Data

There are eight primitive data types in Java.

- Four of them represent integers (no decimal point):

  `1. byte  2. short  3. int  4. long`
- Two of them represent floating point numbers:

  `5. float 6. double`
- One of them represents characters:

  `7. char`
- And one of them represents boolean (true or false) values:

  `8. boolean`

Numeric Primitive Data

Type	Storage	Min Value	Max Value
`byte`	8 bits	−128	127
`short`	16 bits	−32,768	32,767
`int`	32 bits	−2,147,483,648	2,147,483,647
`long`	64 bits	$< -9 \times 10^{18}$	$> 9 \times 10^{18}$
`float`	32 bits	$+/- 3.4 \times 10^{38}$ with 7 significant digits	
`double`	64 bits	$+/- 1.7 \times 10^{308}$ with 15 significant digits	

- The difference between the various numeric primitive types is their size, and therefore the values they can store:

Numeric Primitive Data

- There is an order to the numeric types. byte is the lowest type…double is the highest.

Type

```
byte      lowest
short
int
long

float
double    highest
```

- Java will automatically convert from a lower type to a higher type.

Expressions

- An *expression* is a combination of one or more operators and operands
- *Arithmetic expressions* compute numeric results and make use of the arithmetic operators:

Addition	+
Subtraction	–
Multiplication	*
Division	/
Remainder	%

If either or both operands used by an arithmetic operator are floating point, then the result is a floating point

Division and Remainder

- If both operands to the division operator (/) are integers, the result is an integer (the fractional part is discarded).

```
14 / 3      equals      4

8 / 12      equals      0
```

The modulus operator (%) returns the remainder after dividing the first operand by the second operand.

```
14 % 3      equals      2

8 % 12      equals      8
```

Operator Precedence

- Operators can be combined into complex expressions

```
result  =  total + count / max - offset;
```

- Operator precedence determines the order in which they are evaluated.
- ( ) parentheses first
- * / % (left to right)
- + - and string concatenation (also left to right)

Operator Precedence

- What is the order of evaluation in the following expressions?

```
a + b + c + d + e
  1   2   3   4

a + b * c - d / e
  3   1   4   2

a / (b + c) - d % e
  2    1    4   3

a / (b * (c + (d - e)))
  4    3    2    1
```

Increment and Decrement

- The increment and decrement operators use only one operand.
- The *increment operator* (++) adds one to its operand.
- The *decrement operator* (--) subtracts one from its operand.
- The statement

```
count++;
```

  is equivalent to

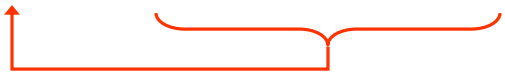
```
count = count + 1;
```

Assignment Revisited

- The assignment operator has a lower precedence than all of the arithmetic operators

First the expression on the right hand side of the = operator is evaluated

```
answer  =  sum / 4 + MAX * lowest;
           4     1   3     2
```



Then the result is stored in the variable on the left hand side

Assignment Revisited

- The right and left hand sides of an assignment statement may contain the same variable

First, one is added to the original value of count

```
count  =  count + 1;
```

Then the result is stored back into count (overwriting the original value)

Data Conversion

- Sometimes you would like to convert one type of data into another type.
- *promotion – conversion from a lower type to a higher type*
- *Narrowing conversions* can lose information because they tend to go from a higher data type to a lower one. Narrowing conversion requires a cast.

```
float x = 3.5f;
int y = (int) x;  // (int) is a cast
```

Assignment Conversion

- *Assignment conversion* occurs when a value of one type is assigned to a variable of another.

```
int dollars = 8;
float money;
money = dollars;  // assignment conversion
```

- What's in money now? `8.0f` (float) What's in dollars? `8` (int)
- Only widening conversions can happen via assignment.

Data Conversion

- *Promotion* happens automatically when operators in expressions convert their operands
- For example, if `sum` is a `float` and `count` is an `int`, the value of `count` is converted to a floating point value to perform the following calculation:

```
result = sum / count;
```

- Promotion only occurs from a lower type to a higher type.

Keyboard Input

- Programs generally need input on which to operate.
- The `Scanner` class provides convenient methods for reading input values of various types.
- A `Scanner` object can be set up to read input from various sources, including the keyboard.
- Keyboard input is represented by the `System.in` object.

Reading Input

- The following line creates a Scanner object that reads from the keyboard:

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

- The `new` operator creates the `Scanner` object.
- Once created, the `Scanner` object can be used to invoke various input methods, such as:

answer = scan.nextLine( );

Inputting Numbers

- The Scanner class has six methods for inputting numbers. Each method corresponds to a specific data type.

```
nextByte
nextShort
nextInt
nextLong
nextFloat
nextDouble
```

Inputting Boolean Values

- The Scanner class has a method for inputting boolean values (true or false).

```
nextBoolean
```

Inputting Strings

- The Scanner class has two methods for inputting Strings.

`nextLine`

reads characters from the input until the end of the line is found

`next`

reads characters from the input until a whitespace character is found

Whitespace includes space, tab, and newline.

What about characters?

- The Scanner class does not have a method for inputting one character.
- So, how can we input one character?

1. Input a String using next or nextLine.
2. Use the String method `charAt(0)` to get the first character of the String.