Week 1 Notes

Reading: 2.1-2.3.3, 2.3.5-2.3.8

What do these mean?

TLA (Three Letter Acronym)  
 JDK (Java Development Kit) – basically a package for you to develop applications w/ Java  
 JRE (Java Runtime Environment) – in jdk; compiler, standard libraries, config tools  
 IDE (Integrated Development Environment) – In this class, we will use jGRASP

How it works: (Bk 1 p1)

Write Example1.txt on board

* go through each word in the code (p 9-10 has all the breakdown)
  + keywords in class header (p 3)
  + Example1 is an identifier (p3)
  + double slashes mean comment
  + main method and input args
  + System.out.println calls the println method from the System.out class (tbd)

Class: Example1.txt

1 public class Example1 // Class header  
2 {  
3 public static void main(String [] args) // Method header  
4 {  
5 System.out.println("Well, well, well, what have we here");  
6 }  
7 }

Procedural Abstraction:

Think of WHAT to do independently of HOW to do it  
 In this example, you know that System.out.println(“Well, well, well, what have we here”); displays the text, but you don’t know HOW it does it

A very useful way of designing a program!  
You will use this method a lot in class.

Basics of Object-Oriented Programming

Bk 1, p.1-flow of control  
 \*\*INCLUDE additional definition of method in p 2

Pseudocode:

Bk 1 p 2  
 Example of pseudocode: write a program so that if a number is greater than or equal to 0, print that it is so. If it is less than zero, print that it is so

We use algorithms to help us design the code.

Algorithm definition (p2)

Variables

* Named olcations in memory where values can be stored
* Uses a name to represent a value that can change
  + eg. in algebra, x can be different values in different equations (x + 4 = 10 OR x – 3 = 1)
  + Not like constants, which cannot change values at all (10 is always 10)
* 8 primitive data types of variables
  + 4 integer data types (Bk 1 p 5)
  + 2 types of floating-point data (p 5)
  + 2 other types, char and Boolean (p6)
* Literals (p 11) are the actual values the variable represents
* Declaring a variable:
  + Syntax p 5)
  + Conventional way of naming variables
  + Java is pretty strict with variables (eg. you cannot declare an int variable w value of 3.14159)
* Can also have variables of type String (p 12)
  + Non-primitive
  + String is an Object (to be discussed)
  + Example
  + In the example, s is a reference variable. Reference means address of. That means ‘s’ points to the location of “abc” and not “abc” itself

String literals and escape sequences are already covered in the reading 2.2.7. Any questions, let me know

One thing to note, String literals are always enclosed in ““ double quotes, chars are enclosed w ‘’

Example: FutureAge class

A very basic code dealing with variables

1 public class FutureAge  
 2 {  
 3 /\*  
 4 Data Table  
 5 Variable or Constant Purpose  
 6 --------------------- -------------------  
 7 args parameter of main  
 8 age my currenet age  
 9 oldAge age in 5 years  
10 \*/  
11   
12 /\*  
13 Algorithm  
14 main(args)  
15 age <-- 27  
16 oldAge <-- age + 5  
17 print oldAge  
18   
19 \*/  
20   
21 public static void main(String [] args)  
22 {  
23 int age; //declare age  
24 age = 27;  
25 int oldAge = age + 5;  
26 System.out.println( oldAge);  
27   
28 }  
29   
30 }

Constants

* Syntax (p7)
* Declaration
* naming convention

Operators, String operators, Shortcut operators (2.3.1)

* Operators “do stuff” to the data
* Binary operators need 2 operands (+, -, /, \*, %)
  + Modulus is remainder after division
  + 5 % 2 = 1; 5 / 2 = 2 r. 1
  + If need more clarification I have a Java file that explains it (Espressions.java 10/12)
* Unary operators need 1 operand (+, -, positive and negative)
* Shortcut operators, incrementing and decrementing (pp 16-18)
  + Only go over ++ and –
  + Others should have been read and understood
* Order of Operators (p 11): More complete table found in book
  + Example in p 10
* Division and Rounding
  + What is 5/2? (Answer is 2 since 2.5 rounded down is 2)
  + In division w integers, round down always
  + What does % = 0 imply about the number being divided?

Exercise1 class to show order of operations

1 public class Exercise1  
 2 {  
 3 public static void main(String [] args)  
 4 {  
 5 int a = -3;  
 6 System.out.println(((7+(6\*4))/(8-(3%2))+ ((17-(4/3))/(6+(3\*2)))/(8-(7%4))));  
 7   
 8 System.out.println( (((7+6\*4)/(8-3%2)) + ((17-4/3)/(6+3\*2))) / (8-7%4));  
 9   
10 System.out.println(7 + 6/4 - 3\*7%2 + 5/2);  
11 System.out.println( 6 + 3/2.0 - 6/4 + 8%10);  
12 System.out.println( 6 + 4/2.0 - 6/4 + 8%10);  
13 String s = "abc";  
14 String t = 6 - 8 % 3 + s + "\n" + -a + 7 / 4;  
15 System.out.println(t);  
17 // t = s \* 3  
18 // The only string operator is +   
19   
20 double inches = 10.0;  
21 System.out.println(inches + " inches is equal to " + 2.54 \* inches + " centimeters");  
22   
23 //constants  
24 final int MAXIMUM\_DAMAGE = 1000000;  
25 }  
26 }

Expressions class to go over unary operators

1 public class Expressions  
 2 {  
 3   
 4 public static void main(String [] args)  
 5 {  
 6 // int x=3.2;  
 7 int a = -3;  
 8 System.out.println(+a);  
 9 System.out.println(-a);  
10 System.out.println(+ - - + - +a); // order of operations goes from right to left. End result is 3 (3 minuses)  
11 System.out.println(7 + -a);  
12   
13 }  
14 }

String Operators

* The only operator for a String is “+”
* Example on p 13
* Naming convention for class names

OldAge class: Dealing w String operators

We have our design document elsewhere. We design our code first before we implement it

OldAge design doc:

Data Table

Variable or Constant Purpose

--------------------- -------------------

args parameter of main

age my currenet age

oldAge age in 5 years

Algorithm

main(args)

age <-- 27

oldAge <-- age + 5

print oldAge

Then we implement our design:

1 public class OldAge  
 2 {  
 3   
 4 public static void main(String [] args)  
 5 {  
 6 int age = 27;  
 7 int oldAge = age+5;  
 8 System.out.println("Right now you are " + age);  
 9 System.out.println("If you are " + oldAge + ", you are officially old. Congratulations!");  
10 }  
11   
12 }

Division by Zero

* int / 0 = error
* Double / 0 =
  + If dividend is nonzero, answer = infinity
  + If dividend is zero, answer = NaN (Not a number)

Explicit/Implicit Type Casting

* We need type casting when we perform operations on data of different types.
  + 8 (int) / 1.25 (double)
  + These types are not compatible, so how do we choose what type the result is?
  + Convert the result to desired variable type
  + Remember, ints are always truncated (cut off)!
* You should have read and understood the Rules for Promotions: (p 14)
* Implicit type casting is performed by the compiler automatically
  + Remember, the data type is temporarily changed and will go back to its old type after the calculation is performed
  + Example on p 15
* Explicit typecasting
  + Instruct compiler specifically to convert variable types
  + Syntax on p 15
  + Only include () around expression when you want to perform expression calculation first THEN change result type
* Go over example on p 16 about explicit vs implicit typecasting

TypeCasting.java: example of typecasting

1 public class TypeCasting  
 2 {  
 3 public static void main(String [] args)  
 4 {  
 5 //syntax:  
 6 //(type)(expression)  
 7 //put a wide value in a narrow box  
 8 double d = 4.3;  
 9 int x = (int)d;  
10 byte a = 1;  
11 byte b = 2;  
12 short c =(short)(a+b);  
13 c = 129;  
14 a = (byte)c;  
15   
16 //make a type wider to store more information,  
17 //typically, fractional parts of answer.  
18   
19 int sum = 6834;  
20 int count = 13;  
21 double average = sum / count;  
22 average = (double)sum / count;  
23 }  
24 }

Different kinds of errors (p 19, copy from book)

1. Syntax error
2. Runtime error
3. Logic error