Object-Oriented Programming

Java Syntax

- Program Structure
- Variables and basic data types.
- Industry standard naming conventions.
- Java syntax and coding conventions
- If … Then … Else …
- Case statements
- Looping (for, while)
- Classes and Methods

Variables & Data Types

Data types

- Java supports 8 basic data types known as **Primitive Types**.
  - Four Integer types
  - Two real (or floating-point) types
  - A boolean type
  - A character type
- Java also supports class and array data types (collectively known as **Reference types**), which will be discussed later.

Primitive Types – Integers

Keyword	Size	Min value	Max value
byte	8-bit	-128	127
short	16-bit	-32768	32767
int	32-bit	-2147483648	2147483647
long	64-bit	- 9223372036854758087	9223372036854775807

- **int daysInYear = 365;**

```
/* declares an integer named
   daysInYear, assigns it the value
   of 365*/
```

Primitive Types – Real Numbers

Keyword	Size	Min value	Max value
float	32-bit	1.4E-45	3.4028235E38
double	64-bit	4.9E-324	1.797693134862315 7E308

- **float price = 8.99f;**

```
/* declares a float named price, assigns
   it the value of 8.99 */
```

Primitive Types – Boolean/char

Keyword	Size or Format	Description
boolean	true/false	true or false
char	16-bit Unicode	A single character \u0000 - \uFFFF

- **boolean trustworthy = true;**
- **char firstLetter = 'a';**

Strings

- A String represents a group of characters.
- String is not a primitive type, it is a Java class.
- Strings can be used like primitive types in many ways.
  - Assign literal values or Add Strings together

```
System.out.println("Joe");
String name = "Joe";
System.out.println(name);
String fullName = name + " Thomas";
```

- We'll discuss String in detail later.

Variable Naming Conventions

- Variables **must** start with a letter, a dollar sign ($), or an underscore (_).
- Followed by any of the above characters and/or numbers.
- Can be any length.
- Avoid short abbreviations - your tools will do most of the typing for you!

Naming Conventions cont

- Variables cannot be a Java keyword, or any Java reserved keyword.
- *By convention,* begin with a lowercase letter and no separating character between words (also called the title case rule).
- Each following word starts with a capital letter.

myVariableName

```
int registration = 100;
short conferenceRoomCapacity = 500;
float feePerPerson = 1199.99f;
boolean conferenceIsCancelled = false;
```

Java and Case-Sensitivity

Java is a case sensitive language. ‘result’ and ‘Result’ are different variables.

```
int result = 123;
int Result = 123;  // don't do this!
```

REMEMBER: Everything in Java is CASE sEnSiTiVe.

Java Program Statements

Java Statements

- Each Java statement ends with a "**;**".

```
int count;

char indicator;
```

- It is permitted to have two or more statements on a single line, but it is not recommended.

```
int x = 0; int y = 1;
```

Block

- A block is zero or more Java statements
- Blocks are defined by { }

```
{
    int x = 5;
    int y = 0;
}
```

Scope Rules

- A variable is available only within its 'scope'.
- Scope is the defining block and all contained blocks.

```
{
      {
            int x = 5;
            System.out.println(x);
      }
      System.out.println(x); // error!
}
```

Scope Rules - Example 2

```
{
   int x = 10;
   {
        int x = 5; // error!
        System.out.println(x);
   }
   System.out.println(x);
}
```

Find the Defect 1

- There is an naming convention mistake in the following program. Can you find it?

```
public class MyClass {
      public static void main(String args[])
      {
            String Name = "Bob";
            System.out.println( name );
      }
}
```

Find the Defect 2

- There is one syntax error in the following code. Can you find it?

```
public class MyClass {
      public static void main(String args[])
      {
            System.out.println( Hello there! );
      }
}
```

Find the Defect 3

- There are two syntax errors in the following code. Can you find them?

```
public class MyClass {
      public static void main(String args[])
      {
            system.out.println("Hello there!")
      }
}
```

Key Points

- Java statements must end with a ‘;’
- Variables must have a **type**
- Variables must have a **name**
  - int count;
- Java is case sensitive

Comments

- Two ways to comment in Java:
  - Single Line Comment

```
int states = 50;  // current number of states
```

  - Multiple Line Comment

```
/* This code computes the total amount due by the
customer including applicable interest, late fees
and principal by the next due date  */

float amountDue = principal + latefees + interest;
```

Performing Calculations in Java

Arithmetic Operators

Operator	Description	Example
+	add	`y = x + 1;`
-	subtract	`y = x - 1;`
*	multiply	`y = x * 10;`
/	divide	`y = x / 5;`
%	modulus – returns remainder	`y = x % 3;`

Operator Precedence

Java's Mathematical Order of Precedence

- The operator precedence defines what operators take precedence over other operators, and hence get executed first.
- Expressions contained within parenthesis are evaluated first.
- The order of precedence for the mathematical operators is from **left to right in the following order**:
  - Multiplication and Division
  - Addition and Subtraction

Operator Precedence

Example	Explanation

`int x = 2;`
`int y = 10;`
`int z = 3;`
`int r;`
`Example 1:`
`r = x + y * z;`
`// result is 32`
`Example 2:`
`r = (x + y) * z;`
`// result is 36`
	Rule: Parenthesis determine 1st order of mathematical operations.
	
	Example 1:
	Multiplication first; then addition
	
	Example 2:
	Math proceeds first with the parenthesis then the multiplication takes place.

Lab #1

- Ask user to state the price of an item to be purchased. Display a bill to the user that shows the original price, the amount of tax to be charged at NJ tax rate, and the total amount the customer will pay.

1. Turn Pseudo-code into Java code
    1. What type of variables need to be declared
    2. What are the key formulas?
    3. What do you have to print out?
2. Create print statement(s) that output:

```
Price: 20
Tax: 1.4
Total: 21.4
```

Branching & Decision-Making

If … Else …

If / Else provides a way to make decisions between 2 possible paths

- If (trafficLight = ‘Green’)
  - Drive();
- Else
  - Stop();

if/else Statements

```
if ( test condition ) {
        do this when true;

} else {
        do this when false;

}
```

if/else Example

```
int x;

System.out.println("Enter a number");
x = input.nextInt();

if ( x > 10 ) {
      System.out.println("x is greater than 10");
} else {
      System.out.println("x is NOT greater than
10");
```

Nested if/else Statements

```
if ( test condition ) {
        do this when true;

} else if (2nd test condition){
        do this when true for 2nd condition;

}else{
        do this if both test conditions are false

}
```

Nested if/else Example

```
int x;

System.out.println("Enter a number");
x = input.nextInt();

if ( x > 10 ) {
      System.out.println("x is greater than 10");
} else if ( x < 10 ) {
      System.out.println("x is less than 10");
} else {
      System.out.println("x is equal to 10!");
}
```

- The else can be combined with if. The last else becomes the default condition.

Test Condition Operators

Operator	Description	Example
<	less than	`if ( x < y )`
>	greater than	`if ( x > y )`
>=	greater than or equal	`if ( x >= y )`
<=	less than or equal	`if ( x <= y )`
==	equality (notice there are TWO = signs here)	`if ( x == y )`
=	assignment	`x = y;`
!=	not equal	`if ( x != y )`
!	negation “not”	`if ( !false )`

Equality Operator

- '==' is very different from "=".
- Rule:
  - '==' is used to compare two variables.
  - '=' is used **only** for assignment.

```
if (a = 0)    // compiler error!


if (a == 0)   // compiler happy!
```

Logical Operators

Operator	Example	Description
`&&`	`((x==5)&&(y==3))`	Conditional statements using **&& (Logical AND)** or **|| (Logical OR)** are only evaluated as much as needed to make a **DEFINITE** decision. These are known as short-circuit operators, and should **USUALLY** be used in conditional statements.
`||`	`((x==5)||(y==3))`	

Branching & Decision-Making

case / switch statements

switch statement

- *switch* statement is used when multiple, discreet options exist

```
System.out.println("Pick a number from 1 to 3");
int number = input.nextInt();
String winner = "";
switch (number) {
case 1: winner = "Big Prize!"; break;
case 2: winner = "Medium Prize!"; break;
case 3: winner = "Small Prize!"; break;
default: winner = "No Prize!"; break;
}
System.out.println("You win: " + winner);
```

switch statement

- General format:

```
Initialize option variable;

switch (option variable) {
case choice1: do something; break;
case choice2 : do something; break;
case choice3 : do something; break;
… and so on …
default: do default action; break;
}
```

- default is used when there is no match on any other choice

Repeating your program steps

Looping

for Loops

- A Java 'for' loop repeats a block of statements a fixed number of times.

```
for (int i = 0; i < 4; i++ ) {
     System.out.println(i);
}
```

Output:

0
1
2
3

What is the value of 'i' after the for loop?

for Loops

- General format:

```
for (initialize control variable; specify test
condition; increment or decrement control variable ) {
      do something;
}
```

- As long as the for loop *test condition* remains true, the loop will continue to run

Increment/Decrement Operators

Keyword	Description	Code	Equivalent
`++var`	Pre-Increment	`++x;`	`x = x + 1;`
`var++`	Post-Increment	`x++;`	`x = x + 1;`
`--var`	Pre-Decrement	`--x;`	`x = x - 1;`
`var--`	Post-Decrement	`x--;`	`x = x - 1;`

- With the "Pre" operators the variable is inc/decremented BEFORE it is used.
- With the "Post" operators the variable is inc/decremented AFTER it is used.

Pre-Post Increment/Decrement Example

Initial x	Expression	Final y	Final x
3	`y=x++`	3	4
3	`y=++x`	4	4
3	`y=x--`	3	2
3	`y=--x`	2	2

while Loops

- A Java 'while' loop continually executes a block of statements while a condition remains true.
- A while loop executes zero or more times.

```
int rightAnswer = 3;
int userAnswer = 0;
boolean correct = false;
while (not correct) {
        System.out.println("Guess a number 0 to 9");
        userAnswer = input.nextInt();
        if (userAnswer == rightAnswer) {
          correct = true;
          System.out.println("Congratulations! You are right!");
        } else {
           System.out.println("Wrong.  Try again.");
            }
}
```

while Loops – 2nd Example

```
int rightAnswer = 3;
int userAnswer = 0;
int numTries = 1;  // counts number of attempts by the user
int triesLeft;
boolean correct = false;
while (not correct) && (numTries < 4) {
        System.out.println("Guess a number 0 to 9");
        userAnswer = input.nextInt();
        if (userAnswer == rightAnswer) {
          correct = true;
        System.out.println("Congratulations! You are right!");
        } else {
           System.out.println("Sorry. Wrong answer.");
          triesLeft = 3 - numTries;
           System.out.println("You get " + triesLeft + " more
          try / tries.")
          numTries ++;
          }
}
```

while Loops

- General format:

```
initialize control variable to true state ;
while  (specify test condition)  {
          do something ;
        if (end loop test condition) {
                set control variable to opposite state;
        }
}
```

- As long as the while loop *test condition* remains true, the loop will continue to run

break Statement and Loops

- The 'break' statement immediately ends a loop. What is the output?

```
int i = 0
while(i < 5) {
      if (i == 2) {
            break;
      }
      i++;
      System.out.println(i);
}
```

continue Statement and Loops

- A ‘continue’ statement allows one to skip the rest of the current loop iteration. What is the output?

```
for (int i = 0; i < 5; i++ ) {
     if (i == 2) {
          continue;
     }
     System.out.println(i);
}
```

Classes & Methods

Classes & Methods

- Up until now, we've been creating simple programs with a single method - main

```
public Class Classname {

public static void main (String[] args) {
   Do something;
      }
}
```

Classes & Methods

- We now want to create classes with specialized methods

```
public Class MyClass {

public static void method1 () {
    Do something;
    // this method has no inputs and no return value
        }

public static float method2 (int varName) {
       float newValue = 0;
      newValue = Do something with varName;
      return(newValue);
      // this method takes an integer and returns a float
        }
}
```

Classes & Methods

- We want our main program to call those classes and their methods

```
public Class MainClass {

public static void main (String[] args) {
    MyClass.method1(); //no inputs to method1

    System.out.println("Enter your selection: ");
    int i = input.nextInt();
    int k;

    k = MyClass.method2(i); //the input to method2 is (i)

        }
}
```

Classes & Methods

- Method that has no input and no return value

```
public Class MyClass {

public static void method1 () {
    do something;

    // the empty parentheses, (), indicates no input value
    // The use of void indicates no return value
        }
}
```

Classes & Methods

- Method that has both an input and a return value

```
public Class MyClass {

public static returnValueType method2 (type inputVariable) {
       returnValueType returnVariable; // declare variable

       returnVariable = do something with inputVariable;

      return (returnVariable);

      /* this method takes inputVariable, manipulates it
       and sends back returnVariable  */
       }
}
```

Extra, unused slides

Lab Practice – if/else

- Let's practice using if then/else and loops. We'll test integers from 1 to 10 to determine if the number is even or odd and print it. HINT: We need to us % operator. Output:

  1 is odd

  2 is even

  3 is odd

  …

  10 is even

Our first program: Hello World

- print( ), println( )
- Use escape character to format output: \n \t

```
class HelloWorld{
    static public void main(String[] args){
        System.out.print("Hello");
        System.out.print(", World");
        System.out.print("\nHello\n");
        // \n means new      line
        System.out.println("Hello!");
        //println prints a line followed by a line break
        System.out.println();
        //this prints an empty line
        System.out.print("H\te\tl\tl\to");
        // \t means   tab
    }
}
```

Lab #1

- Edit your HelloWorld program to practice declaring and using Java primitive types.

1. declare an int named age, assign it a value.
2. declare a double named d, assign it a value.
3. declare a boolean named isCrazy, assign it a true value.
4. declare an char named exclaim, assign it a value '!'.
5. declare a string named name, assign it your name.
6. Create a print statement that outputs:

```
Hello Lori, You are 18 years old!
```

Shorthand Operators in Java

Java Shorthand	Equivalent Expanded Java
`x += 2; or x+=2;`	`x = x + 2;`
`x -= 3; or x-=3;`	`x = x - 3;`
`x *= 4; or x*=4;`	`x = x * 4;`
`x /= 5; or x/=5;`	`x = x / 5;`
`x %= 6; or x%=6;`	`x = x % 6;`

- You may code either way, code performance is not affected.

Lab

- Write new class named operTest
- Declare variables x and y.
- Set x to 1.
- Set y = ++x;
- Print “y = ++x is” + y.
- Set x = 1.
- Set y = x++;
- Print “y = x++ is” + y.

& 'AND' Example

```
int x = 5;
int y = 0;

if ((y > 0) & ((x / y) > 6))
     x = x + 1;
```

- Why is this a problem?

Short Circuit 'AND' Example

```
int x = 5;
int y = 0;

if ((y > 0) && ((x / y) > 6))
     x = x + 1;
```

- Why does this work better?

| 'OR' Example

```
boolean cash = true;
boolean credit = false;
boolean check = true;

if (cash | credit | check)
    //do something
```

- Why is this inefficient?

Short Circuit 'OR' Example

```
boolean cash = true;
boolean credit = false;
boolean check = true;

if (cash || credit || check)
    //do something
```

- Why is this more efficient?