Chapter 3

Flow of Control – Part 1

- Branching
- Loops
- exit(n) method
- **Boolean** data type and logic expressions

"Flow of Control" - control of the execution of instructions in a program

1. **Sequence** - execute next instruction
2. **Selection** - conditional selection of next
   – if, switch/case, try/catch
3. **Repetition** - repeat (loop) code block until a particular condition is met; either,
   – proceed to next (or first) instruction in block
   – jump back to beginning (first instruction) of block
   – exit block, continue with the next instruction after block

- Selection & Repetition are called *Branching Controls* (there may be more than one path)

Java Flow Control Statements

Sequence
» default; Java automatically executes the next instruction

Branching: Selection
» **if**, **if-else** [and **if-else if**]
» **switch/case**

Branching: Repetition (Looping)
» **while, for** ← "pre-check" loops
» **do-while** ← "post-check" loop

Definition of Boolean Values

- Branching
  » decisions are based on the outcome of a **boolean** expression that evaluates in either true or false
  » boolean expression also called a "condition"
- All branching controls utilise boolean expressions,
  » **if** (*boolean expr.*)
  » **while** (*boolean expr.*), **for** (__, *boolean expr.*, __)
  » **do** { } **while** (*boolean expr.*)
  » **switch/case** is special because is uses an *implicit* boolean expression (discussed later)

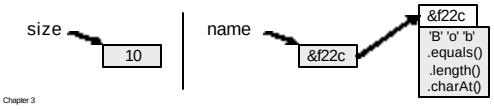
Comparison Operators

(each results in a boolean value)

In Math	Name	In Java	Java Examples
=	equal to	==	balance == 0 answer = 'y'
≠	not equal to	!=	income != tax answer != 'y'
>	greater than	>	income > outgo
≥	greater than or equal to	>=	points >= 60
<	less than	<	pressure < max
≤	less than or equal to	<=	income <= outgo

Variables and Objects: *memory!*

- a variable points directly to its data,
  » the variable directly stores its data
  » this is called a "direct reference" to the variable data
- an object identifier is a "reference variable" that stores a memory address as its data,
  » the memory address is the location where the object's data is stored (as variable values and methods)
  » this is called an "indirect reference" to the object data
- example: int size=10; String name="Bob";



Object Comparisons

- **==** does not work as expected for objects!!
  - » unlike variables, objects store memory addresses
  - » in using "==" to test if objects are "equal to each other," the test actually performed is,
    - – if *"the memory addresses stored by the object reference variables are the same" (is this ever true?!)*
  - » *note: all objects reference variables (regardless of the class they are instances of) have the same data type; and so can be compared together (but is this useful?)*
- Most classes/objects have a method to test for "data equality" rather than "memory address equality" to test if the *objects share similar data*

String Comparisons: special method

- for String objects, use their **.equals( )** method,

```
String s1 = "Mondo", s2;
s2 = SavitchIn.readLine();
if ( s1.equals(s2) )
  System.out.print("User typed 'Mondo');
else
  System.out.print("User did not type..");
```

  - » `s1.equals(s2)` is **true** both Strings have the same data; false otherwise
- **.equals()** is case sensitive, so to compare *meaning* an ignore case, use **.equalsIgnoreCase()**
  - » *how do you think .equalsIgnoreCase() works?*

Compound Boolean Expressions

- To build longer, more complex, boolean expressions,
  [boolean LHS] logic operator [boolean RHS]
  - » **&&** or **&** (AND) – both LHS and RHS must be true
  - » **||** or **|** (OR) – either LHS or RHS is true
- example: a test to see "if A is either equal to 0, or between the values of B and C (equal to neither)",
  `(A == 0) || (A < B && B < C)`
- parentheses are sometime not required but always added for clarity (*along with a comment!!)*
  - » see text (and later slides) for **Precedence rules**
  - » single & and | are used to avoid *short-circuit evaluation* and force *complete evaluation* of a boolean expression

`if` statement

- "do the next statement if test is true or skip it if false"
- Syntax:

```
if ( boolean_expression )
   action if true;  //only if true
next action;  //always executed
```

  - » indentation for *program readability*

```
if (eggsPerBasket < 12)  // if less than a dozen eggs
   System.out.println("Less than a dozen eggs per basket");

totalEggs = numberOfEggs * eggsPerBasket;
System.out.println("A total of "+ totalEggs + " eggs.");
```


More than 1 statement: Compound Statements:

- although the **if** can perform only a single statement, statements can be grouped together and treated as one
- these are called *compound statements*, *code blocks*, or just a *blocks*, and are defined by enclosing the statements in curly brackets (like the **main()** method)
- example:

```
if (eggsPerBasket < 12)  // if less than 12 eggs
{
   System.out.println("Less than a dozen ...");
   costPerBasket = 1.1 * costPerBasket;
} //end of if statement
```


Two-way Selection: `if-else`

- Select either one of two options,
  - » either perform the true statement block or the false statement block
- syntax:

```
if (Boolean_Expression)  // expression is true
{
   statement block if true
}
else  // assumption is that expression is false
{
   statement block if false
}

statements outside of/after if; always executed
```

`if-else` Examples

- example with single-statement blocks:

```
if(time < limit) // time less than deadline
   System.out.println("You made it.");
else
   System.out.println("Missed deadline.");
```

- example with compound statements:

```
if(time < limit) // time less than deadline
{
   System.out.println("You made it.");
   bonus = 100;   // sweet!!
}
else  // time is >= than deadline
{
   System.out.println("Missed deadline.");
   bonus = 0;     // doh!!
} // end of if
```


Multi-way if `if-else if`

- an extended form of "two-way if" (also called a *cascading if-else),*

```
if (score >= 90       // scr >= 90
   grade = 'A');
else if (score >= 80) // 80<=scr<90
   grade = 'B';
else if (score >= 70) // 70<=scr<80
   grade = 'C';
else if (score >= 60) // 60<=scr<70
   grade = 'D';
else                  // scr<60
   grade = 'E';
```

» to see how if works, match up each **else** to an **if**

Multibranch selection: `switch`

- an other branching technique, similar to multi-way if
- comments on switch/case
  - » the *controlling_expression* must be an integer data type: **byte, char, short, int, or long**
  - » controlling_expression and *case_label* have same type
  - » when a **break** statement is encountered, control immediately goes to the first statement after the **switch statement**,
    - – if no break is encountered to program proceeds directly to the next instruction in the following ***case*** statement
    - – the **switch/case** statement is very old (written for C/C++) but is very efficient after being compiled

`switch` Example

```
switch ( seatLocationCode )
{
 case 1:
    System.out.println("Orchestra");
    price = 40.00;
    break;
 case 2:
    System.out.println("Mezzanine");
    price = 30.00;
    break;
 case 3:
    System.out.println("Balcony");
    price = 15.00;
    break;
 default:
    System.out.println("Unknown seat");
    break;  // last statement; optional
}// end of switch()
```