Chapter 3

Flow of Control – Part 2

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

Repetition: Loops

- Structure of a general "loop":
  - *initialisation code (counting variable(s))*
  - *body of loop*
  - *loop continuation or termination condition*
- Types of "loops"
  - » counting loops
  - » sentinel-controlled loops ("flag" controlled)
  - » infinite loops
- Several programming statement variations
  - » **while & for** (perform at least zero loops)
  - » **do-while** (perform at least one loop)

`while`

- Syntax:

```
while(boolean_expression)
{
   //body of loop
   First_Statement;
   ...
   Last_Statement;
}
```

Something in body of loop should eventually cause the *`boolean_expression`* to become **`false`**.

- initialisation statements usually precede the loop
- while the **bool_expr** is true the loops continues; when it become false, the loop ends
- **while** can be used as either a counting loop or sentinel loop

`while`: a counting loop example

- A loop to sum 10 numbers entered by user

```
int next;  // next value from user
int count, total;  // loop counter & sum

//init loop
count = 1;   // init. Count to 1 (1st)
total = 0;   // init. Total to zero (empty)

while (count <= 10)  //count from 1 to 10
{ //get next value, add to total (sum)
   next = SavitchIn.readLineInt();
   total = total + next;   // summation
   count++;    //incr. loop counter
} // end of loop
```


`while`: a sentinel controlled loop example

- A loop to sum positive integers entered by the user
- `next` holds the user's input, sentinal
- loop terminates when a negative number is input

```
int next = 0;   // user input
int total = 0;  // summation

  // loop while next is not zero or neg.
while (next >= 0)
{
  total += next;   // sum
  next = SavitchIn.readLineInt();
}// end of loop
```

- *why is the summation before the data input?*

`while`: minimum of zero iterations

- **while** is a "pre-check" loop
- it checks whether to loop or not before performing the loop body**—it may never loop!**

```
int next = 0, int total = 0;
  // initial read, called "priming next"
next = SavitchIn.readLineInt();

while (next >= 0)  // loop while next >= 0
{
  total += next;
  next = SavitchIn.readLineInt();
}// end of loop
```

- if the first user value is negative—no loop.

do-while Loop

- syntax

```
do
{  //body of loop
   First_Statement;
   ...
   Last_Statement;
} while(Boolean_Expression);
```

Something in body of loop should eventually cause *Boolean_Expression* to be *false*.

- test for loop is after body so the loop must execute at least once
- may be either counting or sentinel loop
  - » good choice for sentinel loop

do-while Example

```
int count = 1;
int number = 10;
do //display 1 thru 10 on one line
{
   System.out.print(count + ", " );
   count++;
}
while (count <= number); // end of loop
```

- why is `System.out.print()` used rather than `System.out.println()`?

for Loop

- Excellent choice for counting loop!
- syntax includes: *initialisation*, *test,* and *incr.*
- syntax:

```
for (decl & init; bool_expr; incr/decr.)
{
   body;
}// end of for
```

- *note: the incr/decr. happens after the body of the loop is executed—**for** is similar to **while***

for Example

- Count down from 9 to 0,

```
     decl & init    test     decr.
for (int count=9; count>=0; count--)
{
   System.out.print("T = " + count);
   System.out.println(" and counting");
}

System.out.println("Blast off!");
```


Comments on Loops

- common loop errors:
  - » unintended infinite loops
  - » "off-by-one errors" in counting loops
- when a loop terminating condition is not met—result: **infinite loop**
  - » CLI programs in an infinite loop--press **^C** (Control-C)
- *loops should always be tested thoroughly*
  - » especially at the boundaries of the loop test, to check for off-by-one and other possible errors

The Type boolean

- only two values: **`true`** and **`false`**
- **Boolean** variables can,
  - » hold the result of a logical expression
  - » be used as expression in **if, while, do_while**, **for**
    - – in this case, storing the result of an expression and using the result more than once—*advantage?*

```
if ( systemsAreOK )
  System.out.println("Initiate launch sequence.");
else
  System.out.println("Abort launching sequence");
```

Truth Tables for `boolean` Operators

&&, & (and)

Value of A	Value of B	A && B
true	true	**true**
true	false	**false**
false	true	**false**
false	false	**false**

||, | (or)

Value of A	Value of B	A || B
true	true	**true**
true	false	**true**
false	true	**true**
false	false	**false**

! (not)

Value of A	!A
true	**false**
false	**true**

Precedence Rules

Highest Precedence

- » 1st : unary operators: +, -, ++, --, and !
- » 2nd: binary arithmetic operators: *, /, %
- » 3rd : binary arithmetic operators: +, -
- » 4th : boolean operators: <, >, =<, >=
- » 5th : boolean operators: ==, !=
- » 6th : boolean operator: & (full eval. and)
- » 7th : boolean operator: | (full eval. and)
- » 8th : boolean operator: && (short eval. and)
- » 9th : boolean operator: || (short eval. and)

Lowest Precedence

Short-Circuit Evaluation

- *Short-circuit evaluation:*
  - » "evaluate as much of an expression as necessary"
- example:

```
if ((assign > 0) && ((total/assign) > 60))
   System.out.println("Good work");
else
   System.out.println("Work harder.");
```

- if **(assign>0)** is false, the complete expression will also be false, regardless of the rest
- with && Java will not evaluate the full expression
- *short-circuit eval. would prevent a "divide-by-zero" exception when `assign` = 0.*

The `exit` Method

- if you have a program situation where it is pointless to continue execution you can terminate the program with the *`exit(n)`* method
- *n* is often used to identify to the OS if the program ended normally or abnormally
- *n* is *conventionally* 0 for "normal termination" and a non-zero for abnormal termination
  - » sometimes a non-zero indicates a special circumstance

`exit` Method Example

```
char userIn=' ';
System.out.println("e to exit, c to cont.");

userIn = SavitchInReadLineChar();

if (userIn == 'e')
   System.exit(0);
else if (userIn == 'c')
{
   //statements to do work
}
else
{
   System.out.println("Invalid entry");
   //statements to something appropriate
}
```


Summary

- selection: **if**, **if-else**, **if-else if**, and **switch**
- repetition (loop): **while**, **do-while**, and **for**
- loops are usually,
  - » counting loops (loop controlled by an inc./dec. value)
  - » sentinel controlled, terminating once a given value is seen
- any loop can be written with any of the three loop statements, but
  - » **while, do-while** are good choices for sentinel loops
  - » **for** is a good choice for counting loops

Summary

- unintended "infinite loops" can usually be terminated by entering *^C* (control-C) at the prompt
- the most common loop errors are unintended infinite loops and *off-by-one* errors in counting loops
- branching and loops are controlled by **boolean** expressions
  - » **boolean** expressions are either **`true`** or **`false`**
  - » **`boolean`** is a primitive data type in Java
- the **System.exit(n)** method terminates the running program
  - » required when using GUI in your application