**Administrative**

**Today’s session**

Homework 1 key

Ternary operator

*while* statement

*do-while* statement

Validation loop

Sentinel loop

Infinite loop

*for* statement

Nested loop

**Session Topics**

**Homework 1 key**

● The Homework 1 key is available on Blackboard.

**Ternary operator**

● A **ternary operator** is an expression.

● A ternary operator enables a branch to represent an expression or be placed within an expression.

● A ternary operator has a condition that is tested. If the condition is true, the operator returns the specified **true-value-expression**. If the condition is false, the operator returns the specified **false-value-expression**.

● A ternary operator has syntax:

<condition> ? <true-value-expression> : <false-value-expression>

**Where …**

**<condition>** is a Boolean expression.

**<true-value-expression>** and **<false-value-expression>** have data types compatible with the code the ternary operator is placed within. The data types of the two expressions do not have to match.

● Although not required, parentheses should be placed around the ternary operator to separate its logic from surrounding code.

● See **Ifs, switches, and ternary operators** sample application on Blackboard.

***while* statement**

● A **while statement** is an iterative (looping) statement with a condition and a block of code.

● A while statement is a **pre-test loop**: the condition is tested *before* the block of code executes.

● A while statement does **indefinite iteration**: the number of times it loops is unknown ahead of time.

● A while statement loops **zero or more times**.

● A while statement continues to execute the block of code while the condition is true.

**●** A while statement has syntax:

while (<condition>)

{

<block>

}

● *while* example:

int i = 1;

while (i <= 10)

{

System.out.println(“i is now ” + i);

i = i + 1;

**}**

***do-while* statement**

● A **do-while statement** is an iterative (looping) statement with a condition and a block of code.

● A do-while statement is a **post-test loop**: the condition is tested *after* the block of code executes.

● A do-while statement does **indefinite iteration**: the number of times it loops is unknown ahead of time.

● A do-while statement loops **one or more times**.

● A do-while statement continues to execute the block of code while the condition is true.

**●** A do-while statement has syntax:

do

{

<block>

} while (<condition>)

● *do-while* example:

int i = 1;

do

{

System.out.println(“i is now ” + i);

i = i + 1;

} while (i <= 10)

**Validation loop**

● A **validation loop** reads and checks values entered at the keyboard.

● A validation loop continues to loop until the user enters a valid value.

● A valid value is defined by the user and/or developer.

● A validation loop is implemented with a while or do-while statement. A while statement is preferred since the user can be told why their input was invalid.

● Validation loop example:

Scanner keyboard = new Scanner(System.in);

int month;

System.out.print("Enter a month (between 1 and 12): ");

month = keyboard.nextInt();

while (month < 1 || month > 12)

{

System.out.println(“Error: month must be between 1 and 12.”);

System.out.print("Enter a month (between 1 and 12): ");

month = keyboard.nextInt();

}

System.out.println(“Congrats! You entered a valid month!”);

**Sentinel loop**

● A **sentinel loop** reads input until a special value (the sentinel) is read from the keyboard.

● A sentinel value is defined by the user and/or developer.

● Sentinel loop example:

Scanner keyboard = new Scanner(System.in);

int number;

int sum = 0;

System.out.print("Enter a number (-99 to exit): ");

number = keyboard.nextInt();

while (number != -99)

{

sum = sum + number;

System.out.print("Enter a number(-99 to exit): ");

number = keyboard.nextInt();

}

System.out.println("The sum of the numbers entered is " + sum + ".");

● See **While loops** sample application on Blackboard.

**Infinite loop**

● An infinite loop is a loop that (unintentionally) never ends.

● Every loop must have:

✓ An initialization.

✓ A condition.

✓ An update.

● If any of these three are missing or incorrect, an infinite loop will result.

***for* statement**

● A **for statement** is an **iterative** (looping) statement with a condition and a block of code.

● A for statement is a **pre-test loop**: the condition is tested *before* the block of code executes.

● A for statement does **definite iteration**: the number of times it loops is known ahead of time.

● A for statement continues to execute the block of code while the condition is true.

**●** A for statement has syntax:

for (<initialization>; <condition>; <update>)

{

<block>

}

● *for* example:

int i;

for (i = 1; i <= 10; i++)

{

System.out.println(“i is now ” + i);

}

● See **For loops** sample application on Blackboard.

**Nested loop**

● A **nested loop** is one loop with another.

● Any loop type (*while*, *do-while*, *for*) may be placed within any other loop type (*while*, *do-while*, *for*).

● Nested loop example:

int i;

int j;

for (i = 1; i <= 2; i = i + 1)

{

for (j = 1; j <= 4; j = j + 1)

{

System.out.print(“\*”);

}

System.out.println();

}