Computer Programming I - CSC111

Chapter 4 – Flow of Control (Loops)

Dr. Mejdl Safran
mejdl@ksu.edu.sa
Outline

• The `while` statement
• The `do-while` statement
• The `for` statement
• The loop body
• Initializing statements
• Controlling loop iterations
• Loop bugs
• Tracing variables
Java loop statements

• A portion of a program that repeats a statement or a group of statements is called a loop.
• The statement or group of statements to be repeated is called the body of the loop.
• A loop could be used to compute grades for each student in a class.
• There must be a means of exiting the loop.
The **while** statement

- Also called a **while** loop
- A **while** statement repeats while a controlling boolean expression remains true
- The loop body typically contains an action that ultimately causes the controlling boolean expression to become false.
Syntax of the **while** statement

• Syntax

```plaintext
while(Boolean_Epression)
  Body_Statement
or
while(Boolean_Epression)
{
  First_Statement
  Second_Statement
  ...
}
```
Semantics of the **while** statement

```
while (Boolean_Expression)
    Body
```

Start → Evaluate Boolean_Expression → True -> Execute Body → False → End loop
Example

- Write a Java program that reads a positive number from the user and prints all numbers starting from 1 to the entered number.

```java
int number, count=1;
Scanner keyboard = new Scanner(System.in);
System.out.println("Enter a number:");
number = keyboard.nextInt();

while(count <= number)
{
    System.out.print(count + ", ");
    count++;
}
```

Sample Output

Enter a number:
5
1, 2, 3, 4, 5,
Action of the **while** statement

```java
while (count <= number) {
    System.out.print(count + ", ");
    count++;
}
```
Example

```
EXAMPLE

//Sum positive integers read until one is not positive
int total = 0;
int next = keyboard.nextInt();
while (next > 0) {
    total = total + next;
    next = keyboard.nextInt();
}
```
Another example

```java
public class StarsPrint {
    public static void main(String args[]) {
        int count=1, number=10;
        while (count++ <= number) {
            System.out.print("*");
        }
        System.out.println();
    }
}
```

************
Example

- Write Java statements that compute the product $1 \times 2 \times \ldots \times 10$ and print its value

```java
int product = 1, number = 1;
while (number <= 10) {
    product = product * number;
    number++;
}
System.out.println(product);
```

Output

3628800
Example

• Write Java statements that compute the sum 1+2+ ... +10 and print its value

```java
int sum = 0, number = 1;
while(number <= 10){
    sum = sum + number;
    number++;
}
System.out.println(sum);
```

Output

55
Caution

```java
int product = 1, number = 1;
while (number <= 10) {
    product = product * number;
    number++;
}
System.out.println("Product of the numbers 1 through 10 is "+ product);
```

Do not write a semicolon after the beginning of a while statement.
Nested loops

• The body of a loop can contain any kind of statements, including another loop.

• How many times will the string “Here” be printed?

```java
int count1 = 1, count2 = 1;
while(count1 <= 10){
    count2 = 1;
    while(count2 <= 20){
        System.out.println("Here");
        count2++;
    }
    count1++;
}
```

$10 \times 20 = 200$
Rectangle drawing example

Enter the length of the rectangle: 5
Enter the width of the rectangle: 5
#
#
#
#
#
#
#
#
#
#
#
#
#
#

Enter the length of the rectangle: 2
Enter the width of the rectangle: 4
#
#
#
#
import java.util.Scanner;
public class RectangleDrawing {
    public static void main(String[] args) {
        Scanner keyboard = new Scanner(System.in);
        int length, width, currentL=0, currentW=0;

        System.out.print("Enter the length of the rectangle: ");
        length = keyboard.nextInt();
        System.out.print("Enter the width of the rectangle: ");
        width = keyboard.nextInt();

        if(length>0 && width>0)
        {
            while(currentL<length)
            {
                currentW=0;
                while(currentW<width)
                {
                    System.out.print("# ");
                    currentW++;
                }
                System.out.println();
                currentL++;
            }
        }
    }
}
import java.util.Scanner;
public class RectangleDrawing {
    public static void main(String[] args) {
        Scanner keyboard = new Scanner(System.in);
        int length, width, currentL=0, currentW=0;

        System.out.print("Enter the Length of the rectangle: ");
        length = keyboard.nextInt();
        System.out.print("Enter the width of the rectangle: ");
        width = keyboard.nextInt();

        if(length>0 && width >0)
            while(currentL++<length)
                {currentW=0;
                while(currentW++<width)
                    System.out.print("# ");
                    System.out.println();
                }
    }
}
The **do-while** statement

- Also called a **do-while** loop
- Similar to a **while** statement, except that the loop body is executed at least once
- Syntax
  ```
  do
    Body_Statement
  while (Boolean_Expression);
  ```
- Don’t forget the semicolon!
The **do-while** statement

- First, the loop body is executed.
- Then the boolean expression is checked.
  - As long as it is true, the loop is executed again.
  - If it is false, the loop is exited.
- Equivalent **while** statement
  
  ```
  Statement(s)_S1
  while (Boolean_Condition)
   Statement(s)_S1
  ```
Semantics of the **do-while** statement

```
do
    Body
while (Boolean_Expression)
```
Example

```java
int number, count=1;
Scanner keyboard = new Scanner(System.in);
System.out.println("Enter a number:");
number = keyboard.nextInt();

do
{  
    System.out.print(count + ", ");
    count++;
} while(count <= number);
```

Sample Output

Enter a number: 0
0
1,
Action of the **do-while** statement

```java
do {
    System.out.print(count + ", ");
    count++;
} while (count <= number);
```
Example

- A great use for the Do-While loop is the menu: The menu is displayed continuously until the end user picks a value. If they type a certain menu item, then the code stops.

***** MAIN MENU *****
1. Print Good Morning!
2. Print Good Evening!
3. Print Good Bye! and Exit the program

Select an Option:
1
Good Morning!

***** MAIN MENU *****
1. Print Good Morning!
2. Print Good Evening!
3. Print Good Bye! and Exit the program

Select an Option:
2
Good Evening!

***** MAIN MENU *****
1. Print Good Morning!
2. Print Good Evening!
3. Print Good Bye! and Exit the program

Select an Option:
3
Good Bye!
Example

```java
int userOption;
Scanner keyboard = new Scanner(System.in);
    do {
        System.out.println("***** MAIN MENU *****");
        System.out.println("1. Print Good Morning!");
        System.out.println("2. Print Good Evening!");
        System.out.println("3. Print Good Bye! and Exit the program");
        System.out.println("\nSelect an Option: ");
        userOption = keyboard.nextInt();
        if(userOption == 1)
            System.out.println("Good Morning!");
        else
            if(userOption == 2)
                System.out.println("Good Evening!");
            else
                if(userOption == 3)
                    System.out.println("Good Bye!");
        } while(userOption != 3);
```
while vs. do-while

```java
public class WriteWhileAndDoWhileLoops {
    public static void main (String[] args) {
        int i=0;
        System.out.println("Try while loop:");
        while (i < 5) {
            System.out.println("Iteration " + ++i);
        }
        System.out.println("Try do while loop:");
        i=0;
        do {
            System.out.println("Iteration " + ++i);
        } while (i < 5);
    }
}
```

```java
public class WriteWhileAndDoWhileLoops {
    public static void main (String[] args) {
        int i=0;
        System.out.println("Try while loop:");
        while (i < 5) {
            System.out.println("Iteration " + ++i);
        }
        System.out.println("Try do while loop:");
        i=0;
        do {
            System.out.println("Iteration " + ++i);
        } while (i < 5);
    }
}
```
Infinite loops

• A loop which repeats without ever ending is called an infinite loop.

• If the controlling boolean expression never becomes false, a **while** loop or a **do-while** loop will repeat without ending.

```java
int count = 1;
while(count <= 25){
    System.out.println(count);
    count--;
}
```
Exam average calculation example

• Compute the average of a list of (nonnegative) exam scores
• Repeats computation for more exams until the user says to stop
import java.util.Scanner;

/**
 * Computes the average of a list of (nonnegative) exam scores. Repeats computation for more exams until the user says to stop.
 */

public class ExamAverager {
    public static void main(String[] args) {
        System.out.println("This program computes the average of");
        System.out.println("a list of (nonnegative) exam scores.");
        double sum;
        int numberOfStudents;
        double next;
        String answer;
        Scanner keyboard = new Scanner(System.in);
do
{
    System.out.println();
    System.out.println("Enter all the scores to be averaged.");
    System.out.println("Enter a negative number after");
    System.out.println("you have entered all the scores.");
    sum = 0;
    numberOfStudents = 0;
    next = keyboard.nextDouble();
    while (next >= 0)
    {
        sum = sum + next;
        numberOfStudents++;
        next = keyboard.nextDouble();
    }
    if (numberOfStudents > 0)
        System.out.println("The average is "+
                        (sum / numberOfStudents));
    else
        System.out.println("No scores to average.");
    System.out.println("Want to average another exam?");
    System.out.println("Enter yes or no.");
    answer = keyboard.next();
    while (answer.equalsIgnoreCase("yes"));
}
Sample Screen Output

This program computes the average of a list of (nonnegative) exam scores.

Enter all the scores to be averaged. Enter a negative number after you have entered all the scores.

100
90
100
90
-1

The average is 95.0

Want to average another exam?
Enter yes or no.

yes

Enter all the scores to be averaged. Enter a negative number after you have entered all the scores.

90
70
80
-1

The average is 80.0

Want to average another exam?
Enter yes or no.

no
The **for** Statement

- A **for** statement executes the body of a loop a fixed number of times.

- Example
  ```java
  for (count = 1; count < 3; count++)
    System.out.println(count);
  ```
The **for** Statement

- **Syntax**
  
  ```
  for (Initialization, Condition, Update) 
  Body_Statement
  ```

- **Body_Statement** can be either a simple statement or a compound statement in `{}`.
The **for** statement

- Corresponding **while** statement

\[
\text{Initialization} \\
\text{while (Condition)} \\
\text{Body Statement Including Update}
\]
Semantic of the **for** statement
Example

class ForDemo
{
    public static void main(String[] args)
    {
        for(int i=1; i<11; i++)
        {
            System.out.println("Count is: " + i);
        }
    }
}
Example

```java
public class ForDemo {
    public static void main(String[] args) {
        int countDown;
        for (countDown = 3; countDown >= 0; countDown--){
            System.out.println(countDown);
            System.out.println("and counting.");
        }
        System.out.println("Blast off!");
    }
}
```

**Screen Output**

3
and counting.
2
and counting.
1
and counting.
0
and counting.
Blast off!
Action of the **for** loop

```
for (countDown = 3; countDown >= 0; countDown--)
{
    System.out.println(countDown);
    System.out.println("and counting.");
}
```
The **for** statement

• Possible to declare variables within a **for** statement

```java
int sum = 0;
for (int n = 1 ; n <= 10 ; n++)
    sum = sum + n * n;
```

• Note that variable **n** is local to the loop
The `for` statement

• A comma separates multiple initializations

• Example

  ```
  for (n = 1, product = 1; n <= 10; n++)
      product = product * n;
  ```

• Only one boolean expression is allowed, but it can consist of `&&` s, `||` s, and `!` s.

• Multiple update actions are allowed, too.

  ```
  for (n = 1, product = 1; n <= 10;
       product = product * n, n++);
  ```
The **for** statement

Do not write a semicolon after the beginning of a for statement

```java
for (number = 1; number <= 10; number++) {
    product = product * number;
}
```

```java
int product = 1;
for (int number = 1; number <= 10; number++)
    product = product * number; //Invalid
```

```java
for (int n = 1; n <= 10; n++)
    sum = sum + n * n;
System.out.println(n); //Invalid
```
What does it display? How would you fix it?

```java
int product = 1;
int max = 20;
for (int i = 0; i <= max; i++)
    product = product * i;
System.out.println("The product is " + product);

int sum = 0;
int product = 1;
int max = 20;
for (int i = 1; i <= max; i++)
    sum = sum + i;
    product = product * i;
System.out.println("The sum is " + sum + " and the product is " + product);
```
Convert it to while loop

```java
int s = 0;
int t = 1;
for (int i = 0; i < 10; i++)
{
    s = s + i;
    for (int j = i; j > 0; j--)
    {
        t = t * (j - i);
    }
    s = s * t;
    System.out.println("T is "+ t);
}
System.out.println("S is "+ s);
```
The loop body

• To design the loop body, write out the actions the code must accomplish.

• Then look for a repeated pattern.
  • The pattern need not start with the first action.
  • The repeated pattern will form the body of the loop.
  • Some actions may need to be done after the pattern stops repeating.
Initializing statements

• Some variables need to have a value before the loop begins.
  • Sometimes this is determined by what is supposed to happen after one loop iteration.
  • Often variables have an initial value of zero or one, but not always.

• Other variables get values only while the loop is iterating.
Controlling number of loop iterations

• If the number of iterations is known before the loop starts, the loop is called a *count-controlled loop.*
  • Use a *for* loop.
• Asking the user before each iteration if it is time to end the loop is called the *ask-before-iterating technique.*
  • Appropriate for a small number of iterations
  • Use a *while* loop or a *do-while* loop.
Controlling number of loop iterations

• For large input lists, a sentinel value can be used to signal the end of the list.
  • The sentinel value must be different from all the other possible inputs.
  • A negative number following a long list of nonnegative exam scores could be suitable.

90 0
10
-1
Controlling number of loop iterations

• Example - reading a list of scores followed by a sentinel value

```java
int next = keyboard.nextInt();
while (next >= 0) {
    Process_The_Score
    next = keyboard.nextInt();
}
```
Controlling number of loop iterations

• Using a boolean variable to end the loop

```java
import java.util.Scanner;
/**
 * Illustrates the use of a boolean variable to end loop iteration.
 */
public class BooleanDemo {
    public static void main(String[] args) {
        System.out.println("Enter nonnegative numbers.");
        System.out.println("Place a negative number at the end");
        System.out.println("to serve as an end marker.");
        int sum = 0;
        boolean areMore = true;
        Scanner keyboard = new Scanner(System.in);
        while (areMore) {
            int next = keyboard.nextInt();
            if (next < 0)
                areMore = false;
            else
                sum = sum + next;
        }
        System.out.println("The sum of the numbers is "+sum);
    }
}
```

Sample Screen Output

Enter nonnegative numbers.
Place a negative number at the end to serve as an end marker.
1 2 3 -1
The sum of the numbers is 6
Programming example

• Spending Spree
  • You have $100 to spend in a store
  • Maximum 3 items
  • Computer tracks spending and item count
  • When item chosen, computer tells you whether or not you can buy it

• Client wants adaptable program
  • Able to change amount and maximum number of items
```java
import java.util.Scanner;
public class SpendingSpree
{
    public static final int SPENDING_MONEY = 100;
    public static final int MAX_ITEMS = 3;
    public static void main(String[] args)
    {
        Scanner keyboard = new Scanner(System.in);
        boolean haveMoney = true;
        int leftToSpend = SPENDING_MONEY;
        int totalSpent = 0;
        int itemNumber = 1;
        while (haveMoney && (itemNumber <= MAX_ITEMS))
        {
            System.out.println("You may buy up to " +
                                (MAX_ITEMS - itemNumber + 1) +
                                " items");
            System.out.println("costing no more than $" +
                                leftToSpend + ".");
            System.out.print("Enter cost of item #" +
                             itemNumber + ": ");
            int itemCost = keyboard.nextInt();
            if (itemCost <= leftToSpend)
            {
                System.out.println("You may buy this item.");
                totalSpent = totalSpent + itemCost;
                System.out.println("You spent $" + totalSpent +
                                    " so far.");
                leftToSpend = SPENDING_MONEY - totalSpent;
                if (leftToSpend > 0)
                {
                    itemNumber++;
                }
            }
            else
            {
                System.out.println("You are out of money.");
                haveMoney = false;
            }
        }
    }
```
else
    System.out.println("You cannot buy that item.");
}
System.out.println("You spent "+ totalSpent + ", and are done shopping.");
}

---

**Sample Screen Output**

You may buy up to 3 items costing no more than $100.
Enter cost of item #1: $80
You may buy this item.
You spent $80 so far.
You may buy up to 2 items costing no more than $20.
Enter cost of item #2: $20
You may buy this item.
You spent $100 so far.
You are out of money.
You spent $100, and are done shopping.
Tracing variables

- *Tracing variables* means watching the variables change while the program is running.
  - Simply insert temporary output statements in your program to print the values of variables of interest
  - Or, learn to use the debugging facility that may be provided by your system.
Loop bugs

• Common loop bugs
  • Unintended infinite loops
  • Off-by-one errors
  • Testing equality of floating-point numbers

• Subtle infinite loops
  • The loop may terminate for some input values, but not for others.
  • For example, you can’t get out of debt when the monthly penalty exceeds the monthly payment.