Loop variations

do-while and for loops

Do-while loops

- Slight variation of while loops
- Instead of testing condition, then performing loop body, the loop body is performed first, then condition is tested
- A do-while loop is guaranteed to perform one iteration, because the validity of the condition is not known until after the iteration is complete

Syntax for do-while loop

do {
   /* loop body statements */
} while (expression);

Notes:
- do-while loops are almost always event-controlled
- Often not necessary to initialize loop control
- Good idea to keep end bracket and while(expression) on same line
Applications for do-while loops

- Often used to display an initial menu of choices, one of which is "quit program" – you want user to see this at least once, even if the option they pick is to quit
- Useful for checking validity of input value – eliminates having to prompt twice for same data (see examples, next two slides)

Validating input – while loop

Scanner kb = new Scanner(System.in);
int x;
System.out.print("Enter a number between 1 and 100: ");
x = kb.nextInt();
while(x < 1 || x > 100) {
    System.out.println("Value out of range");
    System.out.print("Enter a number between 1 and 100: ");
x = kb.nextInt();
}

Validating input – do-while loop

Scanner kb = new Scanner(System.in);
int x;

do {
    System.out.print("Enter a number between 1 and 100: ");
x = kb.nextInt();
    if(x < 1 || x > 100)
        System.out.println("Value out of range");
} while(x < 1 || x > 100);
while loop

- **PRE-TEST loop** (entry-condition)
- The looping condition is tested before executing the loop body.
- Loop body may not be executed at all.

---

do-while loop

- **POST-TEST loop** (exit-condition)
- The looping condition is tested after executing the loop body.
- Loop body is always executed at least once.

---

For loops

- Count-controlled loops are so common in programming that most languages have a special set of syntax designed specifically for this construct
- In Java, the special syntax for a count-controlled loop is called a **for loop**
Count-controlled loops

- Count-controlled while loop:
  - initialize counter before loop starts
  - test counter in loop control expression
  - update counter inside loop body
- A for loop is a stylized version of a count-controlled while loop; all of the elements above appear at the top of the loop

For Loop Syntax

```java
for (initialization ; test expression ; update )
{
  0 or more statements to repeat
}
```

Count-controlled while loop example

```java
int x=0;  /* step 1: initialize counter */
while (x < 100) /* step 2: test counter value */
{
    System.out.println("I will be a good student");
    x++;   /* step 3: increment counter */
}
```
Same logic using for loop

```java
for (int x=0; x<100; x++)
    System.out.println("I will be a good student");
```

Step 1: initialize loop counter (performed once)
Step 2: test counter value (performed once for each iteration)
Step 3: increment loop counter (performed once per iteration)

Body of loop is performed between steps 2 and 3, just as in the while loop version

Notes on for loop

- Just stylized version of while loop; condition test occurs before each iteration
- Can contain single statement (making brackets unnecessary) because increment occurs in loop heading
- Each section of loop heading can contain multiple parts, separated by commas; each section can also be omitted

Example using decrement

```java
for (int count = 4 ; count > 0 ; count-- )
{
    System.out.println("" + count);
}
System.out.println("Done");
```

OUTPUT:

```
4
3
2
1
Done
```
Write a loop to produce the following output:

1Potato
2Potato
3Potato
4
5Potato
6Potato
7Potato
More

What is output?

```java
int count=0;
for (; count < 10 ; count++ )
{
    System.out.println("*");
}
```

A. 10 stars
B. 11 stars
C. 1 star
D. No stars

What is output?

```java
int count=0;
for (; count < 10 ; count++ )
{
    System.out.println("*");
}
```
Example – multiplication table

The loop displays a specified multiplication table. For example, if the user enters 6, the program displays this table:

1 x 6 = 6
2 x 6 = 12
3 x 6 = 18
.
.
12 x 6 = 72

import java.util.*;

public class Mtable {
    public static void main(String [] args) {
        int value;
        Scanner kb = new Scanner(System.in);
        System.out.print("Enter value for table: ");
        value = kb.nextInt();
        for (int ct = 1; ct <= 12; ct++)
            System.out.println("" + ct + " x " + value + " = " + ct * value);
    }
}

A “safe” input routine for numbers

- Input streams can get “clogged” by bad data – like the user entering a letter when a number is expected
- Using do-while loops, we can write an input routine for int or floating-point numbers that can’t be “clogged"
**Code for safe integer input**

```java
int n=0, sct =0;
char c;
Scanner kb = new Scanner(System.in);
System.out.print("Enter a whole number: ");
String s = kb.nextLine();
do {
    c = s.charAt(sct);
    if (c >= '0' && c <= '9')
        n = n * 10;
    n += (int)(c - '0');
    sct++;
} while(sct < s.length());
```

**Safe floating-point input**

- How would you do it?
- One approach:
  - 2 loops: first reads whole part of number, second reads fractional part
  - When both loops are finished, assemble the number from the sum of the two
  - How would we deal with numbers that don't have a whole part?