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

```plaintext
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
- Looping condition tested **before** execution of loop body
- Loop body may not be executed at all
Do-while loop

- Post-test loop
- Loop condition is checked AFTER execution of loop body
- Loop body guaranteed to execute at least once
A “safe” input routine for numbers

- Input streams can be “clogged” by bad data – e.g. the user enters 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
```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:
  – two loops: first reads part before decimal point, second reads part after decimal point
  – When both loops are finished, assemble number
  – What if number doesn’t have a whole part?
For Loop Syntax

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

1 Potato
2 Potato
3 Potato
4
5 Potato
6 Potato
7 Potato
More
What is output?

```java
int count=0;

for (; count < 10 ; count++)
{
    System.out.println("*");
}
```
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:

\[
\begin{align*}
1 \times 6 &= 6 \\
2 \times 6 &= 12 \\
3 \times 6 &= 18 \\
&\vdots \\
12 \times 6 &= 72 \\
\end{align*}
\]
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);
    }
}