Repetition using Java – `do` Statements

The following is a practical example to illustrate the `while statement`.

```
while (there are lumps in the gravy)
  give the gravy a stir
end while
```

First we check for lumps. If there are any lumps, start stirring. The idea is to keep checking for lumps, and if there are still lumps, stir the gravy. If the lumps are gone, the loop ends.

In the kitchen, you might not be able to detect the lumps just by looking. In that case, you might start by giving the gravy a stir, and then checking if any lumps are still present. Our example *pseudocode* would have to change slightly to accomplish this new behaviour.

```
start
  give the gravy a stir
while (there are lumps in the gravy)
```

The Do Loop – Decision at the End

The `do` statement is similar to the `while` statement. Both check a *boolean expression*, and if the expression is `true`, the loop continues. Once the statement becomes `false`, the loop exits.

```
while (<boolean expression>)
{
  <statements>
}
```

```
do
{
  <statements>
} while (<boolean expression>);
```

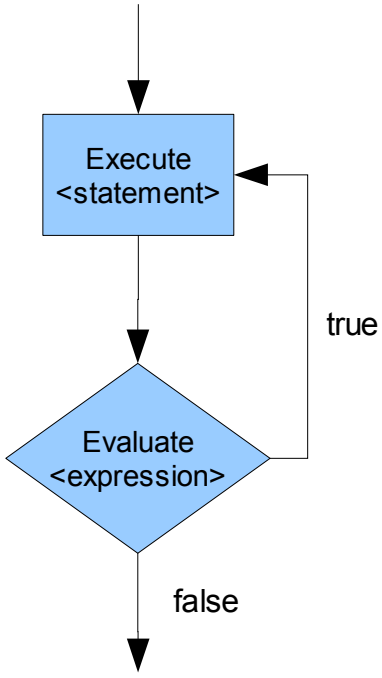
Note the semicolon (;) at the end of the `do` statement.

We can also represent a `do` loop using a *flowchart*, as we did with `while` statements.

Since the `do` statement executes the statement(s) in its body before evaluating the boolean expression, the loop will always be entered at least once.

This makes the `do` statement particularly useful in situations where we know we want to perform the loop at least once.

Otherwise, it is generally interchangeable with the `while` statement.



Repetition using Java – `do` Statements

Example 1 – Ask the user for integer values, and then outputs the square of the value. Continues to do this until the user enters zero, which causes the loop to exit. Compare the code containing while and do statements which yield similar, but not identical, results.

while statement – The user is asked for a number before entering the loop to print out a value. If the number is zero, nothing will be printed.

```
System.out.print("Give an integer (zero to stop): ");
int value = In.getInt();
while (value != 0)
{
   System.out.println(value + "   " + value*value);
   System.out.print("Next integer (zero to stop): ");
   value = In.getInt();
}
```

do statement – The user is asked for a value inside the loop, which means something will be printed no matter what they enter. Also notice that the order of the statements inside the loop had to be rearranged.

```
do
{
   System.out.print("Give an integer (zero to stop): ");
   value = In.getInt();
   System.out.println(value + "   " + value*value);
} while (value != 0);
```

Example 2 – This fragment forces a user to supply a value that lies in the range from one to ten. If the user provides a value outside this range, they are asked to enter their value again (and again) until they get it right. That is why the *boolean expression* is the negative of what we want the user to enter.

```
int value;
do
{
     System.out.println("Give an integer from one to ten");
     value = In.getInt();
} while (value < 1 || value > 10);
```

Exercises

1. For what kind of looping situation is a `do` usually preferable to a `while`?

2. What would be output by the program fragment shown below is it were given the following input sequence?

input	code fragment

-5
0
25
	`System.out.println("How many items to add?");`
	`do`
	`{`
	`howMany = In.getInt();`
	`if (howMany <= 0)`
	`{`
	`System.out.println("Give a positive value.");`
	`}`
	`} while (howMany <= 0);`

3. Write a fragment that forces a user to supply either `'y'` or `'n'` in response to the question, `"Continue? Respond with y or n"`.

4. Write a program fragment that asks the user to supply a letter of the alphabet, rejecting responses until the user gives either an upper or a lower case letter of the alphabet (i.e., rejecting numbers, punctuation, or spaces).

5. Write a program that first forces the user to supply a positive integer and then prints the number and the sum of its digits.

6. Write a program that asks the user for the secret password until they provide it.

Solutions

1. When you must enter the loop at least once.
2. Give a positive value
   Give a positive value
3. ```
   char answer;
   do
   {
     System.out.println("Continue? Respond with y or n");
     answer = In.getChar();
   }
   while (answer != 'y' && answer != 'n');
   ```
4. ```
   class DoLoopExercise4
   {
     public static void main(String [] args)
     {
       char letter;
       do
       {
         System.out.println("Give a letter of the alphabet");
         letter = In.getChar();
       }
       while ( ! (('A'<= letter && letter <= 'Z') ||
                  ('a' <= letter && letter <= 'z')));
       // letters are between 'A' and 'Z', or 'a' and 'z'
       // so we stay in the loop when we DON'T have a letter
     }
   }
   ```
5. ```
   class DoLoopExercise5
   {
     public static void main(String [] args)
     {
       int number, lowDigit;
       int sum = 0;
       System.out.println("Give a POSITIVE integer");
       number = In.getInt();
       System.out.print(number + " has a sum of digits = ");
       do
       {
         // divide any number by 10, and the remainder is the
         // number in the ones column
         lowDigit = number % 10;
         sum += lowDigit;
         // integer division by 10 will move the decimal place
         // left by one, and drop the decimal value
         number = number / 10;
       }
       while (number > 0);
       System.out.println(sum);
     }
   }
   ```
   ```

6.
```
class DoLoopExercise6
{
  public static void main(String [] args)
  {
    String userPassword = "";
    String actualPassword = "please";
    do
    {
      System.out.print("What is the password? ");
      userPassword = In.getString();
    }
    while (!userPassword.equals(actualPassword));
    System.out.println("Password accepted");
  }
}
```