09 Repeating Actions with Loops

[Close all sections] Only visible sections will be included when printing this document.

References: L&L 5.1, 5.3–5.4, 6.3–6.4

▽ Repetition statements

Repetition statements (like conditional statements) change the flow of execution, by allowing us to repeat a set of statements multiple times. They are often just called *loops*.

Whether or not the statements are executed (or executed again) is controlled by a boolean expression (just like a conditional statement uses a boolean expression to determine whether or not to execute a block of statements at all).

Pre- and post-tested loops

Loops can be pre-tested or post-tested:

- Pre-tested: The boolean expression is evaluated at the beginning, so the loop body may never be executed. This kind of loop will repeat zero or more times.
- Post-tested: The boolean expression is evaluated at the end of the loop. This look will be executed one or more times.

Loop Components (IBUT)

All loops have four components:

- **I**nitialisation:
  - Preparing for the *first* step of repetition
  - May declare additional variables to control loop
- **B**ody:
  - The *work* done inside the loop
  - Same *kind* of action every time, but applied to different data or in a different state (of variable values)
- **U**pdate:
  - Something must *change* as a result of the loop (a counter, position in a list, etc.)
- **T**est:
  - Should we continue to repeat, or even start?
  - The **update** affects the **test**

▽ Loop Constructs in Java

while

Executes some statements *while* something is true.

- Pre-tested
- Syntax:

```
while ( boolean expression ) {
    statement(s)
}
```

- Initialisation must be done outside the loop (before the `while`)
- Update must be done inside the loop

Real-life example: cleaning the dishes

- **Initialisation**: Walk to the sink
- **Test**: Are there any dirty dishes left? Washing dishes is a *pre-tested* loop because there may not be any dishes to wash
- **Body**: Take next dish and wash it (this is also the **Update** because it affects the thing we test)

do-while

Executes some statements *at least once* and then continues to repeats them *while* something is true.

- Post-tested
- Syntax:

```
do {
    statement(s)
} while ( boolean expression );
```

- Initialisation must be done outside the loop (before the `do`)
- Update must be done inside the loop

Real-life (and computer) example: guess a number

In this game the player must always make at least one guess.

- **Initialisation**: Game selects a secret number
- **Body**: Ask player for their next guess.
  Give them some feedback if it's wrong ("higher", "lower"). This is also the **Update** because…
- **Test**: Is their guess incorrect? If it is then repeat the body of the loop.

for

Useful for executing some statements *a fixed number of times*.

- Pre-tested
- Syntax:

```
for ( initialisation ; boolean expression ; increment ) {
    statement(s)
}
```

- Initialisation done inside the `for` statement.
  - Can only be an assignment or declaration and assignment (e.g., `int i = 0`)
- The Update is done in *increment* (e.g., `i = i + 1`)

`for` loops are used most commonly to perform actions a fixed number of times. For instance, to repeat some statements 10 times, we would write:

```java
for (int i = 0; i < 10; i = i + 1) {
    /* statements to execute */
}
```

Real-life example: lifting weights

The task is to lift a weight 15 times.

- **Initialisation**: Set counter to 0
- **Test**: Is counter still less than 15?
- **Body**: Lift weight once
- **Update**: Increment counter

Alternative ways to increment an int

Java contains some shorthands for modifying a primitive variable's value.

1. A statement of the form `count = count + 1` can be rewritten as `count += 1`. This works for all compatible binary operators (operators that take two operands): +, -, / and *.
2. To increment an `int` (called `count` in this example) by 1 you can also write `count++;`
   - This is the same as `count = count + 1`

▽ ☑ Test Yourself: Trace loop code

For each of the following create a tracing table on paper (as there's no need to keep it) to help you answer the questions about each loop. Every time a line is repeated simply add another row to the table with the same line number.

Activity: A little counting

What is the value of `a` after the following loop has executed?

```java
int a = 1;

while (a < 10) {
    a = a + 1;
}
```

Hide Solution

10, as when `a` becomes 10 the loop condition becomes false and the loop ends.

Activity: Repeated multiplication

Read the following Java statements.

```java
final int TIMES = 3;
int n = 1;

for (int i = 0; i < TIMES; i++) {
    n = n * 2;
}

System.out.println("After loop, n is " + n);
```

How many times will the `for`-loop be executed?

Hide Solution

Three times, for `i` equal to 0, 1 and 2.

What would the output be if the statements were executed?

Hide Solution

The output is "After loop, n is 8", because the loop executes three times, each time doubling the previous value of `n`.

Activity: Disappearing cat

This one is a *little* harder. What is the output from executing the following Java statements? (Predict first, then run them in a program to find out.)

```java
1  String s = "cat";
2
3  while (s.length() > 0) {
4      System.out.println(s);
5      s = s.substring(1, s.length());
6  }
```

Hide Solution

First identify the stopping condition: when the String `s` is empty. Next, identify the statement inside the loop that *updates* the value of `s`: the assignment and `substring()` method call at line 5.

Check your notes to see what `substring()` does to see how `s` is changed each time through the loop. Then you'll see why the output is:

Hide Solution

```
cat
at
t
```

Activity: Countdown

Examine the following code and determine what will be displayed at the end:

```java
final int START = 10; //countdown starts at 10
final int END = 0;    //countdown ends on zero
String output = "";   //output message we will construct

for (int i = START; i >= END; i--) {
    //a += b; is the same as a = a + b;
    output += i + ", ";
}
output += "blast off!"; //add final message

System.out.println(output);
```

Hide Solution

First, what's happening with the loop? It's counting backwards from 10, all the way down to zero (because the test `>=` includes the value of `END`). So `i` will be 10, 9, 8, …, 0.

What's happening inside the loop?

Hide Solution

Each time through the loop it appends the value of `i` and the text ", " to the end of `output`. Here's how the value of `output` changes at each iteration:

i	Value of `output` after loop body executes
10	`"10, "`
9	`"10, 9, "`
8	`"10, 9, 8, "`
…	
1	`"10, 9, 8, 7, 6, 5, 4, 3, 2, 1, "`
0	`"10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0, "`

So what is printed in the end?

Hide Solution

```
10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0, blast off!
```

Because just after the loop the code also appends the text "blast off!" to the String that it has been constructing.

There are some additional code tracing problems involving loops in the appendix of code tracing problems.

▽ Tips for nesting loops (and other constructs)

- Statements in a branch or loop can be *any* statements, including other branches and loops
- The language has no limit to depth, but it's good programming practice to have a maximum of 4 levels
- Syntax issues
  - Inner statement must be complete
- Tips to help
  - Use blocks `{ }`
  - Code incrementally

▽ Loop Design Considerations

- Start with the problem
  - Start state
  - Goal state
- Consider possible algorithms in terms of I B U T
- Choose a suitable loop
  - Is it pre- or post-tested?
  - What kind of test?
    - counter: usually a special variable set up to be the counter (can count forwards or backwards depending on what suits the problem)
    - sentinel value: a special (input) value that marks the end (this is most often used for data input)
    - general condition: the most general test. It is sometimes helpful to think about "what makes the loop continue?" or "what makes the loop stop?"
    - flag: sometimes there are several conditions that will make the loop stop. When this is the case it may be a good idea to set up a boolean variable (the "flag") that controls the loop

▽ Troubleshooting loops

When programs with loops aren't working ask yourself:

Where is the problem?

Symptom	Check
Not entering the loop	Initialisation & Test
Not leaving the loop	Update & Test
Loop producing wrong effect	Body

What is causing it?

Type of error	Indicator/indicates
Syntax error	won't compile
Conceptual error	possibly the wrong construct, wrong kind of loop, or wrong test
Strategic error	wrong algorithm

Tip: To help diagnose problems (in any program), it can help to insert "debugging" statements, such as `System.out.println("Entered the outer loop")`. These help to trace the flow of execution.

▽ ☑ Test Yourself: Debug some loops

These activities should be attempted after you have implemented some loops yourself so that you are familiar with the syntax and the expected behaviour. You will need access to DrJava.

First, create this skeleton program in DrJava. For each activity below replace all code between the two `println`s with the given declarations and statements. Save each program as you go into a separate file, remembering to change the class name to match the file name.

```java
import java.util.Scanner;

public class LoopTest {

    public static void main(String[] args) {
        System.out.println("Program has started");

        //Replace this with the code to test

        System.out.println("Program about to end");
    }
}
```

Activity: Sum of the first *n* integers

Paste the following code into the indicated part of the skeleton program. Compile it, run it, and then answer the questions below.

```java
final int N = 4; //sum up to this value
int total = 0;   //stores sum

//Calculates the sum of the first N positive integers
for (int i = 1; i <= N; i++) {
    total = i;
}
System.out.println("Sum of the first " + N + " positive integers is " + total);
```

- What happens when the program runs?
  Hide Solution
  It prints the wrong answer. 1 + 2 + 3 + 4 = 10, but it displays 4.
- Where are good places to look for the cause of the error?
  Hide Solution
  Try the body first, because the behaviour of the loop is incorrect.
- What's the problem?
  Hide Solution
  The line `total = i;` replaces the value of `total` rather than adding to it.
- What's a suitable correction?
  Hide Solution
  Modify the body to be `total += i;` or `total = total + i;`.

Implement a correction and rerun to test.

Activity: Countdown timer

Now test the following loop code in the test program. Compile it, run it, and then answer the questions below.

```java
//Displays a countdown from 10 to 1, followed by Lift off!
for (int t = 10; t < 0; t--) {
    System.out.println(t + "!");
}
System.out.println("Lift off!");
```

- What happens when the program runs?
  Hide Solution
  It displays "Lift off!" but nothing else.
- Where are good places to look for the cause of the error? And what is the problem?
  Hide Solution
  The test is evaluating to `false` straight away, so the loop is never executed. This means that either we've initialised the loop variable to the wrong value
  Hide Solution
  No, the comment indicates we want to start it at 10 and that's what we set it to.
  or the test is incorrect.
  Hide Solution
  That's it! The expression `t < 0` will never be true because we start `t` at 10 and the for-loop never gets to execute the update.
- What's a suitable correction?
  Hide Solution
  The original author probably meant to write `t > 0`, so we count down *as long as* `t` has not yet reached zero.

Implement a correction and rerun to test.

Activity: Word list

Now test the following loop code in the test program.

```java
final String STOP = "end";
Scanner sc = new Scanner(System.in);
String word;
String list = "";

//Construct list of words entered by the user, separated by commas
//User enters 'end' to stop ('end' is not added to the list)
System.out.print("Enter a word: ");
word = sc.next();
while (!word.equals(STOP)) {
    list += word + ",";
    System.out.print("Enter a word: ");
}

System.out.println("Word list is: " + list);
```

Compile it, run it, and enter the following sequence of words when prompted:
apple
loop
end

- What happens when the program runs?
  Hide Solution
  It never stops! (Or worse, it crashes Java!) Congratulations, you've just found the address of Apple Inc. Press the Reset button in DrJava to stop the program.
- Where are good places to look for the cause of the error? And what is the problem?
  Hide Solution
  If a loop never ends then it means that either the test can never become `false`
  Hide Solution
  No, the user *could* enter the word 'end'. Try this now: rerun and type *end* when first prompted.
  or the update is incorrect.
  Hide Solution
  Wait. What update? The value of `word` never changes. We ask the user for another word but we never actually read it from them!
- What's a suitable correction?
  Hide Solution
  Add `word = sc.next();` inside the loop just after we prompt the user.

Implement a correction and rerun to test.

Activity: The Mysterious Case of the Disappearing Cat That Didn't Disappear

Here is a slight variation on the Disappearing cat code from above. It differs by only four characters. Place it in new program as the sole contents of main(). Compile and run the code and observe how it behaves.

```java
String s = "cat";

while (s.length() > 0) {
    System.out.println(s);
    s.substring(1, s.length());
}
```

- What happens when the program runs?
  Hide Solution
  It prints 'cat' repeatedly but never stops. It's another infinite loop, so press the Reset button in DrJava to stop the program.
- Where are good places to look for the cause of the error? And what is the problem?
  Hide Solution
  If a loop never ends then it means that either the test can never become `false` or the update is incorrect. Since we *could* have initialised `s` to be an empty String the test could conceivably be false, so let's check the update…
  Hide Solution
  In this modified version of the loop the 'update' is `s.substring(1, s.length());` but there's a problem: Strings are immutable, so the substring method does not modify the String we call it on, but instead returns a new String that holds some of the characters from the origial.
  Look back at the original *disappearing cat* code and notice the `s =` that *replaces* the value of `s` with the result of substring.
- What's a suitable correction?
  Hide Solution
  Adding the `s =` again so that `s` is actually updated each time through the loop.