

## Revision

- To write loops in Java, we use while statements.
- To design while statements, we follow two steps: to decide what each iteration do, and to design the loop invariants.
- Loop invariants are those conditions that are guaranteed to be true when a loop starts executes.
- With loops, we usually add one to or subtract one from a variable.
- The increment and decrement operators perform exactly this.
- These operators come in two flavors. The prefix operator takes the value after the variable is changed, while the postfix operator takes the value before the variable is changed.

0911B L11-1

## Formatting output

- To output something in a report format, we usually want to control how much space the output takes.
- Unluckily, standard Java give the user little control. Again, we will rely on the library in the package of the book.
- The `Fmt` class is an easy-to-use facility to format output. To use it, all we need is to `import chapman.io.*`.
- Instead of `"System.out.print(something);"`, we perform output like `"Fmt.printf(format.string, something);"`.
- The `format.string` is used to control how to format the `something`. E.g., `"HK$%3d.00"` will print `"HK$"` first, and then the number with at least 3 columns, and finally `".00"`.

0911B L11-2

### Example: our multiplication table

```
// Print a multiplication table
int i = 1;
while (i <= 10) { // Print row i.
    int j = 1;
    while (j <= 10) { // Print column j. Space is already printed if needed.
        Fmt.printf("%3d", i * j);
        if (j++ != 10)
            System.out.print(" ");
        else
            System.out.println("");
    }
    ++i;
}
```

Format strings			
(Modifier)	(Width)	(Precision)	Format
e.g. + = leading + space = leading space - = left align numbers 0 = show leading 0			f = fixed point e = exponential g = general d = decimal integer c = character s = string

0911B L11-3

### Loop control take 1: break

- Like switch/case, sometimes you want to stop executing a while statement before it reaches the end.
- You can use a `break` statement in such cases.
- When a `break` statement is executed, the loop stops immediately.
- The statements after `break` will not be executed. And no more iterations will follow.
- We call such statements a "loop control" statement, as it controls how the loop is executed.

0911B L11-4

## Example

Find the average of 10 numbers, or until a negative number is read.

```
StdIn in = new StdIn();
double number;
double total = 0;
int count = 0;
while (count < 10) {
    System.out.println("Input a number, or stop with a negative number.");
    number = in.readDouble();
    if (number < 0)
        break;
    total += number;
    ++count;
}
Fmt.printf("The average is %.2f\n", total/count);
```

Note the usage. Using "break" makes sense only if everything in the loop following "break" should not be executed.

0911B L11-5

### Loop control take 2: continue

- Unlike switch/case statements, sometimes you want to stop executing the loop but not completely.
- You might have detected some condition that requires special treatments. After that you want to stop executing the current iteration, and continue with the next.
- In such situations you cannot use `break`, which stop executing the remaining iterations as well as the remainder of the current iteration.
- Instead, you can use a `continue` statement. Whenever `continue` is executed, the current iteration is stopped, and the loop condition is checked again to see whether there should be a new iteration.
- So it works like a direct jump to the end of the loop body.

0911B L11-6

### Example: printing a tan function table

Suppose we want to print the values of  $\tan 10^\circ$  to  $\tan 180^\circ$  in steps of  $10^\circ$ . We know that calculating  $\tan 90^\circ$  is a bad idea, so we want to skip that particular value.

```
int i = 0;
while (i < 180) {
    i += 10;
    if (i == 90)
        continue;
    Fmt.printf("%3d ", i);
    Fmt.printf("%8.2f\n", Math.tan(i * Math.PI / 180));
}
```

Again, it only makes sense to use `continue` if all the remainder of the iteration should not be executed. (This example is in fact a bit strange: why the “`i += 10`” done at the beginning?)

Since we usually need to restore invariants, `continue` is not usually used with “`while`”. It is usually used with “`for`”, which we will see later.

0911B L11-7

### Loop control take 3: labelled break/continue

- If a `break` or `continue` statement appear within a nested `while` loop, or if a `break` statement appear within a `while` statement inside a `switch/case` statement, which will it `break/continue`?
- The rule: the inner-most one, if it is not specified otherwise.
- Some times the inner-most loop is not the loop that you want to `break/continue`. In this case you need a labelled loop, and a labelled `break`.
- Any statement in Java can be labelled, by writing “`label:`” before the statement (label is any identifier). In practice it only makes sense to label a looping or a `switch/case` statement.
- A `break` statement can specify the label of the statement to `break`, like “`break label;`”. Similar for `continue`.

0911B L11-8

### Example: guessing birthdays

The following program guesses the birthday of somebody, after he calculates a number from his birthday and give it to your program. Your program tries every day of every month to see whether it works. Of course you want to stop once it is found.

```
System.out.print("Multiply the month of your birthday by 13, ");
System.out.println("and the day by 17.");
System.out.print("Add them up. What is it? ");
int result = in.readInt();
int month = 1;
guess:
while (month <= 12) {
    int day = 1;
    while (day <= 31) {
        if (13 * month + 17 * day == result) {
            System.out.println("Your birthday is " + day + "/" + month);
            break guess;
        }
        ++day;
    }
    ++month;
}
if (month == 13)
    System.out.println("You cheat!");
```

0911B L11-9