

While statements: repeating with a condition

COMP112: 137

- **For** statements: repetition over a list of values.
- **While** statements: general repetition, subject to a condition.

```
while (condition-to-do-it-again) {  
    actions to perform each time round  
}
```

Similar structure to the if statement

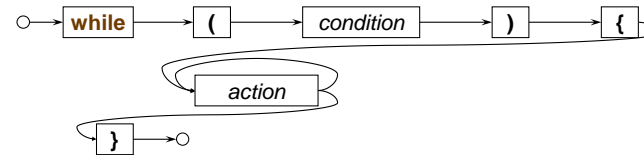
```
while (true) {  
    UI.println("this repeats forever!");  
}
```

```
int n = 1;  
while (n <= 100) {  
    UI.println(n);  
    n = n + 1;  
}
```

© Peter Andreae

While statement

COMP112: 138



- Meaning:
 - Repeatedly
 - If the condition is still true, do the actions another time
 - If the condition is false, stop and go on to the next statement.
 - Note: don't do actions at all if the condition is initially false
- Similar to **if**, but NOT THE SAME!
 - keeps repeating the actions,
 - as long as the condition is still true each time round
 - no **else** — just skips to next statement when condition is false

© Peter Andreae

While with numbers #1

COMP112: 139

- Print a table of numbers and their squares:

```
public void printTable(int max){  
    int num = 1;           Initialise  
    while (num <= max) {    Test  
        UI.printf(" %3d  %6d  %n", num, (num*num)); Body  
        num = num + 1;      Increment  
    }  
}
```

- Repetition with **while** generally involves

- initialisation: get ready for the loop
- test: whether to repeat
- body: what to repeat
- "increment": get ready for the next iteration

© Peter Andreae

While with numbers #2

COMP112: 140

- Draw a row of squares:

```
public static final double SIZE = 20;  
:  
/* Draws n squares in a horizontal row, starting at (left,top) */  
public void drawSquares (int left, int top, int n){  
    int count = 0;           Initialise  
    while (count < n) {      Test  
        double x = left + count * SIZE;  
        UI.drawRect(x, top, SIZE, SIZE); Body  
        count ++;           Increment  
    }  
}
```



Scope of variables declared in loop is limited to the loop

Shorthand for
count = count + 1
except value is value of x before adding

© Peter Andreae

While with numbers #3

COMP112: 141

- Counting down:

```
public void countDown(int start){
    int count = start;
    while ( count >= 1) {
        UI.println( count );
        count = count - 1;
    }
    UI.println(" GO");
}

:
this.countDown(5);
:
```

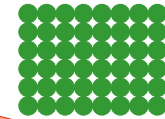
© Peter Andreae

Nested while loops with numbers

COMP112: 142

Draw a grid of circles

```
public void drawCircles(int rows, int cols, int diam ) {
    int row = 0;
    while (row < rows) {
        int col = 0;
        while ( col < cols ) {
            int x = LEFT + row*diam;
            int y = TOP +col*diam;
            UI.fillOval(x, y, diam, diam);
            col++;
        }
        row++;
    }
}
```



Outside loop:
do each row

Inside loop:
do each column within the
current row

© Peter Andreae

Designing loops with numbers

COMP112: 144

When the number of steps is known at the beginning of the loop:

```
int count = 0;
while ( count < number) {
    <do actions>
    count = count + 1;
}

OR

int num = 1;
while ( num <= number) {
    <do actions>
    num = num + 1;
}
```

- Can count from 0 or from 1
 - If counting from 0, loop while count is **less than** target:
(count is the number of iterations that have been completed)
 - If counting from 1, loop while num is **less than or equal to** target:
(num is the iteration it is about to do)

© Peter Andreae

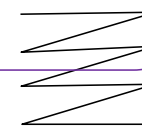
Designing nested loops with numbers

COMP112: 145

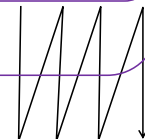
2D structures, eg table of rows and columns:

- Can do rows in the outside loop and columns in the inside loop, or vice versa

```
int row = 0;
while ( row < numberOfRows ) {
    int col = 0;
    while ( col < numberOfCols ) {
        <do actions for row, col>
        col++;
    }
    row++;
}
```



```
int col = 0;
while ( col < numberOfCols ) {
    int row = 0;
    while ( row < numberOfRows ) {
        <do actions for row, col>
        row++;
    }
    col++;
}
```



© Peter Andreae

General while loops

COMP112: 146

```
/** Practice times-tables until got 5 answers correct */
public void playArithmeticGame (){
    int score = 0;
    while ( score < 5) {
        // ask an arithmetic question
        int a = this.randomInteger(10);
        int b = this.randomInteger(10);
        int ans = UI.askInteger("What is " + a + " times " + b + "?");
        if ( ans == a * b ) {
            score = score + 1;
        }
    }
    UI.println("You got 5 right answers" );
}
```

```
public int randomInteger(int max) {
    return (int) (Math.random() * max ) + 1;
}
```

© Peter Andreae

General while loops

COMP112: 147

```
/** Ask a multiplication problem until got it right */
public void practiceArithmetic (){
    int a = this.randomInteger(10);
    int b = this.randomInteger(10);
    String question = "What is " + a + " times " + b + "?";
    boolean correct = false;
    while ( ! correct) {
        int ans = UI.askInteger(question);
        if ( ans == a * b ) {
            correct = true;
        }
    }
    UI.println("You got it right!");
}
```

- This seems unnecessarily complex!!

© Peter Andreae

Loops with the test "in the middle"

COMP112: 148

If the condition for exiting the loop depends on the actions, need to exit in the middle!
Common with loops asking for user input.

- **break** allows you to exit a loop (**while**, or **for**) (or a **switch**)
 - Must be inside a loop
 - Ignores any **if**'s
 - Does not exit the method (**return** does that)

```
while ( true ) {
    actions to set up for the test
    if ( exit-test ) {
        break;
    }
    additional actions
}
```

continue means exit this iteration of the loop, and jump to the next iteration.

© Peter Andreae

General while loops with break

COMP112: 149

```
/** Ask a multiplication problem until got it right */
public void practiceArithmetic (){
    int a = this.randomInteger(10);
    int b = this.randomInteger(10);
    String question = "What is " + a + " times " + b + "?";
    boolean correct = false;
    while ( true ) {
        int ans = UI.askInteger(question);
        if ( ans == a * b ) {
            break;
        }
    }
    UI.println("You got it right!");
}
```

Setting up for test

Test and break

no additional actions

- Only use **break** when the exit is not at the beginning of the loop.

© Peter Andreae

More loops with user input

COMP112: 150

- Make user guess a magic word:

```
public void playGuessingGame(String magicWord){
    UI.println("Guess the magic word:");
    while (true) {
        String guess = UI.askString("your guess: ");
        if ( guess.equalsIgnoreCase(magicWord) ){
            UI.println("You guessed it!");
            break;
        }
        UI.println("No, that wasn't right. Try again!");
    }
}
```

Setting up for test

Test and break

Additional actions

© Peter Andreae

Testing your program

COMP112: 151

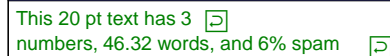
- A) Need to try out your program on sample input while removing the "easy" bugs.
 - Can be a pain if need lots of input (eg TemperatureAnalyser)
 - UI window has a menu item – "set input" – to get input from a text file instead of user typing it.
 - ⇒ don't have to type lots of data each time
 - Create the text file, eg in Notepad
 - Select file using menu before the program has started asking for input.
 - File can contain multiple sequences of data.
 - B) Need to test your program on a range of inputs
 - Easy, "ordinary", inputs
 - Boundary cases — values that are only just in range, or just out of range
 - Need to check that your **if** conditions are right
 - Invalid data—does your program handle invalid input correctly?
- Creating test cases involves creativity – have to try to come up with ways to break your program.

© Peter Andreae

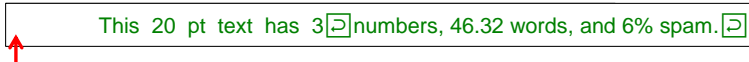
Text Input: reading multiple values

COMP112: 152

- When the user types into the text pane:



- nothing happens until they type a newline ("enter")
- Then all the characters on the line are put into a buffer that the program can access



- The program can access the buffer using the "UI.next..." methods:
 - UI.next() → "This" reads next "token" as a string
 - UI.nextInt() → 20 reads next "token" as an integer
 - UI.nextDouble() → ERROR! reads next token as a double
 - UI.nextLine() → "pt text has 3" reads up to next as a string
- all the methods move the cursor forward, past what was read.

© Peter Andreae

Text Input: reading multiple values

COMP112: 153

- If there is no input yet, the UI.next...() methods will just wait.
 - ⇒ Always print a prompt to the user before you try to read!
- It is not safe to call UI.nextInt() or UI.nextDouble() unless you can be certain the next token is an integer / double!
- How can you tell?
 - UI.hasNextInt() → boolean true if next token is an integer
 - UI.hasNextDouble() → boolean true if next token is a double
 - UI.hasNext() → boolean true if there is a next token (always true for text pane)

© Peter Andreae

next vs. nextLine()

COMP112: 154

- next(), nextInt(), nextDouble()
 - picks up any spaces, discards them,
 - picks up characters to make next "token" (until it reaches a space),
 - returns the token
 - next() returns it as a String
 - nextInt() returns it as an int,
 - nextDouble() returns it as a double.
- nextLine()
 - Picks up all the characters (including spaces) until it reaches end-of-line character,
 - throws away end-of-line, and
 - returns all the characters (including spaces) as a String.

© Peter Andreae

Input with "next" methods

COMP112: 155

Method	What it does	Returns
next()	Read and return next token of user's input	String
hasNext()	Returns true if there is another token in the user input. Waits for the user to type something if necessary.	boolean
nextInt()	Read the next token of the user's input. Return it as an integer if it is a number. Throws an exception if it is not a number.	int
nextDouble()	Read the next token of the user's input. Return it as a double if it is a number. Throws an exception if it is not a number.	double
hasNextInt()	Returns true if next token in the input is an int / double. Waits for user to type something if necessary.	boolean
hasNextDouble()	Returns true if next token in the input is a double. Waits for user to type something if necessary.	boolean
nextBoolean()	Read the next token of the user's input. Return true if it is "yes", "y", or "true", return false if it is "no", "n", or "false" (case insensitive). Throws an exception if it is anything else.	boolean
nextLine()	Read the remaining characters of the user's input up to (but not including) the next end-of-line and return them as a string. Reads and throws away the end-of-line character. If there are no characters on the line, then it returns an empty string ("").	String

© Peter Andreae

Reading words from user

COMP112: 156

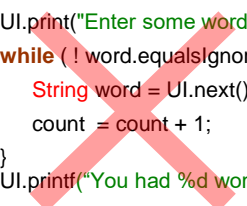
```
public void countWordsBeforeThe() {  
    int count = 0;  
    UI.print("Enter some words: ");  
    // loop, stopping when you get to 'the'  
    while (true) {  
        // read next token  
        String word = UI.next();  
        // increment count  
        if (!word.equalsIgnoreCase("the")) {  
            count++;  
        }  
    }  
    UI.printf("You had %d words before 'the'. %n", count);  
}
```

© Peter Andreae

Reading words from user: BAD

COMP112: 157

```
public void countWordsBeforeThe() {  
    int count = 0;  
    UI.print("Enter some words: ");  
    while (!UI.hasNextInt()) {  
        String word = UI.next();  
        count = count + 1;  
    }  
    UI.printf("You had %d words before 'the'. %n", count);  
}
```



© Peter Andreae

Reading words from user: BAD

COMP112: 158

```
public void countWordsBeforeThe() {
    int count = 0;
    UI.print("Enter some words: ");
    String word = UI.next();
    while ( ! word.equalsIgnoreCase("the") ) {
        count = count + 1;
    }
    UI.printf("You had %d words before 'the'. %n", count);
}
```

Reading words from user: Fixed

COMP112: 159

```
public void countWordsBeforeThe() {
    int count = 0;
    UI.print("Enter some words: ");
    String word = UI.next();
    while ( ! word.equalsIgnoreCase("the") ) {
        count = count + 1;
        word = UI.next();
    }
    UI.printf("You had %d words before 'the'. %n", count);
}
```

read first word before loop

read next word at end of loop
("increment")

© Peter Andreae

© Peter Andreae

Alternate design: using break.

COMP112: 160

```
public void countWordsBeforeThe() {
    int count = 0;
    UI.print("Enter some words: ");
    while ( true ) {
        String word = UI.next();
        if ( word.equalsIgnoreCase("the") ){
            break;
        }
        count = count + 1;
    }
    UI.printf("You had %d words before 'the'. %n", count);
}
```

gets out of the
enclosing **while** loop

- Note: Textbook does not like this style; I do
- Only use when the test has to be in the middle of the loop
- Typically only use with a **while** (true) {...}
- The condition is an *exit* condition, not a *keep going* condition

© Peter Andreae

Using next... methods

COMP112: 161

```
/** sum up all numbers entered by user */
:
UI.print("Enter numbers: end with 'done':");
double sum = 0;
while (UI.hasNextDouble() ) { //peeking at next value or "token"
    double amt = UI.nextDouble(); //getting the next value and move pointer
    sum = sum + amt;
}
UI.nextLine(); // throw away the 'done'
UI.printf("Total of all numbers entered: %.2f %n", sum);
```

```
Enter numbers: end with 'done': 40 60
30 50 done
Total of all numbers entered: 180.00
```

© Peter Andreae