Another Java Program

- Design a Java program to measure reaction time of users responding to true and false "facts".
  - Ask the user about a fact: "Is it true that the BE is a 4 Year degree?"
  - Measure the time they took
  - Print out how much time.
- Need a class
  - what name?
- Need a method
  - what name?
  - what parameters?
  - what actions?

ReactionTimeMeasurer

```
/** Measures reaction times for responding to true-false statements */
public class ReactionTimeMeasurer {
    public ReactionTimeMeasurer(){
        UI.addButton("Measure Time", this::measureReactionTime);
    }
    /** Measure and report the time taken to react to a question */
    public void measureReactionTime() {
[        ].  // find out the current time and remember it
            // ask the question and wait for answer
[        ].  // find out (and remember) the current time
            // print the difference between the two times
    }
}
```

Write the method body in comments first,

(to plan the method without worrying about syntax)

Work out what information needs to be stored (ie, variables)

ReactionTimeMeasurer

```
    /** Measure and report the time taken to react to a question */
    public void measureReactionTime() {
        long startTime = System.currentTimeMillis();
        UI.askString("Is it true that the sky is blue?");
        long endTime = System.currentTimeMillis();
        UI.printf("Reaction time = %d milliseconds \n",  (endTime - startTime) );
    }
}
```

Returns a very big integer
⇒ long
(milliseconds since 1/1/1970

Just asking one question is not enough for an experiment.

➔ need to ask a sequence of questions.

Multiple questions, the bad way

```
/** Measure and report the time taken to react to a question */
public void measureReactionTime(){
    long startTime = System.currentTimeMillis();
    UI.askString( "Is it true that John Quay is the Prime Minister");
    long endTime = System.currentTimeMillis();
    UI.printf("You took %d milliseconds \n",  (endTime - startTime) );

    startTime = System.currentTimeMillis();
    UI.askString( "Is it true that 6 x 4 = 23");
    endTime = System.currentTimeMillis();
    UI.printf("You took %d milliseconds \n",  (endTime - startTime) );

    startTime = System.currentTimeMillis();
    UI.askString( "Is it true that summer is warmer than winter");
    endTime = System.currentTimeMillis();
    UI.printf("You took %d milliseconds \n",  (endTime - startTime) );

    startTime = System.currentTimeMillis();
    UI.askString( "Is it true that Wellington's population > 1,000,000");
    endTime = System.currentTimeMillis();
    UI.printf("You took %d milliseconds \n",  (endTime - startTime) );
}
```

Lots of repetition.
But not exact repetition.
How can we improve it?

Good design with methods

- Key design principle:
  - Wrap up repeated sections of code into a separate method,
  - Call the method several times:

```
public void measureReactionTime ( ) {
    this.measureQuestion( "John Quay is the Prime Minister");
    this.measureQuestion( "6 x 4 = 23");
    this.measureQuestion( “Summer is warmer than winter");
    this.measureQuestion( "Wellington’s population > 1,000,000 ");
}
public void measureQuestion (String fact      ) {
    long startTime = System.currentTimeMillis();
    UI.askString("Is it true that "  fact    . );
    long endTime = System.currentTimeMillis();
    UI.printf("You took %d milliseconds \n",  (endTime - startTime) );
}
```

We need to *parameterise* the method

Improving ReactionTimeMeasurer (1)

```
public void measureReactionTime() {
    this.measureQuestion("John Quay is the Prime Minister");
    this.measureQuestion(“6  x 4 = 23");
    this.measureQuestion(“Summer is warmer than Winter");
    this.measureQuestion("Wellington’s population > 1,000,000 ");
}
public void measureQuestion(String fact) {
    long startTime = System.currentTimeMillis();
    UI.askString("Is it true that" + fact);
    long endTime = System.currentTimeMillis();
    UI.printf("You took %d milliseconds \n",  (endTime - startTime) );
}
```

"this" and method calls

- When you call a method on an object, the method "knows" which object it was called on.
  - stored in the "special variable": this
- If the method needs to call another method from the same class, it generally needs to call it on the same object.

```
public class MyObjects {
        :
    public void method1(){
        …
        this.method2(45,  "name");
        …
    }
    public void method2(int num, String n){
        …
    }
}
```

But, this. is optional!
If you leave the object out of a method call, Java will assume you meant this!

To be safe: always put the this. in, until you really know what you are doing.

Problem

- A good experiment would measure the average time over a series of trials
  - Our program measures and reports for each trial.
- Need to add up all the times, and compute average:
  - problem:
    - MeasureReactionTime needs to add up the times
    - MeasureQuestion actually measures the time, but prints it out.
    - How do we get the time back from MeasureQuestion to MeasureTime?

Methods that return values

- Some methods just have "effects":

```
UI.println("Hello there!");
UI.printf("%4.2f miles is the same as %4.2f km\n", mile, km);
UI.fillRect(100, 100, wd, ht);
UI.sleep(1000);
```

- Some methods just return a value:

```
long  now = System.currentTimeMillis();
double distance = 20 * Math.random();
double ans = Math.pow(3.5, 17.3);
```

- Some methods do both:

```
double height = UI.askDouble("How tall are you");
Color col =JColorChooser.showDialog(UI.getFrame(),  "paintbrush",   Color.red);
```

Defining methods to return values

Improving ReactionTimeMeasurer:

make measureQuestion **return** a value instead of just printing it out.

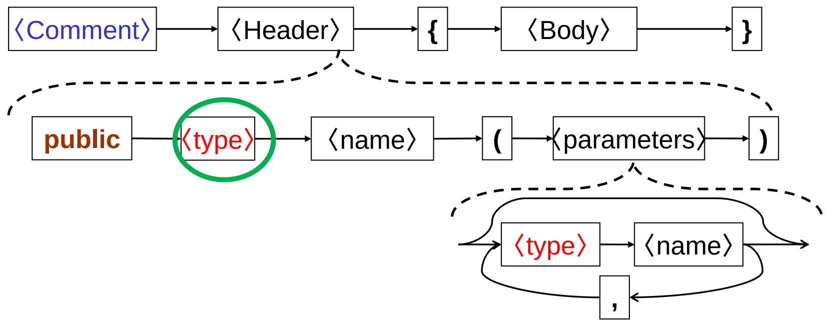
```
public void measureReactionTime() {
    long time = 0;
    time = time + this.measureQuestion("John Quay is the Prime Minister");
    time = time + this.measureQuestion("11 x 13 = 143");
    time = time + this.measureQuestion("Summer is warmer than Winter");
    time = time + this.measureQuestion(" Wellington's pop > 1,000,000 ");
    UI.printf("Average reaction time = %d milliseconds\n", (time / 4));
}
```

Specifies the type of value returned.
void means "no value returned"

```
public  long  measureQuestion(String fact) {
    long startTime = System.currentTimeMillis();
    ……
}
```

Syntax: Method Definitions (v3)

```
/** Measure time taken to answer a question*/
      public long measureQuestion ( String fact ){
            long startTime = System.currentTimeMillis();
               :
```



Defining methods to return values

If you declare that a method returns a value,
then the method body must return one!

```
public  long  measureQuestion(String fact) {
    long startTime = System.currentTimeMillis();
    String  ans = UI.askString("Is it true that " + fact);
    long endTime = System.currentTimeMillis();
    return  (endTime - startTime) ;
}
```

New kind of statement
  Means: exit the method and return the value
  The value must be of the right type

Returning values.

- What happens if we call the method:
  RTM-1 . askQuestions();

```
public void measureReactionTime(){                    this:
                                                      RTM-1
----------------------------------------------------
✓ long time = 0;          0•

  time = time + this.measureQuestion("John Quay is the Prime Minister");
  time = time + this.measureQuestion("6 x 4 = 23");
  time = time + this.measureQuestion("summer is warmer than Winter");
  time = time + this.measureQuestion("Wellington's pop > 1,000,000");
```

Returning values

```
return value:    •
                                                              this:
   public  long  measureQn(String fact){  "            "      RTM-1
---------------------------------------------------------------------
     •        long startTime = System.currentTimeMillis();
"        "    UI.askString("Is it true that " + fact);
     •        long endTime = System.currentTimeMillis();
              return  (endTime - startTime) ;
   }
```

Returning values.

- What happens if we call the method:
  RTM-1 . askQuestions();

```
public void measureReactionTime(){                    this:
                                                      RTM-1
----------------------------------------------------
✓ long time = 0;        0•
✓ time = time + this.measureQuestion("John Quay is the Prime Minister");
  time = time + this.measureQuestion("6 x 4 = 23");
  time = time + this.measureQuestion("summer is warmer than Winter");
  time = time + this.measureQuestion(" Wellington's pop > 1,000,000");
```

Aside: Random numbers

- Math.random() computes and returns a random double
  - between 0.0 and 1.0
- To get a random number between min and max:
  - min + random number * (max-min)

    (50.0 + Math.random() * 70.0)

    gives a value between 50.0 and 120.0
- This is an expression:
  - can assign it to a variable to remember it
  - can use it inside a larger expression
  - can pass it directly to a method

Menu

- Repetition/Iteration

Admin:

- Test
- Submission
- When the assignments are marked, marks and comments are available via the link on the Assignments page

Repetition / Iteration

Doing some action repeatedly:

- “Polish each of the cups on the shelf”
- “Put every chair on top of its desk”
- “Give a ticket to everyone who passes you”
- “Keep patrolling around the building until midnight”
- “Practice the music until you can play it perfectly”

Two patterns:

- Do something to each thing in a collection
- Do something until some condition changes

Repetion/Iteration in Java LDC 4.5

Several different ways of specifying repetition.

- For statement:
  - Do something to each element of a list

```
for ( type value  :  listOfValues ) {
    do something to value
}
```

- While statement:
  - Repeat some action until some condition becomes false

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

For statement

Three components

- a list of values
- a variable that is assigned each value of the list in turn.
- actions to perform for each value in the list

```
// print each number in a list of numbers:
for ( Double num  :  listOfNumbers ) {
    UI.println(num);
}
```

listOfNumbers: 150.0, 32.2, 6.9, 49.5, 83.4, -21.0, 1.0

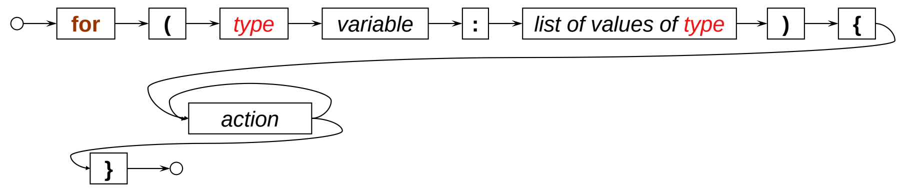
num: •

```
// print each string in a list of numbers that starts with "A":
for ( String str  :  listOfStrings ) {
    if ( str.startsWith("A") ) {
        UI.println(str);
}
```

listOfStrings: "Jamie", "Andie", "Jules", "Amy", "Mark"

str: "   "

For statement ("foreach" version)



```
for ( Double   num  :   listOfNumbers   ) {
      UI.println(num);
}
```

- Meaning:

  Repeatedly (for each value in the list)
  - put the next value of the list into the variable
  - do the actions.

Lists of values

- What type is a **list** of values?
- How do we get a list of values?

Have to use Double, not double
Double is the "wrapped-up" version of double, for putting into a list

List of doubles

```
ArrayList <Double>  numberList   =  UI.askNumbers("Enter numbers");
for (double num : numberList)  {
    UI.println(num);
}
UI.setColor(Color.red);
UI.setLineWidth(5);
for (double radius : numberList)  {
    if (radius> 20  &&  radius < 200) {
        UI.drawOval( 300 – radius,  250 – radius,  radius * 2.0, radius * 2.0);
}
```

Asks for a list of numbers, ending with 'done'

Lists of values

- What type is a **list** of values?
- How do we get a list of values?

List of String values

```
ArrayList <String>  nameList   =  UI.askStrings("Enter names");
for (String name : nameList)  {
    UI.println("Hello " + name);
}
UI.println("=========== Long names ============");
for (String name : nameList)  {
    if (name.length() > 6 ) {   UI.println(name);    }
}
UI.println("=========== Short names ============");
for (String name : nameList)  {
    if (name.length() <= 6 ) {   UI.print(name + ", ");    }
}
UI.println();
```

Asks for a list of strings, ending with empty line

print without a new line

print just a new line

Doing more with the loops: using Variables

- Add up all the numbers in a list:

numberList: | 150.0, 32.2, 6.9, 49.5, 83.4, -21.0, 1.0 |

```
ArrayList <Double>  numberList   =  UI.askNumbers("Enter numbers");

double total = 0.0;
for (double num : numberList)  {
    total = total + num;
}
UI.println("Total of numbers = " + total );
```

Declare and initialise variable

Add each number into the total:
- Uses current value in <u>total</u>
- Adds the next number to it
- Puts result back into <u>total</u>

Doing more with the loops: using Variables

- Count the number of long names in a list.

```
ArrayList <String>  nameList   = UI.askStrings("Enter names");

int count = 0;
for (String name : nameList)  {
     if (name.length() > 6 ) {
          count = count + 1;
     }
}
UI.printf("There were %d long names out of %d names \n", count,  nameList.size() );
```

Declare and initialise variable

Add 1 to the count

Number of values in a list

Lists are values too: passing lists around

```
public void analyseNames() {
     ArrayList <String>  nameList   = UI.askStrings("Enter names");
     UI.println("Total characters: "  + this.totalChars (nameList) );
     UI.println("Starts with A: " + this.wordStartingWith(nameList, "A") );
}
public int totalChars(ArrayList <String>  strings ){
     int count = 0;
     for (String str : strings)  {
          count = count + str.length();
     }
     return count;
}
public String wordStartingWith(ArrayList <String>  strings,  String pattern ){
     for (String str : strings)  {
          if ( str.startsWith(pattern) )  {  return str;  }      // returns first word starting with the pattern
     }
     return "<none>";
}
```

While statements: repeating with a condition

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

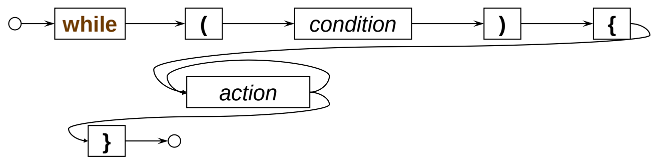
```
while (condition-to-do-it-again )  {
     actions to perform each time round
 }
```

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;
}
```

While statement



- 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

While with numbers #1

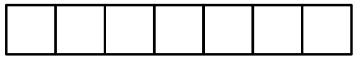
- Print a table of numbers and their squares:

```
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

While with numbers #2

- 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
    }
}
```

Shorthand for
count = count + 1

While with numbers #3

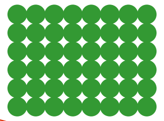
- Counting down:

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

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

Nested while loops with numbers

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