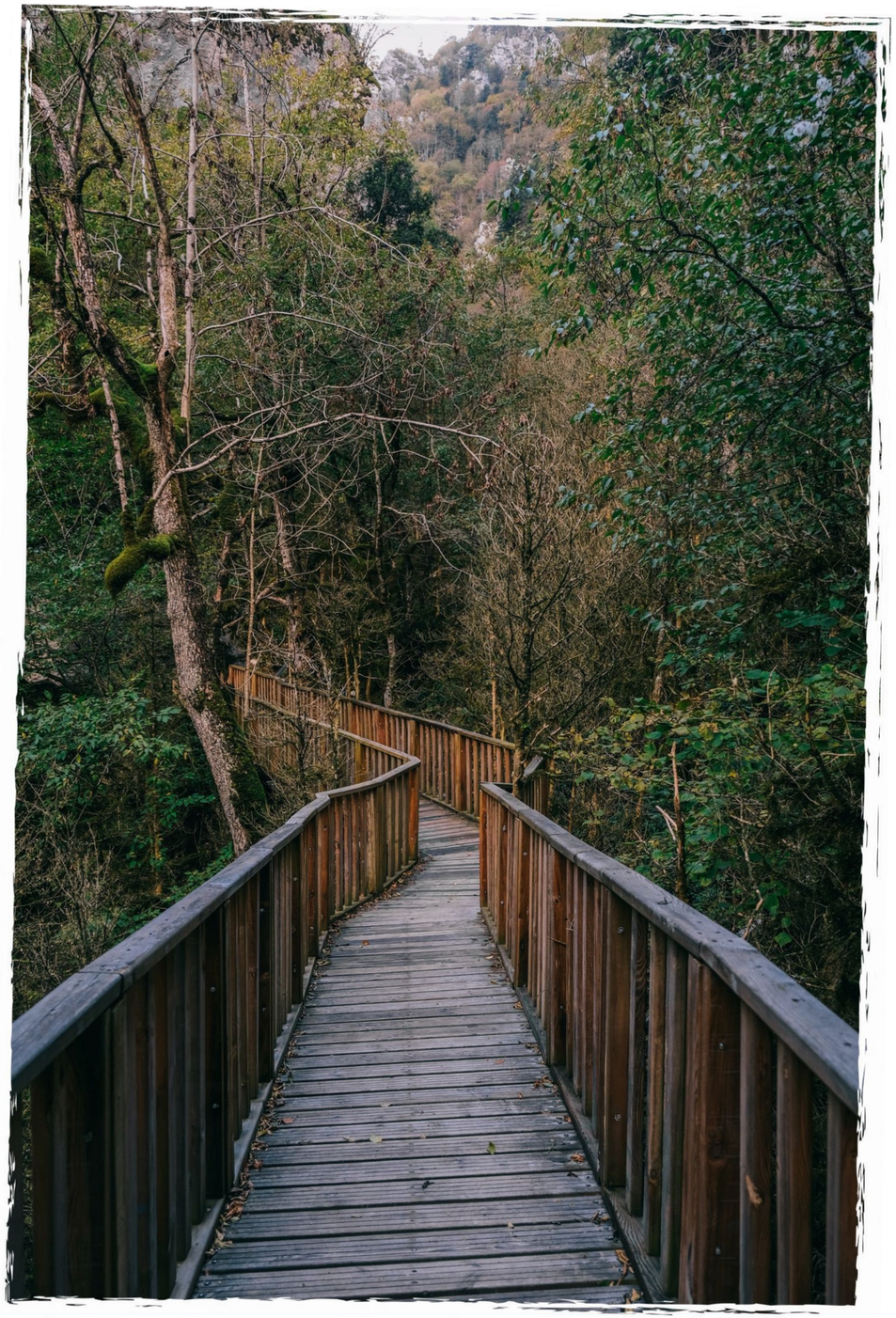
Dr David A McMeekin

Nesting, 2D Arrays &

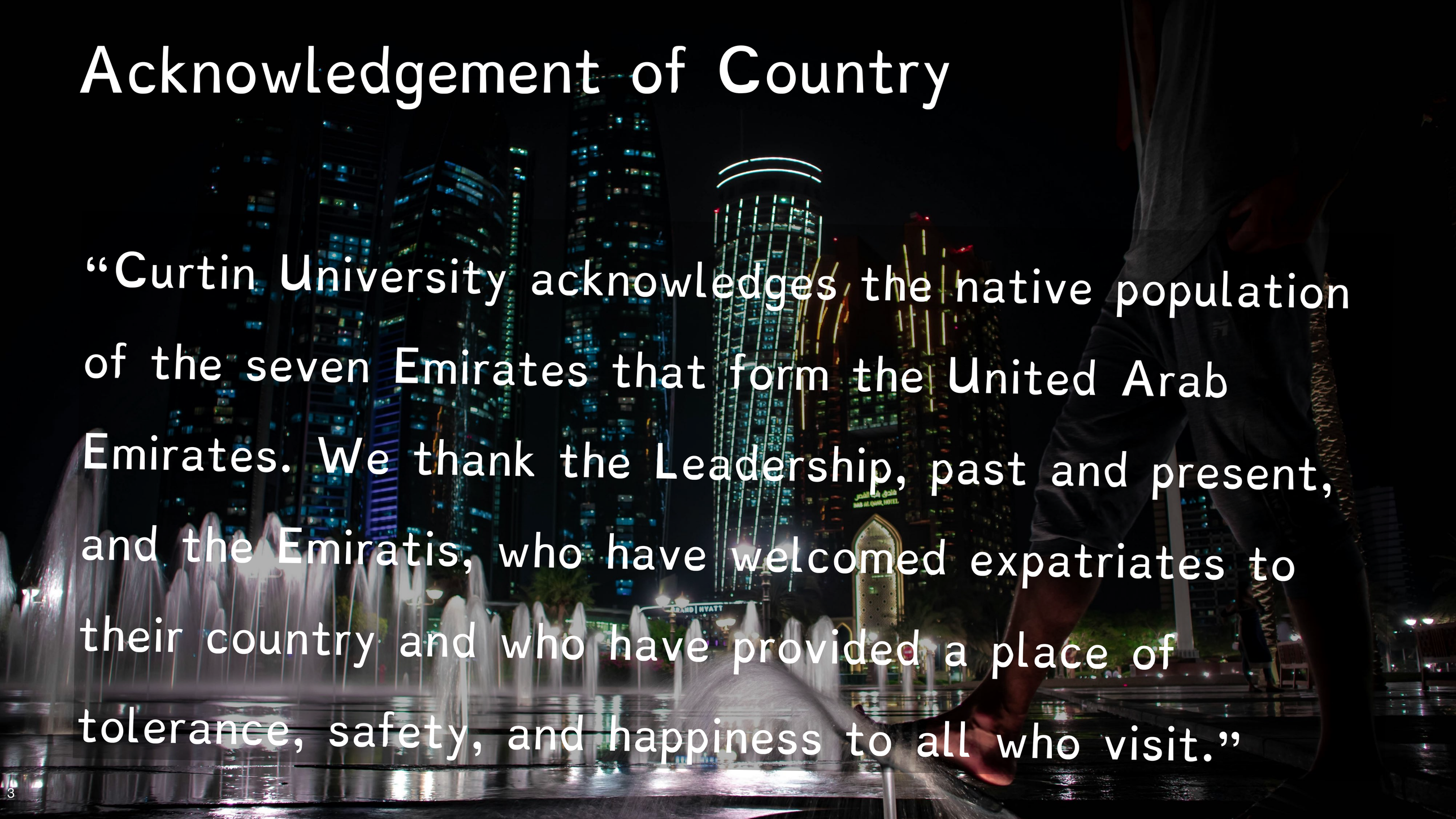
COMP1007



Acknowledgement of Country

“I acknowledge the Wadjuk people of the Noongar nation on which Curtin University’s Boorloo Campus sits, and the Wangkatja people where the Karlkurla Campus sits, as the traditional custodians of the lands and waterways, and pay my respect to elders past, present and emerging.”

Acknowledgement of Country



“Curtin University acknowledges the native population of the seven Emirates that form the United Arab Emirates. We thank the Leadership, past and present, and the Emiratis, who have welcomed expatriates to their country and who have provided a place of tolerance, safety, and happiness to all who visit.”

COMP1007 - Unit Learning Outcomes

- Identify appropriate primitive data types required for the translation of pseudocode algorithms into Java;
- Design in pseudocode simple classes and implement them in Java in aLinux command-line environment;
- Design in pseudocode and implement in Java structured procedural algorithms in a Linux command-line environment;
- Apply design and programming skills to implement known algorithms in real world applications; and
- Reflect on design choices and communicate design and design decisions in a manner appropriate to the audience.

Outline

- Nested Loops
- 2D Arrays
- Methods & Design
- Methods
- Parameters
- Example Code

Nested Loops

Nested Loops

- Control structures can be nested inside other control structures;
- `IF-THEN-ELSE` inside `IF-THEN-ELSE`
- A loop inside a loop;
- Be careful of algorithm efficiency (inefficiency);
- Nesting loops inside another loop exponentially increases the number of processing steps;
- Good use of indentation is essential for human readability.

Nested Loop Example

- Write an algorithm that receives a number between 1 and 12 (inclusive), then outputs the times tables (1 to 12) between 1 and the input number.

```
Enter a number in the range 1 to 12: 3
The 1 Times Table
1 x 1 = 1
...
1 x 12 = 12
The 2 Times Table
2 x 1 = 2
...
2 x 12 = 24
The 3 Times Table
3 x 1 = 3
...
3 x 12 = 36
```

Nested Loop Example: Pseudocode

```
num = -1
 WHILE(num < 1) OR (num > 12)
      OUTPUT 'Enter a number in the range 1 to 12'
      num = GET user input
 ASSERTION: num in the range 1 to 12 inclusive

 FOR table = 1 TO num CHANGEBY 1
      OUTPUT 'The ' table ' Times Table'
      FOR number = 1 TO 12 CHANGEBY 1
          OUPTUT table " x " number " = " (table * number)
      ENDFOR
      ASSERTION: table Times Table is output to the user
ENDFOR
ASSERTION: one to n Times Table is output to the user
```

Nested Loop Example: Java

```java
import java.util.*;
public class NestedLoop
{
    public static void main(String[] args)
    {
        int num = -1;
        Scanner sc = new Scanner(System.in);
        while ((num < 1)||(num > 12))
        {
            System.out.print("Enter a number in the range 1 to 12: ");
            num = sc.nextInt ();
        }
        for (int table = 1; table <= num ; table ++)
        {
            System.out.println("The " + table + " Times Table ");
            for (int number = 1; number <= 12; number ++)
            {
                System.out.println(table + " x " + number + " = " + (table * number ));
            }
        }
        sc.close();
    }
}
```

Nested Loops and Algorithm Complexity

- Algorithm Complexity can indicate algorithm efficiency;
- Attempts to show the rate of increase in processing steps as a function of the amount of data being processed;
- Algorithm complexity covered in DSA (COMP1002).

Algorithm Complexity

- Let's consider the previous example with nested **FOR** loops;

```
for (int table = 1; table <= num ; table ++)
{
    System.out.println("The " + table + " Times Table ");
    for (int number = 1; number <= 12; number ++)
    {
        System.out.println(table + " x " + number + " = " + (table * number ));
    }
}
```

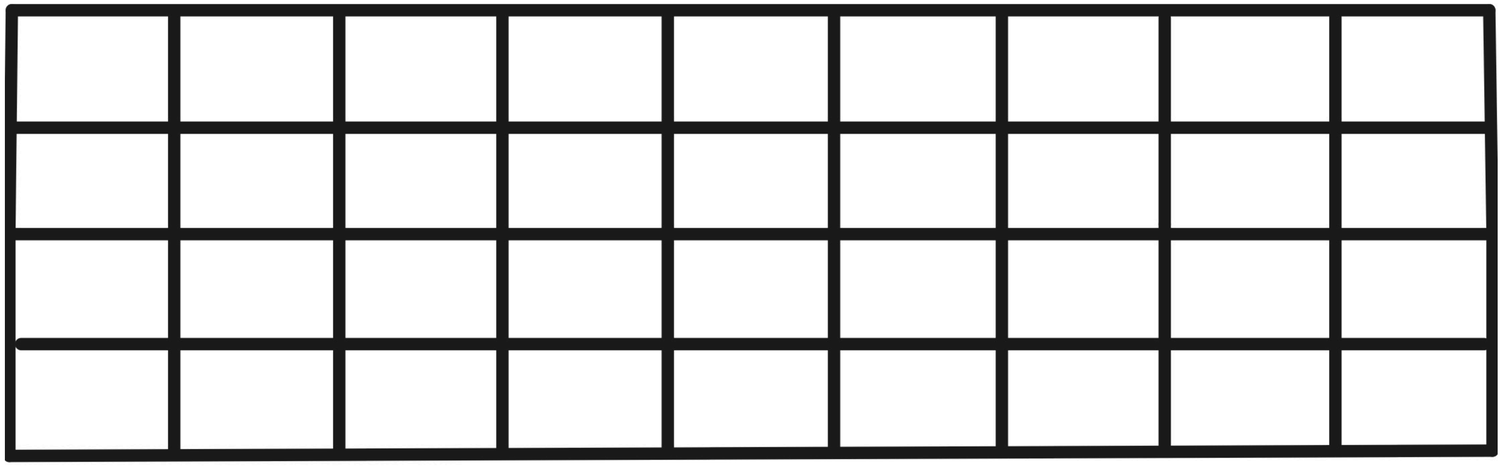
- When the user inputs 6, how many times did the statement **System.out.println(table + …);** (Line 6) execute?

- How about when the user inputs 12?

2D Arrays

Two-Dimensional Arrays (2D arrays)

- A 2D array expands the concept of the array:

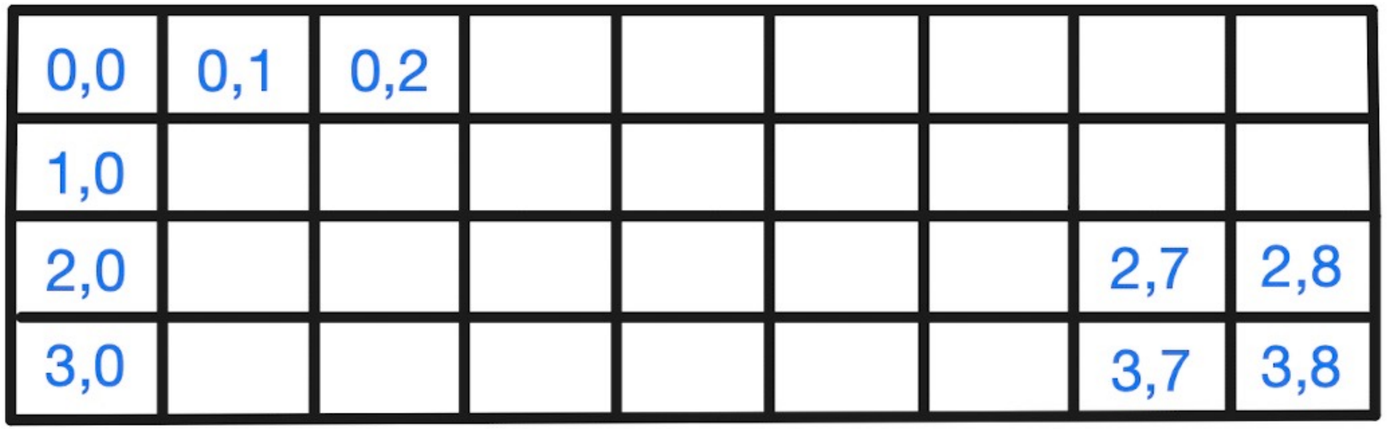


- Access to array elements: still via an index or subscript;
- The index contains 2 numbers, one for the row, one for the column;
- Row numbering starts at 0;
- Column numbering starts at 0.

Creating a 2D Array in Java

- Creating a 2D array with 4 rows and 9 columns for integers:

```
int [][] twoDArray = new int [4][9];
```



- Above is the 2D array (with some index numbering);
- A way to access elements is via a nested for loop.

Java Code

```java
import java.util.*;
public class UsingTwoDArrays
{
    public static void main(String[] args)
    {
        int [][] myTwoDArray;
        myTwoDArray = new int[10][10];

        for(int i = 0; i < 10; i++)
        {
            for(int j = 0; j < 10; j++)
            {
                myTwoDArray[i][j] = j;
            }
        }
        for(int i = 0; i < 10; i++)
        {
            for(int j = 0; j < 10; j++)
            {
                System.out.print(myTwoDArray[i][j] + " ");
            }
            System.out.println();//Print on the next line
        }
    }
}
```

Methods & Design

Modularity

- Methods: used to break algorithms into smaller parts;
- Each method performs one task and one task only:
  - Ease of design;
  - Readability;
  - Code re-use;
  - Debugging.

main() Method

- `main()` is a Java program's starting point;
- Within `main()`:
  - Other methods are invoked;
  - Objects can be created;

```
      public static              void          main       (String[] args)    {...}
| method type/modifiers |  | return type |  | name |    | parameters |    | code |
```

- Parameters for `main()` are covered later.

Methods (also known as functions)

```
public double multiplyTwoNumbers(double numOne, double numTwo)
{
    double product = 0.0;
    product = numOne * numTwo;
    return product;
}
```

- A method has a header consisting of:
  - Modifiers: public, private (there are others);
  - Data type the method returns (void if nothing returned);
  - Method name;
  - Parameter list;
  - Exception list (not shown & covered later).
- A method has a body enclosed in braces { } that does the computation.

Methods

- Plan and design your methods;
  - Use pseudocode (especially in PDI);
- Look at the requirements;
- For each task in the requirement specification, ask the question: “Is there a written & tested method that does this already?”
  - Yes: Use it;
  - No: Design, create and test one.

Designing Methods

- Think: what does the algorithm supply to the method?
- Think: what does the method `return` to the algorithm?
- When designing it, create it like this:
  - **IMPORT:** data supplied to the method;
  - **EXPORT:** data `returned` from the method to the calling method;
  - **ASSERTION:** what is true after the method executes.
    - Assertions statements assist in debugging the algorithm
    - Sometimes no assertion can be made.

Comments

- Comment blocks are used to describe methods;
- Generally contain:
  - Method Contract:
  - IMPORT, EXPORT and ASSERTION.
  - Purpose of the method (its job);
  - Authors;
  - Dates created and modified.
- Generally omitted from lecture notes due to space;
- Expectation is you use them in your programs/classes.

Comment Blocks

```
/**************************************************
 * Author : David A McMeekin                      *
 * Created: 29/02/2003                            *
 * Purpose: To do something amazing               *
 **************************************************/
public class MyJavaApp
{
    public static void main(String[] args)
    {
        // Code
    }// End main

    /**************************************************
     * Name: myMethod                                 *
     * Date: 29/02/2003                               *
     * Import: a (int), b (int)                       *
     * Export: val (double)                           *
     * Purpose: Do a part of something                *
     **************************************************/
    public static double myMethod(int a, int b)
    {
        // Code
    }// End myMethod
}// End Class
```

`void` Methods

- Do their job and pass control back to calling method;
- Have no return value: i.e., no export.

Example void method: Pseudo Code

```
MAIN:
    sayHello("David")
END MAIN

METHOD: sayHello
IMPORT: aParam
EXPORT: none
ALGORITHM:
    DISPLAY "I'm saying hello to " aParam
END METHOD
```

Example **void** method: Java

```java
import java.util.*;
public class MyWorldClass
{
    public static void main(String[] args)
    {
        sayHello("David"); //Calling sayHello() method
    }    // "David" is the argument

    // METHOD: sayHello
    // IMPORT: aParam (String)
    // EXPORT: none
    public static void sayHello(String aParam)
    {
        System.out.println("I'm saying hello to " + aParam);
    }
}
```

Invoking or Calling methods

- Two possibilities when calling methods:
  - The calling and called method are in the same class; or
  - The calling and called method are in different classes.

- Methods in the same class are invoked as in previous example;
- Non static methods in a different class are invoked via an Object variable:
  - `objectName.methodName();`
  - `sc.nextDouble();`

- Static methods in a different class are invoked by specifying the class name:
  - `Math.sqrt(9);`

More on Methods

Methods

- In Java methods can return a single piece of data to the calling method:
- The return data type is shown in the method header:

```
public int calculateMyAge(String name)
```

- Remember void methods do not return anything:

```
public void sayHello(String aParam)
```

Functions/Methods

- Most non-trivial arithmetic uses various functions:
  - sin, log, sqrt, etc.
- Calculators have predefined functions that return a value when given argument(s):
  - Provide x and it returns sin(x);
  - Arguments are passed to methods as parameters.

Functions/Methods Java

- In Java, functions are implemented as non-void methods;
- The Java `Math` class provides a library math methods;
- Predefined functions avoid reinventing the wheel & allow reusability of written and tested code:
  - If a function does not exist then write the method for it.
- Values are passed to a function in the argument list;
- A single result (value) is returned from the function.

Java Math Class

- In maths a function maps a set of inputs to an output value:

  z = f(x, y);

- Java Math class contains methods that perform non-trivial mathematical calculations;

- Is part of the **package.java.lang** which is automatically imported into every Java program.

Java Math Class

- It is **final**, has no subclasses (see later), has no constructors and methods are static:
  - The class name must is always used before the method name (slide 28);
  - e.g., **Math.sqrt(x);**
- Two constants:
  - **Math.E**
  - **Math.PI**

Some Math class Methods

Function	Argument	Constraints	Result
abs(x)	int/long/float/double		Same as arg
ceil(x)	double		double
floor(x)	double		double
round(x)	float/double		int/long
rint(x)	double		double
os(x)	double	x in radians	double
exp(x)	double		double
log(x)	double	x > 0	double
pow(x,y)	double	x^y, x > 0	double
sqrt(x)	double	x >= 0	double
max(x,y)	int/long/float/double		Same as arg
min(x,y)	int/long/float/double		Same as arg

Examples

Expression	Result
`Math.round(2.6)`	`3`
`Math.round(-3.15)`	`-3`
`Math.rint(-3.7)`	`-4.0`
`Math.ceil(3.7)`	`4.0`
`Math.floor(3.7)`	`3.0`
`Math.pow(2.0, 3.0)`	`8.0`
`Math.sqrt(-9)`	`NaN`
`Math.abs(-4)`	`4`
`Math.abs(4.0)`	`4.0`

Method Example: Pseudocode

```
MAIN:
    INPUT inches
    cms = convertInchesToCms <- inches
    OUTPUT "CMs are: " + cms
END MAIN

METHOD: convertInchesToCms
IMPORT: ins
EXPORT: cm
ALGORITHM:
    cm = ins * 2.54
END convertInchesToCms
```

Method Example: Java

```java
/* Title:       InchesToCmConverter
   Author:      David A. McMeekin
   Created:     06/09/2020
   Modified:    23/03/2022
   Desc:        Simple example class for Lecture Five*/

import java.util.*;
public class InchesToCmConverter
{
    public static void main(String[] args)
    {
        double inches = 0.0, cms = 0.0;
        Scanner sc = new Scanner(System.in);
        System.out.print("Input inches: ");
        inches = sc.nextDouble();
        cms = convertInchesToCms(inches);
        System.out.println("CMs are: " + cms);
    }

    private static double convertInchesToCms(double pIns)
    {
        double cm;
        cm = pIns * 2.54;
        return cm;
    }
}
```

Method Example 2: Pseudocode

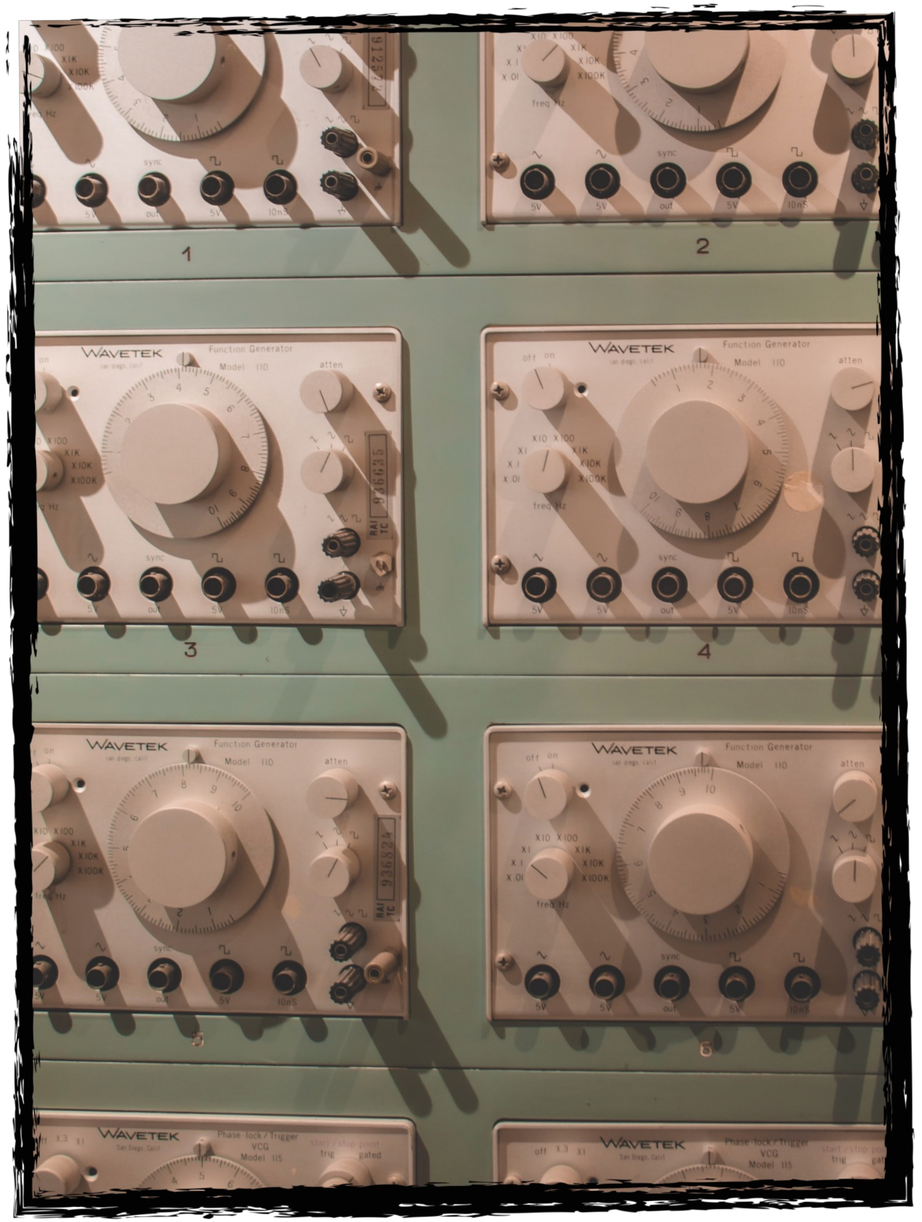
```
MAIN:
    INPUT x
    INPUT y
    average = calculateMean(x, y)
    OUTPUT average
END MAIN

METHOD: calculateMean
IMPORT: a, b
EXPORT: mean
ALGORITHM:
    mean = (a + b) / 2.0
END calculateMean
```

Method Example (2): Java

```java
/* Title:       CalculateTheMean
   Author:      David A. McMeekin
   Created:     06/09/2020
   Modified:    23/03/2022 */
import java.util.*;
public class CalculateTheMean
{
    public static void main(String[] args)
    {
        int x, y; double avg;
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter an integer: ");
        x = sc.nextInt();
        System.out.print("Enter another integer: ");
        y = sc.nextInt();
        avg = calculateMean(x, y);
        System.out.println("Mean " + x + " & " + y + " is: " + avg);
    }
    /*********************************************************
    * METHOD: calculateMean*
    * IMPORT: pA (int), pB (int)   *
    * EXPORT: mean (double)*
    *********************************************************/
    private static double calculateMean(int pA, int pB)
    {
        return (double)(pA + pB) / 2.0;
    }
}
```

Parameters


WAVETEK
Function Generator
Model 110
WAVETEK
Function Generator
Model 110
WAVETEK
Function Generator
Model 110
WAVETEK
Function Generator
Model 110
WAVETEK
Phase lock/Trigger
VCG
Model 115
WAVETEK
Phase lock/Trigger
VCG
Model 115

Method Parameters

- For some methods to work, they require data when called;
- Data used in the method call is an argument;
- This data arriving in the method is called a parameter;
- Parameters are local variables to the method.

Method Parameters (2)

- Parameters are initialised with the argument values;
- **pIns** and **localCM** are local variables to the method **convertInchesToCms();**
- **'p'** is the parameter's first letter indicating it's a parameter (optional).

```
…
    public static void main(String[] args)
    {
        double inches = 0.0, cms = 0.0;
        Scanner sc = new Scanner(System.in);
        System.out.print("Input inches: ");
        inches = sc.nextDouble();
        cms = convertInchesToCms(inches);
        System.out.println("CMs are: " + cms);
    }

    private static double convertInchesToCms(double pIns)
    {
        double localCM;
        localCM = pIns * 2.54;
        return localCM;
    }
…
```

Modular Grade Example Pseudocode

```
METHOD: markGrade
IMPORT: pMark (Integer)
EXPORT: grade (Character)
ALGORITHM:
    newMark = pMark DIV 10
    CASE newMark OF
  8, 9 or 10
grade = 'H'
  7
grade = 'D'
  6
grade = 'C'
  5
grade = 'P'
  DEFAULT
grade = 'F'
    ENDCASE
END markGrade
```

Modular Grade Example Java Code

```
public static char markGrade(int pMark)
{
    int newMark = pMark / 10; char grade;
    switch(newMark)
    {
        case 8: case 9: case 10:
            grade = 'H';
            break;
        case 7:
            grade = 'D';
            break;
        case 6:
            grade = 'C';
            break;
        case 5:
            grade = 'P';
            break;
        default:
            grade = 'F';
    }
    return grade;
}
```

Special/Interesting Cases

- Some programming languages pass parameters in a different manner:
  - It's called passing by reference (covered in UCP (COMP1000)).
- In Java, two data types are automatically passed by reference:
  - Objects; and
  - Arrays (a special type of object).
- A parameter which is of a particular class type will contain the object's address (a reference to the original object);
- Modification to the object are reflected in the original method;
- RELAX: Classes are covered in another lecture.

Worked Example: Times Table

Modular Algorithm Design: Pseudocode

```
MAIN:
    OUTPUT 'Welcome to the Times Tables!'
    maxTable = userInput(1, 12)
    FOR table = 1 TO maxTable CHANGEBY 1
        outputTable(table)
    ENDFOR
    ASSERTION: 1 to maxTable times table output to user.
END MAIN
```

Modular Algorithm Design: Pseudocode

- A possible algorithm to get user input:

```
METHOD: userInput
IMPORT: pLower (Integer), pUpper (Integer)
EXPORT: number (Integer)
ASSERTION: value will be in the range of lower and upper
ALGORITHM:
START:
    WHILE((number <= pLower) OR (number => pUpper)) DO
        OUTPUT Enter a number in the range 'pLower' to 'pUpper'
        number = GET user input
    ENDWHILE
    ASSERTION: pLower <= value <= pUpper
END inputValueFromUser
```

Modular Output Table: Pseudocode

```
METHOD: outputTable
IMPORT: pTable (Integer)
EXPORT: none
ASSERTION: pTable is in the range of 1 to 12
ALGORITHM:
    OUTPUT 'The ' pTable ' Times Table'
    FOR number = 1 TO 12 CHANGEBY 1
        OUPTUT pTable ' x ' number ' = ' (pTable x number)
    ENDFOR
    ASSERTION: pTable Times Table is output to the user
END outputTable
```

Modular Algorithm Design in Java: main()

```
import java.util.*;
public class TimesTable
{
    public static void main(String[] args)
    {
        int maxTable;
        System.out.println("Welcome to the times tables!");
        maxTable = userInput(1, 12);

        for(int table = 1; table <= maxTable; table++)
        {
            outputTable(table);
        }
    }
```

Modular Algorithm Design in Java: userInput()

```
public static int userInput(int lower, int upper)
{
    int number = -1;
    Scanner sc = new Scanner(System.in);

    while((number <= lower) || (upper <= number))
    {
        System.out.print("Enter a value between " + lower + " and " + upper + ":");
        number = sc.nextInt();
    }
    return number;
}
```

Modular Algorithm Design in Java: outputTable()

```
public static void outputTable(int table)
    {
        System.out.println("The " + table + " Times Table");
        for(int number = 1; number <= 12; number++)
        {
            System.out.println(table + " x " + number + " = " + (table * number));
        }
    }
}
```