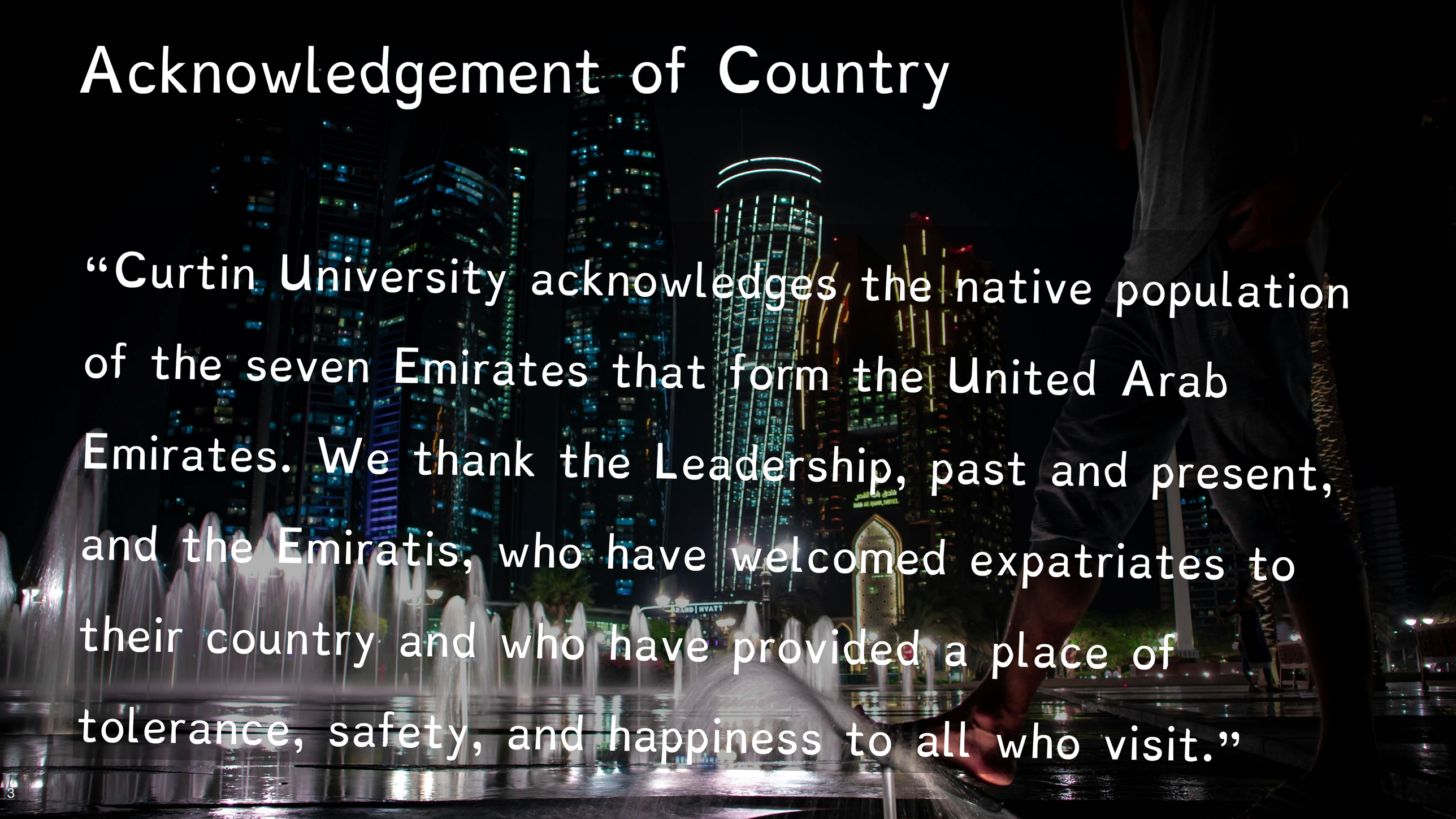
Dr David A. McMeekin

Coding Standards &

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.”

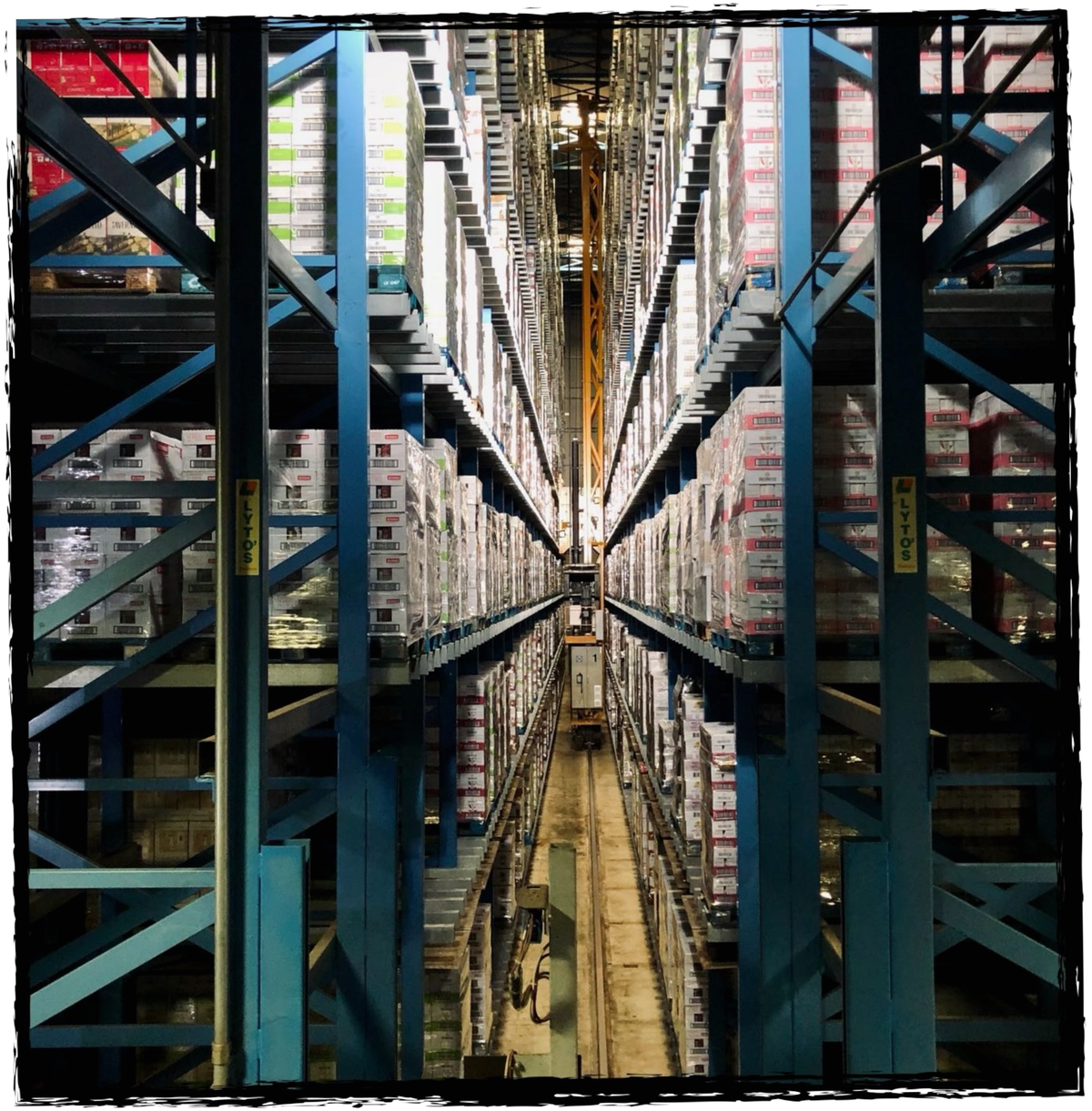
COMP1007 - Unit Learning Outcomes

- 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

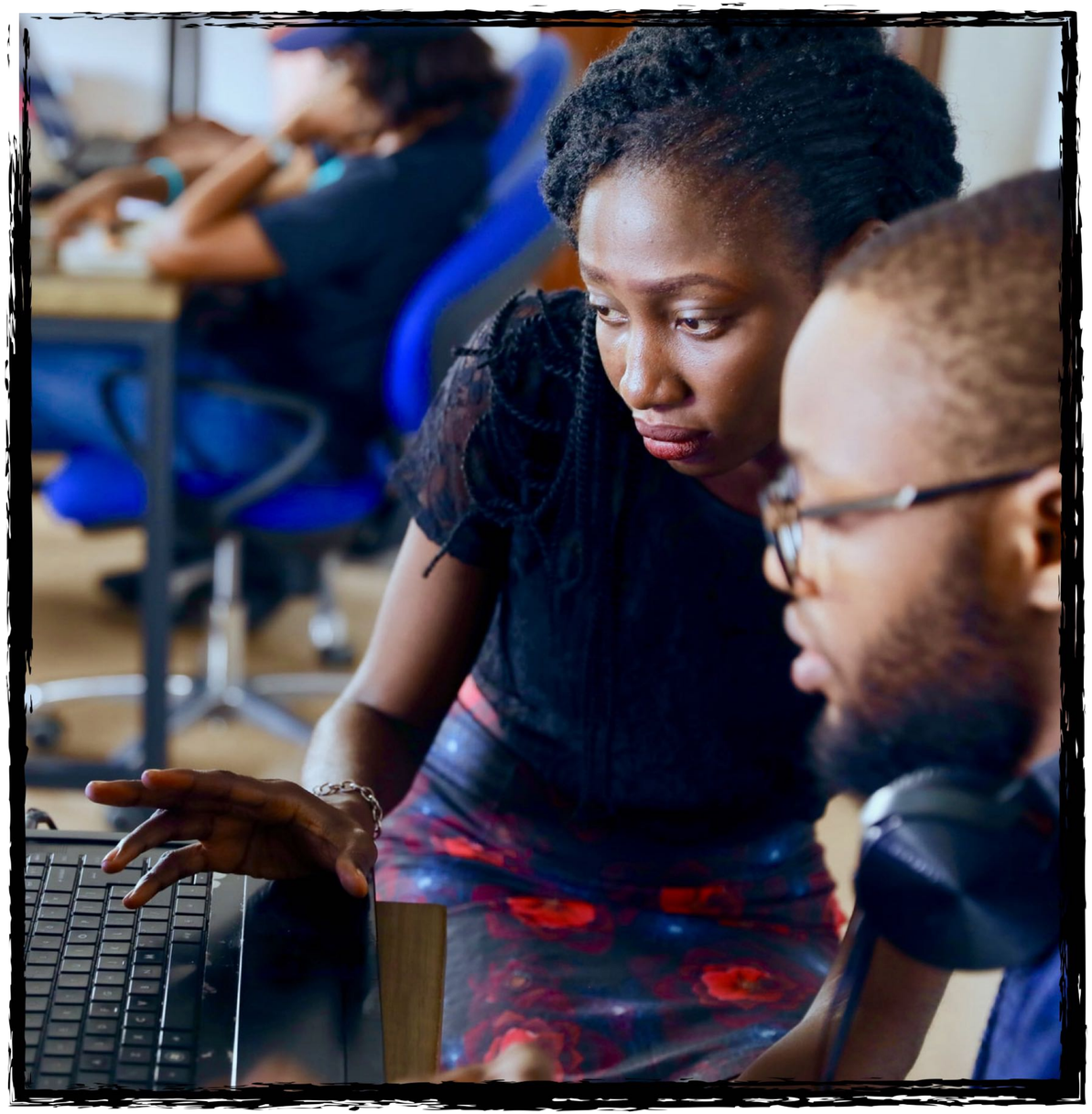
- Coding Standards;
- Boolean Operations;
- IF-THEN-ELSE; and
- Switch Statements.

Coding Standards



Coding Standards

- Code for Your Future Self
- What your code does today, may not be obvious tomorrow;
- What you were thinking today when you wrote this?
- Create code your future self will thank you for;
- Create code your team will thank you for.



In the Form of Naming

- Programmers constantly invent names for classes, methods, variables, constants, etc.
  - These names are called **identifiers**.
- Considerable thought must be given to each identifier;
- Ensures code is readable, understandable & maintainable;
- Names should be:
  - Unique;
  - Meaningful;
  - Unambiguous;
  - Consistent; and
  - Enhanced by case.

Rules for Identifiers

- Consists only of letters, digits, _ or $;
- Cannot start with a digit;
- Cannot be a reserved word;
- Case sensitive:

  StNo;

  stno;

  STNO;

  Stno;

- Are all different identifiers.
- Can be any length.

Guidelines for Identifiers

- Meaningful: name reflects the nature of the value it holds:
  `studentNum` is different to `numOfStudents`
- Readable:
  `studentNum` not `stdnbr`
- Consistent: be consistent in all aspects
  - Abbreviations, case, indentation, etc.
- Avoid overly long names:
  `theNumberOfStudentsAtTheUniversity`

Examples

- Good: `studentNum`
- Bad: `identification_number_of_the_student`
- Use underscore to good effect (Rarely);
- Use capitalisation to good effect (Always).

Java Identifier Naming Conventions

- Constants should be all uppercase:

```
public static final int MAXSTUDENTS = 30000;
```

- Class names should be Capitalised:

```
public class ThisIsAClass
```

- Internally capitalise methods and variables(camel case):

```
public void thisIsAMethod()
private double thisIsAVariable;
```

Indentation

- Indenting and spacing statements reduce cognitive load;
- Reduced cognitive load increases readability & maintainability;
- Indent code as you go, DON'T leave it for later.

Good Java Code

```java
import java.util.*;

public class ExampleOne
{
  public static void main(String[] args)
  {
      int x, y;
      double avg;

      Scanner sc = new Scanner(System.in);
      System.out.print("Enter 1st Number: ");
      x = sc.nextInt();
      System.out.print("Enter 2nd Number: ");
      y = sc.nextInt();
      avg = calculateMean(x, y);
      System.out.println("Mean of " + x + " & " + y + " = " + avg);
  } // End main

  public static double calculateMean(int a, int b)
  {
      return (double) (a + b) / 2.0;
  } // End calculateMean
} // End class
```

Bad Java Code

```
import java.util.*;
public class ExampleOne
{
public static void main(String[] args)
{
int x, y;
double avg;
Scanner sc = new Scanner(System.in);
System.out.print("Enter 1st Number: ");
x = sc.nextInt();
System.out.print("Enter 2nd Number: ");
y = sc.nextInt();
avg = calculateMean(x, y);
System.out.println("Mean of " + x + " & " + y + " = " + avg);
} // End main
public static double calculateMean(int a, int b)
{
return (double) (a + b) / 2.0;
} // End calculateMean
} // End class
```

No indentation, no line spacing and brackets not aligning make it very difficult to read and follow

Comments

- Statements in non-programming language describing the code;
- Comment blocks should be used:
  - For the program or class (later)
  - Include the authors name
  - Description of the purpose of the overall program/class
  - Dates and who modified

Comments

- Statements in non-programming language describing the code;

- Comment blocks should be used:
- For the program or class include:
  - Authors name;

  - Description of purpose of overall program/class; and

  - Dates and who modified.

Comments (2)

- To describe all methods:
  - Method Contract:

    `IMPORT`, `EXPORT` and `ASSERTION`
  - Method purpose; and
  - Dates and who modified.

- REQUIRED for ALL of your programs/classes.

Comment Your Code

- One comment type; inline comments:

```
// Anything in this line after the double slash is
// treated as a comment

int posTally; // Will keep a count of the positive
              // values > 0
```

- Another comment type; comment blocks:

```
/* Anything up to and including the close comment
   is treated as a comment */

/*************************************************
* Variables:                                     *
*       posTally; Will keep a count of the       *
*                 positive values > 0            *
*************************************************/
```

Comment Blocks

```
/**************************************************
* Author: David A. McMeekin *
* Purpose: To do something with my App.          *
* Date: 31/03/2029 *
**************************************************/
public class MyJavaApp
{
    public static void main(String [] args)
    {
        // Variable Declarations
        int a, b;
        double result;

        // Algorithm
        ... // Code
        result = myMethod(a, b);
    } // End Main

      //Continued on Next Slide
```

Comment Blocks (2)

```
/**************************************************
 * Purpose: To divide two integers (as Reals)    *
 * Date: 31/03/2029 *
 * Import: a (Integer)                           *
 *         b (Integer)                           *
 * Export: myVal (Real)                          *
 **************************************************/
    public static double myMethod(int a, int b)
    {
        // Variable Declarations
        double myVal;

        // Algorithm
        ... // Code
        return myVal;
    } // End myMethod

} // End Class
```

Boolean Operations

Boolean

- A data type that is either `true` or `false`;
- Can be used to reflect an on/off type status;
- Often used with control structures to decide:
  - between code blocks;
  - whether or not to repeat a code block.
- Example:

```
boolean isOdd, isPositive;
isOdd = false;
isPositive = true;
```

Equals Operator ==

- Testing for true or false, often equality is tested;
- The assignment operator is `=`
- To test for equality in primitive data types **==** is used;
- Example: `x == y;`
- This will make so much more sense shortly.

NB: The `==` operator cannot be used to test object equality (later).

Relational Operators

- Each evaluates to `true` or `false`;
- Each operator has an exact opposite;

Operator	Meaning	Example
==	is equal to	age == 50
>	is greater than	xPos > MAX
<	is less than	yPos < MIN
>=	is greater than or equal to	age >= 75
<=	is less than or equal to	age <= 19
!=	is not equal to	roofColour != RED

Logical Operators

- Three types:
  - Logical `AND` both are `true`;
  - Logical `OR` either are `true`;
  - Logical Negation `NOT` the opposite of;
- Used to combine relational operators to create more complex boolean expressions;
- Use parenthesis to ensure:
  - Evaluation occurs in correct order; and
  - Expression is readable.

Logical Operators

- Logical `AND` is `&&`
- Logical `OR` is `||`
- Logical Negation is `!`
- These expressions evaluate to **true** or **false**.

Logical Operators - Truth Table

Truth Table				
a	**b**	**a && b**	**a || b**	**!b**
TRUE	TRUE	TRUE	TRUE	FALSE
TRUE	FALSE	FALSE	TRUE	TRUE
FALSE	TRUE	FALSE	TRUE	FALSE
FALSE	FALSE	FALSE	FALSE	TRUE

Boolean Expression Examples

- Given that:

```
int a = 5, b = 10, c = 3;
boolean red = true, brown = false, blue = true;
```

Expression	Result
`red`	true
`a > c`	true
`(b - a) == c`	false
`red || brown`	true
`!(red && brown)`	true
`(brown && red) || blue`	true
`brown && (red || blue)`	false
`(a > c) && blue`	true
`!(!((b - a) == c))`	false
`blue && (b != c)`	true

Short Circuit Evaluation

- Minimises processing evaluating boolean expressions;
- Only evaluates as much as required;
- Logical `AND`:
  - `if first operand is false, entire expression is false, evaluation stops.`
- Logical `OR`:
  - `if first operand is true, entire expression is true, evaluation stops.`

Short Circuit Evaluation Examples

```
(x < 10) && (y > 50)
// When (x < 10) is false the expression is false

(life == 42) || (age > 100)
// When (life == 42) is true, the expression is true
```

IF-THEN-ELSE

Control Structures

- Allow portions of an algorithm to executed under specific conditions;

- Two basic control structure types:

  1. Selection: Given one or more possible choices: choose which section (if any) of an algorithm to execute;

  2. Repetition: Repeat a section of an algorithm while required conditions are met, iteration or looping.

- Boolean expressions used to make a choice (selection) or whether to repeat an algorithm section (repetition).

Selection Control Structures

- Two basic types:
  1. IF-THEN-ELSE statement:
     - provides between two possible alternatives;
  2. SWITCH/CASE statement:
     - provides multiple possible alternatives;

IF-THEN-ELSE

1. **IF-THEN**
   - Choice: execute code statements or not;

2. **IF-THEN-ELSE**
   - Choice: execute code statements one or code statements two;

- Writing control structure pseudocode, it must be clear for:
  - what statements are encapsulated in the control structure;
  - the logical expression controlling the actions taken by the control structure.

IF-THEN-ELSE (in Pseudocode)

- If `boolean_expression` is true, then execute statement(s):

```
IF boolean_expression THEN
    statement(s)
ENDIF
```

- If `boolean_expression` is true, then execute statement(s) `else` execute other statement(s):

```
IF boolean_expression THEN
    statement(s)
ELSE
    other statement(s)
ENDIF
```

Example: IF-THEN-ELSE (in Pseudocode)

```
IF (result < 0) THEN
    result = -result
ENDIF
// Assertion: result will be positive
```

```
IF inputPasswd = usrPasswd THEN
    OUTPUT "Access Granted"
// Do really cool things
ELSE
    OUTPUT "Invalid Entry"
ENDIF
// Assertion: Access granted with correct password entered
```

IF-THEN: Java Implementation

- Without `else`, the code will continues to the next code block;
- It is always good to use `{}` to ensure what code executes;

```
import java.util.*;

public class IfThen
{
    public static void main(String[] args)
    {
        int x = 3; int y = 5; int result = 0;
        result = x - y;
        if (result < 0)
        {
            result = -result;
        }
        System.out.println("result is:" + result);
    }
}
```

Multiple Decisions - Same Data

- For multiple alternative decisions where all decisions relate to the same data:

```
IF x is positive THEN
    INCREMENT posTally
ELSE IF x is negative THEN
    INCREMENT negTally
ELSE
    INCREMENT zeroTally
ENDIF
```

```
if(x > 0)
{
    posTally++;
}
else if(x < 0)
{
    negTally++;
}
else
{
    zeroTally++;
}
```

Multiple Decisions - Different Data

- For multiple alternative decisions where all decisions relate to different data:

```
IF age < 18 THEN
    INCREMENT childTally
ELSE
    IF criminal THEN
      INCREMENT crimTally
    ELSE
      INCREMENT nonCrimTally
    ENDIF
ENDIF
```

```
if(age < 18)
{
    childTally++;
}
else
{
    if(criminal)
    {
        crimTally++;

    }
    else
    {
        nonCrimTally++;

    }
}
```

Nesting IF-THEN-ELSE

- `if` statements can be nested, one inside the other;
- Example:
- `else` statements always matched to the nearest unmatched `if`
- Hence, use blocks `{ ... }`

```
if(x > 0)
{
    posTally++;
}
else
{
    if(x < 0)
    {
        negTally++;
    }
    else
    {
        zeroTally++;
    }
}
```

Order of Boolean Expressions

- Order of questions asked is extremely important;
- Before asking your partner about the wedding location, ask them to marry you;
- Consider the example:

```
IF mark >= 80 THEN
    grade = 'H'
ELSE IF mark >= 70 THEN
    grade = 'D'
ELSE IF mark >= 60 THEN
    grade = 'C'
ELSE IF mark >= 50 THEN
    grade = 'P'
ELSE
    grade = 'F'
ENDIF
```

```
IF mark >= 50 THEN
    grade = 'P'
ELSE IF mark >= 60 THEN
    grade = 'C'
ELSE IF mark >= 70 THEN
    grade = 'D'
ELSE IF mark >= 80 THEN
    grade = 'H'
ELSE
    grade = 'F'
ENDIF
```

- Are both of these alternatives valid?

Efficiency Considerations

- Order questions from most likely to least likely where possible;
- In the previous example, suppose the marks' distribution is:

Grade	% of Students
High Distinction	10
Distinction	15
Credit	30
Pass	35
Fail	10

- What impact does this have on the ordering of your choices?

Efficiency Considerations (2)

```
IF mark < 50 THEN
  grade = 'F'
ELSE IF mark < 60 THEN
  grade = 'P'
ELSE IF mark < 70 THEN
  grade = 'C'
ELSE IF mark < 80 THEN
  grade = 'D'
ELSE
  grade = 'H'
END IF
```

- Time for more testing.

Sequential vs Nested IF's

- Why is this code segment inefficient?

```
IF x IS POSITIVE THEN
  INCREMENT posTally
END IF
IF x IS NEGATIVE THEN
  INCREMENT negTally
END IF
IF x IS ZERO THEN
  INCREMENT zeroTally
END IF
```

```
if(x > 0)
  posTally++;

if(x < 0)
  negTally++;

if(x == 0)
  zeroTally++;
```

- If you are unsure, look at the same code in a previous slide;
- Novice programmers often use sequential rather than nested if's.

Beware of Impossible Conditions

- Consider this example:

```
if((y / x > 10) && (x != 0)) { ... }
```

- What will happen if x is equal to zero?

- Taking advantage of short circuit evaluation could lead to:

```
if((x != 0) && (y / x > 10)) { ... }
```

- This can be dangerous, a better alternative would be:

```
if(x != 0)
{
  if(y / x > 10)
  {
      ...
  }
}
```

Real Numbers and Equality

- Comparing real numbers in boolean expressions is possible;
- Comparing real numbers equality operator **==** is problematic;
- Numbers may only be ”equal” within a specified tolerance level;

$$\sqrt{3} = 1.732050 \qquad \sqrt{3} \times \sqrt{3} = 2.999997$$

- Deciding tolerance is crucial:
  - Too large: almost all numbers are considered equal;
  - Too small: almost all numbers are considered **NOT** equal;
- Hence: test whole numbers or to **N** decimal places.

Comparing Whole Numbers

- Two alternatives:

1. Type conversion (truncation will occur):

```
if((int)amount == value)
{
    ...
}
```

2. Use the **round()** method in the **Math** class

```
if(Math.round(amount) == value)
{
    ...
}
```

Comparing Reals within a Tolerance

- The tolerance must be valid for the application;
- Two decimal places, sufficient for money, not for turbine blade inspection!
- **IF amount IS WITHIN 0.01 OF 10.53 THEN**

```
if(Math.abs(amount - 10.53) < 0.01)
{
    ...
}
```

- **IF THE DIFFERENCE BETWEEN observedX AND trueX IS LESS THAN blade TOLERANCE THEN**

```
if(Math.abs(trueX - observedX) < blade.TOLERANCE)
{
    ...
}
```

Be Careful...

- Do NOT test if a boolean is equal to **true**/**false**, they are!

Good		Bad
if(x > 3 == true)	=>	if(x > 3)
if(y < 3 == false)	=>	if(!(y < 3))
if(((x - 3) == 0) == true)	=>	if(x - 3 == 0)
if(y < 3 != true)	=>	if(!(y < 3))

Switch Statements

Switch in Pseudocode

- Provides several alternatives where choice is based upon a single expression result:

```
Case expression
     case 1    Action1
     case 2    Action2
     ...        ...
     case n    ActionN
     else      ActionN+1
END Case
```

- Generic properties of a Switch statement:
  - Expression must involve a discrete data type:

    int, char, byte, short (i.e., non-real)
  - Each case must be a list of one or more constant values;
  - Equality is the only possible comparison;
  - Which statement executed for each constant value set must be clear.

Switch: Java

```
switch(expression)
{
   case const1:
        Statement_Set_1;
   break;
   case const2:
        Statement_Set_2;
   break;
   ...
   case constN:
        Statement_Set_N;
  break;
   default:
        Statement_Set_N+1;
}
```

Switch: Java (2)

- When the expression matches a case, the statements in the case are executed until:
  - A **break** statement is encountered; or
  - The **switch** statement's end is encountered.

- The **default** clause is optional:
  - **default** is executed if the **switch** expression does not match any case;
  - With no **default** and the **switch** expression is not matched the switch statement is exited (nothing happens).

Example

```
char status = 'G';
String studentStatus;
switch(status)
{
     case 'G':
          studentStatus = new String("Good Standing");
     break;
     case 'C':
          studentStatus = new String("Conditional");
    break;
     case 'T':
          studentStatus = new String("Terminated");
     break;
     default:
          studentStatus = new String("No Status");
     break;
}
```

Multiple case for Same Action

- If same action required for multiple cases, list all cases followed by the required action:

```
switch(month)
{
   case 1: case 3: case 5: case 7: case 8: case 10: case 12:
        daysInMonth = 31;
   break;
   case 4: case 6: case 9: case 11:
        daysInMonth = 30;
   break;
   case 2:
        daysInMonth = daysInFeb(year);
   break;  // Not optional as default is not supplied
}
// Assertion: Days in Month will contain
// 31: Jan, Mar, May, Jul, Aug, Oct and Dec
// 30: Apr, Jun, Sep and Nov
// 28/29: Feb
```

Common Mistakes with switch

- Ensure break statements are correctly placed;
- Compiler does not check this as break statement is not required to follow each case;

```
switch(choice)
{
    case '+':
        result = addition(numOne, numTwo);
    case '-':
        result = subtraction(numOne, numTwo);
}
```

- What is the error in the above statement?

IF-THEN-ELSE vs SWITCH

- Every switch can be converted to an if-else;
- Every if-else can NOT be converted to a switch;
- Consider the 'Mark/Grade' example:
  - Mark is a discreet datatype, so it can be used in a switch;
  - But there are too many possibilities;
  - Need to use a "trick".

Mark/Grade Example

```
newMark = mark DIV 10
CASE newMark

    8: 9: 10:
        grade = 'H'

    7:
        grade = 'D'

    6:
        grade = 'C'

    5:
        grade = 'P'

    DEFAULT:
        grade = 'F'
END CASE
```

```
newMark = mark / 10;
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';
}
```

References

Photo by Lagos Techie on Unsplash
Photo by Jon Tyson on Unsplash
Photo by Arno Senoner on Unsplash
Photo by ThisIsEngineering from Pexels
Photo by Nick Fewings on Unsplash
Photo by cottonbro from Pexels
Photo by emre keshavarz from Pexels
Photo by Vladislav Babienko on Unsplash
Photo by Alexander Schimmeck on Unsplash
Photo by Shawn Sim on Unsplash