Programming Design and Implementation

Lecture 4: Looping and Arrays

Dr David A. McMeekin
Discipline of Computing
School of Electrical Engineering, Computing and Mathematical Sciences (EECMS)

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 a Linux 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.

COMP5011 - Unit Learning Outcomes

- Develop and apply simple non-object oriented algorithms;
- Develop and implement simple classes in an object oriented language;
- Create object oriented designs consisting of classes connected by aggregation; and
- Communicate design and design decisions in a manner appropriate to the audience.

Outline

WHILE Loops

DO-WHILE Loops

Validation

FOR Loops

Arrays

WHILE Loops

Repetition AKA Looping

- Loop: a block of code repeated `0` to many times;
- Three available loops are:

  The difference is how the repetition is controlled

  1. `WHILE`:
     - Executes zero or more times
  2. `DO-WHILE`:
     - Executes one or more times
  3. `FOR`:
     - Executes a fixed number of times
- Choose the appropriate loop based what you want to do.

WHILE Loop

- Repetition controlled by a logical expression at top of loop;
- If the logical expression is `true`, the loop is entered, code inside loop is executed.

Pseudo Code:

```
WHILE boolExpression DO
    Body of loop
ENDWHILE
```

Java:

```java
while(boolExpression)
{
    statements;
}
```

`WHILE` Loop (2)

- The logical expression is checked **before** entering the loop:
  - If the logical expression is `false` the loop is **NOT** entered, program jumps to first statement after loop's body;
  - If the logical expression is `true`, the loop **IS** entered, body of loop is executed.

- After executing the code in the loop, the logical expression is checked again:
  - If the logical expression **IS** still `true`, execute code inside loop again;
  - If the logical expression is `false` the loop is **NOT** entered, program jumps to first statement after loop's body.

WHILE Loop - Menu Example - Pseudocode

```
close = FALSE
WHILE NOT close DO
   OUTPUT 'Enter Choice'
   INPUT choice
   CASE choice OF
      a OR A
         OUTPUT 'You entered' choice
      e OR E
         OUTPUT 'You entered' choice
         close = TRUE
      DEFUALT
         OUTPUT 'Invalid Choice'
   ENDCASE
ENDWHILE
```

WHILE Loop - Menu Example - Java Code

```java
public class WhileLoop
{
 public static void main(String[] args)
 {
     char choice;
     boolean close = false;
     while(!close)
     {
         Scanner input = new Scanner(System.in);
         System.out.print("Enter letter: ");
         choice = input.next().charAt(0);
         switch(choice)
         {
             case 'a': case 'A':
                 System.out.println("You entered:" + choice);
             break;
             case 'e': case 'E':
                 System.out.println("You entered:" + choice);
                 close = true;
             break;
             default:
                 System.out.println("Invalid Choice");
         }
     }
     input.close(); // Close the Scanner object.
 }
```

Infinite Loop

- Is one that can never end
- Three major causes:
  1. **Logical** expression can never be `false` (logical error);
  2. The **variable** within the logical expression never changes in the loop code (logical error); or
  3. **Semi-colon** in the wrong place (Syntax error).

- Good assertion statements usually mean that:
  - Infinite Loops rarely occur within your algorithm;
  - Infinite Loops occur because of typos;
  - **REASON**: you see what should be true for the loop to stop.

Logical Error (1)

Logical Expression can never be `false`

```
x = 0
WHILE x NOT EQUAL TO 11 DO
  OUTPUT x
  INCREMENT x BY 2
ENDWHILE
ASSERTION: x is equal to 11
```

Should be:

```
x = 0
WHILE x < 11 DO
  OUTPUT x
  INCREMENT x BY 2
ENDWHILE
ASSERTION: x >= 11
```

Logical Error (2)

- The variable within the logical expression never changes to eventually become `false`

```
INPUT x
WHILE x < 0 OR x > 10 DO
  OUTPUT "Invalid Input"
ENDWHILE
ASSERTION: 0 <= x <= 10
```

```
x = 0
WHILE x < 11 DO
  OUTPUT x
ENDWHILE
ASSERTION: x >= 11
```

Corrected:

```
INPUT x
WHILE x < 0 OR x > 10 DO
  OUTPUT "Invalid Input"
  INPUT x
ENDWHILE
ASSERTION: 0 <= x <= 10
```

Corrected:

```
x = 0
WHILE x < 11 DO
  OUTPUT x
  INCREMENT x BY 2
ENDWHILE
ASSERTION: x >= 11
```

Syntax Error

- Semi colon in wrong place, loop body is now outside the loop;

```
evensSum = 0;
nextNo = 0;
while(nextNo <= 100);
{
    evensSum = evensSum + nextNo;
    nextNo += 2; // add two to nextNo
} // Assertion: nextNo > 100
System.out.println(evensSum);
```

- The loop ends after the semi-colon following the while loop i.e., there are no statements in the loop
- The boolean expression is continually checked, nothing else.

Live Demo

- In this live demo we will look at:
  - The `while` loop; and
  - Infinite loops with logical and syntax errors.

DO-WHILE Loops

DO-WHILE

- Repetition controlled by a logical expression at bottom of loop;
- Loop body executes once before logical expression is checked;
- If logical expression is `true` the loop code executes again

Pseudo Code:

```
DO
    Body of loop
WHILE boolExpression
```

Java:

```java
do
{
    statements;
} while(boolExpression);
```

DO-WHILE (2)

- Logical expression is **NOT** checked before entering the loop
    - Loop is executed once prior to logical expression evaluation
    - If logical expression still `true`, execute code inside loop again
    - If logical expression is `false`, program is finished looping and jumps to the first statement after the body of the loop.
- The logical expression is repeatedly checked after the last statement in the loop is executed.

Example: Algorithm

```
DO
    INPUT age
WHILE age <= 0 OR age >= 110
ASSERTION: 0 < age < 110
```

- What is potentially wrong with this algorithm?

Example: Java

```java
int age;
Scanner sc = new Scanner(System.in);

do
{
    System.out.println("Enter Age");
    age = sc.nextInt();
} while((age <= 0) || (age >= 110));
sc.close();
// Assertion: 0 < age < 110
```

- The logic is correct, but no indication given to the user of what went wrong

Example: Possible Solution

- A possible starting template for you to use:
  - **NB**: it will evolve as we cover submodules, and again as we cover Exceptions

```
DO
    DISPLAY 'Please enter a value between X & Y'
    num = GET num
WHILE((num < x) OR (num > y))
ASSERTION: lower <= value <= upper
```

- The displayed message (prompt) works even on first loop;
- Creating accurate messages here is difficult.

Loop Equivalency

- A WHILE loop can be expressed as a DO-WHILE loop

```
WHILE x < 10 DO
    INCREMENT x BY 2
ENDWHILE
ASSERTION: x >= 10
```

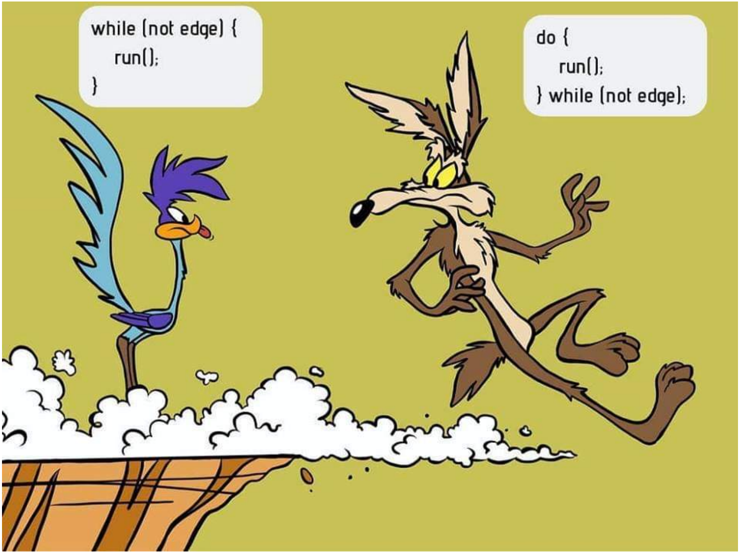
```
IF x < 10 THEN
    DO
        INCREMENT x BY 2
    WHILE x < 10
    ASSERTION: x >= 10
ENDIF
ASSERTION: x >= 10
```

- A DO-WHILE loop can be expressed as a WHILE loop

```
DO
    INCREMENT x BY 2
WHILE x < 10
ASSERTION: x >= 10
```

```
INCREMENT x BY 2
WHILE x < 10 DO
    INCREMENT x BY 2
ENDWHILE
ASSERTION: x >= 10
```

Think before you design



```
while (not edge) {
    run();
}
```

```
do {
    run();
} while (not edge);
```

Live Demo

- In this live demo we will look at:
  - The `do-while` loop; and
  - Infinite loops with logical and syntax errors.

Validation

Validating User Input

- Programs must protect against a unique error: `ID10T`;
- Is the input correct?
- Is the input actually correct?
- Are you sure the input is really correct?

Image from IMDB

Can You Repeat that Please?

- Not all users get it right the first time;
- Not all files actually exist;
- Not all files that exist permit you to access them;
- Not all data files contain the correct data;
- Validating the input is crucial and can save lives.

Validation Using an IF Statement - Pseudocode

```
number = 0
'Enter number between 1 and 6'
GET number
IF number between 1 and 6 THEN
   the magic happens here here
   run your code
ELSE
    PRINT 'Input was not in the required range'
ENDIF
```

Validation Using an IF Statement - Java

```java
import java.util.*;
public class IfValidation
{
    public static void main(String[] args)
    {
        int number = 0;
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a number between 1 and 6: ");
        number = sc.nextInt();
        sc.close();
        if(number < 7 && number > 0)
        {
            System.out.print("Look what happens now: AMAZING");
        }
        else
        {
            System.out.println("Input was outside of range!");
        }
    }
}
```

Validation Using a loop - Pseudocode

```
DO
  'Enter number between 1 and 6'
   GET number
   the magic happens here here
   run your code
WHILE number NOT between 1 and 6
END DO-WHILE
```

Validation Using a loop - Java

```
import java.util.*;
public class WhileValidation
{
    public static void main(String[] args)
    {
        int number = 0;
        Scanner sc = new Scanner(System.in);
        do
        {
            System.out.print("Enter a number between 1 and 6: ");
            number = sc.nextInt();
        }while(number > 7 && number < 0);
        sc.close();
    }
}
```

Exception

- '...something that doesn't follow a rule';
- Something that wasn't meant to happen;

- Possible exceptions for programmers:
  - user enters wrong data type;
  - file doesn't exist;
  - divide by zero.

- Dealing with this is called: **Exception Handling.**

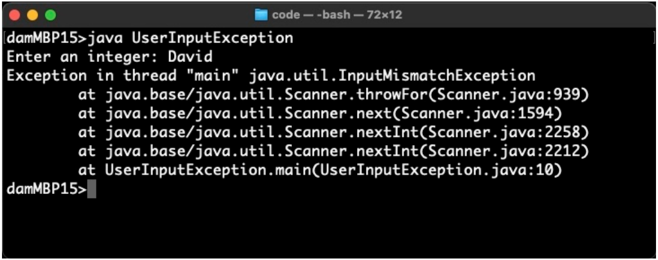
A Java Program - no Exception Handinling

```
import java.util.*;

public class UserInputException
{
   public static void main(String[] args)
   {
      int number = 0;
      Scanner sc = new Scanner(System.in);
      System.out.print("Enter an integer: ");
      number = sc.nextInt();
      System.out.println("The integer is:" + number);
      sc.close();
   }
 }
```

Exception Message

- Displayed by Java when the error was preventable;



```
damMBP15>java UserInputException
Enter an integer: David
Exception in thread "main" java.util.InputMismatchException
        at java.base/java.util.Scanner.throwFor(Scanner.java:939)
        at java.base/java.util.Scanner.next(Scanner.java:1594)
        at java.base/java.util.Scanner.nextInt(Scanner.java:2258)
        at java.base/java.util.Scanner.nextInt(Scanner.java:2212)
        at UserInputException.main(UserInputException.java:10)
damMBP15>
```

- Program asked for a number, user entered ‘David’

```
Exception in thread "main" java.util.InputMismatchException
```

Exception Handling

- try a method that may facilitate an error;
- The method throws an exception;
- Create a code block to catch the exception

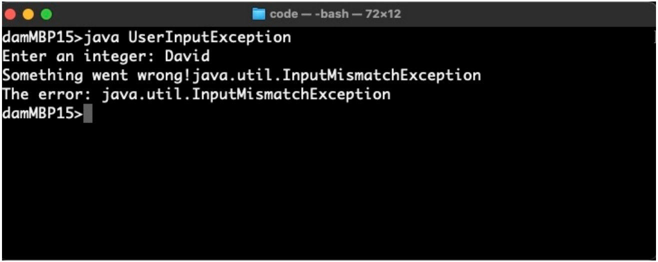
```
try
{
     //code that may throw an exception
}
catch(Exception someException)
{
     //code execute when exception happens
}
```

A Java Program - with Exception Handling

```
import java.util.*;
public class UserInputException
{
   public static void main(String[] args)
   {
      int number = 0;
      Scanner sc = new Scanner(System.in);
      try
      {
         System.out.print("Enter an integer: ");
         number = sc.nextInt();
         System.out.println("The integer is:" + number);
         sc.close();
      }
      catch(InputMismatchException error)
      {
         System.out.println("Something went wrong!" + error);
         System.out.println("The error: " + error);
      }
   }
 }
```

Error Handled by Exception handling

- A clean error message to the user;
- Program did NOT crash;
- Program exitedgracefully;
- Message could be written to log file.



```
damMBP15>java UserInputException
Enter an integer: David
Something went wrong!java.util.InputMismatchException
The error: java.util.InputMismatchException
damMBP15>
```

There are many more exceptions to be handled (some coming later).

Your Challenge

- Change the `UserInputException` program to loop back for user input when an exception is thrown.

Live Demo

- In this live demo we will look at:
  - Validation using loops; and
  - Exceptions related to user input.

FOR Loops

FOR Loop

- Is an extremely useful loop;
- Pseudo Code:

```
FOR count = startVal TO stopVal CHANGEBY increment
    OTHER_ACTIONS
ENDFOR
```

- The variable **increment** can be positive or negative
- **count** is known as the for loop **index**

Properties of a FOR Loop

- Loop index should always be a local variable;
- Loop index is **never** a Real number;
- Loop index is never explicitly modified inside the loop;
- The value of the loop index is undefined outside of the loop;
- `for` loop never executes if:
    - Positive increment and `stopVal` < `startVal`
    - Negative increment and `startVal` < `stopVal`

FOR Loops in Java

- Syntax:

```
for(initialisation; booleanExpression; increment)
{
    body_of_loop;
}
```

- Example:

```
sum = 0;
for(int count = 0; count < 10; count++)
{
    System.out.println("Count is: " + count);
    sum += count;
}
```

Declaring Loop Indexes

- Good programming practice says declare all local variables at start of method block
- A loop index is an exception because it is never referred to outside the for loop
- Java allows us to declare our variables anywhere:

```java
int sum = 0;
for(int count = 1; count <= 10; count++)
{
    System.out.println("Count is: " + count);
    sum += count;
}
```

- An attempt to refer to the loop index outside of the for loop will incur a compiler error

What Not to Use in Loops

- Three statements that *can* be used but should **NOT** in loops:

break	exit loop
continue	skip to next iteration of the loop
goto	go to LABEL (but not in Java)

```
for( ; ; )                    // Infinite Loop
{
    ...
    if(cond1) continue;       // Start the next Iteration
    else if(cond2) break;     // Exit the loop now
    else if(cond3) goto FRED; // Go to label FRED (Somewhere)
    ...                       // Neither cond1 or cond2 true
}
```

- Programming languages allow poor algorithm design and programming style;
- You should design great algorithms using great programming style.

FOR Loop Example: Algorithm

```
FOR index = 0 TO userNumber LENGTH CHANGEBY 1
    OUTPUT userNumber * 2
ENDFOR
```

FOR Loop Example: Java

```
Scanner input = new Scanner(System.in);
System.out.print("Enter an integer: "); //Prompt for user input
int userNumber = input.nextInt();
for(int i = 0; i < userNumber; i++)
{
    System.out.println(i * 2);
}
input.close();
//ASSERTION: Output from 0 - userNumber will be doubled
```

FOR Loop Example (2): Algorithm

```
ASSERTION: if n is 0 or negative, then nFactorial is 1

ALGORITHM:
    nFactorial = 1
    FOR i = 2 TO n CHANGEBY 1
        nFactorial = nFactorial * i
    ENDFOR

ALTERNATE ALGORITHM:
    nFactorial = 1
    FOR i = n DOWNTO 2 CHANGEBY -1
        nFactorial = nFactorial * i
    ENDFOR
```

FOR Loop Example (2): Java

```
/*******************************************************
 * ASSERTION: if n 0 or negative, then nFactorial is 1 *
 *******************************************************/
int n = 5;
long nFactorial = 1;
for(int i = 2; i <= n; i++)
{
    nFactorial *= (long)i;
    System.out.println(nFactorial);
}

// ------------------------------------------------------
// Try this one

long nFactorial = 1;
for(int i = n; i >= 2; i--)
{
    nFactorial *= (long)i;
}
```

Live Demo

- In this live demo we will look at:
  - The `for` loop;
  - Logical and syntax errors.

Arrays

Arrays

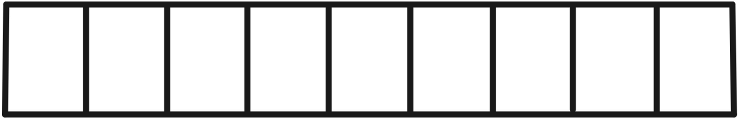
- Variables represent a single item:
  - e.g., `int numTimes;` is a single Integer number
- We also work with *sets* of similar data:
  - e.g., a list of student marks in PDI.
  - How do we work with this?

    ```
    double student1Mark,student2Mark, ..., studentXMark;
    ```

- Calculating the average involves a massive amount of typing;
- Can't conveniently pass the student set around.

Arrays (2)

- Arrays solve this problem;
- A simple data structure to store sets of data;
- An array is a variable that contains *many* elements.

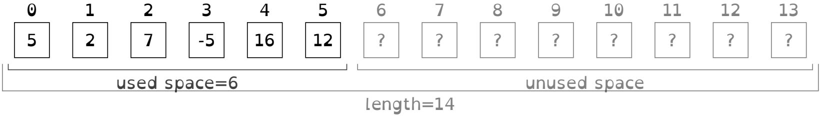


Array Properties

- Elements located sequentially in memory:
  - the array is a *contiguous* block of memory
- All elements must have same data type
  - e.g., `double`
- Arrays can be initialised to any size (within memory limits);
- Once initialised they **cannot** be resized;
- A new array must be created and the old array contents copied over to it.

Arrays - Accessing Elements

- Once created, you need to work with the array elements;
- Elements are accessed via an *index* or *subscript*;
- The *index* is the element number in the array;
- Arrays are numbered: 0...,to N-1, N is the allocated length;
- To access an element: **theArrayName[elementNumber]**
- In the below example, **theArray**[0] contains 5.



0	1	2	3	4	5	6	7	8	9	10	11	12	13
5	2	7	-5	16	12	?	?	?	?	?	?	?	?

used space=6

unused space

length=14

Array Properties

- Array **capacity** (length) vs **actually used** elements;
- Initialisation, also referred to as *allocation*;
- An array initialised to `length = 20;` *reserves space* for **20** elements;
- Typically keep track of how many array elements are *actually used*:
  - i.e., the count of used elements, as distinct from array size.
- Allocate more space than required, as arrays cannot be resized.

Declaring and Allocating Arrays in Java

- Declaring arrays use: []

  ```
  double[] theArray;
  ```

  - Arrays of any data type can be created (including classes);

- Allocating arrays: use keyword `new` with [] syntax;

  ```
  theArray = new double[100];
  ```

- `theArray` now has `100` elements of data type `double`

Accessing and Copying Arrays

- To access elements: `theArray[index]`, `index` must be a positive `int`;
- `index` is the element to access in the array;
- Negative indexes or indexes past the array end (i.e., `>= length`) cause a runtime error.

- Assignment: `sameArray = theArray;` does NOT copy `theArray` into `sameArray`;
  - `sameArray` points to `theArray`;
  - The L.H.S variable points to the R.H.S variable;
  - Same when an array is a method parameter (covered later).

Java Code - Arrays

```
import java.util.*;
public class UserInputException
{
   public static void main(String[] args)
   {
      int []  theArray;
      theArray = new int[100];
      int theArrayLength = theArray.length;

      for(int i = 0; i < theArrayLength; i++)
      {
          theArray[i] = i * i;
      }
      for(int i = 0; i < theArrayLength; i++)
      {
          System.out.println("theArray["+i+"] is: " + theArray[i]);
      }
   }
 }
```

Live Demo

- In this live demo we will look at:
  - Arrays;
  - Accessing elements outside of the array; and
  - Objects of Primitive data types.