Activity: Evaluating Criterion A Samples in Groups

You will work in groups of five to evaluate two IA Criterion A examples.

1. Form groups and open the document

    - Your teacher will assign you to a breakout room.
    - One person in the group opens this document and makes a copy for the group:

      https://docs.google.com/document/d/1fph1LKZMnHGNvolP7b4erjEmRL7XwyzKXVMi70sTdPQ/edit?usp=sharing

2. Share screen and read together

    - The same person should share their screen in the breakout room.
    - As a group, read Example 1 and Example 2 slowly and discuss what you notice.

3. Use the checklist and rubric

    - For each example, work through the checklist in the document.
    - Discuss every statement and tick the check box as Met or Not met based on the text.
    - Using the rubric in the document, agree on the mark out of 4 that you think each example deserves for Criterion A and type that mark in the space provided, with a short reason if you can.

4. Return and share

    - When we return from breakout rooms, your group will briefly share:

        - The mark you gave each example
        - One strength and one weakness you noticed for each example

Total time for this activity: 20 minutes
(About 3–4 minutes to set up, 6–7 minutes per example, and 3–4 minutes to prepare what you will share.)

1. B2.3.3 Construct programs that utilize looping structures

1.1 What is a loop

A loop is a control structure that allows a program to repeat a block of code several times.

In Java there are three common loop structures:

- `for`
- `while`
- `do while`

The syllabus expects you to understand:

- Types of loops, including counted loops and conditional loops, and when each is appropriate
- How to place conditional statements inside loops that use Boolean and relational operators to control behaviour

You should be able to:

- Read and trace existing loops
- Construct your own loops to solve simple problems
- Recognize infinite loops and avoid them

1.2 Counted loops in Java the for loop

A counted loop is used when you know in advance how many times you want to repeat an action.

1.2.1 General form

```
for (initialization; condition; update) {

    // statements to repeat

}
```

Order of execution:

1. Initialization runs once at the beginning

2. Condition is checked
3. If condition is true, the body runs
4. Update runs
5. Go back to step 2

1.2.2 Example 1 sum of the first five natural numbers

```
public class SumFirstFive {

    public static void main(String[] args) {

        int sum = 0;


        for (int i = 1; i <= 5; i++) {

            sum = sum + i;

        }


        System.out.println("Sum is " + sum);

    }

}
```

Explanation:

• `i` is the loop counter
• The loop starts at `i = 1`
• As long as `i <= 5` is true, the body executes
• In each iteration, `sum` is increased by the current value of `i`
• At the end, `sum` equals `1 + 2 + 3 + 4 + 5`

Trace table:

Iteration	i before test	Condition i <= 5	sum before	sum after	i after update
1	1	true	0	1	2
2	2	true	1	3	3
3	3	true	3	6	4
4	4	true	6	10	5
5	5	true	10	15	6
6	6	false	15	15	stop

Appropriate uses of counted loops:

• Repeating a task a fixed number of times
• Traversing arrays and ArrayLists from index zero to the last index

---

1.3 Conditional loops while and do while

Conditional loops are used when you do not know exactly how many repetitions are needed. Instead, you keep repeating while a condition is true.

1.3.1 The while loop

General form:

```
while (condition) {

    // statements to repeat

}
```

The body may run zero times or many times.
The condition is checked at the top of the loop.

Example 2 input validation with a while loop

```
import java.util.Scanner;


public class InputValidation {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        int number = -1;


        while (number < 0 || number > 10) {

            System.out.print("Enter a number from 0 to 10: ");

            number = sc.nextInt();

        }


        System.out.println("You entered: " + number);

        sc.close();

    }

}
```

Key ideas:

• The condition uses relational operators < and >
• The condition also uses Boolean operator || which means logical OR
• The loop repeats while the number is outside the allowed range

Trace table for user inputs 20, 11, 5:

Loop pass	number before test	Condition (number < 0 or number > 10)	User enters	number after input	Loop continues
1	minus 1	true	20	20	yes
2	20	true	11	11	yes
3	11	true	5	5	yes
4	5	false	no input	5	no

Appropriate uses of a while loop:

• Repeat until input becomes valid
• Repeat until a sentinel value is entered
• Repeat until a computation reaches a certain state

1.3.2 The do while loop

General form:

```
do {

    // statements

} while (condition);
```

The difference from `while` is that the body always runs at least once.
The condition is checked at the bottom.

Example 3 simple menu with do while

```
import java.util.Scanner;
```

```
public class MenuExample {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int choice;

        do {
            System.out.println("1. Say Hello");
            System.out.println("2. Say Goodbye");
            System.out.println("3. Exit");
            System.out.print("Enter choice: ");
            choice = sc.nextInt();

            if (choice == 1) {
                System.out.println("Hello");
            } else if (choice == 2) {
                System.out.println("Goodbye");
            } else if (choice != 3) {
                System.out.println("Invalid choice");
            }

        } while (choice != 3);
```

```
        sc.close();
    }
}
```

Here you see:

• A conditional loop that repeats until `choice` equals 3
• Conditional statements inside the loop with relational operators `==` and `!=`
• Use of `if`, `else if`, and `else` to control what happens in each repetition

---

1.4 Nested loops

A nested loop is a loop inside another loop.

Every time the outer loop runs once, the inner loop runs through all its iterations.
Nested loops are very common when working with tables, grids, or two dimensional arrays.

1.4.1 Concept of nested loops

Think of nested loops as:

• Outer loop controls rows
• Inner loop controls columns

or

• Outer loop controls days
• Inner loop controls hours

1.4.2 Example 4 multiplication table using nested for loops

```
public class MultiplicationTable {

    public static void main(String[] args) {

        for (int row = 1; row <= 3; row++) {

            for (int col = 1; col <= 3; col++) {
```

```
                int product = row * col;
                System.out.print(product + " ");
            }
            System.out.println();
        }
    }
}
```

Output:

```
1 2 3
2 4 6
3 6 9
```

Explanation:

- Outer loop variable `row` goes from 1 to 3
- For each row, inner loop variable `col` goes from 1 to 3
- For each pair `(row, col)`, we compute `row * col` and print it

High level trace of loop structure:

row	col values executed	Printed values
1	1, 2, 3	`1 2 3`
2	1, 2, 3	`2 4 6`
3	1, 2, 3	`3 6 9`

1.4.3 Nested loops with conditions inside

You can place `if` statements inside nested loops.
This allows you to create patterns or selectively process elements.

Example 5 diagonal pattern with stars and dots

```
public class DiagonalPattern {

    public static void main(String[] args) {

        int size = 4;


        for (int row = 0; row < size; row++) {

            for (int col = 0; col < size; col++) {

                if (row == col) {

                    System.out.print("* ");

                } else {

                    System.out.print(". ");

                }
```

```
            }
            System.out.println();
        }
    }
}
```

Here:

• The condition `row == col` uses a relational operator
• When `row` equals `col`, a star is printed
• Otherwise a dot is printed
• The outer loop controls which row you are in
• The inner loop controls the column in that row

1.5 Using while loops for numeric processing extracting digits

A classic pattern is to use a while loop to extract digits from a number.

1.5.1 Example 6 print digits of a number from last to first

```
public class ExtractDigits {
    public static void main(String[] args) {
        int number = 4725;

        while (number > 0) {
            int digit = number % 10;
            System.out.println(digit);
            number = number / 10;
        }
```

```
    }
}
```

Explanation:

• Condition `number > 0` is a relational test
• `number % 10` gives the last digit of the number
• `number / 10` removes the last digit
• The loop continues until `number` becomes zero

Trace table:

Loop pass	number before	digit = number % 10	number after division	Printed digit
1	4725	5	472	5
2	472	2	47	2
3	47	7	4	7
4	4	4	0	4
5	0	loop ends		

This shows a conditional loop in action with arithmetic and relational operators.

---

2. B2.2.2 Construct programs that apply arrays and ArrayLists

The syllabus requires you to understand:

• One dimensional arrays in Java
• Two dimensional arrays in Java

• ArrayList in Java as a dynamic list
• How to add, remove and traverse elements in a dynamic list

Everything in this section will use Java.

2.1 One dimensional arrays in Java

2.1.1 Concept

An array is a collection of elements of the same type stored in a fixed sequence.
A one dimensional array is like a row of boxes, each box containing a value.

Properties:

• Fixed length once created
• Elements are accessed using an integer index
• Indexes start at zero and go up to `length minus one`

2.1.2 Declaring, creating and initializing

Declare and create an array with a specific size:

```
int[] scores = new int[5];   // 5 integers, all initialized to 0
```

Array literal with initial values:

```
int[] scores = {80, 92, 75, 60, 88};
```

2.1.3 Traversing a one dimensional array

To process each element, you usually use a counted `for` loop.

Example 7 printing all scores

```
public class ArrayTraverse {

    public static void main(String[] args) {

        int[] scores = {80, 92, 75, 60, 88};
```

```
        for (int i = 0; i < scores.length; i++) {
            System.out.println("Score " + i + " is " + scores[i]);
        }
    }
}
```

Explanation:

- `scores.length` gives the number of elements
- The loop iterates from `i = 0` to `i = scores.length minus 1`
- `scores[i]` accesses the element at index `i`

Traversing arrays is directly linked to the idea of counted loops from B2.3.3.

2.2 Two dimensional arrays in Java

2.2.1 Concept

A two dimensional array can be thought of as a table of values with rows and columns.

Visual example:

```
1  2  3
4  5  6
```

This can be stored as `int[][] grid`.

2.2.2 Declaring and initializing a 2D array

Example with values:

```
int[][] grid = {
```

```
    {1, 2, 3},
    {4, 5, 6}
};
```

• `grid.length` is the number of rows
• `grid[row].length` is the number of columns in that row

2.2.3 Traversing a 2D array with nested loops

Example 8 printing a 2D array

```
public class TwoDArrayExample {
    public static void main(String[] args) {
        int[][] grid = {
            {1, 2, 3},
            {4, 5, 6}
        };

        for (int row = 0; row < grid.length; row++) {
            for (int col = 0; col < grid[row].length; col++) {
                System.out.print(grid[row][col] + " ");
            }
            System.out.println();
        }
    }
}
```

Explanation:

• The outer loop goes through each row
• The inner loop goes through each column in the current row
• This is a perfect example of nested loops from B2.3.3 used together with arrays from B2.2.2

2.3 ArrayList in Java a dynamic list

2.3.1 Concept

An `ArrayList` is a class in the Java Collections framework. It behaves like a resizable array.

Differences from arrays:

• Size can grow and shrink at run time
• Methods are used to add and remove elements
• You can store objects, usually with a type parameter such as `ArrayList<String>`

2.3.2 Creating an ArrayList

```
import java.util.ArrayList;


public class ArrayListCreate {

    public static void main(String[] args) {

        ArrayList<String> names = new ArrayList<>();


        names.add("Alice");

        names.add("Bob");

        names.add("Charlie");


        System.out.println(names);
```

```
    }
}
```

Here:

• `ArrayList<String>` is a list that will store strings
• `add` appends a new element at the end

2.3.3 Traversing an ArrayList

Indexed traversal:

```
for (int i = 0; i < names.size(); i++) {
    System.out.println("Name " + i + ": " + names.get(i));
}
```

Enhanced for loop:

```
for (String name : names) {
    System.out.println(name);
}
```

Both methods are important at IB level.

2.3.4 Adding and removing elements

Adding:

```
names.add("Dana");          // add at the end
names.add(1, "Ethan");      // insert at index 1
```

Removing:

```
names.remove("Bob");       // remove first occurrence of "Bob"

names.remove(0);           // remove element at index 0
```

Dynamic behaviour:

• After insertion or removal, the list automatically adjusts
• You do not need to manage the size manually as with arrays

Example 9 working with an ArrayList

```
import java.util.ArrayList;


public class ArrayListExample {

    public static void main(String[] args) {

        ArrayList<String> names = new ArrayList<>();


        names.add("Alice");

        names.add("Bob");

        names.add("Charlie");


        System.out.println("Initial list: " + names);


        names.add(1, "Dana");

        System.out.println("After inserting Dana: " + names);


        names.remove("Bob");
```

```
        System.out.println("After removing Bob: " + names);


        System.out.println("Traversing list:");
        for (String name : names) {
            System.out.println(name);
        }
    }
}
```

This example demonstrates:

• How a dynamic list grows
• How an element can be inserted in the middle
• How an element can be removed
• How to traverse all elements

---

3. IB style practice questions

Below are practice questions using IB command terms.
You can convert some of them into formative assessments or homework.

3.1 Questions for B2.3.3 looping structures

1. Define the term counted loop as used in Java. [2]

2. Define the term conditional loop as used in Java. [2]

The following Java code is intended to compute the sum of integers from 1 to 4 inclusive.

```
int sum = 0;
for (int i = 1; i < 4; i++) {
```

```
    sum = sum + i;
}
System.out.println(sum);
```

3. a. Trace the execution of this program by constructing a table showing the values of `i` and `sum` for each iteration of the loop. [4]
   b. State the output of the program. [1]
   c. Explain how to correct the loop so that it correctly sums the integers from 1 to 4 inclusive. [3]

4. Explain one situation where a `for` loop is more appropriate than a `while` loop, and one situation where a `while` loop is more appropriate than a `for` loop. [4]

5. Construct a Java program that repeatedly asks the user for a positive integer and stops when the user enters zero. At the end, the program should print the total number of positive integers that were entered. Use a conditional loop. [6]

Consider the following code fragment:

```
int number = 753;
while (number > 0) {
    int digit = number % 10;
    number = number / 10;
}
```

6. a. Describe what this loop does to the variable `number`. [3]
   b. Explain how you could modify the code so that all digits of the initial number are printed. [4]

7. Describe what is meant by a nested loop. [2]

Construct a Java program that uses nested `for` loops to print the following pattern:

```
*
* *
```

```
* * *
```

8. Each line contains one more star than the previous line. [6]

9. In the menu program using a `do while` loop, the condition is `choice != 3`.

a. Explain how this condition governs the execution of the loop. [3]
b. Discuss one advantage and one disadvantage of using a `do while` loop rather than a `while` loop for such menu driven programs. [4]

3.2 Questions for B2.2.2 arrays and ArrayLists in Java

10. Define a one dimensional array in Java. [2]

11. Define a two dimensional array in Java. [2]

12. Describe the difference between an array and an `ArrayList` in Java. In your answer, refer to size and operations on elements. [4]

13. Construct a Java method `printScores` that receives an array of integers and prints each score on a separate line with its index. Example output for `{80, 92}` should be:

```
Score 0 is 80

Score 1 is 92
```

[6]

Consider the following 2D array:

```
int[][] grid = {
    {1, 2, 3},
    {4, 5, 6}
};
```

14. a. State the value of `grid[0][2]`. [1]
b. Construct a pair of nested `for` loops that prints all the values in `grid` in row major order (row by row). [5]

15. Construct a Java program that uses an `ArrayList<String>` to store the names of students in a class. The program should:

- start with an empty list
- add three names
- remove one of the names
- print the resulting list

[6]

16. Explain how an `ArrayList<String>` can be used as a dynamic list of names to which new entries can be added or removed at run time. Your answer should mention at least two relevant `ArrayList` methods. [4]

A student writes the following code:

```
ArrayList<Integer> values = new ArrayList<>();

values.add(10);

values.add(20);

values.add(30);


for (int i = 0; i <= values.size(); i++) {

    System.out.println(values.get(i));

}
```

17. a. Identify the error that will occur when this code runs. [1]
b. Explain why this error occurs. [3]
c. Construct a corrected version of the loop. [3]

18. Discuss the factors that a programmer should consider when choosing between a fixed size array and an `ArrayList` for storing data in an IB Computer Science solution. [6]