

Surname: .....

First Name: .....

Student ID: .....



VICTORIA UNIVERSITY OF  
**WELLINGTON**  
TE HERENGA WAKA

**EXAMINATIONS – 2024**  
**TRIMESTER 2 (TEST 1)**  
**FRONT PAGE**

**COMP 103**  
**INTRODUCTION TO**  
**DATA STRUCTURES**  
**AND ALGORITHMS**  
**SEP 5, 2024**

**Time allowed:**           **50 MINUTES**

**Permitted**               **CLOSED BOOK**

**materials:**            You are allowed to use a (paper) language dictionary during the test. Electronic dictionaries, apps, and other digital resources are not permitted.

**Instructions:**        Attempt ALL **5** questions  
The test will be marked out of a total of **50** marks.  
You will be provided with a *concise Java documentation for Collections* and a *brief Java documentation*.  
Record your answers in their desinated spaces.  
If you find any question unclear, request clarification from the invigilator.  
Assume that all necessary libraries for programming are already imported.

<b>Question</b>	<b>#1</b>	<b>#2</b>	<b>#3</b>	<b>#4</b>	<b>#5</b>	<b>Total</b>
Max Points	9	8	17	10	6	<b>50</b>
Mark						

**SPARE PAGE FOR EXTRA ANSWERS**

Cross out rough working that you do not want marked. Specify the question number for work that you do want marked.

---

**1. Properties of Collections****[9 marks]**

In questions (a) - (c), circle the *best* answer:

- (a) To quickly retrieve a student's name by their ID (Integer type), while maintaining the natural order of the IDs, which collection from the list below should you use? Assume that the IDs are unique for each student. **[2 marks]**
- i. ArrayList
  - ii. HashMap
  - iii. TreeSet
  - iv. TreeMap
- (b) In an online auction system, you need to keep track of active bids on items, and you need to efficiently retrieve and remove the highest bid when it is won. Which collection would you use for this purpose? **[2 marks]**
- i. HashSet
  - ii. PriorityQueue
  - iii. TreeMap
  - iv. Stack
- (c) You are building a spell-checking feature for a text editor. You need to store a dictionary of words in a way that allows for efficient lookup of words (i.e., check words are contained in the dictionary) and doesn't store duplicate words. Which one would you use for this purpose? **[2 marks]**
- i. ArrayDeque
  - ii. HashMap
  - iii. HashSet
  - iv. TreeMap
- (d) Consider the piece of code given below:

```
List<String> st = new Stack<>();  
st.add("First");  
st.add("Second");
```

Which of the following statements might result in an error? Justify your answers. **[3 marks]**

- i. String topElement = st.pop();
- ii. String topElement = st.remove("First");
- iii. st.clear();

Justification:

**2. Using collections****[8 marks]**

Mathematical expressions often use various types of brackets (e.g., parentheses `()`, square brackets `[]`, curly braces `{}`). An expression has balanced brackets if each opening bracket has a corresponding matching closing bracket in the correct order.

Examples of unbalanced brackets are:

- `((a + b) * t / 2 * (1 - t))` where there are 3 opening brackets and 2 closing brackets,
- `{ [b * b - (4 * a * c) ] / (2 * a) }` where there is no closing bracket corresponding to `[`, and no opening bracket corresponding to the last `}`.

To detect unbalanced brackets in expressions with multiple bracket types, we can use a stack data structure. Here's the algorithm:

1. Scan the expression from left to right
2. If an opening bracket is encountered, push it onto the stack
3. If a closing bracket is encountered:
  - a. If the stack is empty, the expression is unbalanced
  - b. Pop the top element from the stack
  - c. If the popped bracket doesn't match the current closing bracket, the expression is unbalanced
4. After scanning the entire expression, if the stack is not empty, the expression is unbalanced

Complete the `isBalanced` method on the facing page to detect unbalanced brackets in mathematical expressions using a stack. Your method should:

- Handle parentheses `()`, square brackets `[]`, and curly braces `{}`.
- Return `true` if the expression has balanced brackets, `false` otherwise.
- Follow the algorithm described above.

Note that the method uses a `Map` to record the pairs of corresponding brackets which are of `Character` type.

Complete the `isBalanced` method in the provided template.

**Note:** `char` is a primitive data type that represents a single character. `Character` is the wrapper class for the `char` type. To retrieve a character from a specific index `ind` in a `String`, use the `String` method `charAt(ind)`. For example:

```
String str = "Hello, World!";
char ch = str.charAt(7);
UI.println(ch); // Output: 'W'
```

Note that in Java, single quotes are used to enclose char literals, such as `'a'`, while double quotes are used for String literals, such as `"Hello"`.

```
public boolean isBalanced(String expression) {
    Stack<Character> stack = new Stack<>();
    Map<Character, Character> brackets = new HashMap<>();
    brackets.put(')', '(');
    brackets.put(']', '[');
    brackets.put('}', '{');
    // YOUR CODE HERE

}
```

**3. More on using collections****[17 marks]**

You are maintaining a library management program written in Java. The program currently lacks the required functionalities, and the new manager has requested several updates.

In the program, there is a Book class:

```
public class Book {
    private String title;
    private int isbn;
    private int year;
    private boolean available;

    public Book(String title, int isbn, int year, boolean available) {
        this.title = title;
        this.isbn = isbn;
        this.year = year;
        this.available = available;
    }

    public String getTitle() { return title;}
    public int getISBN() { return isbn;}
    public int getYear() { return year;}
    public boolean isAvailable() { return available;}
    public String toString() { return title + ": " + isbn;}
}
```

Additionally, there is a field `allBooks` which is a list of all books in the library's inventory:

```
private List<Book> allBooks;
```

**a. Sort books alphabetically by title: [3 marks]**

Implement a method that returns a list of books sorted by their titles alphabetically (ascending). Use a List of Book objects and the `Collections.sort()` method with a custom Comparator named `titleComparator` to achieve this.

```
private Comparator<Book> titleComparator = (b1, b2) ->
    // YOUR CODE HERE

public List<Book> sortByTitle() {
    // YOUR CODE HERE

}
```







**f. Check if a book is available by ISBN: [2 marks]**

Implement a method that checks if a book with a given ISBN is available in the library.

```
public boolean isBookAvailableByISBN(int isbn) {
    // YOUR CODE HERE

}
```

**4. Complexity analysis****[10 marks]**

Analyse the time complexity of the findPairs method given below. For each line, specify the cost and the number of times that code executes. Then, calculate the total time complexity for the entire method. Assume that the input size is  $n$ .

```
public List<List<Integer>> findPairs(List<Integer> numbers) {

    List<List<Integer>> pairs = new ArrayList<>();    // cost=O( 1 ), times= 1

    for (int i = 0; i < numbers.size(); i++) {      // cost=O(    ), times=

        for (int j = i + 1; j < numbers.size(); j++) { // cost=O(    ), times=

            List<Integer> pair = new ArrayList<>(); // cost=O(    ), times=

            pair.add(numbers.get(i));                // cost=O(    ), times=

            pair.add(numbers.get(j));                // cost=O(    ), times=
            pairs.add(pair);

        }
    }
    return pairs;

} // total cost = O(    )
```



