



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

COMP 103
SOLUTIONS
SEP 5, 2024

Time allowed: 50 MINUTES

Permitted materials: **CLOSED BOOK**

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.

Question	#1	#2	#3	#4	#5	Total
Max Points	9	8	17	10	6	50
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:

```
-> i. This statement will result in a compilation error because st is of the
  ↪ reference type List<String>, and pop() is a method specific to the Stack
  ↪ class.

-> ii. The remove(Object o) method when called on a list returns a boolean so it
  ↪ cannot be assigned to topElement which is defined to be a String reference
  ↪ (incompatible types).
```

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

    //Step 1
    for (int i = 0; i < expression.length(); i++) {
        char ch = expression.charAt(i);
        // Step 2
        if (ch == '(' || ch == '[' || ch == '{') {
            stack.push(ch);
        }
        //Step 3
        else if (ch == ')' || ch == ']' || ch == '}') {
            // Substep 3.a
            if (stack.isEmpty()) {
                return false;
            }
            // Substep 3.b
            char top = stack.pop();
            // Substep 3.c
            if (top != brackets.get(ch)) {
                return false;
            }
        }
    }
    // Step 4
    return stack.isEmpty();
}
```

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
    b1.getTitle().compareTo(b2.getTitle());

public List<Book> sortByTitle() {
    // YOUR CODE HERE
    List<Book> sortedBooks = new ArrayList<>(allBooks);
    Collections.sort(sortedBooks, titleComparator);
    return sortedBooks;
}
```

b. Find the list of all available books: [3 marks]

Write a method that filters out all available books, stores them in a new List and returns it.

```
public List<Book> availableBooks() {  
    // YOUR CODE HERE  
  
    List<Book> availableBooks = new ArrayList<>();  
    for (Book book : allBooks) {  
        if (book.isAvailable()) {  
            availableBooks.add(book);  
        }  
    }  
    return availableBooks;  
}
```

c. Search books by title keyword: [3 marks]

Implement a method that searches for books by a keyword in their titles and returns a list of all matching books. The search should be case-insensitive.

```
public List<Book> findMatch(String keyword) {  
    // YOUR CODE HERE  
  
    List<Book> matchingBooks = new ArrayList<>();  
    for (Book book : allBooks) {  
        if (book.getTitle().toLowerCase().contains(keyword.toLowerCase())) {  
            matchingBooks.add(book);  
        }  
    }  
    return matchingBooks;  
}
```

d. Count books by year: [3 marks]

Create a method that counts the number of books published each year. It returns a `Map<Integer, Integer>` where the key is the publication year and the value is the count of books published in that year.

```
public Map<Integer, Integer> countByYear() {
    // YOUR CODE HERE

    Map<Integer, Integer> yearCountMap = new HashMap<>();
    for (Book book : allBooks) {
        int year = book.getYear();
        if (yearCountMap.containsKey(year)){
            yearCountMap.put(year, yearCountMap.get(year)+1);
        } else {
            yearCountMap.put(year, 1);
        }
    }
    return yearCountMap;
}
```

e. Use the ISBN numbers as the keys: [3 marks]

List is not the best collection for recording the books because searching is not efficient on a list. To improve that create a field of type `Map<Integer, Book>` called `bookByISBN` and implement a method to copy all books from `allBooks` list to this map. Use the ISBN numbers as the keys.

```
// YOUR CODE HERE

private Map<Integer, Book> bookByISBN = new HashMap<>();

public void moveToMap(List<Book> books) {
    for (Book book : books) {
        bookByISBN.put(book.getISBN(), book);
    }
}
```


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

    Book book = bookByISBN.get(isbn);
    if (book != null) {
        return book.isAvailable();
    }
    return false;
}
```

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( 1 ), times= n

        for (int j = i + 1; j < numbers.size(); j++) { // cost=O( 1 ), times= n(n-1)/2

            List<Integer> pair = new ArrayList<>(); // cost=O( 1 ), times= n(n-1)/2

            pair.add(numbers.get(i)); // cost=O( 1 ), times= n(n-1)/2

            pair.add(numbers.get(j)); // cost=O( 1 ), times= n(n-1)/2
            pairs.add(pair); // cost=O( 1 ), times= n(n-1)/2
        }
    }
    return pairs; // cost=O( 1 ), times= 1
} // total cost = O( n^2 )
```

Note regarding the for loops:

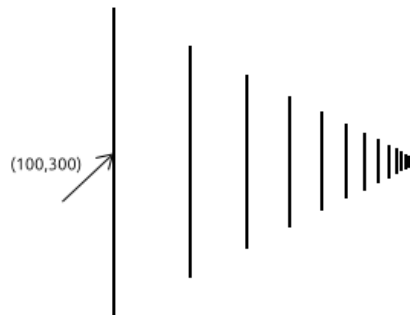
The complexity of the lines starting with `for` is implicitly included inside the for
 ↪ loops. As such it is not necessary to include them in the calculations explicitly
 ↪ as above.

5. Recursion**[6 marks]**

a. Complete the following recursive `drawLines(x,y,length)` method to draw a sequence of vertical lines. **[3 marks]**

- You must use a *first-and-rest* recursion: `drawLines` should draw the first line and then call itself recursively to draw the rest of the lines.
- The first vertical line to the left should be centered at (x,y) with length equal to `length` pixels.
- The next line should be `length/4` pixels to the right from the previous line *and* its length should be three quarters ($3/4$) of the previous line's length.
- It should continue to draw lines as long as the length is larger than 3.

For example, `drawLines(100, 300, 200)` should output the following:



```
public void drawLines(double x, double y, double length) {
    // YOUR CODE HERE

    if (length <= 3) {
        return; // Stop recursion if length is too small
    }

    // Draw the vertical line
    UI.drawLine(x, y - length / 2, x, y + length / 2);

    // Recursive call for the next line
    drawLines(x + length / 4, y, 3*length / 4);
}
```

- b. A *palindrome* is a word that reads the same left to right as right to left. Examples are “racecar”, “radar”, “level”, “noon” and “madam”. Write a recursive method that determines whether its argument string `str` is a palindrome. If it is a palindrome it returns `true` otherwise `false`. For simplicity, assume that `str` consists only of the lowercase letters a to z. [3 marks]

Hint: You may want to use the `String` methods `charAt(int ind)` (see page 5) and/or `substring(int j, int k)` (see the documentation).

```
private static boolean isPalindrome(String str) {
    // YOUR CODE HERE

    // Base case 1: if the string is empty or has one character
    if (str.length() <= 1) {
        return true;
    }

    if (str.charAt(0) != str.charAt(str.length() - 1)) {
        return false;
    }
    // Recursive case
    return isPalindrome(str.substring(1, str.length() - 1));
}
```
