STATE UNIVERSITY OF ZANZIBAR (SUZA)
School of Computing Communication and Media Studies (SCCMS)

CycleSheet II — Programming Assignment

Course Code:	IT6003	**Programme:**	MSc. Information
Course Name:	Advanced Programming Using Java	**Semester:**	I
Total Questions:	10	**Total Marks:**	100

AI Usage Policy: The use of AI tools (ChatGPT, GitHub Copilot, etc.) is permitted as a *learning aid* only. You must:

-2
1. Declare any AI usage in your submission
2. Be able to explain every line of your code during viva
3. Understand that plagiarism detection tools will be used
4. Accept that inability to explain your code results in **zero marks**

Submission Format: ZIP file named `RegNo_CycleSheet2.zip` containing all `.java` source files organized by question number.

Grading Criteria:

Criterion	Weight
Compilation & Execution	15%
Correctness of Output	30%
Concept Implementation	25%
Code Quality & Documentation	15%
Edge Case Handling	10%
Originality	5%

Section A: Collections Framework [20 marks]

Question 1: Library Book Management System [10 marks]

Design a **Library Book Management System** using the Java Collections Framework.

- Create a `Book` class with fields: `isbn` (String), `title` (String), `author` (String), `year` (int), `copies` (int).
- `Book` must implement `Comparable<Book>` (compare by title alphabetically).
- Override `equals()` and `hashCode()` based on `isbn`.

Implement the following using appropriate collection types:

a) Use a `HashMap<String, Book>` to store books by ISBN. Implement add, remove, and search operations. [2 marks]

b) Use a `TreeSet<Book>` to maintain books sorted by title. Display all books in alphabetical order. [2 marks]

c) Use a `LinkedList<Book>` as a queue for book reservation requests. Implement `reserveBook()` and `processReservation()`. [2 marks]

d) Implement a method that uses `Collections.sort()` with a custom `Comparator` to sort books by year (newest first). [2 marks]

e) Use an `Iterator` to traverse the HashMap and remove all books with zero copies. Print removed books. [2 marks]

Question 2: Student Course Registration System [10 marks]

Build a course registration system using multiple collection types:

- `Student` class: `regNo` (String), `name` (String), `gpa` (double)
- `Course` class: `code` (String), `name` (String), `maxCapacity` (int)

a) Use a `HashMap<String, Set<Student>>` to map course codes to enrolled students. Ensure no duplicate students per course (use `HashSet`). [2 marks]

b) Use a `PriorityQueue<Student>` (ordered by GPA descending) for a waitlist when a course is full. [3 marks]

c) Implement `dropCourse()` that removes a student and automatically enrolls the highest-GPA student from the waitlist. [3 marks]

d) Use `Collections.unmodifiableMap()` to provide a read-only view of the registration data. Demonstrate that modifications throw `UnsupportedOperationException`. [2 marks]

Section B: Generics [16 marks]

Question 3: Generic Data Structure Library [8 marks]

Implement a generic **Pair** and **Stack**:

a) Create a generic class `Pair<K, V>` with methods `getFirst()`, `getSecond()`, `swap()` (returns new Pair with swapped values), and `toString()`. [2 marks]

b) Create a generic class `BoundedStack<T extends Comparable<T>>` that:

- Has a maximum capacity (set in constructor)
- Implements `push()`, `pop()`, `peek()`, `isEmpty()`, `isFull()`
- Has a method `findMin()` that returns the minimum element using `compareTo()`

[3 marks]

c) Write a generic method `<T> void printArray(T[] array)` that prints any array type. [1 mark]

d) Write a generic method with wildcards: `double sumOfList(List<? extends Number> list)` that sums any list of numbers (Integer, Double, Float, etc.). [2 marks]

Question 4: Generic Repository Pattern [8 marks]

Implement a generic data access pattern:

a) Define a generic interface:

```
public interface Repository<T, ID> {
    void save(T entity);
    T findById(ID id);
    List<T> findAll();
    void delete(ID id);
    boolean exists(ID id);
}
```

[1 mark]

b) Create `InMemoryRepository<T, ID>` that implements this interface using a `HashMap`. The entity must implement a `Identifiable<ID>` interface with method `ID getId()`. [3 marks]

c) Create `Employee` and `Department` classes implementing `Identifiable<Integer>`. [2 marks]

d) Demonstrate polymorphism by using `Repository<Employee, Integer>` and `Repository<Department, Integer>` references. [2 marks]

Section C: Lambda Expressions and Stream API [20 marks]

Question 5: Functional Programming with Lambdas [10 marks]

a) Create a functional interface `MathOperation` with method `double operate(double a, double b)`. Use lambda expressions to implement: addition, subtraction, multiplication, division, power,

and modulus. Store them in a `Map<String, MathOperation>` and build a calculator that takes operator name and operands. [3 marks]

b) Given a list of strings, use `Predicate<String>`, `Function<String, String>`, and `Consumer<String>` to:
   - Filter strings longer than 5 characters
   - Transform them to uppercase
   - Print each with its length

   Chain these using `and()`, `or()`, `compose()`, `andThen()`. [3 marks]

c) Implement a method that accepts a `Comparator<Student>` as a parameter. Call it with different lambda expressions to sort by: name, GPA, registration number. Use method references where possible. [2 marks]

d) Create a generic `filter()` method that takes `List<T>` and `Predicate<T>` and returns a filtered list. Test with Integer and String lists. [2 marks]

Question 6: Stream API Data Processing [10 marks]

Given a CSV file `employees.csv` with columns: `id,name,department,salary,yearsOfService,city`

Create sample data with at least 20 employees, then use the Stream API to:

a) Read the CSV file and create a `List<Employee>` using `Files.lines()` and `stream().map()`. [2 marks]

b) Find the top 3 highest-paid employees using `sorted()` and `limit()`. [1 mark]

c) Group employees by department and calculate average salary per department using `Collectors.groupingBy` and `Collectors.averagingDouble()`. [2 marks]

d) Partition employees into two groups: salary above and below the overall average, using `Collectors.partitio` [1 mark]

e) Create a salary report string using `Collectors.joining()`: each line shows "Name — Department — Salary". [1 mark]

f) Use `reduce()` to find the total salary expenditure. [1 mark]

g) Use parallel streams to process the data. Compare execution time with sequential streams using `System.nanoTime()`. Print both times. [2 marks]

Section D: File I/O and NIO [16 marks]

Question 7: File Processing System [8 marks]

Build a log file analyzer:

a) Generate a sample log file (`server.log`) with 1000+ lines in the format:

```
2025-01-15 10:23:45 [INFO] User login:  user123
2025-01-15 10:24:12 [ERROR] Database connection failed
2025-01-15 10:25:00 [WARNING] Memory usage at 85%
```

   Include at least 4 log levels: INFO, ERROR, WARNING, DEBUG. [1 mark]

b) Use `BufferedReader` to read and count occurrences of each log level. Display a summary table. [2 marks]

c) Use `java.nio.file.Files` and `Path` to:
   - Extract all ERROR lines to a separate file `errors.log`
   - Extract all lines from a specific date range to `filtered.log`

   [3 marks]

d) Use `try-with-resources` throughout. Implement proper exception handling with custom exception `LogParseException`. [2 marks]

Question 8: Object Serialization System [8 marks]

a) Create a `StudentRecord` class implementing `Serializable` with fields: regNo, name, courses (List¡String¿), grades (Map¡String, Double¿). Mark the `transient` keyword on a `password` field. [2 marks]
b) Implement `saveRecords(List<StudentRecord>, String filename)` using `ObjectOutputStream`. [1 mark]
c) Implement `loadRecords(String filename)` using `ObjectInputStream`. Handle `ClassNotFoundException` and `InvalidClassException`. [2 marks]
d) Add a `serialVersionUID`. Demonstrate what happens when you modify the class and try to deserialize old data (change a field type). Explain in comments. [1 mark]
e) Implement the same save/load using **JSON format** (manually format as text, no external libraries). Compare file sizes. [2 marks]

Section E: JDBC and Database Connectivity [16 marks]

Question 9: Student Records Database Application [8 marks]

Build a complete CRUD application using JDBC with SQLite:

a) Create a database with two tables:

```
CREATE TABLE departments (
    dept_id INTEGER PRIMARY KEY AUTOINCREMENT,
    dept_name TEXT NOT NULL UNIQUE
);

CREATE TABLE students (
    reg_no TEXT PRIMARY KEY,
    name TEXT NOT NULL,
    email TEXT,
    dept_id INTEGER,
    gpa REAL DEFAULT 0.0,
    FOREIGN KEY (dept_id) REFERENCES departments(dept_id)
);
```

Use `DriverManager.getConnection()` and `Statement` to create tables. [2 marks]
b) Implement all CRUD operations using `PreparedStatement` (prevent SQL injection). Create a `StudentDAO` class with methods: `insert()`, `findByRegNo()`, `findAll()`, `update()`, `delete()`. [3 marks]
c) Implement a method that joins both tables to display students with their department names. Use `ResultSet` to process results. [1 mark]
d) Implement **transaction management**: a method `transferStudent()` that moves a student to a new department. Use `setAutoCommit(false)`, `commit()`, and `rollback()` in case of failure. [2 marks]

Question 10: Database Report Generator [8 marks]

Extend Question 9:

a) Insert at least 30 sample students across 5 departments using `batch processing` (`addBatch()` and `executeBatch()`). [2 marks]
b) Generate the following reports using SQL queries via JDBC:
- Department-wise student count
- Top 5 students by GPA
- Students with GPA below 2.0 (academic probation)
- Average GPA per department

Format output as aligned tables using `System.out.printf()`. [3 marks]

c) Use `ResultSetMetaData` to dynamically print column names and types for any query result. [1 mark]

d) Implement connection pooling simulation: create a class `SimpleConnectionPool` that manages a fixed number of connections using a `BlockingQueue<Connection>`. Implement `getConnection()` and `releaseConnection()`. [2 marks]

Section F: Design Patterns [12 marks]

This section is a bonus/advanced section. Attempting it demonstrates mastery.

Question 11: Implement Three Design Patterns [12 marks]

Choose and implement **three** of the following design patterns in a unified **Online Shopping System** context:

a) **Singleton Pattern:** Create a `ShoppingCart` class that ensures only one cart instance exists per session. Include `addItem()`, `removeItem()`, `getTotal()`. Make it thread-safe using double-checked locking. [4 marks]

b) **Factory Pattern:** Create a `PaymentFactory` that produces different payment processors (`CreditCardPayme` `MobileMoneyPayment`, `BankTransferPayment`) based on a string parameter. Each implements a `Payment` interface with `processPayment(double amount)`. [4 marks]

c) **Observer Pattern:** Implement a notification system where `Product` is the subject. When price changes, notify all registered `Observer` objects (`EmailNotifier`, `SMSNotifier`, `AppNotifier`). [4 marks]

d) **Strategy Pattern:** Create different shipping strategies (`StandardShipping`, `ExpressShipping`, `FreeShipping`) implementing a `ShippingStrategy` interface. The `Order` class uses the strategy to calculate shipping cost. [4 marks]

Provide a `Main.java` that demonstrates all three chosen patterns working together in a single order workflow.

End of CycleSheet II