FPT Education
FPT UNIVERSITY

CSD201 - PRACTICAL EXAM
SUMMER 2024
Duration: 85 minutes

Problem 1: Task Management System using Linear Data Structures (5 points)

A task management system needs to manage tasks with their priorities. Each task has the following attributes:

- Task ID (string)
- Task description (string)
- Priority (integer)

Requirements:

1. **Data Structure:**
   - Design an appropriate linear data structure to store and manage tasks.
2. **Add Task: (1 point)**
   - Write a function `void addTask(String id, String description, int priority)` to add a new task to the system. If the task ID already exists, update the task information.
3. **Remove Task: (1 point)**
   - Write a function `void removeTask(String id)` to remove a task from the system based on the task ID.
4. **Search Task: (1 point)**
   - Write a function `Task searchTask(String id)` to search and return the information of a task based on the task ID.
5. **List Tasks: (1 point)**
   - Write a function `List<Task> listTasksByPriority()` to return a list of all tasks sorted by their priority in descending order.
6. **List High Priority Tasks: (1 point)**
   - Write a function `List<Task> listHighPriorityTasks(int minPriority)` to return a list of tasks that have a priority equal to or higher than `minPriority`.
7. **Main function:**
   - Insert the main function below into the `class TaskManagement` to check the function calls.

```java
public static void main(String[] args) {
    TaskManagement system = new TaskManagement();
    system.addTask("T1", "Task 1 description", 5);
    system.addTask("T2", "Task 2 description", 8);
    system.addTask("T3", "Task 3 description", 3);

    System.out.println("List of Tasks by Priority: " +
        system.listTasksByPriority());
    System.out.println("Search Task T1: " + system.searchTask("T1"));

    system.removeTask("T2");
    System.out.println("List of Tasks after removing T2: " +
        system.listTasksByPriority());

    System.out.println("List of High Priority Tasks (priority >= 4): "
        + system.listHighPriorityTasks(4));
}
```

Note: Use appropriate linear data structures to ensure the add, remove, and search operations have the lowest possible complexity. Write the code on 1 file, named the file using your student ID: StudentID_P1.java.

Problem 2: Address Book Management using Binary Search Tree (BST) (5 points)

An address book needs to manage contacts by phone number. Each contact has the following attributes:

- Phone number (string)
- Contact name (string)
- Email address (string)

Requirements:

1. **Data Structure:**
   - Design an appropriate Binary Search Tree (BST) data structure to store and manage contacts by phone number.
2. **Add Contact: (1.5 points)**
   - Write a function `void addContact(String phone, String name, String email)` to add a new contact to the address book. If the phone number already exists, update the contact information.
3. **Remove Contact: (1.5 points)**
   - Write a function `void removeContact(String phone)` to remove a contact from the address book based on the phone number.
4. **Search Contact: (1 point)**
   - Write a function `Contact searchContact(String phone)` to search and return the information of a contact based on the phone number.
5. **List Contacts: (1 point)**
   - Write a function `List<Contact> listContacts()` to return a list of all contacts in the address book in ascending order of phone numbers.
6. **Main Function:**
   - Insert the main function below into the `class AddressBook` to check the function calls.

```java
public static void main(String[] args) {
    AddressBook addressBook = new AddressBook();
    addressBook.addContact("0999123456", "Dao Tao",
        "daotao@fe.edu.vn");
    addressBook.addContact("0900121212", "Khao Thi", "Khao
        Thi@fe.edu.vn");
    addressBook.addContact("0999123456", "Phong Dao Tao",
        "phongdaotao@fe.edu.vn");

    System.out.println("List of Contacts: " +
        addressBook.listContacts());
    System.out.println("Search Contact 0999123456: " +
        addressBook.searchContact("0999123456"));
    addressBook.removeContact("0900121212");
    System.out.println("List of Contacts after removal: " +
        addressBook.listContacts());
}
```

Note: Use the Binary Search Tree (BST) data structure to ensure the add, remove, and search operations have the lowest possible complexity. Write the code on 1 file, named the file using your student ID: StudentID_P2.java.