قوليتيکنيك بروني
POLITEKNIK BRUNEI

END-OF-SEMESTER EXAMINATION (FORMAL)

SEMESTER 2, 2022-23

SCHOOL OF INFORMATION & COMMUNICATION TECHNOLOGY

NS4307

NETWORK PROGRAMMING

TIME ALLOWED: 2 HOURS

INSTRUCTIONS TO CANDIDATES:

- **DO NOT TURN OVER THE PAGE UNTIL YOU ARE TOLD TO DO SO.**
- Write down the details required on the front cover of the answer booklet provided. Make sure you complete the details on any additional answer booklets that you use.
- Answer ALL questions.
- The total mark for this paper is **70**. The number of marks for each question or part question is shown in brackets.
- Your answer booklets and any other paper used in the examination must be handed in to the invigilators before leaving the room or hall. You may take your own examination paper with you unless otherwise instructed.
- Write in **dark blue** or **black** ink pen.

MATERIALS SUPPLIED:

Answer booklet (1).

MATERIALS PERMITTED:

Calculator.

STRUCTURED QUESTIONS [TOTAL MARKS: 70]

Answer **ALL** questions.

QUESTION 1 [Total marks: 14]

a) Describe Version Control. **[2 marks]**

b) There are **three (3)** types of Version Control Systems (VCS) and one of them is Local Version Control Systems. List the other **two (2)** types of VCS. **[1 mark]**

c) Illustrate the basic structure of Local Version Control Systems. **[3 marks]**

d) Explain Git's **three (3)** main states: modified, staged **and** committed, that your files can reside in. **[3 marks]**

e) State **and** explain the **two (2)** types of files in Git working directory. **[3 marks]**

f) Explain the difference between the command git commit and git push. **[2 marks]**

QUESTION 2 [Total marks: 14]

a) "Java Binary Input/Output is more efficient than Text Input/Output." Justify the statement.

[1 mark]

b) The following Table 1 and Table 2 are two Java classes named SaveStudentFile.java and LoadStudentFile.java, respectively, with an empty binary file, Table 3, named student.dat.

```
SaveStudentFile.java
public class SaveStudentFile {
  public static void main(String[] args) {
    try(DataOutputStream output = new DataOutputStream(new
        FileOutputStream("student.dat", false))) {
      output.writeInt(123);
      output.writeUTF("Jailani");
      output.writeDouble(98.5);
      output.writeUTF("SICT");
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
}
```

Table 1

```
LoadStudentFile.java
public class LoadStudentFile {
  public static void main(String[] args) {
    try(/*TODO1*/) {
    /*TODO2*/
  } catch (IOException e) {
    e.printStackTrace();
  }
   }
}
```

Table 2

student.dat

Table 3

Complete the implementation of LoadStudentFile.java (Table 2) by replacing /*TODO1*/ and /*TODO2*/ with proper input streams to read and print to console all the contents of student.dat (Table 3) after SaveStudentFile.java (Table 1) was executed once. (Note: you are not required to rewrite the whole class, just write the codes needed in /*TODO1*/ and /*TODO2*/).

[4 marks]

[Turn over

c) Your colleague has just started to learn Java Binary Input / Output and tried to implement a console application that read/write object data types from/to a file in a Windows operating system. Currently, your colleague needs your help to complete the implementation where it is commented with TODO.

The following Table 4 and Table 5 are two (2) Java classes called ChildrenDetails.java and Children.java, respectively, which your colleague is currently implementing.

ChildrenDetails.java

```java
public class ChildrenDetails {
  public static void main(String[] args) throws
      FileNotFoundException, IOException, ClassNotFoundException {
    Scanner scanner = new Scanner(System.in);
    // The directory is valid and JaiChildren.dat already exists
    File file = new File("JaiChildren.dat");
    try(
      // TODO 1
    ) {
      System.out.println("Children Details in the file.");
      while(readData.available() != 0) {
        System.out.println((Children) readData.readObject());
      }
      System.out.println("######################");
    }
    try(
      // TODO 2
    ) {
      System.out.println("Filling in more children details:");
      System.out.println("Name:");
      String name = scanner.nextLine();
      System.out.println("Date of Birth (dd/mm/yyyy):");
      String[] dob = scanner.nextLine().split("/");
      int day = Integer.parseInt(dob[0]);
      int month = Integer.parseInt(dob[1]);
      int year = Integer.parseInt(dob[2]);
      // TODO 3
    }
    scanner.close();
  }
}
```

Table 4

Children.java

```java
public class Children implements Serializable {
  String name;
  int yearOfBirth;
  int monthOfBirth;
  int dayOfBirth;

  public Children(String name, int yearOfBirth,
      int monthOfBirth, int dayOfBirth) {
    this.name = name;
    this.yearOfBirth = yearOfBirth;
    this.monthOfBirth = monthOfBirth;
    this.dayOfBirth = dayOfBirth;
  }

  public String toString() {
    return name + " -> " + dayOfBirth + "/" +
        monthOfBirth + "/" + yearOfBirth;
  }
}
```

Table 5

i. Based on Table 4, implement the appropriate InputStream object in Line 8, commented with TODO 1, that will read the File object, file, in Line 6.

[3 marks]

ii. Based on Table 4, implement the appropriate OutputStream object in Line 17, commented with TODO 2, that will append the File object, file, in Line 6.

[3 marks]

iii. Based on Table 4, implement the appropriate statements in Line 27 commented with TODO 3 that will write Children.java, Table 5, object once with the data in Line 21, Line 24, Line 25, and Line 26, using the OutputStream object created in **Question 2.c)ii.** to the File object, file, in Line 7 accordingly. The data written into the file must follow the order of it being read in Line 13 **with** its integrity still maintained.

[3 marks]

[Turn over

QUESTION 3 [Total marks: 14]

a) Explain the usual process of client / server computing in the correct order. **[4 marks]**

b) The following Table 6, Table 7 and Table 8 are three (3) Java classes called ChatServer.java, ChatHandler.java and ChatClient.java, respectively.

ChatServer.java

```java
public class ChatServer {
  public static void main(String[] args) throws Exception {
    ServerSocket serverSocket = new ServerSocket(9103);
    System.out.println("Server started.");

    while(true) {
      Socket chatClient1 = serverSocket.accept();
      Socket chatClient2 = serverSocket.accept();

      new Thread(new ChatHandler(chatClient1, chatClient2)).start();
    }
  }
}
```

Table 6

ChatHandler.java

```java
public class ChatHandler implements Runnable {
  Socket client1, client2, client3;
  DataInputStream fromClient1, fromClient2;
  DataOutputStream toClient1, toClient2;

  public ChatHandler(Socket client1, Socket client2) {
    this.client1 = client1;
    this.client2 = client2;
  }

  public void run() {
    try {
      fromClient1 = new DataInputStream(client1.getInputStream());
      fromClient2 = new DataInputStream(client2.getInputStream());
      toClient1 = new DataOutputStream(client1.getOutputStream());
      toClient2 = new DataOutputStream(client2.getOutputStream());

      while(true) {
        if(fromClient1.available() > 0) {
          String client1Msg = fromClient1.readUTF();
          toClient2.writeUTF(client1Msg);
        }
        if(fromClient2.available() > 0) {
          String client2Msg = fromClient2.readUTF();
          toClient1.writeUTF(client2Msg);
        }
      }
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

Table 7

[Turn over

ChatClient.java

```
1.   public class ChatClient {
2.     public static void main(String[] args) throws Exception {
3.       Socket socket = new Socket("localhost", 9103);
4.       System.out.println("Connected to the server.");
5.
6.       DataInputStream fromServer =
7.         new DataInputStream(socket.getInputStream());
8.       DataOutputStream toServer =
9.         new DataOutputStream(socket.getOutputStream());
10.
11.      Scanner scanner = new Scanner(System.in);
12.
13.      while(true) {
14.        System.out.println("Please type message:");
15.        String msgToSend = scanner.nextLine();
16.        toServer.writeUTF(msgToSend);
17.
18.        if(fromServer.available() > 0) {
19.          String receivedMsg = fromServer.readUTF();
20.          System.out.println(">> " + receivedMsg);
21.        }
22.      }
23.    }
24.  }
```

Table 8

When testing the client server application in the developer's own computer, the developer encounters a problem. Both clients have been connected to the server, but when one client types a String into the console and presses Enter, the other client does not receive the String immediately until it presses Enter.

i. Explain the reason why it happens. (Note: mention the lines of codes and the class name in your explanation.) **[4 marks]**

ii. Suggest a solution to allow the client to concurrently send String and receive String. (Note: it is not required to rewrite the whole class. Just rewrite the relevant code and mention the location using line number and class name which this implementation should be rewritten.) **[6 marks]**

QUESTION 4 [Total marks: 14]

a) Define Thymeleaf. **[1 mark]**

b) The following Table 9 is a HyperText Markup Language (HTML) file called index.html and Table 10 is a Java class called ControllerClass. Both are used for Spring Application (with Thymeleaf).

index.html

```html
<!DOCTYPE html>
<html lang="en" xmlns:th="http://thymeleaf.org">
<head></head>
<body>
<h1 th:text="${u.id} + ' ' + ${u.name}"></h1>
<p th:text="${u.percentage}"></p>
<p th:text="${u.active}"></p>
</body>
</html>
```

Table 9

ControllerClass.java

```java
@Controller
public class ControllerClass {
  @RequestMapping("/")
  public String home(ModelMap modelMap) {
    User user = new User(551, "Abu", 77.5, true);
    modelMap.put("u", user);
    return "index";
  }
}
```

Table 10

By analysing the above codes, create a Plain Old Java Object (POJO) class with only instance variables that are stated, including constructor, getter and setter methods.

[13 marks]

[Turn over

QUESTION 5 [Total marks: 14]

a) The following Table 11 is a Java class called ControllerClass.java.

ControllerClass.java
```java
@Controller
public class ControllerClass {

}
```
Table 11

UserRepository.java
```java
public interface UserRepository
    extends CrudRepository<User, Integer> {

}
```
Table 12

User.java
```java
@Entity
@Table(name="users")
public class User {
  /*
  * Contain proper POJO implementation of
  * instance variables, constructors,
  * getters and setters as of
  * question 4.b) but with additional
  * implementation for Java Persistence API
  */
}
```
Table 13

registeruser.html
```html
<!DOCTYPE html>
<html lang="en" xmlns:th="http://thymeleaf.org">
<head></head>
<body>
<h1>Registration</h1>
<form action="/register" method="post">
 id<input type="text" name="id" /><br>
 name<input type="text" name="name" /><br>
 percentage<input type="text" name="id" /><br>
 active
 <select name="active">
      <option value="true">Yes</option>
      <option value="false">No</option>
 </select><br>
 <button type="submit">Submit</button>
</form>
</body>
</html>
```
Table 14

i. Implement a method in Table 11 to handle Uniform Resource Identifier (URI) "/registeruser" where it will respond with registeruser.html, Table 14. (Note: You are not required to rewrite the whole class.) **[3 marks]**

ii. Implement a method in Table 11 to handle HTTP Post Request when the button (Line 15) in Table 14 is pressed where it will retrieve the data inputted in the form elements in Table 14, Line 7 – 14, and add new user into the database. Finally, it will redirect back to URI "/registeruser" (Note: You are not required to rewrite the whole class.) **[11 marks]**

[END OF PAPER]