ڤوليتيكنيك بروني
POLITEKNIK BRUNEI

END-OF-SEMESTER EXAMINATION (FORMAL)

SEMESTER 2, 2022-23

SCHOOL OF INFORMATION & COMMUNICATION TECHNOLOGY

NS4307

NETWORK PROGRAMMING

TIME ALLOWED: 2 HOURS

(MARKING SCHEME)

PREPARED BY MOHAMMAD JAILANI BIN HAJI ABDUL RAHMAN

INSTRUCTIONS TO MARKERS:

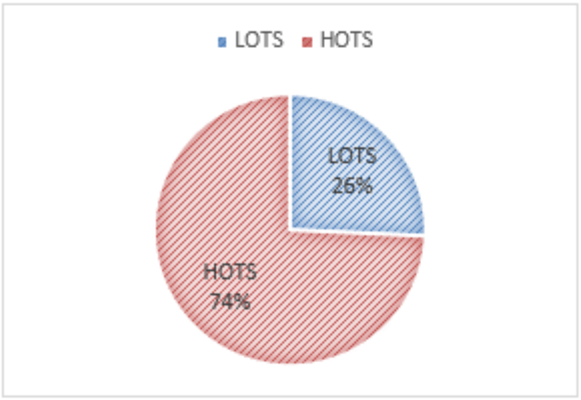
- Students must answer ALL questions.
- Students have 2 Hours to attempt this paper unless otherwise specified.
- The total mark for this paper is **70**. The number of marks for each question or part question is shown in brackets.
- The answers to the paper are shown in blue colour fonts.
- Keywords/ key points are shown in bold/underlined.
- Answer in italic represents an alternate answer.
- Only answers written in blue or black ink will be assessed except for diagram, where students are allowed to use pencil.
- The Blooms indicators after each question represents the Bloom's taxonomy classifying the cognitive level required in answering the questions.
- The assessment specification table provides the summary of the paper.
- Only use **RED** colour ink to mark.

No	Question	Marks	Bloom's Taxanomy
1	1.a.	2	Remember
2	1.b.	1	Remember
3	1.c.	3	Apply
4	1.d.	3	Remember
5	1.e.	3	Remember
6	2.a.	1	Remember
7	2.b.i.	4	Evaluate
8	2.b.ii.	2	Evaluate
9	2.c.i.	3	Evaluate
10	2.c.ii.	3	Evaluate
11	2.c.iii.	3	Evaluate
12	3.a.	4	Remember
13	3.b.i.	4	Analyze
14	3.b.ii.	6	Evaluate
15	4.a.	1	Remember
16	4.b.	13	Analyze
17	5.a.i.	3	Evaluate
18	5.a.ii.	11	Evaluate
19			
20			
21			
22			
23			
24			
25			
26			
27			
28			
29			
30			
	Total Marks	70	

Template by Tan Szu Tak

Bloom	No. Q	Marks
Remember	7	15
Understand	0	0
Apply	1	3
Analyze	2	17
Evaluate	8	35
Create	0	0

	Marks	%
LOTS	18	25.7%
HOTS	52	74.3%
Total	70	100.0%



STRUCTURED QUESTIONS [TOTAL MARKS: 70]

Answer **ALL** questions.

QUESTION 1 [Total marks: 14]

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

Note: Students answers may varies in terms but only the right content in the right context will be given marks.	
Version control is a system that records changes to a file or set of files	[1 mark]
over time so that you can recall specific versions later.	[1 mark]
Blooms: Remember (LOT)	

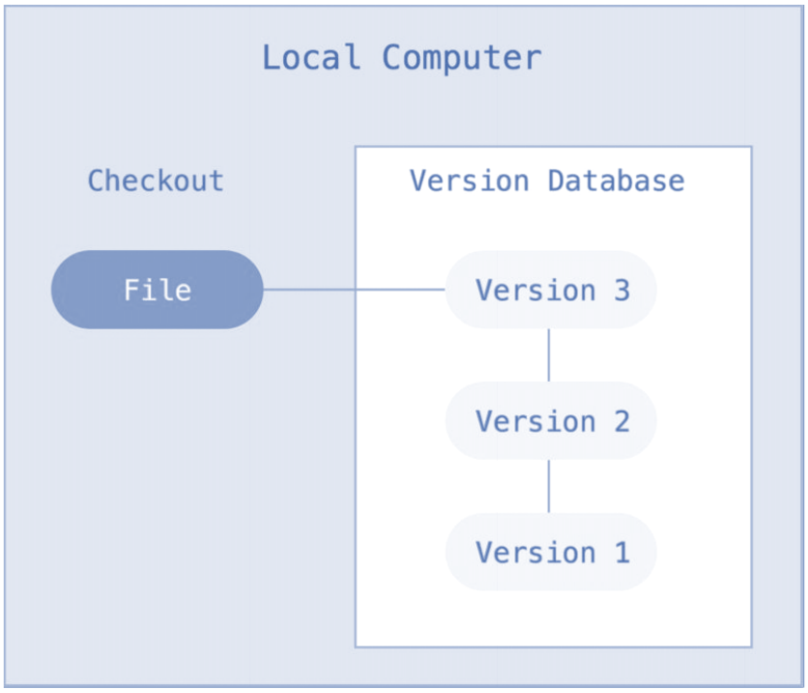
b) There are **three (3)** types of Version Control Systems (VCS) and one of them is Local Version Control Systems. List the other **two (2)** types of VCS. **[1 mark]**

Note: Students answers may varies in terms but only the right content in the right context will be given marks.	
- Centralised version control systems	[0.5 marks]
- Distributed version control systems	[0.5 marks]
Blooms: Remember (LOT)	

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

Student's solution has to satisfy the following criteria:	
- Illustrated local version control systems (LVCS) computer	[0.5 marks]
- Inside the LVCS computer contain version database	[0.5 marks]
- Inside version database contain at least one version	[0.5 marks]
- Inside the LVCS computer contain file(s)	[0.5 marks]

[Turn over

- Draw line from version database to each file(s) in the computers	[0.5 marks]
- No additional computer(s) or server(s) in the illustration.	[0.5 marks]
Sample solution:  Local Computer Checkout File Version Database Version 3 Version 2 Version 1	
Blooms: Apply (LOT)	

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

Note: Students answers may varies in terms but only the right content in the right context will be given marks.	
Modified You have <u>changed the file but have not committed it</u> to your database yet	[1 mark]
Staged You have <u>marked a modified file in its current version to go into your next commit</u> snapshot.	[1 mark]

Committed The data is safely stored in your local database. **Blooms:** Remember (LOT)	 [1 mark]

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

Note: Students answers may varies in terms but only the right content in the right context will be given marks.	
Tracked	[0.5 marks]
- Files that Git knows about	[1 mark]
Untracked	[0.5 marks]
- Files that have still not been added, so Git doesn't know about	[1 mark]
Blooms: Remember (LOT)	

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

Note: Students answers may varies in terms but only the right content in the right context will be given marks.	
git commit	
- Taking the snapshot of all changes in the working directory.	[1 mark]
git push	
- Upload local repository content to a remote repository.	[1 mark]
Blooms: Remember (LOT)	

[Turn over

QUESTION 2 [Total marks: 14]

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

[1 mark]

Note: Students answers may varies in terms but only the right content in the right context will be given marks. Because Binary Input/Output does not involve encoding or decoding the file. **Blooms:** Remember (LOT)	[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

```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

```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]**

Student's code has to satisfy the following criteria: /*TODO1*/ • Create appropriate InputStream object to read a file student.dat • The InputStream object created must be appropriate to read the data in the file /*TODO2*/ • Read the content of the file in the correct order. (Deduct 0.5 marks for each incorrect order. Maximum deduction 1 mark.) • Output all the content of the file to the console. (Deduct 0.5 marks for each content not outputted. Maximum deduction 1 mark.) Sample solution: /*TODO1*/ DataInputStream input = new DataInputStream(new FileInputStream("student.dat")) /*TODO2*/ System.out.println(input.readInt()); System.out.println(input.readUTF()); System.out.println(input.readDouble()); System.out.println(input.readUTF()); **Blooms:** Evaluate (HOT)	 [1 mark] [1 mark] [1 mark] [1 mark]

[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. In Table 4, implement the appropriate InputStream object at Line 8 commented with TODO 1 that will read the File object, file, in Line 6. **[3 marks]**

The student's solution should satisfy the following criteria:	
• Declare a variable with identifier, readData, with data type ObjectInputStream.	[0.5 marks]
• Create a ObjectInputStream object.	[0.5 marks]
• Create a FileInputStream object.	[0.5 marks]
• Pass the FileInputStream object as the argument when creating a ObjectInputStream object.	[0.5 marks]
• Pass the File object, file, as the argument when creating the FileInputStream object.	[0.5 marks]
• The ObjectInputStream object must be assigned to the variable.	[0.5 marks]
Example Solution: `ObjectInputStream readData = new ObjectInputStream(new FileInputStream(file));`	
Blooms: Evaluate (HOT)	

[Turn over

ii. In Table 4, implement the appropriate OutputStream object at Line 17 commented with TODO 2 that will append the File object, file, in Line 6. **[3 marks]**

The student's solution should satisfy the following criteria:	
• Declare a variable with identifier with data type ObjectOutputStream.	[0.5 marks]
• Create a ObjectOutputStream object.	[0.5 marks]
• Create a FileOutputStream object.	[0.5 marks]
• Pass the FileOutputStream object as the argument when creating a ObjectOutputStream object.	[0.5 marks]
• Pass the File object, file, as the first argument and Boolean value true as second argument when creating the FileOutputStream object.	[0.5 marks]
• The ObjectOutputStream object must be assigned to the variable.	[0.5 marks]
Example Solution: `ObjectOutputStream writeData = new ObjectOutputStream(new FileOutputStream(file));`	
Blooms: Evaluate (HOT)	

iii. In Table 4, implement the appropriate statements at Line 27 commented with TODO 3 that will write a Children, Table 2, 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 **and** its integrity still maintained. **[3 marks]**

The student's solution should satisfy the following criteria:	
• Create a Children object.	[0.5 marks]
• Pass the arguments following the order of data when creating the Children object:	
i. Data in variable, name.	[0.5 marks]
ii. Data in variable, year.	[0.5 marks]
iii. Data in variable, month.	[0.5 marks]
iv. Data in variable, day.	[0.5 marks]
• Use the variable in Question 2.c)ii. to call the method writeObject and pass the created Children object as its argument.	[0.5 marks]

Example Solution:
`Children children = new Children(name, year, month, day);`
`writeData.writeObject(children);`
 	
Blooms: Evaluate (HOT)	

QUESTION 3 [Total marks: 14]

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

Note: Students answers may varies in terms but only the right content in the right context will be given marks.	
• The client begins by attempting to establish a connection to the server.	[1 mark]
• The server can accept or deny the connection.	[1 mark]
• Once a connection is established, the client and the server communicate through sockets.	[1 mark]
The order of the process is as above.	[1 mark]
Blooms: Remember (LOT)	

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

[Turn over

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

```
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. When both clients have been connected to the server, when one client type a String into the console and press 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]**

Note: Students answers may varies in terms but only the right content in the right context will be given marks.	
It is due to the implementation of ChatClient class	[1 mark]
where in Line 15,	[0.5 marks]
the application will stop and wait for user input first	[1 mark]
then once the user presses Enter, then it proceeds to receive the String	[1 mark]
in Line 18 to 21.	[0.5 marks]
Blooms: Analyse (HOT)	

[Turn over

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]**

The student's solution should satisfy the following criteria:	
• ChatClient class	[0.5 marks]
• Rewrite Line 13 to Line 22.	[0.5 marks]
• Create a new Thread object.	[0.5 marks]
• Start the Thread object.	[0.5 marks]
• Create a new Runnable object (Using anonymous class or separate class)	[0.5 marks]
• Define the run method.	[0.5 marks]
• Implement a proper loop in the run method.	[0.5 marks]
• Move either codes in Line 14 – 16 or Line 18 – 21 into the loop.	[0.5 marks]
• The moved codes should remain the same structure.	[0.5 marks]
• The other codes that are not moved.	
i. It should have the same code structure as Line 14 – 16 or Line 18 – 21.	[0.5 marks]
ii. It should be inside the original loop Line 13.	[0.5 marks]
• There is no syntax error (No deduction of marks if the student does not implement try-catch statement in the run method.	[0.5 marks]

Example Solution:

```
Rewrite ChatClient class in Line 13 to Line 22.

new Thread(new Runnable() {
  public void run() {
    try {
      while(true) {
        if(fromServer.available() > 0) {
          String receivedMsg = fromServer.readUTF();
          System.out.println(">> " + receivedMsg);
        }
      }
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}).start();

while(true) {
      System.out.println("Please type message:");
      String msgToSend = scanner.nextLine();
      toServer.writeUTF(msgToSend);
}
```

Blooms: Evaluate (HOT)

QUESTION 4 [Total marks: 14]

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

Note: Students answers may varies in terms but only the right content in the right context will be given marks. Thymeleaf is a modern server-side Java template engine for both web and standalone environments. **Blooms:** Remember (LOT)	[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 that are used for Spring Application (with Thymeleaf).

```
index.html
1.   <!DOCTYPE html>
2.   <html lang="en" xmlns:th="http://thymeleaf.org">
3.   <head></head>
4.   <body>
5.   <h1 th:text="${u.id} + ' ' + ${u.name}"></h1>
6.   <p th:text="${u.percentage}"></p>
7.   <p th:text="${u.active}"></p>
8.   </body>
9.   </html>
```

Table 9

[Turn over

```
ControllerClass.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]

Student's Plain Old Java Object Class has to satisfy the following criteria:	
• Define the POJO class.	[0.5 marks]
• Declare four instance variables using the valid data types. [0.5 marks] each	[2 marks]
• Define constructor for POJO class with parameter variable.	[0.5 marks]
• Initialise the appropriate instance variables with the appropriate parameter variables. [0.5 marks] each	[2 marks]
• Implement the getter methods [0.5 marks] each and return data [0.5 marks] each.	[4 marks]
• Implement the setter methods [0.5 marks] each and setting its appropriate variable [0.5 marks]	[4 marks]

Sample solution:

```
public class User {
  int id; String name; double percentage; boolean active;

  public User(int id, String name, double percentage, boolean active) {
    this.id = id;
    this.name = name;
    this.percentage = percentage;
    this.active = active;
  }
  public int getId() {return id;}
  public void setId(int id) {this.id = id;}
  public String getName() {return name;}
  public void setName(String name) {this.name = name;}
  public double getPercentage() {return percentage;}
  public void setPercentage(double percentage) {
    this.percentage = percentage;}
  public boolean isActive() {return active;}
  public void setActive(boolean active) {this.active = active;}
}
```

Blooms: Analyse (HOT)

QUESTION 5 [Total marks: 14]

a) The following Table 5 is a Java class called MainController.java.

ControllerClass.java

```
@Controller
public class ControllerClass {

}
```

Table 11

UserRepository.java

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

}
```

Table 12

User.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

[Turn over

registeruser.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]**

The student's solution should satisfy the following criteria:

- Implements a method with correct syntax,
  - With any identifier [0.5 marks]
  - No parameter [0.5 marks]
  - String return data type [0.5 marks]
- Implements return "registeruser" inside the method block. [0.5 marks]
- Implements @RequestMapping annotation with value "/registeruser" [0.5 marks]
- The @RequestMapping annotation is on top of the method definition. [0.5 marks]

Sample solution:

```
@RequestMapping("/registeruser")
public String registeruser() {
  return "registeruser";
}
```

Blooms: Evaluate (HOT)

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]**

The student's solution should satisfy the following criteria:	
• Implements a method with correct syntax,	
○ With any identifier	[0.5 marks]
○ Four parameter variables declared with appropriate data type using the names of each form elements as its identifiers respectively. [0.5 marks] each.	[2 marks]
○ All four parameter variables have @RequestParam annotation declared before the data type. [0.5 marks] each.	[2 marks]
○ String return data type	[0.5 marks]
• Implements @RequestMapping annotation with value "/register"	[0.5 marks]
• The @RequestMapping annotation is on top of the method definition.	[0.5 marks]
• Declare an instance variable with any identifier of UserRepository data type.	[0.5 marks]
• Declare @Autowired annotation for the UserRepository instance variable declaration.	[0.5 marks]
• Create User object in the method.	[0.5 marks]
• Correctly pass appropriate data from the form to the User object constructor. [0.5 marks] each.	[2 marks]
• The order of the data being passed follows User class construction answered in question 4.b).	[0.5 marks]
• Use the save method from UserRepository variable to save the User object.	[0.5 marks]
• Implements return "redirect: /registeruser" inside the method block.	[0.5 marks]

[Turn over

Sample solution:
`@Autowired`
`UserRepository userRepo;`
 	
`@RequestMapping("/register")`
`public String register(`
`    @RequestParam int id,`
`    @RequestParam String name,`
`    @RequestParam double percentage,`
`    @RequestParam boolean active) {`
`  User user = new User(id, name, percentage, active);`
`  userRepo.save(user);`
`  return "redirect:/registeruser";`
`}`
 	
Blooms: Evaluate (HOT)	

[END OF MARKING SCHEME]