NS4307 Network Programming Sample Examination Questions

2.25. Explain the large part of what network programs do.
2.26. Differentiate Uniform Resource Locator (URL) and Uniform Resource Identifier (URI) and provide an example for each URL and URI.
2.27. List and explain the two (2) methods to send data with HTTP Request.
2.28. List and explain the four (4) different ways to implement Thymeleaf URL Expression.
2.29. Show example of the four (4) different ways to implement Thymeleaf URL Expression; Absolute URLs, Context-relative URLs, Server-relative URLs and Protocol relative URLs; and show the result of the URL Expression.
2.30. Define Plain Old Java Object (POJO).
2.31. In Java Socket, server and client application. When you run the client application, it terminates and produced an error with ConnectException. Explain what causes this exception error.
2.32. Define Threads in Java.
2.33. Define Spring Framework for Java.
2.34. Before a Spring Web Application with Hibernate can connect to a database management system, the web application needs to be configured with some properties first. Explain the purpose of the following properties.
    2.34.1. spring.datasource.driver-class-name
    2.34.2. spring.datasource.url
    2.34.3. spring.datasource.username
    2.34.4. spring.datasource.password

3. Coding Questions
3.1. Your colleague just started to learn Java Binary Input / Output and tried to implement a console application that can copy and paste a file in a Windows operating system. Currently your colleague encountered a syntax error and unable to compile the application.
    3.1.1. Identify by stating the line number of the statement that causes the syntax error, in CopyPasteApp.java and explain why the syntax error happened.
    3.1.2. Suggest a solution to fix the syntax error to allow the application to copy Copy.docx file in Line 6 and paste it as Paste.docx in Line 8, for the application CopyPasteApp.java. (Note: You are not required to rewrite the whole class. Just state the line number(s) that needs to be replaced with your solution.)
3.2. The following are two (2) Java classes called LearningDOS.java and LearningDIS.java with an empty binary file called data.dat.
3.3. The following are two (2) Java classes called Server.java and Client.java.
    3.3.1. Server.java is a server application. Complete the implementation of Server.java based on the following.
    3.3.2. Client.java is a client server that connects to Server.java, a server application. Complete the implementation of Client.java based on the following.
3.4. The following is a Java class called MainController.java.
3.5. The following are Java classes called MainController.java and Module.java and is a Hypertext Markup Language (HTML) called modules.html, that are used for Spring Web Application (with Thymeleaf).
    3.5.1. Complete the Plain Old Java Object class, Module.java by analysing MainController.java and modules.html implementations. (Note: You are not required to rewrite the whole class.)
    3.5.2. Implement a method in MainController.java to handle Uniform Resource Identifier (URI) "/modules" where it will response with the HTML file, modules.html with the data Line 4 to Line 8 in MainController passed to the HTML file. (Note: You are not required to rewrite the whole class.)
    3.5.3. In line 2 of modules.html, a thymeleaf namespace, th, is being declared into the XML name space. Explain the purpose of this declaration.
3.6. The following is a Java class called Student.java and is a command line (terminal) screenshot of a table details queried in MySQL Database Management System server, that are used for Spring Web Application with Hibernate.

3.7. The following are three (3) Java classes called LearningOOS.java, LearningOIS.java and Student.java with an empty binary file called object.dat.

    3.7.1. When executing LearningOOS.java, it threw an exception, java.io.NotSerializableException. Explain why that is the case.

    3.7.2. Suggest a solution to prevent LearningOOS.java from throwing an exception, java.io.NotSerializableException. (Note: You are not required to rewrite the whole class. Just state the Java class and line number(s) that needs to be replaced with your solution.)

    3.7.3. Produce the output generated by the code in LearningOIS.java, assuming that LearningOOS.java has been executed once without any issue before executing LearningOIS.java.

3.8. The following are two (2) Java classes called LearningDOS.java and LearningDIS.java respectively with an empty binary file called data.dat.

3.9. The following are two (2) Java classes called Server.java and Client.java.

    3.9.1. Server.java is a server application. Complete the implementation of Server.java based on the following.

    3.9.2. Client.java is a client server that connects to Server.java, a server application. Complete the implementation of Client.java based on the following.

3.10. The following is a Java class called MainController.java

3.11. The following are Java classes called MainController.java and Stock.java and is a Hypertext Markup Language (HTML) called stocks.html, that are used for Spring Web Application (with Thymeleaf).

    3.11.1. Complete the implementation of Line 14 to Line 18 of the HTML file, stocks.html which it should display all data passed to stocks.html through MainController.java where each Stock.java object is displayed as table row and each field of Stock.java object is displayed in its respective table column. (Note: You are not required to rewrite the whole html file.)

    3.11.2. The Integrated Development Environment (IDE) that you are using to develop this Spring application is showing an error after you completed Question 3.11.1. This is because you haven't declared Thymeleaf namespace into your stocks.html. State where you need to declare Thymeleaf namespace in stocks.html and implement the Thymeleaf namespace in stocks.html. (Note: You are not required to rewrite the whole html file.)

3.12. The following is a Java class called Student.java and is a command line (terminal) screenshot of a table details queried in MySQL Database Management System server, that are used for Spring Web Application with Hibernate.

3.13. The following is a Spring Web Application with Hibernate class named Employee.java

3.14. Before a Spring Web Application with Hibernate can connect to a Database Management System, the web application need to be configured first. The configurations are implemented in application.properties file as follows.

3.15. The following is a Java class named MyController.java and is a Hypertext Markup Language (HTML) named index.html. Both files are used for Spring Web Application (with Thymeleaf).

    3.15.1. Implement a method in MyController.java to handle Uniform Resource Identifier (URI) "/home" where it will response with index.html (Note: you are not required to rewrite the whole class)

    3.15.2. Implement a method in MyController.java (Note: you are not required to rewrite the whole class):

3.16. The following are two Java classes named Server.java and Client.java.

3.17. The following are three Java classes named Student.java SaveStudentFile.java and LoadStudentFile.java with an empty binary file named student.dat.

3.18. Your colleague is implementing a Java server application. He implemented it in a Java class called Server.java by creating a ServerSocket object with port 8080.

    3.18.1. When he tries to execute the class above, it produced a Runtime error called BindException. Explain to him what causes this error.

    3.18.2. Suggest a solution that will fix the error in Question 3.18.1.

3.18.3. Complete the implementation of Server.java where the server application should accept connection from two client applications and the server application should be able to communicate with each client application separately. (Note: you are not required to rewrite the whole class, just write the codes needed in `/*TODO*/`).

3.18.4. After successfully executing Server.java, he needs your help to implement a Java client application to connect and communicate to the server in the same network. All you need to do is to write the code (statement) needed to connect to the server application. The server application Internet Protocol (IP) address is 192.168.1.1. (Note: you are not required to write the whole client class)

3.18.5. When you run the client application, it terminates and produced an error with ConnectException. Explain what causes this exception error.

3.19. The following are three Java classes named Product.java, Employee.java and MyController.java and are two Hypertext Markup Language (HTML) files named abc.html and def.html, that are used for Spring Web Application (with Thymeleaf).

3.19.1 Analyse the code above and complete the implementation of abc.html and def.html (using paragraph tag `<p></p>` and Thymeleaf syntax) where it should display into the page all the data inside each Model that passed through the ModelMap respectively. (Note: you do not need to rewrite the whole HTML file. Specify the file and parent tag which the paragraph should be implemented. Arrangement of data does not matter)

3.19.2. Explain why in both html files implements xmlns:th="http://thymeleaf.org" in the html tag and what happens if it is omitted.

3.19.3. Explain the use of @Controller and @RequestMapping implemented in MyController.java.

3.20. The following are two Java classes named SaveStudentFile.java and LoadStudentFile.java with an empty binary file named student.dat.

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

3.20.2. Explain the meaning of the second argument (second parameter), in SaveStudentFile.java when creating FileOutputStream object and what the boolean value false means.

3.21. The following is a HTML file called index.html that is used for Spring Web Application (with Thymeleaf).

3.22. The following is a Spring Web Application (with Thymeleaf) Controller file called MyController.java.

3.22.1. Explain what does @Controller (Line 1), @RequestMapping (Line 3 and Line 8), @ResponseBody (Line 9) and @RequestParam(required=false) (Line 10) means in MyController.java.

3.22.2. Explain what Line 3 to Line 6 in MyController.java does.

3.22.3. Explain what Line 8 to Line 16 in MyController.java does.

3.23. Your colleague is implementing a Java server application. He implemented it in a Java class called Server.java by creating a ServerSocket object with port 9921.

3.23.1. Complete the implementation of Server.java where the server application should accept connection from three (3) client applications and the server application should be able to communicate with each client application separately. (Note: you are not required to rewrite the whole class, just write the codes needed in `/*TODO*/`).

3.23.2. After successfully executing Server.java, he needs your help to implement a Java client application to connect and communicate to the server in the same network. All you need to do is to write the code needed to connect to the server application. The server application Internet Protocol (IP) address is 192.168.100.10. (Note: you are not required to write the whole client class).

3.23.3. When you run the client application, it terminates and produced an error with ConnectException. Explain what causes this exception error.

3.24. The following is a Spring Web Application with Hibernate class called Student.java.

3.25. The following are two (2) Java classes called Server.java and Client.java.

1. Examination Details

- There are 5 Questions where you need to answer all.
- The total mark is 70 marks.
- Each question is 14 marks.
- You must write in dark blue or black ink pen.
- Calculator is allowed.

2. Theory Questions

2.1. Describe Version Control

> Version control is a system that records changes to a file or set of files over time so that you can recall specific version later.

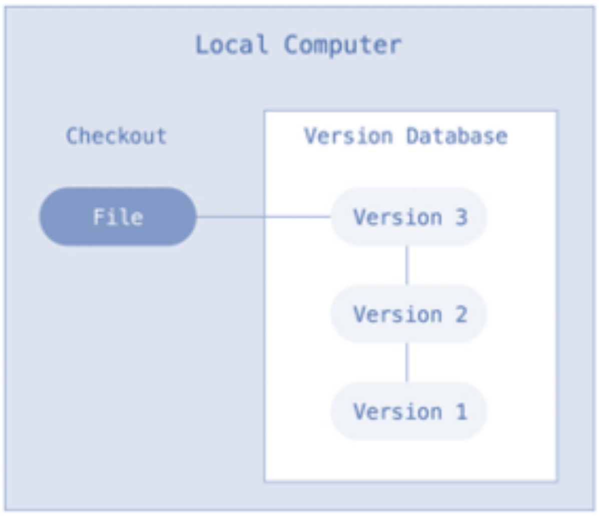
2.2. State any three (3) typical features in Version Control Systems.

> - It allows you to revert selected files back to a previous state
> - Revert the entire project back to a previous state
> - Compare changes over time
> - See who last modified something that might cause a problem
> - Who introduced an issue and when.

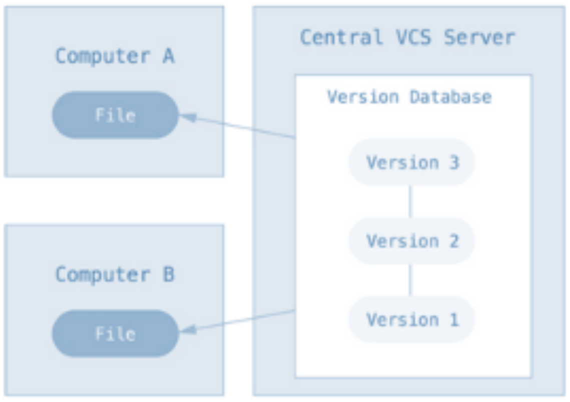
2.3. List the three (3) types of Version Control Systems (VCS).

> - Distributed Version Control Systems
> - Centralised Version Control Systems
> - Local Version Control Systems

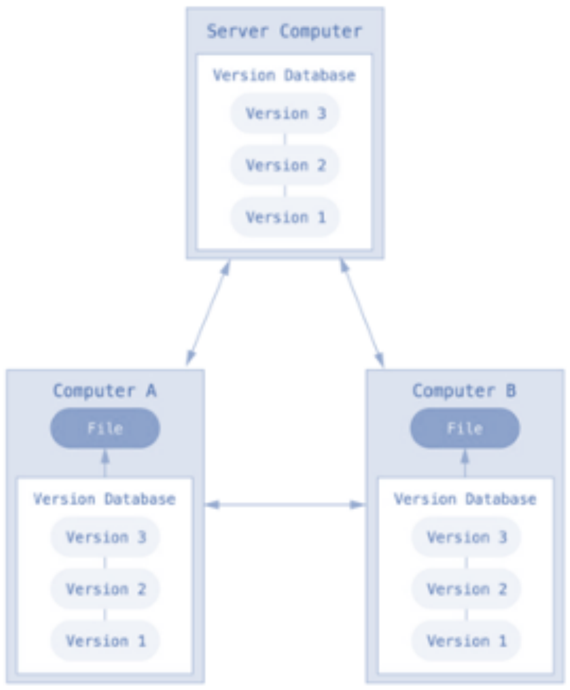
2.4. Illustrate the basic structure of Local Version Control Systems



2.5. Illustrate the basic structure of Centralised Version Control Systems.



2.6. Illustrate the basic structure of Distributed Version Control Systems.



2.7. List and explain the two (2) types of files in Git working directory.

> Tracked
>
> - Files that Git knows about
>
> Untracked
>
> - Files that have still not been added, so Git doesn't know about

2.8. Explain Git's three (3) main states; modified, staged and committed; that your files can reside in.

> Modified
> You have changed the file but have not committed it to your database yet
>
> Staged
> You have marked a modified file in its current version to go into your next commit snapshot.
>
> Committed
> The data is safely stored in your local database.

2.9. Explain in details how Java Text Input / Output and Java Binary Input / Output work in terms of writing and reading to/from a file. (Note: You may illustrate using diagram or use an example to further enhance your explanation.)

> Java Text Input / Output will consider the value that being store/read as Unicode of the characters and the binary representation of each Unicode of character is being stored in the file.
>
> Java Binary Input / Output will store/read the binary representation of Java data type to/from the file.

2.10. Explain the usual process of client / server computing in the correct order.

> - The client begins by attempting to establish a connection to the server.
> - The server can accept or deny the connection.
> - Once a connection is established, the client and the server communicate through sockets.

2.11. Explain the typical case where a Java server application in a client / server computing would throw a java.net.BindException.

> Attempting to create a server socket on a port already in use.

2.12. Explain the purpose of InetAddress class in java.net package.

> It is to know who is connecting to the server by finding the client's host name and IP address.

2.13. Define a web application.

> A web application is a client / server application where user communicates with a web server through a web browser which the web server send back a response.

2.14. Briefly explain the concept of Model View Controller.

> Model View Controller (MVC) is basically a design pattern for implementing user interfaces on computers.

2.15. State and explain the components in Spring Web Application (with Thymeleaf) that represents each component in Model View Controller (MVC) concept.

> The HTML file is the view that contains the description of the user interface
>
> The controller is a Java class, implementing components that handle the URI, POST and GET Request
>
> The model consists of domain objects, defined on the Java side, that you connect to the view through the controller.

2.16. Define the term Thymeleaf.

> Thymeleaf is a modern server-side Java template engine for both web and standalone environments.

2.17. Before a Spring Web Application with Hibernate can connect to a database management system, the web application need to be configured first. One of the properties that can be configured is spring.jpa.hibernate.ddl-auto. List and explain the four (4) values that can be assigned to the property.

> create
> creates the database tables, destroying previous data.
>
> create-drop
> Drop the database tables when the application is stopped.
>
> update
> update the database tables.
>
> validate
> validate the database tables, makes no changes to the database.

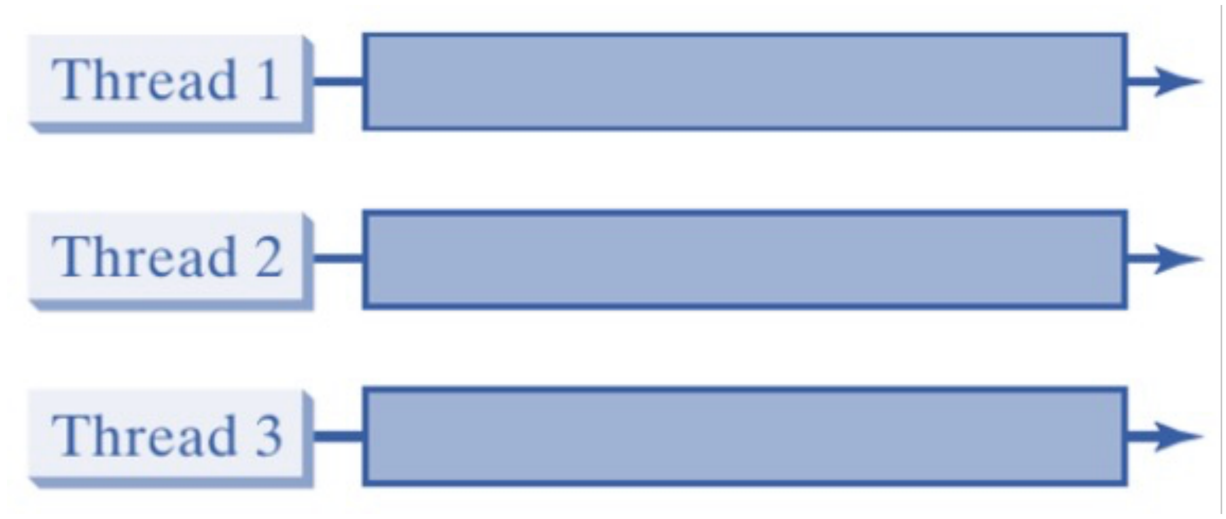
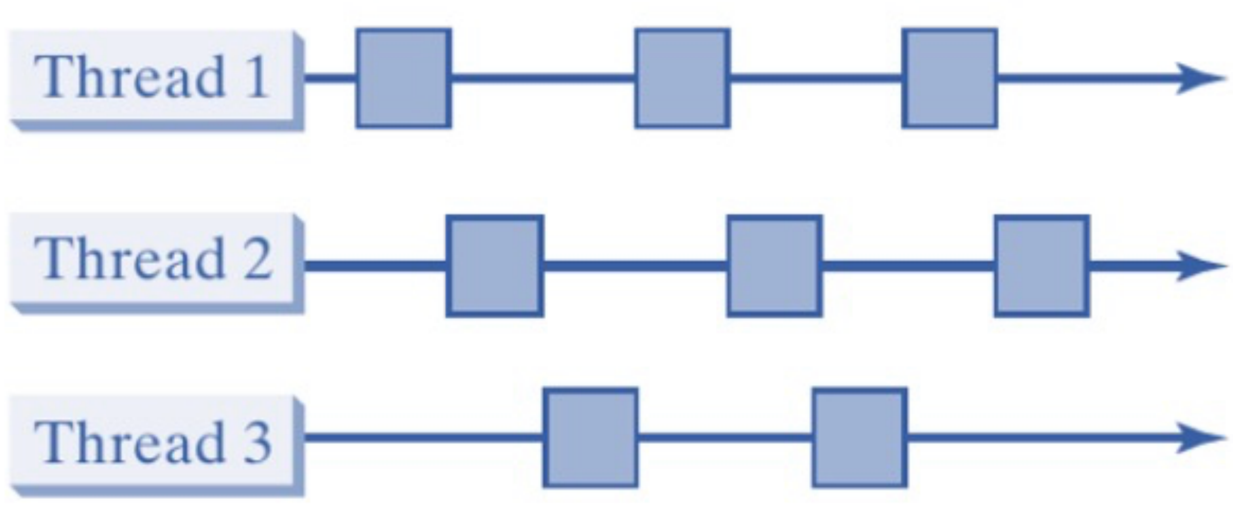
2.18. Networking is tightly integrated in Java where the Java Application Programming Interface (API) provides the classes for creating sockets to facilitate program communications over the Internet. Explain the term Sockets.

> Sockets are the endpoints of logical connections between two hosts can be used to send and receive data.

2.19. BufferedInputStream and BufferedOutputStream can be used to make input and output more efficient by reducing the number of disk reads and writes. Explain how that is the case.

> BufferedInputStream
> The whole block of data on the disk read into the buffer in the memory once.
> The individual data are then delivered to your program from the buffer.
>
> BufferedOutputStream
> The individual data first written to the buffer in the memory.
> When the buffer is full, all data in the buffer are written to the disk once.

2.20. For a server to handle multiple clients simultaneously, it is recommended to use Threads. Simulate using diagram three (3) threads running on multiple CPUs and single CPU.

> Three (3) threads running on multiple CPUs:
>
> 
>
>
> Three (3) threads running on single CPUs
>
> 
>

2.21. Explain why in a Spring Web Application with Thymeleaf, in the html files, it is recommended to implements xmlns:th="http://thymeleaf.org" in the html tag and what happens if it is omitted.

> To identify "th" into the XML name space
> If this is omitted, some IDE might consider attributes with "th:" as an error.

2.22. Explain the use of the following annotations that can be implemented in Spring Controller Class.

2.22.1 @Controller

> To indicate the class is Spring Controller.

2.22.2. @RequestMapping

> To handle the URI request

2.22.3. @ResponseBody

> To indicate the return value is to be response body.

2.22.4. @RequestParam(required=false)

> To handle the GET/POST HTTP request where the HTTP request is optional.

2.22.5. @PathVariable

> To notifying that the method will get the data stated in URI.

2.23. One of Maven feature is its dependency management. Explain what is dependency management and the benefit of using Maven.

> It allows you to download any JARs required for building your project from a central JAR repository.
>
> The benefit of using Maven is you don't need to manually download each third party libraries import it manually into your project.

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

> Because Binary Input/Output does not involve encoding or decoding the file.

2.25. Explain the large part of what network programs do.

> The large part of what network programs do are simple input and output which involves moving bytes from one system to another.

2.26. Differentiate Uniform Resource Locator (URL) and Uniform Resource Identifier (URI) and provide an example for each URL and URI.

> URL is an address that includes a method for locating a resource with a protocol like HTTP or FTP.
> Example URL: http://www.pb.edu.bn/app
>
> URI is the resource location where no host name and protocol are included.
> Example URI: /app

2.27. List and explain the two (2) methods to send data with HTTP Request.

> - GET Method
>   Retrieve whatever information is identified by the Request-URI.
> - POST Method
>   Used to request that the origin server accept the entity enclosed in the request as a new subordinate of the resource identified by the Request-URI in the Request-Line.

2.28. List and explain the four (4) different ways to implement Thymeleaf URL Expression.

> - Absolute URLs:
>
>   Thymeleaf will not modify if the content contain Absolute URL.
>
> - Context-relative URLs
>
>   These are URLs which are supposed to be relative to the web application root once it is installed on the server.
>
> - Server-relative URLs
>
>   These are URLs which are supposed to be relative to the server root.
>
> - Protocol-relative URLs
>
>   Protocol-relative URLs are in fact absolute URLs which will keep the protocol (HTTP, HTTPS) being used for displaying the current page. They
>   are typically used for including external resources like styles, scripts, etc.

2.29. Show example of the four (4) different ways to implement Thymeleaf URL Expression; Absolute URLs, Context-relative URLs, Server-relative URLs and Protocol relative URLs; and show the result of the URL Expression.

> - Absolute URLs
>
>   Example:
>
>   Result: No changes will be made to the content.
>
> - Context-relative URLs
>
>   Example:
>
>   Result: If our app is installed at http://localhost:8080/myapp, this URL will output:
>
> - Server-relative URLs
>
>   Example:
>   Result: The current application's context will be ignored, therefore although our application is deployed at http://localhost:8080/myapp, this URL will output.
>
> - Protocol-relative URLs
>
>   Example:
>
>   Result: No changes will be made to the content.

2.30. Define Plain Old Java Object (POJO).

> A Plain Old Java Object (POJO) is a Java object whose class is coded for it natural functionality and not for the framework which it will be used in.

2.31. In Java Socket, server and client application. When you run the client application, it terminates and produced an error with ConnectException. Explain what causes this exception error.

> The server application is not running or cannot be reached.

2.32. Define Threads in Java.

> A thread provides the mechanism for running a task. With Java, you can launch multiple threads from a program concurrently.

2.33. Define Spring Framework for Java.

> Spring Framework is a tool to help your write a Java Application that can run on a web server.

2.34. Before a Spring Web Application with Hibernate can connect to a database management system, the web application needs to be configured with some properties first. Explain the purpose of the following properties.

2.34.1. spring.datasource.driver-class-name

> The driver that will be used to connect to a database.

2.34.2. spring.datasource.url

> URL location the database is hosted and connects to that database.

2.34.3. spring.datasource.username

> Username used to connect to the database server.

2.34.4. spring.datasource.password

> Password used to connect to the database server.

3. Coding Questions

3.1. Your colleague just started to learn Java Binary Input / Output and tried to implement a console application that can copy and paste a file in a Windows operating system. Currently your colleague encountered a syntax error and unable to compile the application.

The following is a Java class called CopyPasteApp.java which your colleague is currently implementing.

```java
import java.io.*;

public class CopyPasteApp {
  public static void main(String[] args) throws Exception {
    // The directory is valid and Copy.docx exists
    File sourceFile = new File("C:\\Users\\Ali\\Documents\\Copy.docx");
    // The directory is valid and Paste.docx does not exists
    File targetFile = new File("C:\\Users\\Ali\\Documents\\Paste.docx");
    try(
      InputStream input = new InputStream();
      OutputStream output = new OutputStream();
    ) {
      int fileContent = 0;
      int bytesCopied = 0;
      while((fileContent = input.read()) != -1) {
        output.write((byte) fileContent);
17          bytesCopied++;
18        }
19        System.out.println("Copy paste complete.");
20        System.out.println("No of bytes copied: " + bytesCopied);
21      }
22    }
23  }
```

3.1.1. Identify by stating the line number of the statement that causes the syntax error, in CopyPasteApp.java and explain why the syntax error happened.

> Line 10
>
> Line 11
>
> Because InputStream and OutputStream is an abstract class where it cannot be instantiated.

3.1.2. Suggest a solution to fix the syntax error to allow the application to copy Copy.docx file in Line 6 and paste it as Paste.docx in Line 8, for the application CopyPasteApp.java. (Note: You are not required to rewrite the whole class. Just state the line number(s) that needs to be replaced with your solution.)

> Line 10 and 11:
>
> InputStream input = new FileInputStream(sourceFile);
>
> OutputStream output = new FileOutputStream(targetFile);

3.2. The following are two (2) Java classes called LearningDOS.java and LearningDIS.java with an empty binary file called data.dat.

LearningDOS.java

```
import java.io.*;

public class LearningDOS {
  public static void main(String[] args) {
    try(DataOutputStream output = new DataOutputStream(
        new FileOutputStream("data.dat", true))) {
      output.writeByte(1);
      output.writeUTF("Nike");
      output.writeUTF("Shirt");
      output.writeDouble(9.99);
      output.writeByte(2);
      output.writeUTF("Adidas");
      output.writeUTF("Pants");
      output.writeDouble(19.99);
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
}
```

LearningDIS.java

```java
import java.io.*;

public class LearningDIS {
  public static void main(String[] args) {
    try(DataInputStream output = new DataInputStream(
        new FileInputStream("data.dat"))) {
      while(output.available() > 0) {
        System.out.println(output.readByte());
        System.out.println(output.readUTF());
        System.out.println(output.readUTF());
        System.out.println(output.readDouble());
      }
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
}
```

data.dat

```

```

Produce the output generated by the code in LearningDIS.java, assuming that LearningDOS.java has been executed once before executing LearningDIS.java.

> 1
>
> Nike
>
> Shirt
>
> 9.99
>
> 2
>
> Adidas
>
> Pants
>
> 19.99

3.3. The following are two (2) Java classes called Server.java and Client.java.

Server.java

```java
public class Server {
  public static void main(String[] args) {

  }
}
```

Client.java

```java
public class Client {
  public static void main(String[] args) {

  }
}
```

3.3.1. Server.java is a server application. Complete the implementation of Server.java based on the following.

When the server application started, it will listen to port number 9921. The server application should accept connection from a client application and it should be able to send Java primitive and String data to and receive Java primitive and String data from the client application. (Note: You are not required to rewrite the whole class, just write the code statements in proper order. You are not required to handle any exceptions being thrown by any of the statements or import any of the classes.)

> ```java
> ServerSocket serverSocket = new ServerSocket(9921);
> Socket socket = serverSocket.accept();
> DataInputStream input =
>   new DataInputStream(socket.getInputStream());
> DataOutputStream output =
>   new DataOutputStream(socket.getOutputStream());
> ```

3.3.2. Client.java is a client server that connects to Server.java, a server application. Complete the implementation of Client.java based on the following.

It will request a connection to the server application hosted in IP address 10.106.132.123. The client application should be able to send Java primitive and String data to and receive Java primitive and String data from the server application. (Note: You are not required to rewrite the whole class, just write the code statements in proper order. You are not required to handle any exceptions being thrown by any of the statements or import any of the classes.)

```java
Socket socket = new Socket("10.106.132.123", 9921);
DataInputStream input =
  new DataInputStream(socket.getInputStream());
DataOutputStream output =
  new DataOutputStream(socket.getOutputStream());
```

3.4. The following is a Java class called MainController.java.

```java
@Controller
public class MainController {

}
```

Implement a method in MainController.java to handle Uniform Resource Identifier (URI) "/hello" where it will response with a literal String "Hello World". (Note: You are not required to rewrite the whole class.)

```java
@RequestMapping(value="/hello")
@ResponseBody
public String hello() {
  return "Hello World";
}
```

3.5. The following are Java classes called MainController.java and Module.java and is a Hypertext Markup Language (HTML) called modules.html, that are used for Spring Web Application (with Thymeleaf).

MainController.java

```java
@Controller
public class MainController {
  List<Module> modules = (List<Module>) Arrays.asList(
      new Module("NS4305", "DNM", 4),
      new Module("NS4306", "MWC", 4),
      new Module("NS4307", "NEP", 4),
      new Module("IS4211", "PMGT", 2)
      );
}
```

Module.java

```java
public class Module {

}
```

modules.html

```html
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<title>NS4307 NEP</title>
</head>
<body>
<h1>List of Modules</h1>
<table>
<tr>
<th>Code</th>
<th>Name</th>
<th>Credit Value</th>
</tr>
<tr th:each="module : ${ modules }">
<td th:text="${ module.code }"></td>
<td th:text="${ module.name }"></td>
<td th:text="${ module.creditValue }"></td>
</tr>
</table>
</body>
</html>
```

3.5.1. Complete the Plain Old Java Object class, Module.java by analysing MainController.java and modules.html implementations. (Note: You are not required to rewrite the whole class.)

```java
public class Module {
  private String code;
  private String name;
  private int creditValue;

  public Module(String code, String name, int creditValue) {
    this.code = code;
    this.name = name;
    this.creditValue = creditValue;
  }

  public String getCode() {
    return code;
  }

  public void setCode(String code) {
    this.code = code;
  }

  public String getName() {
    return name;
  }

  public void setName(String name) {
    this.name = name;
  }

  public int getCreditValue() {
    return creditValue;
  }

  public void setCreditValue(int creditValue) {
    this.creditValue = creditValue;
  }
}
```

3.5.2. Implement a method in MainController.java to handle Uniform Resource Identifier (URI) "/modules" where it will response with the HTML file, modules.html with the data Line 4 to Line 8 in MainController passed to the HTML file. (Note: You are not required to rewrite the whole class.)

```java
@RequestMapping(value="/modules")
public String modules(ModelMap modelMap) {
  modelMap.put("modules", modules);
  return "home";
}
```

3.5.3. In line 2 of modules.html, a thymeleaf namespace, th, is being declared into the XML name space. Explain the purpose of this declaration.

> This is to prevent some IDE to consider attributes with th: as error.

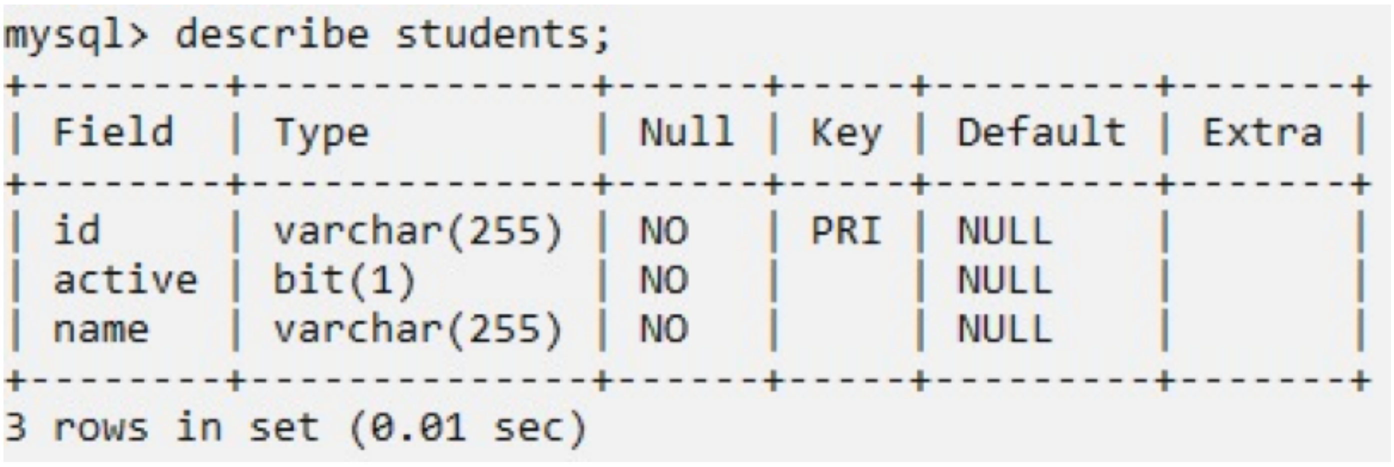
3.6. The following is a Java class called Student.java and is a command line (terminal) screenshot of a table details queried in MySQL Database Management System server, that are used for Spring Web Application with Hibernate.

Student.java

```java
public class Student {

}
```

students MySQL table



```
mysql> describe students;
+--------+--------------+------+-----+---------+-------+
| Field  | Type         | Null | Key | Default | Extra |
+--------+--------------+------+-----+---------+-------+
| id     | varchar(255) | NO   | PRI | NULL    |       |
| active | bit(1)       | NO   |     | NULL    |       |
| name   | varchar(255) | NO   |     | NULL    |       |
+--------+--------------+------+-----+---------+-------+
3 rows in set (0.01 sec)
```

Rewrite the Plain Old Java Object class, Student.java by making the class to be a Database entity that will be used by Hibernate to create the Database table with its field and constraints accordingly.

```java
@Entity
@Table(name = "students")
public class Student {
  @Id
  private String id;

  @NotNull
  private String name;

  @NotNull
  private boolean active;

  public Student(String id, String name, boolean active) {
    this.id = id;
    this.name = name;
    this.active = active;
  }

  public String getId() { return id; }
  public void setId(String id) { this.id = id; }
  public String getName() { return name; }
  public void setName(String name) { this.name = name; }
  public boolean isActive() { return active; }
  public void setActive(boolean active) { this.active = active; }
}
```

3.7. The following are three (3) Java classes called LearningOOS.java, LearningOIS.java and Student.java with an empty binary file called object.dat.

LearningOOS.java

```java
package abc;

import java.io.*;

public class LearningOOS {
  public static void main(String[] args) throws Exception {
    try(
      ObjectOutputStream output = new ObjectOutputStream(
        new FileOutputStream("object.dat"))
    ) {
      output.writeUTF("Students");
      output.writeUTF("DITN01");
      output.writeObject(new Student(1, "Jailani"));
      output.writeObject(new Student(2, "Rahman"));
    }
  }
}
```

LearningOIS.java

```java
package abc;

import java.io.*;

public class LearningOIS {
  public static void main(String[] args) throws Exception{
    try(
      ObjectInputStream input = new ObjectInputStream(
        new FileInputStream("object.dat"))
    ) {
      System.out.println(input.readUTF());
      System.out.println(input.readUTF());
      Student student = (Student) input.readObject();
      System.out.println(student.id);
      System.out.println(student.name);
      Student student2 = (Student) input.readObject();
      System.out.println(student2.id);
      System.out.println(student2.name);
    }
  }
}
```

Student.java

```java
package abc;

public class Student {
  public int id;
  public String name;

  public Student(int id, String name) {
    this.id = id;
    this.name = name;
  }
}
```

object.dat

```

```

3.7.1. When executing LearningOOS.java, it threw an exception, java.io.NotSerializableException. Explain why that is the case.

> LearningOOS.java is implementing an OutputStream for storing objects where it tries to store an object from Student class.
>
> Since Student class does not support the Serializable interface, attempting to store Student object would cause the NotSerializableException.

3.7.2. Suggest a solution to prevent LearningOOS.java from throwing an exception, java.io.NotSerializableException. (Note: You are not required to rewrite the whole class. Just state the Java class and line number(s) that needs to be replaced with your solution.)

```java
// Student.java, Line 3.

public class Student implements Serializable
```

3.7.3. Produce the output generated by the code in LearningOIS.java, assuming that LearningOOS.java has been executed once without any issue before executing LearningOIS.java.

> Students
> DITN01
> 1
> Jailani
> 2
> Rahman

3.8. The following are two (2) Java classes called LearningDOS.java and LearningDIS.java respectively with an empty binary file called data.dat.

LearningDOS.java

```java
import java.io.*;

public class LearningDOS {
  public static void main(String[] args) {
    try(DataOutputStream output = new DataOutputStream(
        new FileOutputStream("data.dat", true))) {
      output.writeUTF("Nike");
      output.writeUTF("Puma");
      output.writeUTF("Adidas");
      output.writeUTF("Umbro");
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
}
```

LearningDIS.java

```java
import java.io.*;

public class LearningDIS {
  public static void main(String[] args) {
    try(DataInputStream output = new DataInputStream(
        new FileInputStream("data.dat"))) {
      while(output.available() > 0) {
        System.out.println(output.readUTF());
      }
    } catch (IOException e) {
    e.printStackTrace();
    }
  }
}
```

data.dat

```

```

Produce the output generated by the code in LearningDIS.java, assuming that LearningDOS.java has been executed once before executing LearningDIS.java.

> Nike

> Puma
>
> Adidas
>
> Umbro

3.9. The following are two (2) Java classes called Server.java and Client.java.

Server.java

```
public class Server {
  public static void main(String[] args) {

  }
}
```

Client.java

```
public class Client {
  public static void main(String[] args) {

  }
}
```

3.9.1. Server.java is a server application. Complete the implementation of Server.java based on the following.

When the server application started it will listen to port number 11234. The server application should accept connection from a client application and it should be able to send Java objects to and receive Java objects from the client application. (Note: You are not required to rewrite the whole class, just write the code statements in proper order. You are not required to handle any exceptions being thrown by any of the statements or import any of the classes.)

```
ServerSocket serverSocket = new ServerSocket(11234);
Socket socket = serverSocket.accept();
ObjectInputStream input =
  new ObjectInputStream (socket.getInputStream());
ObjectOutputStream output =
  new ObjectOutputStream (socket.getOutputStream());
```

3.9.2. Client.java is a client server that connects to Server.java, a server application. Complete the implementation of Client.java based on the following.

It will request a connection to the server application hosted in IP address 202.123.456.789. The client application should be able to send Java objects to and receive Java objects from the server application. (Note: You are not required to rewrite the whole class, just write the code statements in proper order. You are not required to handle any exceptions being thrown by any of the statements or import any of the classes.)

```
Socket socket = new Socket(“202.123.456.789”, 11234);
ObjectInputStream input =
  new ObjectInputStream (socket.getInputStream());
ObjectOutputStream output =
  new ObjectOutputStream (socket.getOutputStream());
```

3.10. The following is a Java class called MainController.java

```
@Controller
public class MainController {

}
```

Implement a method in MainController.java to handle Uniform Resource Identifier (URI) “/message” where it will response with a literal String “Abu Bakar Curi Daging”. (Note: You are not required to rewrite the whole class.)

```
@RequestMapping(value=”/message”)
@ResponseBody
public String hello() {
  return “Abu Bakar Curi Daging”;
}
```

3.11. The following are Java classes called MainController.java and Stock.java and is a Hypertext Markup Language (HTML) called stocks.html, that are used for Spring Web Application (with Thymeleaf).

MainController.java

```
@Controller
public class MainController {
  List<Stock> stocks = (List<Stock>) Arrays.asList(
      new Stock("Laptop", 10, 1999.99),
      new Stock("Solid State Drive", 100, 99.99),
      new Stock("SD Card", 200, 30.99),
      new Stock("Monitor", 100, 299.99)
      );

  @RequestMapping(value="/stocks")
  public String stocks(ModelMap modelMap) {
    modelMap.put("stocks", stocks);
    return "stocks";
  }
}
```

Stock.java

```java
public class Stock {
  private String name;
  private int quantity;
  private double price;

  public Stock(String name, int quantity, double price) {
    this.name = name;
    this.quantity = quantity;
    this.price = price;
  }

  public String getName() { return name; }
  public void setName(String name) { this.name = name; }
  public int getQuantity() { return quantity; }
  public void setQuantity(int quantity) { this.quantity = quantity; }
  public double getPrice() { return price; }
  public void setPrice(double price) { this.price = price; }
}
```

stocks.html

```html
<!DOCTYPE html>
<html>
<head>
<title>NS4307 NEP</title>
</head>
<body>
<h1>List of Stocks</h1>
<table>
<tr>
<th>Name</th>
<th>Quantity</th>
<th>Price</th>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</table>
</body>
</html>
```

3.11.1. Complete the implementation of Line 14 to Line 18 of the HTML file, stocks.html which it should display all data passed to stocks.html through MainController.java where each Stock.java object is displayed as table row and each field of Stock.java object is displayed in its respective table column. (Note: You are not required to rewrite the whole html file.)

```
<tr th:each="stock : ${ stocks }">
<td th:text="${ stock.name }"></td>
<td th:text="${ stock.quantity }"></td>
<td th:text="${ stock.price }"></td>
</tr>
```

3.11.2. The Integrated Development Environment (IDE) that you are using to develop this Spring application is showing an error after you completed Question 3.11.1. This is because you haven't declared Thymeleaf namespace into your stocks.html. State where you need to declare Thymeleaf namespace in stocks.html and implement the Thymeleaf namespace in stocks.html. (Note: You are not required to rewrite the whole html file.)

```
<html xmlns:th="http://www.thymeleaf.org">
```

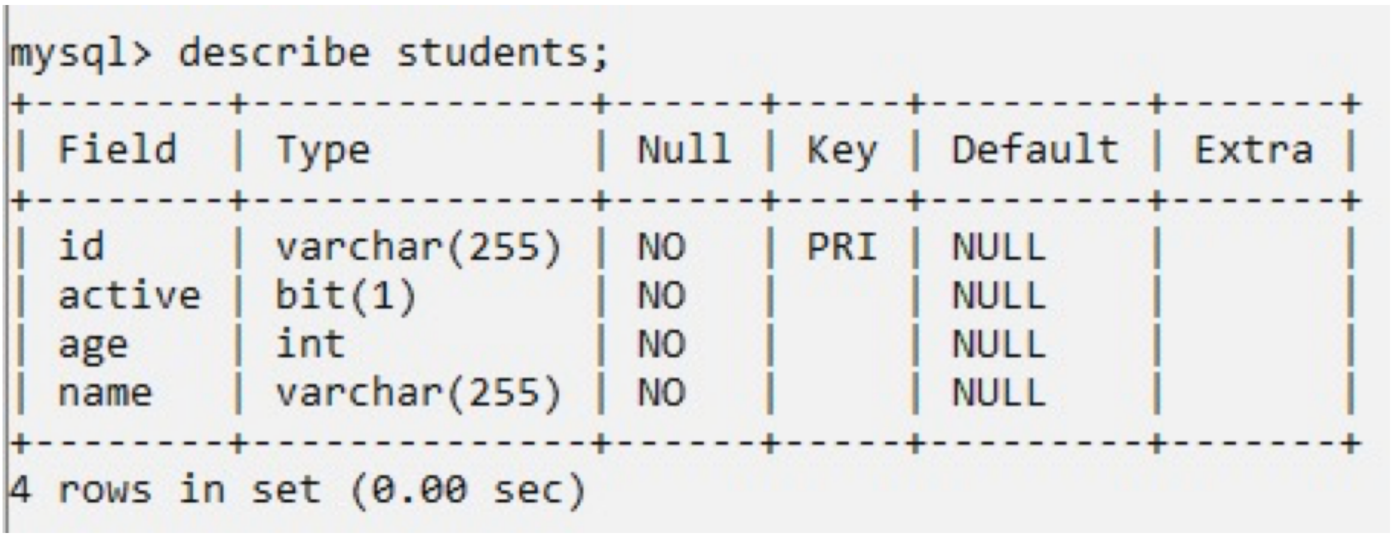
3.12. The following is a Java class called Student.java and is a command line (terminal) screenshot of a table details queried in MySQL Database Management System server, that are used for Spring Web Application with Hibernate.

Student.java

```
public class Student {

}
```

students MySQL table



```
mysql> describe students;
+--------+--------------+------+-----+---------+-------+
| Field  | Type         | Null | Key | Default | Extra |
+--------+--------------+------+-----+---------+-------+
| id     | varchar(255) | NO   | PRI | NULL    |       |
| active | bit(1)       | NO   |     | NULL    |       |
| age    | int          | NO   |     | NULL    |       |
| name   | varchar(255) | NO   |     | NULL    |       |
+--------+--------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

Rewrite the Plain Old Java Object class, Student.java by making the class to be a Database entity that will be used by Hibernate to create the Database table with its field and constraints accordingly.

```java
@Entity
@Table(name = "students")
public class Student {
  @Id
  private String id;

  @NotNull
  private String name;
  private int age;
  private boolean active;

  public Student(String id, String name, int age, boolean active) {
    this.id = id;
    this.name = name;
    this.age = age;
    this.active = active;
  }

  public String getId() { return id; }
  public void setId(String id) { this.id = id; }
  public String getName() { return name; }
  public void setName(String name) { this.name = name; }
  public int getAge() { return age; }
  public void setAge(int age) { this.age = age; }
  public boolean isActive() { return active; }
  public void setActive(boolean active) { this.active = active; }
}
```

3.13. The following is a Spring Web Application with Hibernate class named Employee.java

```java
@Entity
@Table(name = "employees")
public class Employee {
  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  @NotNull
  private String employeeName;
  @NotNull
  private int salary;

  public Employee(String employeeName, int salary) {
    this.employeeName = employeeName; this.salary = salary;
  }

  public String getEmployeeName() { return employeeName; }
  public void setEmployeeName(String employeeName) { this.employeeName =
employeeName; }
  public int getSalary() { return salary; }
  public void setSalary(int salary) { this.salary = salary; }
}
```

The code above is implemented using Java Persistence API (JPA). Explain how JPA structured the database due to Employee.java class assuming this entity does not exist in the database.

> - It will create a new database table.
> - The name of the database table will be students
> - There will be four (4) attributes in the database table, id, employeeName and salary.
> - id will be the primary key for the table.
> - id will be auto-incremented.
> - id, employeeName and salary will not allow null value to be stored into it.

3.14. Before a Spring Web Application with Hibernate can connect to a Database Management System, the web application need to be configured first. The configurations are implemented in application.properties file as follows.

```properties
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
spring.datasource.url=jdbc:mysql://192.168.1.1:3306/nep
spring.datasource.username=jailanihar
spring.datasource.password=Mypassword123-
spring.jpa.hibernate.ddl-auto=create-drop
spring.jpa.show-sql=false
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5Dialect
```

Analyse the above code and explain the purpose of each property in context with the value assigned to the properties.

> spring.datasource.driver-class-name:
>
> Use MySQL JDBC Driver to connect to the database.
>
> spring.datasource.url:
>
> Connect to database called nep hosted in 192.168.1.1 port 3306.
>
> spring.datasource.username:
>
> jailanihar is the username used to connect to the database server.
>
> spring.datasource.password:
>
> MyPassword123- is the password used for jailanihar to connect to the database server.
>
> spring.jpa.hibernate.ddl-auto:
>
> Everytime the web application is started, drop and recreate the database tables.
>
> spring.jpa.show.sql:
>
> Show the sql statement in the console.
>
> spring.jpa.properties.hibernate.dialect:
>
> Hibernate use MySQL v5 SQL statement to access and manipulate data in the database.

3.15. The following is a Java class named MyController.java and is a Hypertext Markup Language (HTML) named index.html. Both files are used for Spring Web Application (with Thymeleaf).

MyController.java

```java
@Controller
public class MyController {

}
```

index.html

```html
<!DOCTYPE html>
<html xmlns:th="http://thymeleaf.org">
<body>
<form action="/hello" method="post">
<p>Username</p>
<input type="text" name="username"/>
<p>Message</p>
<input type="text" name="message"/>
</br>
<button type="submit">Submit</button>
</form>
</body>
</html>
```

3.15.1. Implement a method in MyController.java to handle Uniform Resource Identifier (URI) "/home" where it will response with index.html (Note: you are not required to rewrite the whole class)

> ```java
> @RequestMapping(value="/home")
> public String home() {
>   return "index";
> }
> ```

3.15.2. Implement a method in MyController.java (Note: you are not required to rewrite the whole class):

- To handle the action made by the form in index.html.
- To handle the request parameter sent by the form in index.html.
- Response with a plain text of all the request parameter.

> ```java
> @RequestMapping(value="/hello")
> @ResponseBody
> public String hello(
>   @RequestParam String username,
>   @RequestParam String message
> ) {
>   return username + ": " + message;
> }
> ```

3.16. The following are two Java classes named Server.java and Client.java.

Server.java

```java
public class Server {
  public static void main(String[] args) {
    try {
```

```java
      ServerSocket serverSocket = new ServerSocket(9101);
      Socket socket = serverSocket.accept();
      DataOutputStream toClient = new DataOutputStream(new
          BufferedOutputStream(socket.getOutputStream()));
      toClient.writeUTF("Hello");
      toClient.writeInt(12345);
      toClient.writeUTF("World");
      toClient.flush();
      serverSocket.close();
      socket.close();
    } catch (IOException e) {
    e.printStackTrace();
  }
  }
}
```

Client.java

```java
 1  public class Client {
 2    public static void main(String[] args) {
 3      try {
 4        Socket socket = new Socket("localhost", 9101);
 5        DataInputStream fromServer = new DataInputStream(new
 6            BufferedInputStream(socket.getInputStream()));
 7        /*TODO*/
 8      } catch (IOException e) {
 9        e.printStackTrace();
10    }
11    }
12  }
```

Complete the implementation of Client.java at line 7 commented with /*TODO*/ where it will read all the data sent by Server.java and output to its console.

> ```java
> /*TODO*/
>
> System.out.println(fromServer.readUTF());
> System.out.println(fromServer.readInt());
> System.out.println(fromServer.readUTF());
> ```

3.17. The following are three Java classes named Student.java SaveStudentFile.java and LoadStudentFile.java with an empty binary file named student.dat.

Student.java

```java
public class Student implements Serializable {
  public int id;
  public String name;
  public int cgpa;
  public String school;

  public Student(int id, String name, int cgpa, String school) {
    this.id = id;
    this.name = name;
    this.cgpa = cgpa;
    this.school = school;
  }

  @Override
  public String toString() {
    return "Student [id=" + id + ", name=" + name
      + ", cgpa=" + cgpa + ", school=" + school + "]";
  }
}
```

SaveStudentFile.java

```java
public class SaveStudentFile {
  public static void main(String[] args) {
    try(ObjectOutputStream output = new ObjectOutputStream(new
        FileOutputStream("student.dat"))) {
    output.writeObject(new Student(100, "Jailani", 3.2, "SICT"));
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
}
```

LoadStudentFile.java

```java
public class LoadStudentFile {
  public static void main(String[] args) {
    try(/*TODO1*/) {
    Student s = /*TODO2*/;
    System.out.println(s);
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

student.dat

```

```

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

> ```java
> /*TODO1*/
> ObjectInputStream input = new ObjectInputStream(new FileInputStream("student.dat"))
> /*TODO2*/
> System.out.println((Student) input.readObject());
> ```

3.18. Your colleague is implementing a Java server application. He implemented it in a Java class called Server.java by creating a ServerSocket object with port 8080.

```java
public class Server {
  public static void main(String[] args) throws IOException {
    /*Some Implementation*/
    ServerSocket serverSocket = new ServerSocket(8080);
    /*TODO*/
    /*Some Implementation*/
  }
}
```

3.18.1. When he tries to execute the class above, it produced a Runtime error called BindException. Explain to him what causes this error.

> The port 8080 is currently in use.

3.18.2. Suggest a solution that will fix the error in Question 3.18.1.

> Stop any application that is using that port number or use different port number.

3.18.3. Complete the implementation of Server.java where the server application should accept connection from two client applications and the server application should be able to communicate with each client application separately. (Note: you are not required to rewrite the whole class, just write the codes needed in `/*TODO*/`).

```java
Socket socket1 = serverSocket.accept();
Socket socket2 = serverSocket.accept();
```

3.18.4. After successfully executing Server.java, he needs your help to implement a Java client application to connect and communicate to the server in the same network. All you need to do is to write the code (statement) needed to connect to the server application. The server application Internet Protocol (IP) address is 192.168.1.1. (Note: you are not required to write the whole client class)

```java
Socket socket = new Socket("192.168.1.1", 8080);
```

3.18.5. When you run the client application, it terminates and produced an error with ConnectException. Explain what causes this exception error.

> The server application is not running or cannot be reached.

3.19. The following are three Java classes named Product.java, Employee.java and MyController.java and are two Hypertext Markup Language (HTML) files named abc.html and def.html, that are used for Spring Web Application (with Thymeleaf).

Product.java

```java
public class Product {
  private String productName;
  private int stock;

  public Product(String productName, int stock) {
    this.productName = productName;
    this.stock = stock;
  }

  public String getProductName() { return productName; }
  public void setProductName(String productName) {
    this.productName = productName;
  }
  public int getStock() { return stock; }
  public void setStock(int stock) { this.stock = stock; }
}
```

Employee.java

```java
public class Employee {
  private String employeeName;
  private int salary;

  public Employee(String employeeName, int salary) {
    this.employeeName = employeeName;
    this.salary = salary;
  }

  public String getEmployeeName() { return employeeName; }
  public void setEmployeeName(String employeeName) {
    this.employeeName = employeeName;
  }
  public int getSalary() { return salary; }
  public void setSalary(int salary) { this.salary = salary; }
}
```

MyController.java

```java
@Controller
public class MyController {
  @RequestMapping(value="/employee")
  public String getEmployee(ModelMap modelMap) {
    Employee employee = new Employee("Jailani", 1500);
  modelMap.put("emp", employee);
  return "def";
  }

  @RequestMapping(value="/product")
  public String getProduct(ModelMap modelMap) {
    Product product = new Product("Laptop", 2);
  modelMap.put("prod", product);
  return "abc";
  }
}
```

abc.html

```html
<!DOCTYPE html>
<html xmlns:th="http://thymeleaf.org">
<head></head>
<body></body>
</html>
```

def.html

```html
<!DOCTYPE html>
<html xmlns:th="http://thymeleaf.org">
<head></head>
<body></body>
</html>
```

3.19.1 Analyse the code above and complete the implementation of abc.html and def.html (using paragraph tag `<p></p>` and Thymeleaf syntax) where it should display into the page all the data inside each Model that passed through the ModelMap respectively. (Note: you do not need to rewrite the whole HTML file. Specify the file and parent tag which the paragraph should be implemented. Arrangement of data does not matter)

> abc.html
>
> ```html
> <body>
>   <p th:text="${prod.productName}"></p>
>   <p th:text="${prod.stock}"></p>
> </body>
> ```
>
> def.html
>
> ```html
> <body>
>   <p th:text="${emp.employeeName}"></p>
>   <p th:text="${emp.salary}"></p>
> </body>
> ```

3.19.2. Explain why in both html files implements xmlns:th="http://thymeleaf.org" in the html tag and what happens if it is omitted.

> To identify "th" into the XML name space
> If this is omitted, some IDE might consider attributes with "th:" as an error.

3.19.3. Explain the use of @Controller and @RequestMapping implemented in MyController.java.

> @Controller
>
> To indicate the class is Spring Controller.
>
> @RequestMapping
>
> To handle the URI request

3.20. The following are two Java classes named SaveStudentFile.java and LoadStudentFile.java with an empty binary file 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();
    }
  }
}
```

LoadStudentFile.java

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

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

> ```java
> /*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());
> ```

3.20.2. Explain the meaning of the second argument (second parameter), in SaveStudentFile.java when creating FileOutputStream object and what the boolean value false means.

> It is for FileOuputStream object to decide to append or override the file.
>
> false means override the file.

3.21. The following is a HTML file called index.html that is used for Spring Web Application (with Thymeleaf).

```html
<!DOCTYPE html>
<html lang="en" xmlns:th="http://thymeleaf.org">
<head></head>
<body>
<h1 th:text="${student.name}"></h1>
<p th:text="${student.age}"></p>
<p th:text="${student.school}"></p>
<img th:src="@{'/images/' + ${student.picFile}}"></img>
</body>
</html>
```

By analysing the above code, create a Plain old Java Object (POJO) class called Student.java with only instance variables as stated in index.html. (Note: implement a constructor and the setter and getter methods for the instance variables in Student.java).

> ```java
> public class Student {
>   private String name;
>   private int age;
>   private String school;
>   private String picFile;
>
> ```

```java
  public Student(String name, int age,
                 String school, String picFile) {
    this.name = name;
    this.age = age;
    this.school = school;
    this.picFile = picFile;
  }

  public String getName() { return name; }
  public void setName(String name) { this.name = name; }
  public int getAge() { return age; }
  public void setAge(int age) { this.age = age; }
  public String getSchool() { return school; }
  public void setSchool() { this.school = school; }
  public String getPicFile() { return picFile; }
  public void setPicFile(String picFile) { this.picFile = picFile; }
}
```

3.22. The following is a Spring Web Application (with Thymeleaf) Controller file called MyController.java.

```java
@Controller
public class MyController {
    @RequestMapping(value="/")
    public String index() {
        return "home";
    }

    @RequestMapping(value="/sendmessage")
    @ResponseBody
    public String sendmessage(
      @RequestParam(required=false) String message) {
        if(message == null) {
            return "No message is being sent.";
        } else {
            return "Message sent: " + message;
        }
    }
}
```

3.22.1. Explain what does @Controller (Line 1), @RequestMapping (Line 3 and Line 8), @ResponseBody (Line 9) and @RequestParam(required=false) (Line 10) means in MyController.java.

> @Controller: To indicate the class is Spring Controller
>
> @RequestMapping: To handle the URI request
>
> @ResponseBody: To indicate the return value is to be response body.
>
> @RequestParam(required=false): To handle the GET/POST HTTP request where the HTTP request is optional.

3.22.2. Explain what Line 3 to Line 6 in MyController.java does.

> - If there is request for URI /
> - Spring Web Application will run the index() method
> - returning home.html from its resource template folder.

3.22.3. Explain what Line 8 to Line 16 in MyController.java does.

> - If there is request for URI /sendmessage
> - Spring Application will run the sendmessage() method
> - It will check first if there is GET/POST HTTP request in URI
> - If there is no HTTP request called message
> - it returns the word "No message is being sent." as the response body
> - If there is HTTP request called message with value
> - it returns the word "Message sent: " and
> - whatever value assigned to HTTP request message.

3.23. Your colleague is implementing a Java server application. He implemented it in a Java class called Server.java by creating a ServerSocket object with port 9921.

```java
public class Server {
    public static void main(String[] args) throws IOException {
        /*Some Implementation*/
        ServerSocket serverSocket = new ServerSocket(9921);
        /*TODO*/
        /*Some Implementation*/
    }
}
```

3.23.1. Complete the implementation of Server.java where the server application should accept connection from three (3) client applications and the server application should be able to communicate with each client application separately. (Note: you are not required to rewrite the whole class, just write the codes needed in `/*TODO*/`).

> ```
> Socket socket1 = serverSocket.accept();
> Socket socket2 = serverSocket.accept();
> Socket socket3 = serverSocket.accept();
> ```

3.23.2. After successfully executing Server.java, he needs your help to implement a Java client application to connect and communicate to the server in the same network. All you need to do is to write the code needed to connect to the server application. The server application Internet Protocol (IP) address is 192.168.100.10. (Note: you are not required to write the whole client class).

> ```
> Socket socket = new Socket("192.168.100.10", 9921);
> ```

3.23.3. When you run the client application, it terminates and produced an error with ConnectException. Explain what causes this exception error.

> The server application is not running or cannot be reached.

3.24. The following is a Spring Web Application with Hibernate class called Student.java.

```
@Entity
@Table(name = "students")
public class Student {
  @Id
  @GeneratedValue(strategy = GenerationType.IDENTIFY)
  private int id;
  @Column(nullable=false)
  private String name;
  @Column(nullable=false)
  private int age;
  @Column(nullable=false)
  private String gender;

  public Student(int id, String name, int age, String gender) {
      this.id = id;
      this.name = name;
      this.age = age;
      this.gender = gender;
  }

```

```java
  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 int getAge() { return age; }
  public void setAge(int age) { this.age = age; }
  public String getGender() { return gender; }
  public void setGender(String gender) { this.gender = gender; }
}

```

The code above is implemented using Java Persistence API (JPA) through Hibernate. Explain how JPA structured the database due to Student.java class assuming this entity does not exist in the database.

> - It will create a new database table.
> - The name of the database table will be students
> - There will be four (4) attributes in the database table, id, name, age and gender.
> - id will be the primary key for the table.
> - id will be auto-incremented.
> - name, age and gender will not allow null value to be stored into it.

3.25. The following are two (2) Java classes called Server.java and Client.java.

Server.java

```java
public class Server {
  public static void main(String[] args) {
    try {
      ServerSocket serverSocket = new ServerSocket(9101);
      System.out.println("Waiting for Client.");

      Socket socket = serverSocket.accept();
      System.out.println("Client has connected.");

      DataInputStream fromClient = new DataInputStream(
      new BufferedInputStream(socket.getInputStream()));
      DataOutputStream toClient = new DataOutputStream(
      new BufferedOutputStream(socket.getOutputStream()));

      String firstDataName = fromClient.readUTF();
      System.out.println(firstDataName);
      double firstData = fromClient.readDouble();
      System.out.println(firstData);
      String secondDataName = fromClient.readUTF();
      System.out.println(secondDataName);
      int secondData = fromClient.readInt();
      System.out.println(secondData);

      double total = firstData + secondData;
      String sendData = firstDataName + " + "
        + secondDataName + " = " + total;
      toClient.writeUTF(sendData);
      toClient.flush();
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
}
```

Client.java

```java
public class Client {
  public static void main(String[] args) {
    try {
      Socket socket = new Socket("localhost", 9101);
      System.out.println("Connected to the server.");
      DataInputStream fromServer = new DataInputStream(
      new BufferedInputStream(socket.getInputStream()));
      DataOutputStream toServer = new DataOutputStream(
      new BufferedOutputStream(socket.getOutputStream()));

      Scanner scanner = new Scanner(System.in);

      toServer.writeUTF("X");
      toServer.writeDouble(0.5);
      toServer.writeUTF("Y");
      toServer.writeInt(1);
      toServer.flush();

      System.out.println(fromServer.readUTF());

    } catch (UnknownHostException e) {
      e.printStackTrace();
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
}
```

Server.java will be run followed by Client.java. Both classes are run in the same computer. Produce the output generated by the code in Server.java and Client.java.

> Server.java Output:
>
> Waiting for Client.
>
> Client has connected.

X

0.5

Y

1

Client.java Output:

Connected to the server.

X + Y = 1.5

3.26. The following is a Spring Application (with Thymeleaf) Controller file called HelloController.java.

```java
@Controller
public class HelloController {
  @RequestMapping(value="/index")
  public String index() {
    return "index";
  }

  @RequestMapping(value="/hello/{name}")
  @ResponseBody
  public String hello(@PathVariable String name) {
    return "hello " + name;
  }
}
```

3.26.1. Explain the use of @Controller, @RequestMapping, @ReponseBody and @PathVariable.

@Controller

To indicate the class is Spring Controller.

@RequestMapping

To handle the URI request

@ResponseBody

To indicate the return value is to be response body.

@PathVariable

> To notifying that the method will get the data stated in URI.

3.26.2. Explain what Line 3 to Line 6 does (Note: you may explain using example).

> If there is request for URI /index,
>
> Spring Application will run the index() method
>
> returning index.html .

3.26.3. Explain what Line 8 to Line 12 does (Note: you may explain using example).

> If there is request for URI /hello/jailani,
>
> Spring Application will run the hello() method
>
> the word jailani will be passed into the method
>
> returning the word hello jailani as the response body.

3.27. The following two (2) Java classes called BelajarDataOutputStream.java and BelajarDataInputStream.java.

BelajarDataOutputStream.java

```java
public class BelajarDataOutputStream {
  public static void main(String[] args) {
    try(
      DataOutputStream output =
        new DataOutputStream(new FileOutputStream("data.dat"))
    ) {
      output.writeUTF("Dell");
      output.writeDouble(99.9);
      output.writeUTF("Samsung");
      output.writeDouble(150.3);
      output.writeUTF("Toshiba");
      output.writeDouble(70.2);
    } catch (IOException e) { }
  }
}
```

BelajarDataInputStream.java

```java
public class BelajarDataInputStream {
  public static void main(String[] args) {
    try(
      DataInputStream input =
        new DataInputStream (new FileInputStream(/*TODO-1*/))
    ) {
      /*TODO-2*/
    } catch (IOException e) { }
  }
}
```

Analyse the code in BelajarDataOutputStream.java, complete the implementation of BelajarDataInputStream.java which should read and System.out.println() all the content of the file generated by BelajarDataOutputStream.java. (Note: you don't need to rewrite the whole class, you can use the numbered `/*TODO*/` to identify which part you want your code to be implemented).

> ```java
> System.out.println(input.readUTF());
> System.out.println(input.readDouble());
> System.out.println(input.readUTF());
> System.out.println(input.readDouble());
> System.out.println(input.readUTF());
> System.out.println(input.readDouble());
> ```

3.28. The following is a Java class called BelajarJavaIO.java and binary file called numbers.dat.

BelajarJavaIO.java

```java
public class BelajarJavaIO {
  public static void main(String[] args) {
    try(
      FileInputStream input =
        new FileInputStream("numbers.dat");
      FileOutputStream output =
        new FileOutputStream("numbers.dat", true);
    ) {
      int value = 0;
      while(input.available() > 0) {
        value = input.read();
      }
      value++;
      for(int i = value; i < value+6; i++) {
        output.write(i);
      }
    } catch (IOException e) { }
  }
}
```

numbers.dat (the number shown is just a representation of the data in the file)

```
1
```

3.28.1. By analyzing the above code, explain what the following Line 4, Line 5-6, Line 9-11 and Line 12-15 code does.

> Line 4:
>
> Create an input stream to read numbers.dat file.
>
> Line 5-6
>
> Create an output stream to write into numbers.dat file
>
> If the file exists, it will append the content of the file
>
> If the file does not exists, it will write into a new file called numbers.dat
>
> Line 9-11
>
> While the numbers.dat file has data to be read
>
> Read each byte and store it into integer variable called value.
>
> Line 12-15
>
> Variable "value" is the last integer read in the file.
>
> Increment variable "value" by 1.
>
> Write the incremented variable "value" to the file and then the next 5 increment by 1 integer.

3.28.2. Explain the use of try-with resource in BelajarJavaIO.java.

> This is to automatically closed FileInputStream and FileOutputStream after they are used.

3.28.3. Produce the number representation of the content in the file numbers.dat after BelajarJavaIO.java is executed once.

> 1 2 3 4 5 6 7