Cairo University **Faculty of Computers and Artificial Intelligence**
Computer Science Department – Software Engineering Undergrad Program

SCS 491: Selected Topics in Software Engineering-1, Practical EXAMINATION
Winter 2023 – CLOSED Book Exam – Duration: 40 mins
This exam comes in **five** pages.

STUDENT NAME _____

STUDENT ID # _____ STUDENT Group #

Question 1(2 mark):
Write a one-way client and server setup using TCP socket connection where a client connects, sends a message to the server on IP 192.168.1.67 and port number 5000 and the server prints the message.
Server Side:

```
ServerSocket ss = new ServerSocket(5000);
Socket s = ss.accept();
DataInputStream dis = new DataInputStream(s.getInputStream());
String  str = (String) dis.readUTF();
System.out.println("message= " + str);
ss.close();
```

Client Side:

```
Socket s = new Socket ("192.168.1.67", 5000);
DataOutputStream dos = new DataOutputStream(s.getOutputStream());
dos.writeUTF("Hello Server");
dos.flush();
dos.close();
s.close();
```

Question 2 (4 marks):
Using the classes/interfaces shown below, apply the necessary modifications which you studied to make it running using RMI mechanism. You're required to use and test the two methods inside the "StudentInt". The first one addCourseResult() takes a Result object and inserts it inside the results array. The second method getCourseResult() gets a Result object by searching the array with an input courseId. You're required to implement a server program and make the mentioned operations available via an implementation object placed in the RMI registry by the server.
Access this implementation object via a client program and use the methods of the interface by adding a Result object and displaying its info after being added to the array.

Result.java

```
public class Result implements Serializable {
   private int courseId;
   private String courseName;
```

```
    private int mark;
    public Result(int courseId, String courseName, int mark) {
        this.courseId = courseId;
        this.courseName = courseName;
        this.mark = mark;
    }
    public void setCourseId(int id) {
        courseId = id;
    }
    public int getCourseId() {
        return courseId;
    }
    public void setName(String name) {
        courseName = name;
    }
    public String getName() {
        return courseName;
    }
    public int getMark() {
        return mark;
    }
    public void setMark(int score) {
        mark = score;
    }
}
```

StudentInt.java

```
public interface StudentInt extends Remote
{
    void addCourseResult(Result result) throws RemoteException;
    Result getCourseResult (int courseId) throws RemoteException;
}
```

StudentImpl.java

```
public class StudentImpl extends UnicastRemoteObject implements StudentInt
{
    ArrayList<Result> results;
    protected StudentImpl () throws RemoteException
    {
        super();
        results = new ArrayList<>();
    }
    @Override
    public void addCourseResult(Result result) throws RemoteException
    {
        results.add(result);
    }
```

```
    @Override
    public Result getCourseResult(int courseId) throws RemoteException
    {
        for (int i = 0; i < results.size(); i++) {
            if (results.get(i).getCourseId() == courseId) {
                return results.get(i);
            }
        }
        return null;
    }
```

RMIServer.java

```
public class RMIServer {
    public static void main(String args[]) {
        try
        {
            // Create and start the remote application
            StudentImpl obj = new StudentImpl();
            Registry reg = LocateRegistry.getRegistry(); OR LocateRegistry.createRegistry(4444);
            reg.bind("StudentService", obj);
        }
        catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

RMIClient.java

```
public class RMIClient {
    public static void main(String[] args) {
        try
        {
            // Create and start the client application
            Registry reg = LocateRegistry.getRegistry("localhost"); // OR (“localhost”, 4444);
            StudentInt studentRef = (StudentInt) reg.lookup("StudentService");
            Result res = new Result(111, "Math", 90);
            studentRef.addCourseResult(res); // first method
            System.out.println(studentRef.getCourseResult(111).getMark()); // second method

        }
        catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Question 3 (1 marks):
MyE-Cart is an online e-commerce web store application where a user can buy different types of products by placing them to the shopping cart until the user chooses to checkout and pay for the items he selected. What kind of beans could be most appropriate for adding the product items to the cart?

a. Write a sample class for that "ECartEJB" pojo, assuming that it has one empty method for adding a product to the shopping cart. **Justify** why you chose that bean type specifically.

Stateful – Because the shopping cart component must track the user's state for each request (or any correct justification)

```
@Stateful
public class ECartEJB
{
   public boolean addItemToCart(Product p) {} // parameter and return types may differ
}
```

b. Create an "ECartClient" class to work as a local client for the "ECartEJB", then inject and invoke methods you created in "ECartEJB".

```
public class ECartClient {
   @EJB
   ECartEJB eCartEJB;

   // inside any function
   eCartEJB.addItemToCart(…)
}
```

Question 4 (3 marks):
Assuming that you are required to implement two microservices for the FCI E-COM. Having the below entity classes:

```
class Student {
    int id;
    string name;
    double gpa;
    string email;
}
```

```
class Course {
    int id;
    string name;
    ArrayList<Student> enrolledStudents;
}
```

Write two Spring web service controllers (StudentController and CourseController) providing the below two operations respectively:
Note: For both operations, you're not required to write the functions' body. Only the function header is enough.

a. Get student information by ID, such that it can be exposed using the below link:

   http://localhost:9090/students/find?id=20200100

Note: @RestController = @Controller + @ResponseBody // Both solutions are correct

```
@Controller @Controller OR @RestController
@RequestMapping("/students")
public class StudentController {
{
        @GetMapping("/find")
        @ResponseBody
        public Student findById(@RequestParam int id)
        {}
}
```

b. Register a student to a specific course, such that it can be exposed using the below link:

http://localhost:9090/courses/register?courseId=453&studentId=20200111

```
@Controller @Controller OR @RestController
@RequestMapping("/courses")
public class StudentController {
{
        @GetMapping("/register")
        @ResponseBody
        public boolean register(@RequestParam int courseId, @RequestParam int
studentId)
        {}
}
```

End of Exam