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

SCS 491: Selected Topics in Software Engineering-1, MIDTERM EXAMINATION

Instructor: Dr. Soha Makady

Winter 2023 – CLOSED Book Exam –Total marks: 20 – Duration: 60 mins

This exam comes in **four** pages.

Note: The provided answers within this sheet are NOT complete answers, but *Keys* to give you an idea about the answer. Hence, using them in your answers as is NOT acceptable.

STUDENT NAME ______________________________________

STUDENT ID # ______________________

Note: Some questions demand explanations and/or justification. If no explanation and/or justification is provided, that question will receive a zero grade.

Question 1 - Threads [3 marks]

Consider the following program that involves a CookieIssuer that issues cooies, a CookieMonster that eats cookies, and a CookiePlate.

a) **Modify** the program such that you don't need to extend the Thread class in the CookieIssuer and CookieMonster classes to support multiple threads. Yet, threading should still be supported **[2]**

```
public class CookiesAppTest {
	public static void main(String[] args) {
		CookiePlate c = new CookiePlate();
		CookieIssuer p1 = new CookieIssuer(c, 1);
		CookieMonster c1 = new CookieMonster(c, 1);
		Thread pThread = new Thread (p1);
		Thread cThread = new Thread (c1);
		pThread.start();
		cThread.start();
	}
}
```

```
public class CookieIssuer implements Runnable
{
	private CookiePlate cookiePlate;
	private int number;
	public CookieIssuer
		(CookiePlate c, int   number) {
		  cookiePlate = c;
		  this.number = number;
	}
```

```
public void run() {
 for (int i = 0; i < 10; i++) {
    cookiePlate.put(i);
    System.out.println("Monster #" +
this.number + " put: " + i);
    try {
      sleep((int)(Math.random() * 100));
      } catch (InterruptedException e) { }
    }
  }
}
```

```
public class CookieMonster implements Runnable
{

       private CookiePlate cookiePlate;
       private int number;
       public CookieMonster
                (CookiePlate c, int number) {
                   cookiePlate = c;
                   this.number = number;
       }
```

```
public void run() {
  for ( int i=0; i<10; i++) {
     System.out.println ("CookieMonster# "+
this.number + "get: "+ cookiePlate.get());
       try {
          sleep ((int)(Math.random()*100));
        } catch (InterruptedException e) {}
     }
  }
}
```

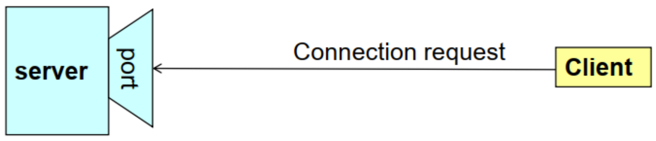
```
public class CookiePlate {

  private int contents;
  private boolean available = false;
  public synchronized int get() {

   //For part (b), add your code here
while (available == false) {
try {
wait();
} catch (InterruptedException e) { }
}
available = false;
notifyAll();
return contents;
}
  }
```

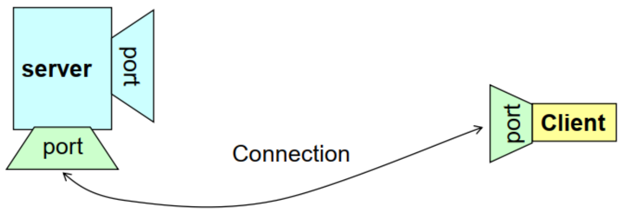
```
public synchronized void put(int value) {
  while (available == true) {
       try {
          wait();
       } catch (InterruptedException e) { }
    }
    contents = value;
    available = true;
    notifyAll();
  }
}
```

b) Complete the **above** CookiePlate.get() method so that it would be used to eat cookies. You need to add your code within the section headed **//For part (b), add your code here [1]**

Question 2 – Sockets and Distributed Systems Basics [7 marks]

a) **[3 marks]** Suppose that a program runs on a specific computer, and there are two client machines who would like to communicate with that program. **Sketch** a diagram to show the **complete sequence of communication**, through several communication requests, between the two clients and that program **using the TCP protocol**. Your diagram needs to clearly show the detailed IP address(es), port number(s), socket(s) as needed. Sketch another diagram if the **UDP protocol** was used instead.

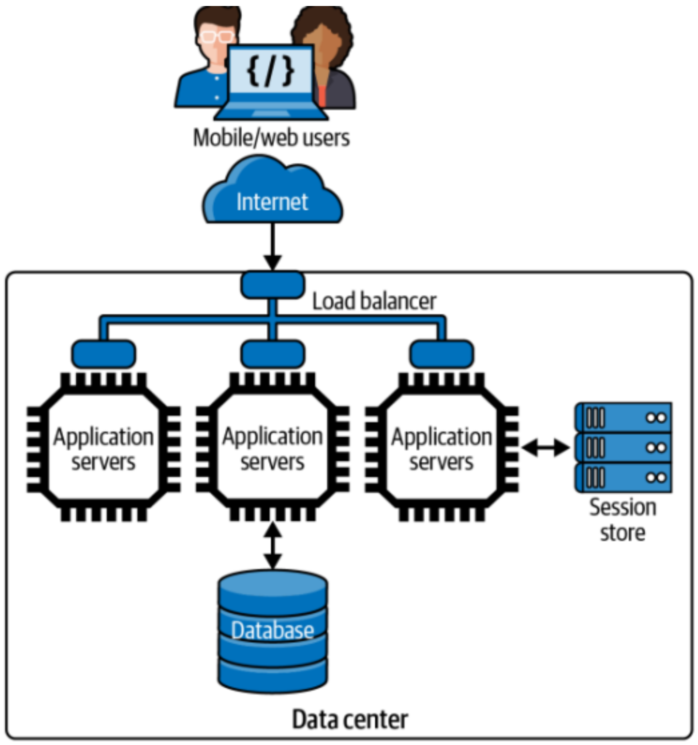




b) **[4 marks]** Within one of the scaling options for distributed systems, using a load balancer and stateless services is needed. **What** is usage of the load balancer? **Why** should the service be stateless? **Sketch** a diagram for the system's architecture when using such scaling option.

Load balancer → receives requests from and distributes them across the application servers (replicas), and sends back the responses from the replicas to the clients.

Stateless service→ To enable the load balancer to send consecutive requests from the same client to different service instances. Hence, the API implementations should retain no knowledge or state associated with an individual client's session



<u>Question 3 – Distributed Objects and Distributed Components [10 marks]</u>

a) **[2 marks]** One of the issues within distributed objects is implicit dependencies. **Explain** such limitation. Clarify how distributed components solved such limitation.

The answer should explain the limitation and then clarify how it got solved using distributed components.

b) **[2 marks]** What is meant by an **idempotent operation** when executed by a server? Provide an **example** for one operation that fits as an idempotent operation, and another example for an operation that does not fit as an idempotent operation. **Justify why** you see each operation fits within its classification.

   An idempotent operation is an operation that can be performed repeatedly with the same effect as if it had been performed exactly once.

c) **[2 marks]** An ideal distributed system should provide several requirements among them is **location transparency**. Describe how **Java RMI** satisfies such requirement using **example source code** of your choice.

*The answer should entail **RMI source code** specifically to clarify how location transparency is achieved.*

d) **[4 marks]** Consider a distributed application for seats reservation within Cineplex movie theatres. The application enables reserving seats, and canceling seat reservations. Furthermore, the application enables registering new customers (who would book the tickets) and updating those customers' information. Assume that the application can be used by any user for **any of the those features individually** (i.e., the user could register himself, without booking tickets at the same time).

   If you would use the RPC option for designing such a distributed application, answer the following questions:

   - **List** the client stub procedure(s) that could be present?

   Reserving seat, canceling seat reservations, registering customer, updating customer information

   - **List** the server stub procedure(s) that could be present?

   Reserving seat, canceling seat reservations, registering customer, updating customer information

   - How many dispatchers would be present, and **why would such number of dispatchers be needed**?

     1 dispatcher because it routes the message to the relevant server stub procedure based on the provided procedure ID within that message

If you would use the RMI option for designing such a distributed application, answer the following questions:

- What remote object(s) could be present? List the detailed skeleton of this/those object(s).
  Two objects: one for the user and one for the seat. The skeleton for each object should be listed as well.

- How many dispatchers would be present, and **why would such number of dispatchers be needed**?

  2 dispatcher because one dispatcher per class, hence it routes the message to the relevant operations within that class.

End of Exam