ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

FACULTAD DE INGENIERÍA EN ELECTRICIDAD Y COMPUTACIÓN

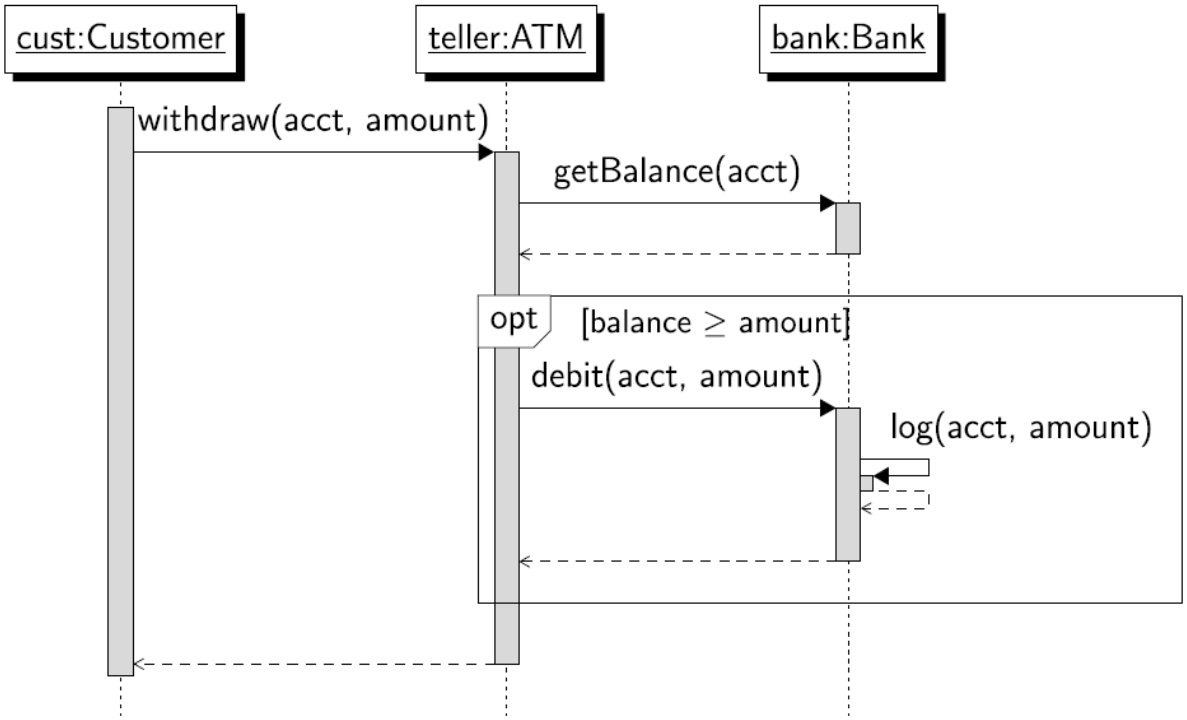
CCPG1042– DISEÑO DE SOFTWARE

PRIMER EXAMEN - I PAO 2024

Nombre: _____ ***Matrícula:*** _____

Sección A

1. Escriba el **mínimo** código fuente en Java para las clases *Customer*, *ATM* y *Bank* que capture el comportamiento del siguiente diagrama de secuencia UML. Los cuerpos de todos los métodos deben dejarse vacíos excepto para las llamadas indicadas en el diagrama. Constructores y propiedades (es decir, métodos *get* y *set*) de las clases no necesitan ser mostradas. **[16%]**



2. A usted se le ha encargado el diseño de un sistema de préstamos y reservas para una biblioteca. La descripción del proyecto es la siguiente:

   La biblioteca presta libros, DVDs y CDs. El sistema debe mantener un catálogo de todos estos elementos. Puede haber múltiples copias de cada elemento en la biblioteca. Una copia puede estar en la percha; en préstamo a un usuario; retenida por un usuario que la ha reservado; retenida para reparación; o podría estar perdida o descartada. Ningún usuario puede tener más de seis copias prestadas en ningún momento. Un usuario puede reservar hasta seis elementos. Si alguna copia de los elementos está disponible (es decir, en la percha) o se vuelve disponible porque fue retornada por otro usuario, entonces la copia es asignada a la primera reservación para ese elemento en la lista de reservaciones, removida de la percha y retenida por el bibliotecario para el usuario. Si el usuario no ha recogido su copia en una semana, entonces la reservación es cancelada y la copia vuelve a estar disponible (posiblemente para ser inmediatamente retenida por otra reservación). Cuando una copia dañada es devuelta, se envía para reparación y descartada si está demasiado dañada. Se asume como perdida la copia cuya fecha de devolución se haya excedido por diez semanas. En cualquier momento, una copia en percha puede ser escogida por el bibliotecario para que sea descartada y removida de la biblioteca.

a. Elabore un **diagrama de clases** UML para este sistema. Haga uso de herencia, agregación o composición, según sea aplicable. Agregue cardinalidades a las relaciones. Agregue atributos y métodos cuando sea necesario. No es necesario incluir *getters* y *setters* de atributos. Modificadores de visibilidad (público, privado, etc.) no son requeridos. Declare cualquier asunción que considere **necesaria**. (La nota será dividida entre su selección de clases, su selección de tipos de relaciones y sus respectivas cardinalidades). **[30%]**

3. Elabore un **diagrama de secuencia** UML para la operación *handle*. **[16%]**

```java
package exam2024;
import java.util.Iterator;

public class RoutingHandler {
    private Router router;

    public RoutingHandler() {
        router = new NullRouter();
    }

    public void handle(Iterator<Message> messages){
        while(messages.hasNext()){
            Message message = messages.next();
            if(message.isEncoded())
                router = new SmsRouter();
            else if(message.isEncrypted())
                router = new JmsRouter();
            router.route(message);
        }
    }
}

interface Message {
    Boolean isEncoded();
    Boolean isEncrypted();
}
interface Router {
    void route(Message msg);
}
class SmsRouter implements Router {
    @Override
    public void route(Message msg) {
        // implementation details
    }
}
class JmsRouter implements Router {
    @Override
    public void route(Message msg) {
        // implementation details
    }
}
class NullRouter implements Router {
    @Override
    public void route(Message msg) {
        // do nothing
    }
}
```

Sección B

4. Considere el caso de uso de ***retiro de dinero*** de una cuenta bancaria desde un cajero automático.

Acción del actor	Respuesta del sistema
1. Introducir tarjeta en el cajero	2. Requerir clave
3. Ingresar la clave	4. Chequear que la cuenta sea válida y activa
	5. Desplegar opciones
6. Seleccionar la opción para retiro	7. Desplegar cuentas asociadas a la tarjeta
8. Escoger la cuenta desde la cual retirar	9. Solicitar el monto a ser retirado
10. Ingresar el monto a ser retirado	11. Contar billetes
	12. Preguntar si se requiere recibo impreso
13. Ingresar decisión	14. Entregar el dinero
	15. Devolver la tarjeta al cliente
16. Remover tarjeta del cajero	17. Imprimir recibo (si se requiere)
	18. Actualizar balance de la cuenta

   a. **Documente** este caso de uso utilizando una plantilla con al menos **seis** elementos, incluyendo actores, precondiciones, postcondiciones, flujo de eventos normal y flujo de eventos alternativo. Indique cualquier asunción **razonable** que realice. **[11%]**
   b. Describa **dos** escenarios de este caso de uso. **[06%]**

5. Considere el código fuente en Java que se muestra a continuación.
   a. **Identifique** los principios SOLID que se están violando. **[03%]**
   b. Para cada principio violado, **justifique** su respuesta. **[06%]**
   c. **Corrija** el código de tal manera que ya no se lo viole. Si lo considera necesario, usted puede crear interfaces, clases u operaciones. Puede utilizar un diagrama de clase UML. **[12%]**

```java
package exam2024;

public class NotificationService {
    private NotificationProvider provider;

    public NotificationService(EmailNotificationProvider provider) {
        this.provider = provider;
    }

    public NotificationService(SMSNotificationProvider provider) {
        this.provider = provider;
    }

    public NotificationService(TelegramNotificationProvider provider) {
        this.provider = provider;
    }

    public void sendNotification() {
        provider.sendNotification();
    }

    public String formatMessage(String message) {
        // Logic to format the message here
        return message;
    }
}

interface NotificationProvider {
    void sendNotification();
}

class EmailNotificationProvider implements NotificationProvider {
    @Override
    public void sendNotification() {
        // Logic to Send an email here
    }
}

class SMSNotificationProvider implements NotificationProvider {
    @Override
    public void sendNotification() {
        // Logic to send an SMS here
    }
}
class TelegramNotificationProvider implements NotificationProvider {
    @Override
    public void sendNotification() {
        // Logic to send a Telegram Message here

        clearBuffer();
    }
    private void clearBuffer() {
        // logic to clear the buffer of messages here
    }
}
```