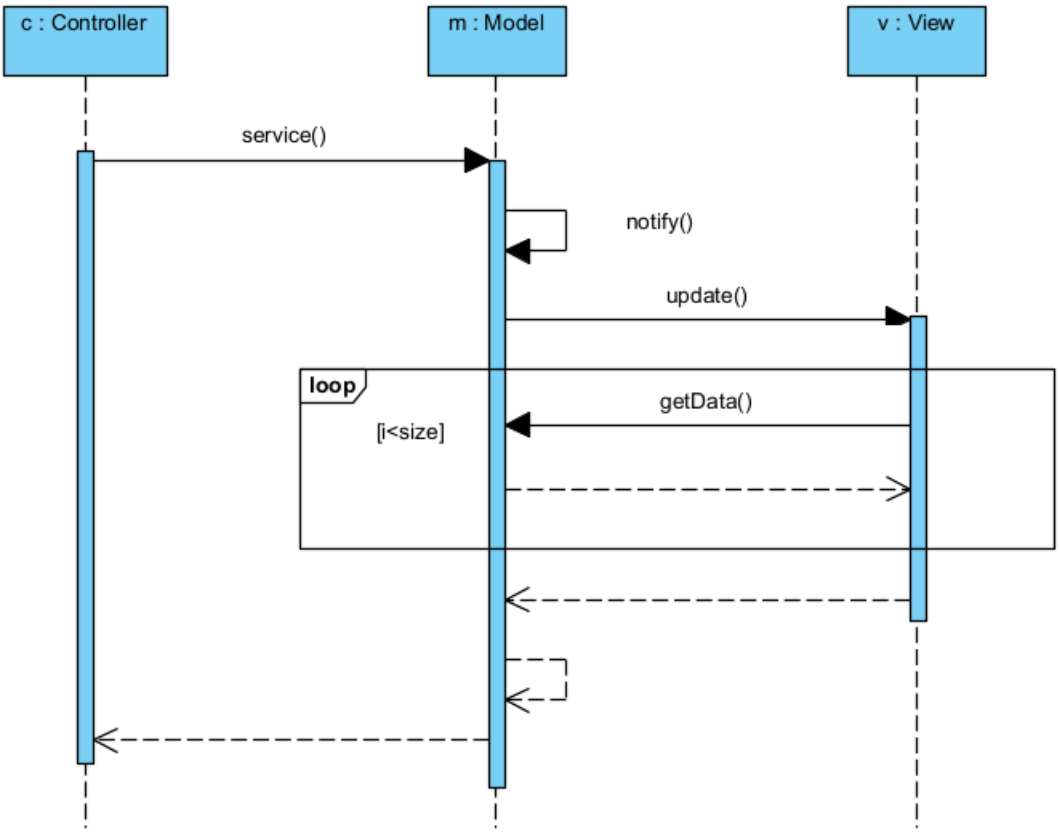
Nombre: _____ ***Matrícula:*** _____

Sección A

1. Escriba el mínimo código fuente en Java para las clases *Controller*, *Model* y *View* 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. *El dueño de un bar de snacks quiere un sistema que permita crear y manejar órdenes de menús. Un menú incluye muchos ítems. El sistema permite operaciones tales como agregar y eliminar ítems en un menú; y ajustar precios. El sistema debe, además, permitir al propietario mantener un historial de la lista de órdenes e ingresar una orden de un cliente y obtener la visualización de la factura*.
   a. Dibuje un **diagrama de clases** 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 de la siguiente manera: 10% por su selección de clases, 10% por su selección de tipos de relaciones y 4% por sus cardinalidades). **[24%]**

3. Escriba un diagrama de secuencia UML para la operación *calculateFee*. **[16%]**

```java
package exam2023_2;

import java.math.BigDecimal;

public class GalapagosEntryFeeCalculator{

    public BigDecimal calculateFee(Flight flight){
        int i=0;
        BigDecimal totalCustomTax = new BigDecimal(0);
        List<Traveller> travellers = flight.getTravellers();
        while(i++<travellers.size()) {
            Traveller traveller = travellers.get(i);
            if(traveller.isResident())
                totalCustomTax.add(getResidentFee());
            else if (!traveller.isResident() && traveller.isAdult())
                totalCustomTax.add(getAdultForeignerFee());
            else
                totalCustomTax.add(getChildForeignerFee());
        }
        return totalCustomTax;
    }

    public BigDecimal getResidentFee(){
        return new BigDecimal(100);
    }

    public BigDecimal getChildForeignerFee(){
        return new BigDecimal(100);
    }

    public BigDecimal getAdultForeignerFee(){
        return new BigDecimal(300);
    }
}
```

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. Entregar el dinero
	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. **[10%]**

b. Describa **dos** escenarios de este caso de uso. **[06%]**

Sección B

5. Considere el código fuente a continuación.
   a. Identifique los principios SOLID que se están violando. **[04%]**
   b. Para cada principio violado, argumente su respuesta. **[08%]**
   c. Corrija el código de tal manera que ya no se lo viole. Si lo considera necesario, usted puede crear interfaces, clases o nombres de métodos. Puede utilizar diagramas de clase. **[16%]**

```java
package solid2023_2;

public interface Customizable {
    public void customizeKeyboard();
    public void customizeStorage();
}
```

```java
public class Computer {
    // TODO
}
```

```java
package solid2023_2;

public class ComputerDirector {

    private ComputerBuilder builder;

    public ComputerDirector(DellInspironBuilder dellInspironBuilder){
        builder = dellInspironBuilder;
    }

    public ComputerDirector(HPSpectreBuilder hpSpectreBuilder) {
        builder = hpSpectreBuilder;
    }

    public void buildComputer() {
        builder.buildScreen();
        builder.buildCPU();
        builder.customizeStorage();
        if (builder instanceof DellInspironBuilder)
            builder.customizeKeyboard();
    }
}
```

```java
public abstract class ComputerBuilder implements Customizable {
    protected Computer computer;

    public Computer getComputer() {
        return computer;
    }

    public void reset() {
        computer = new Computer();
    }

    public abstract void buildScreen();
    public abstract void buildCPU();
}
```

```java
public class UnsupportedFeatureException extends RuntimeException {

    private static final long serialVersionUID = 1L;

}
```

```java
public class DellInspironBuilder extends ComputerBuilder {

    @Override
    public void buildScreen() {
        // TODO Auto-generated method stub
    }

    @Override
    public void buildCPU() {
        // TODO Auto-generated method stub
    }

    @Override
    public void customizeKeyboard() {
        // TODO Auto-generated method stub
    }

    @Override
    public void customizeStorage() {
        // TODO Auto-generated method stub
    }
}
```

```java
public class HPSpectreBuilder extends ComputerBuilder {

    @Override
    public void buildScreen() {
        // TODO Auto-generated method stub
    }
    @Override
    public void buildCPU() {
        // TODO Auto-generated method stub
    }

    @Override
    public void customizeKeyboard() {
        throw new UnsupportedFeatureException();
    }

    @Override
    public void customizeStorage() {
        // TODO Auto-generated method stub
    }
}
```