Trabajo Práctico

Diagrama de Interacción

Lista de Conceptos Tratados:
Clase; Objeto; Rol; Interacción o Colaboración; Mensaje; Secuencia de Mensajes; Línea de vida.

Ejercicio 1

Considere las siguientes afirmaciones sobre los diagramas de interacción:

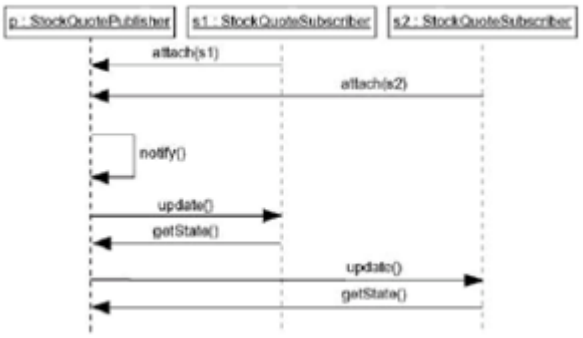
a) Los diagramas de interacción se usan para modelar el aspecto [dinámico] / [estático] de las colaboraciones representando sociedades de objetos jugando roles específicos, trabajando en conjunto para llevar a cabo algún comportamiento que es mayor que la suma de sus elementos.

b) Se puede modelar cada interacción de dos formas:
   1) enfatizando en [la comparación entre los tiempos de vida de los objetos] / [el ordenamiento temporal de los mensajes enviados entre objetos]
   2) enfatizando en [los atributos y las operaciones de los objetos] / [la organización estructural de los objetos]

- **Tache la(s) opción(es) entre corchetes que no corresponda(n) en cada una de ellas.**

Ejercicio 2

Considere el siguiente diagrama de secuencia



Nombre cada uno de los elementos de notación o sintaxis que están presentes en el diagrama.

Ejercicio 3

- **Construya diagramas de secuencia para modelar cada uno de los siguientes comportamientos:**

a) Un objeto de clase *ClaseA* recibe como punto de entrada un mensaje *mensaje1()* y si la condición *cond* se satisface, envía un mensaje *mensaje2()* a un objeto de clase *Clase*B, y en caso contrario, envía un mensaje *mensaje3()* a un objeto de clase *Clase*C.

b) Un objeto de clase *Clase*A, al recibir el mensaje *mensaje1()*, crea una instancia de clase *Clase*B, y se lo envía a un objeto de clase *Clase*C, el cual envía el mensaje *mensaje3()* a la instancia recién creada. Este objeto (la instancia de la clase ClaseB), retorna un valor *valordeRetorno.* Al recibir dicho valor el objeto de clase *ClaseC* envía al objeto de clase *Clase*B un mensaje *destroy()* con el cual se elimina este último objeto.

c) Un objeto de clase *Clase*A, al recibir el mensaje *mensaje1()*, envía *m* veces el mensaje *mensaje2()* a una instancia de clase *Clase*B, el cual a su vez, por cada mensaje recibido, envía un mensaje *mensaje3()* a un objeto de clase *ClaseC* y otro mensaje *mensaje4()*, a uno de clase *Clase*D.

d) Un objeto de clase *ClaseA* recibe como punto de entrada un mensaje *mensaje1()*, y envía a cada objeto de una colección de objetos de clase *ClaseB* el mensaje *mensaje2()*.

Ejercicio 4 (Enunciado 6 TP Base)

A partir del siguiente código realice el diagrama de secuencia correspondiente al momento en que se recibe un mensaje donde se crea un nuevo Episodio *episodio3* con los parámetros *3*, *High Sparrow*, *temporada5*. Tenga en cuenta lo siguiente:

- El método keySet():Set retorna un Set con las claves contenidas en la instancia de la clase Map.
- El método get(key:Object):Object retorna el objeto correspondiente a la clave pasada como parámetro.
- El método iterator():Iterator retorna un Iterator que nos permite recorrer el Set en cuestión.

```
public class Temporada {
   private Map<Suscriptor,Opcion> subscriptos;

   public void agregar(Episodio episodio) {
          this.notificar();
   }

   private void notificar() {
          Set s = this.subscriptos.keySet();
          Iterator it = s.iterator();
          while (it.hasNext()){
                 Usuario k = it.next();
                 Opcion o = this.subscriptos.get(k);
                 Frecuencia f = o.getFrecuencia();
                 if (f == Frecuencia.enElMomento) {
                        TipoEnvio t = o.getTipoEnvio();
                        if (t == TipoEnvio.sms) {
                           k.enviarNotificacionSMS();
                        }
                        if (t == TipoEnvio.email) {
                           k.enviarNotificacionMail();
                        }
                        if (t == TipoEnvio.facebook) {
                           k.enviarNotificacionFB();
                        }
                 } else {
                    //se programa la notificacion futura
                 }
          }
}
```

```
public enum Frecuencia {
   enElMomento, cincoMinDespues, mediaHoraDespues;
}
```

```
public enum TipoEnvio {
          sms, email, facebook;
}
```

```
public class Opcion {
          private Frecuencia  frecuencia;
          private TipoEnvio  tipo;

          public Frecuencia getFrecuencia() {
                 return this.frecuencia;
          }

          public TipoEnvio getTipoEnvio() {
                 return this.tipo;
          }
}
```

```
public class Usuario {

          public void enviarNotificacionSMS()
          {
          //envia una notificacion via sms
          }

          public void enviarNotificacionMail() {
          //envia una notificacion via mail
          }

          public void enviarNotificacionFB() {
          //envia una notificacion via FB
          }
}
```

```
public class Episodio {
          private int numero;
          private String titulo;
          private Temporada temporada;

          public void Epidodio(int num, String
titulo, Temporada temp){
                 this.numero=num;
                 this.titulo=titulo;
                 this.temporada=temp;

this.temporada.agregar(this);
          }
}
```

Ejercicio 5 (Enunciado 8 TP Base)

Teniendo en cuenta el código fuente del ejercicio 2, realice el diagrama de secuencia para el caso que un objeto calendarioGoogle de tipo Calendar recibe el mensaje removeEvent con los parámetros “prefinal” y 27/6/18 de tipo Date. Incluya en su diagrama las clases de java.util intervinientes.

```java
import java.util.ArrayList;
import java.util.Date;
import java.util.Iterator;
import java.util.List;

public class Calendar {
	private String name;
	private ExternalCalendar driver;
	private List<Event> events;
	public Calendar(String name) {
		this.name=name;
		driver=null;
		events=new ArrayList<>();
	}

	public void setDriver(ExternalCalendar driver) {
		this.driver = driver;
	}

	public void addEvent(String title, Date date){
		new Event(title, date, this);
		driver.synchronize(this);
	}
	public boolean removeEvent(String title, Date date){
		Iterator<Event> iterator = events.iterator();
		boolean hasNext= iterator.hasNext();
		while (hasNext) {
			Event event = (Event) iterator.next();
			if(event.getTitle().equals(title)&&event.getDate().equals(date)){
				events.remove(event);
				driver.synchronize(this);
				return true;
			}
			hasNext= iterator.hasNext();
		}
		return false;
	}
	public List<Event> getEvents() {
		return events;
	}
}
```

```java
import java.util.ArrayList;
import java.util.Date;
import java.util.List;

public class CalendarRepository {
	private List<Calendar> calendars;

	public Calendar createLocalCalendar(String name){
		Calendar newCalendar=new Calendar(name);
		calendars.add(newCalendar);
		return newCalendar;
	}
	public List<Event> searchEventsOnDate(Date date){
		List<Event> ret=new ArrayList<>();
		for (Calendar calendar : calendars) {
			for(Event event : calendar.getEvents()){
				if(event.getDate().equals(date))
					ret.add(event);
			}
		}
		return ret;
	}
}
```

```java
import java.util.ArrayList;
import java.util.Date;
import java.util.List;

public class Event {
	private String title;
	private String place;
```

```
        private Date date;
        private List<String> participants;
        private Calendar calendar;
        public Event(String title, Date date, Calendar calendar) {
                this.title=title;
                this.date=date;
                this.calendar=calendar;
                participants=new ArrayList<>();
        }
        public void setPlace(String place) {
                this.place = place;
        }
        public void addParticipant(String participant){
                participants.add(participant);
        }
        public String getPlace() {
                return place;
        }
        public List<String> getParticipants() {
                return participants;
        }
        public String getTitle() {
                return title;
        }
        public Date getDate() {
                return date;
        }
}
```

```
import java.util.Date;

public interface ExternalCalendar {
        public void removeEvent(String title, Date date);
        public boolean synchronize(Calendar calendar);
}
```

```
import java.util.Date;

public class GoogleMailCalendar implements ExternalCalendar {
        @Override
        public void removeEvent(String title, Date date) {
                // Some code using the Google Calendar API
        }
        @Override
        public boolean synchronize(Calendar calendar) {
                // Some code using the Google Calendar API
        }
}
```

Ejercicio 6 (Caso de estudio: Enunciado 7 TP Base)

Realice el diagrama de secuencia para el caso en que una instancia g del tipo GestorPagos recibe el mensaje pagarConPOSNet(juan, visaJuan, pulgarJuan).

```
import java.util.Calendar;
import java.util.Vector;


public class GestorPagos {
    private Vector<Pago> historialPago;
    private POSNet posnet;

    public void pagarConPOSNet(Usuario usuario,TarjetaDeCredito tarjetaSeleccionada, HuellaDigital
                    huella){
        posnet.establecerConexion();
        boolean postnetValido=posnet.testPOSNet();
        if(postnetValido){
```

```
            boolean tarjetaValida=
                    tarjetaSeleccionada.getSistema().validar(tarjetaSeleccionada);
            boolean huellaValida=usuario.validarHuella(huella);
            if(tarjetaValida && huellaValida){
                Pago nuevoPago=new Pago();
                nuevoPago.setDatosComercio(posnet.getDatosComercio());
                nuevoPago.setFechaPago(Calendar.getInstance().getTime());
                nuevoPago.setMonto(posnet.getMonto());
                nuevoPago.setTarjeta(tarjetaSeleccionada);
                historialPago.addElement(nuevoPago);
            }
        }
    }
}
```

```
import java.util.Date;


public class Pago {
    private long monto;
    private Date fechaPago;
    private String datosComercio;
    private TarjetaDeCredito tarjeta;
    public Pago(long monto, Date fechaPago, String datosComercio, TarjetaDeCredito tarjeta) {
        super();
        this.monto = monto;
        this.fechaPago = fechaPago;
        this.datosComercio = datosComercio;
        this.tarjeta=tarjeta;
    }
    public Pago() {

    }
    public void setDatosComercio(String datosComercio) {
        this.datosComercio = datosComercio;
    }
    public void setFechaPago(Date fechaPago) {
        this.fechaPago = fechaPago;
    }
    public void setMonto(long monto) {
        this.monto = monto;
    }
    public void setTarjeta(TarjetaDeCredito tarjeta) {
        this.tarjeta = tarjeta;
    }
}
```

```
public class HuellaDigital {
    private Object huellaRaw;


    @Override
    public boolean equals(Object obj) {
        return huellaRaw.equals(obj);
    }
}
```

```
public interface POSNet {
    public void establecerConexion();
    public long getMonto();
    public String getDatosComercio();
    public boolean testPOSNet();
}
```

```
public class SistemaVISA implements SistemaTarjeta{
```

```java
public interface SistemaTarjeta {
    public boolean validar(TarjetaDeCredito tarjeta);
}
```

```java
    @Override
    public boolean validar(TarjetaDeCredito tarjeta) {
        ...
    }
}
```

```java
import java.util.Date;


public class TarjetaDeCredito {
    private int numero;
    private Date vencimiento;
    private int codigoSeguridad;
    private SistemaTarjeta sistema;

    public SistemaTarjeta getSistema() {
        return sistema;
    }


}
```

```java
import java.util.Vector;


public class Usuario {
    private String nombre;
    private int dni;
    private Vector<TarjetaDeCredito> tarjetas;
    private HuellaDigital huella;

    public void addTarjeta(TarjetaDeCredito nuevaTarjeta){
        tarjetas.add(nuevaTarjeta);
    }
    public boolean validarHuella(HuellaDigital huella2){
        return huella.equals(huella2);
    }
}
```

Ejercicio 7 Dado el siguiente diagrama de secuencia "incompleto", infiera cuáles son las clases involucradas y qué métodos contiene cada clase. Solo tenga en cuenta las clases y métodos indicados en el diagrama. Para ello utilice el siguiente template:

- Clase <nombreClase>
- Métodos clase <nombreClase>: <metodo1>, <metodo2>, …, <metodoN>
- En cada método indicar tipos de parámetros y retorno cuando corresponda.

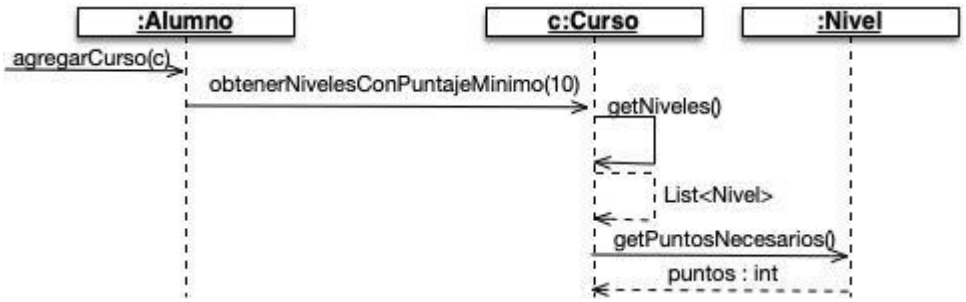
Ejemplo:

Clase A

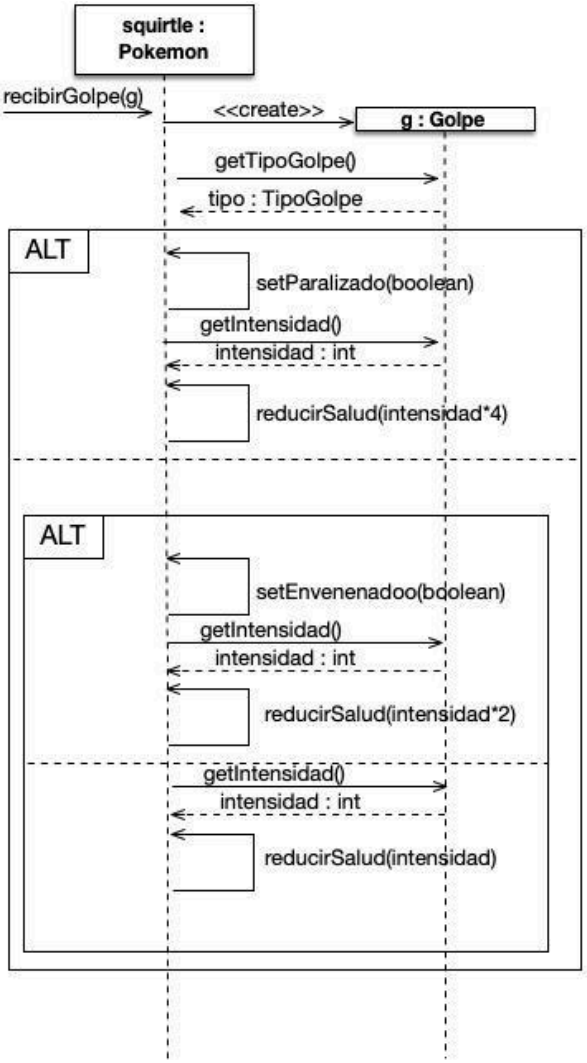
Métodos clase A: int foo1(), foo2(int, int), foo3(String)

Clase B

Métodos clase B: bar1(Object), boolean bar2(Biblioteca)

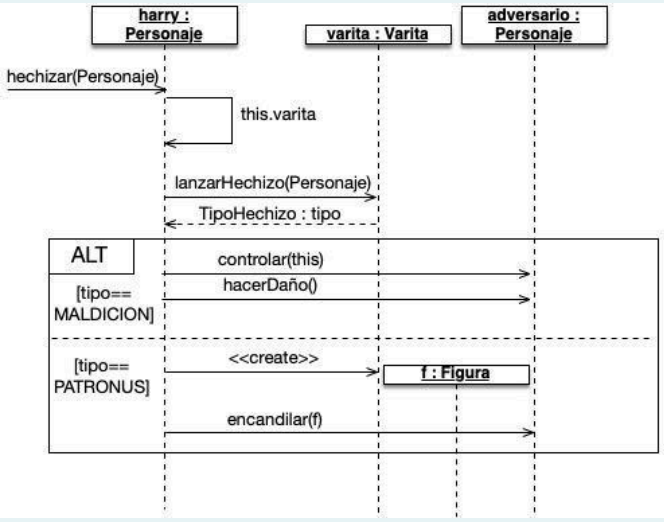


Ejercicio 8 Se desea modelar el diagrama de secuencia para el caso que un objeto squirtle de tipo Pokemon recibe el mensaje recibirGolpe con un golpe g como parámetro. Para ello se cuenta con un fragmento de código fuente de la clase Pokemon el cual es correcto. Teniendo en cuenta el código fuente, indique los errores en el diagrama.



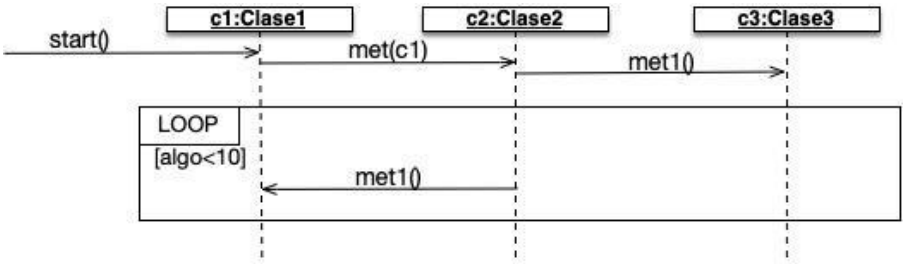

```
CLASE Pokemon
public void recibirGolpe(Golpe g){
    TipoGolpe tipo=g.getTipoGolpe();
    if(tipo==TipoGolpe.PARALIZANTE){
        this.setParalizado(true);
        this.reducirSalud(g.getIntensidad()*4);
    }else{
        if(tipo==TipoGolpe.EVENENANTE){
            this.setEnvenenado(true);
            this.reducirSalud(g.getIntensidad()*2);
        }else{
            //esNormal
            this.reducirSalud(g.getIntensidad());
        }
    }

}
```

Ejercicio 9 Se desea modelar el diagrama de secuencia para el caso que un objeto harry de tipo Personaje recibe el mensaje hechizar con un Personaje adversario como parámetro. Para ello se cuenta con un fragmento de código fuente de la clase Personaje el cual es correcto. Teniendo en cuenta el código fuente, indique los errores en el diagrama.




```
Clase Personaje
private Varita varita;
public void hechizar(Personaje adversario){
    TipoHechizo tipo=this.varita.lanzarHechizo(adversario);
    if(tipo==TipoHechizo.MALDICION){
        adversario.controlar(this);
        adversario.hacerDaño();
    }
    if(tipo==TipoHechizo.PATRONUS){
        adversario.encandilar(new Figura("Ciervo"));
    }
}
```

Ejercicio 10 Dado el siguiente diagrama de secuencia, indique cual es el código fuente correcto:



Opción 1

```
public class Clase1 {
   private Clase2 class2;
   private int algo;
   public void start(){
      class2.met(this);
      while(algo<10){
         class2.met1();
         algo++;
      }
   }
   public void met1(){
      //Hace algo
   }
}
```

```
public class Clase2 {
   private Clase3 class3;

   void met1() {
      //Hace algo
   }

   void met(Clase1 aThis) {
      class3.met1();
   }

}
```

```
public class Clase3 {

   void met1() {
      //Hace algo
   }

}
```

Opción 2

```
public class Clase1 {
   private Clase2 class2;
   private int algo;
   public void start(){
      class2.met(this);
      if(algo<10){
         class2.met1();
      }
   }
   public void met1(){
      //Hace algo
   }
}
```

```
public class Clase2 {
   private Clase3 class3;
   void met(Clase1 aThis) {
      class3.met1();
   }

   public void met1(){
      //Hace algo
   }

}
```

```
public class Clase3 {

   void met1() {
      //Hace algo
   }

}
```

Opción 3

```
public class Clase1 {
   private Clase2 class2;

   public void start(){
      class2.met(this);
   }

   public void met1(){
      //Hace algo
   }
}
```

```
public class Clase2 {
   private Clase3 class3;
   private int algo;
   void met(Clase1 aThis) {
      class3.met1();
      while(algo<10){
         aThis.met1();
         algo++;
      }
   }

   public void met1(){
      //Hace algo
   }

}
```

```
public class Clase3 {

   void met1() {
      //Hace algo
   }

}
```

Opción 4

```
public class Clase1 {
   private Clase2 class2;

   public void start(){
      class2.met(this);
   }

   public void met1(){
      //Hace algo
   }
}
```

```
public class Clase2 {
   private Clase3 class3;
   private int algo;
   void met(Clase1 aThis) {
      class3.met1();
      if(algo<10){
         aThis.met1();
      }
   }

   public void met1(){
      //Hace algo
   }

}
```

```
public class Clase3 {

   void met1() {
      //Hace algo
   }

}
```

Bibliografía de Apoyo Sugerida

[1] – The Unified Modeling Language – Reference Manual. J. Rumbaugh, I. Jacobson and G. Booch. Addison Wesley Longman, Inc. 1999. ISBN 0-201-30998-X. Código de consulta en Biblioteca Central UNICEN: 001.642 R936-1.

Resumen de la notación de UML: Appendix B, páginas 519 a 530.

Interacciones y Diagramas de Interacción: Part 2 – Chapter 8 (Interaction View) páginas 85 a 91.

[2] – The Unified Modeling Language – User Guide. G. Booch, J. Rumbaugh and I. Jacobson. Addison Wesley Longman, Inc. 1999. ISBN 0-201-57168-4. Código de consulta en Biblioteca Central UNICEN: 001.642 B724-3.

Conceptos generales sobre diagramas de UML: Section 2 – Chapter 7, páginas 91 a 104. Resumen de la notación de UML: Appendix A, páginas 435 a 437.

Interacción: Section 4 – Chapter 15, páginas 205 a 217. Diagramas de Interacción: Section 4 – Chapter 18, páginas 243 a 256.