Trabajo práctico integrador

▼ Trabajo práctico integrador

TP Integrador - Paradigmas de Programación.pdf

▼ Objetos

1. ¿Qué es una clase?

   Una clase es un modelo que nos permite representar varios objetos, los cuales tienen estados y comportamientos.

2. ¿Qué hacemos al instanciar?

   Al instanciar una clase obtenemos un objeto en particular de la clase

3. ¿Qué es un objeto?

   Un objeto es una entidad que tiene dos características: estado y comportamiento

4. Definición de estado y comportamiento.

   Estado: representa los atributos del objeto, los cuales se almacenan en variables llamadas variables de instancia.

   Comportamiento: representa la capacidad que tiene un objeto de modificar el estado (atributos) mediante mensajes (métodos).

C1		C2		C3		C4	
m6	^self m8	m9	^self m2	m2	^2	m1	^self m9
m10	^9	m8	^self m7	m5	^self m1	m3	^5
m13	^self m1	m12	^super m13	m7	^super m10	m4	^super m6
						m11	^super m12

Donde C1 es padre de C2, C2 de C3, y C3 de C4. (C1>C2>C3>C4).
Evaluar que devuelven las siguientes líneas
c:= C4 new.
c m4.
c m1.
c m11.

9 | 2 | 2

La UTN de La Plata necesita un sistema de gestión para llevar un registro de los
alumnos, los docentes y las cátedras a las que están asociados.

9. Realizar un cartel que diga “Bienvenido al Sistema de Gestión UTN”.

```
MessageBox notify: "Bienvenido al Sistema de Gestion UT
```

10. Generar la clase persona (legajo, nombre, apellido, materias que cursa/dicta) y sus subclases alumno y docente (sueldo), junto con la clase cátedra (nombre, conjunto de alumnos, docente, comisión). Nota: Asumir que un docente puede dictar una única cátedra y que un alumno puede cursar varias asignaturas distintas.

```

```

▼ **Funcional**

1. Dada una lista de enteros, retornar el promedio de los mismos.

```
promedioLista:: [int]->int
promedioLista(X;Xs):= 0, if longLista(X;Xs)=0
                                := sumarElementos(X;Xs)

sumarElementos:: [num]->num
sumarElementos(X;Xs):= X+sumarElementos(Xs)
sumarElementos([]):=0

longLista::[num]->num
longLista(X;Xs):= 1+longLista(Xs)
longLista([]):=0
```

2. Dada una lista de listas, reemplazar todos los caracteres ‘i’ por la letra ‘y’.

```
reemplazarCaract:: [[char]]->[[char]]
reemplazarCaract(Xs;Xss):= Xs: "y": buscarCaract(X;Xs)
```

```
                                                        := reem

buscarCaract:: [char]->char
buscarCaract(X;Xs):= "i" if X="i"
                                          := buscarCaract(Xs)
```

3. Dada una lista de listas, retornar una lista que contenga los 3eros elementos de
   cada lista, siempre y cuando estos sean mayores a 5.

```
nuevaLista:: [[num]]->[num]
nuevaLista(Xs;Xss):= buscarElemento(X;Xs;1):nuevaLista(
nuevaLista([])=:[]

buscarElemento:: [num],num->num
buscarElemento(X;Xs,K):= X if X>5 && K<4
                                                  := buscarEl
buscarEelemento([], _):=[]
```

4. Dado una lista de listas, concatenar las listas en la que **todos** sus elementos sean pares.

```
concatenarPares:: [[num]]->[[num]]
concatenarPares(Xs;Xss):= esPar(X;Xs):concaterPares(Xss
concatenarPares([]):=[]

esPar:: [num]->[num]
esPar(X;Xs):= [] if X mod%2 != 0
                     := X:esPar(Xs) otherwise
esPar([]):=[]
```

▼ Lógico

▼ Base de conocimientos

```
catedra(1512,"Paradigmas de Programacion").%catedra(codigo, nombre)
catedra(1509,"Sistemas Operativos").
catedra(1511,"Sintaxis y Semántica del Lenguaje").
catedra(1513,"Ingeniería y Sociedad").
catedra(1510,"Análisis Matemático II").
catedra(1508,"Física II").
catedra(1514,"Análisis de Sistemas").

alumno(1,"Rodríguez","Emilio").%alumno(legajo,apellido,nombre)
alumno(2,"Ohara","Maximiliano").
alumno(3,"Herrera","Esteban").
alumno(4,"Iturriaga","Julia").
alumno(5,"Treval","Cecilia").
alumno(6,"Gimenez","Iara").
alumno(7,"Gonzalez","Sonia").
alumno(8,"Martinez","Roberto").

docente(1,"Acosta","Iara").%docente(legajo,apellido,nombre)
docente(2,"Luna","Darío").
docente(3,"Dorado","Luciano").
docente(4,"Salaberri","Tomás").
docente(5,"Bianchi","Stefania").
docente(6,"Draga","Tiara").
docente(7,"Yanini","Pablo").
docente(8,"Carrillo","María").

promociono(5,1509,10).%promociono(legajoAlumno,codigoCatedra,nota)
promociono(2,1512,6).
promociono(4,1512,9).
promociono(6,1508,10).
promociono(4,1511,7).
promociono(3,1513,8).
promociono(1,1511,10).
promociono(1,1514,8).
promociono(1,1508,9).
promociono(7,1509,6).
promociono(8,1512,7).
promociono(8,1511,7).

dicta(1,1514,"S24").%dicta(legajoDocente, codigoCatedra, comision)
dicta(2,1513,"S21").
dicta(7,1511,"S21").
dicta(6,1509,"S24").
dicta(3,1510,"S22").
dicta(4,1512,"S23").
dicta(5,1508,"S21").
dicta(8,1512,"S21").

cursa(1,1513,"S23").%cursa(legajoAlumno,codigoCatedra,comision)
cursa(2,1513,"S23").
cursa(1,1509,"S24").
cursa(8,1513,"S23").
cursa(3,1511,"S21").
cursa(3,1512,"S23").
cursa(5,1514,"S24").
cursa(6,1514,"S24").
cursa(8,1514,"S24").
cursa(4,1508,"S21").
cursa(7,1508,"S21").
cursa(2,1510,"S22").
cursa(1,1510,"S22").
cursa(1,1512,"S21").
cursa(3,1514,"S24").
```

1. Mostrar el legajo del alumno “Ohara”.

```
?-alumno(leg,"Ohara",_)
leg=2, yes
```

```
no
```

2. Mostrar el nombre del docente “Bianchi”.

```
?-docente(_, "Bianchi", nom)
yes, nom=Stefania
no
```

3. Mostrar el legajo de los alumnos que cursen en la S23

```
?-cursa(leg,_,"S23")
leg=1, leg=2, leg=8, leg=3
no
```

4. Generar una regla que dado un legajo retorne nombre y apellido de un docente

```
%la regla esta ya pertenece a la base de conocimient
```

5. Generar una regla que dado un nombre de cátedra, retorna el código de la misma junto con el apellido del docente a cargo.

```
%regla
docenteCatedra(nom, cod, ape):=
catedra(cod, nom), dicta(leg,cod,_), docente(leg, ap
%consulta
?-docenteCatedra("Sistemas Operativos", cod, ape)
cod=1509, ape="Draga"
```

6. Mostrar el código de la materia “Paradigmas de programación”.

```
?-catedra(cod, "Paradigmas de programacion")
cod=1509
no
```

7. Mostrar nombre y apellido del docente que dicta la asignatura “Sistemas Operativos”.

```
?-catedra(cod, "Sistemas Operativos"), dicta(leg, co
leg=6, ape="Draga", nom="Tiara"
no
```

8. Generar una regla que muestre nombre y apellido de los alumnos que cursen en una comisión dada

```
nombreyApePorComision(ape, nom, coms):=
cursa(leg, _, coms),
alumno(leg, ape, nom).
```

9. Generar una regla que dado un apellido de un alumno, retorne las materias que promocionó.

```
materias(ape, mats):= alumno(leg, ape, _), promocion
```

10. Dado un docente (legajo), retornar el nombre y apellido de los alumnos que cursan la materia que dicta.

```
?-dicta(legDoc, cod, _), cursa(legAlu, cod, _), alum
```

▼ Notas de errores comunes *practica*.

1. carga de instacias de clase simple en la coleccion

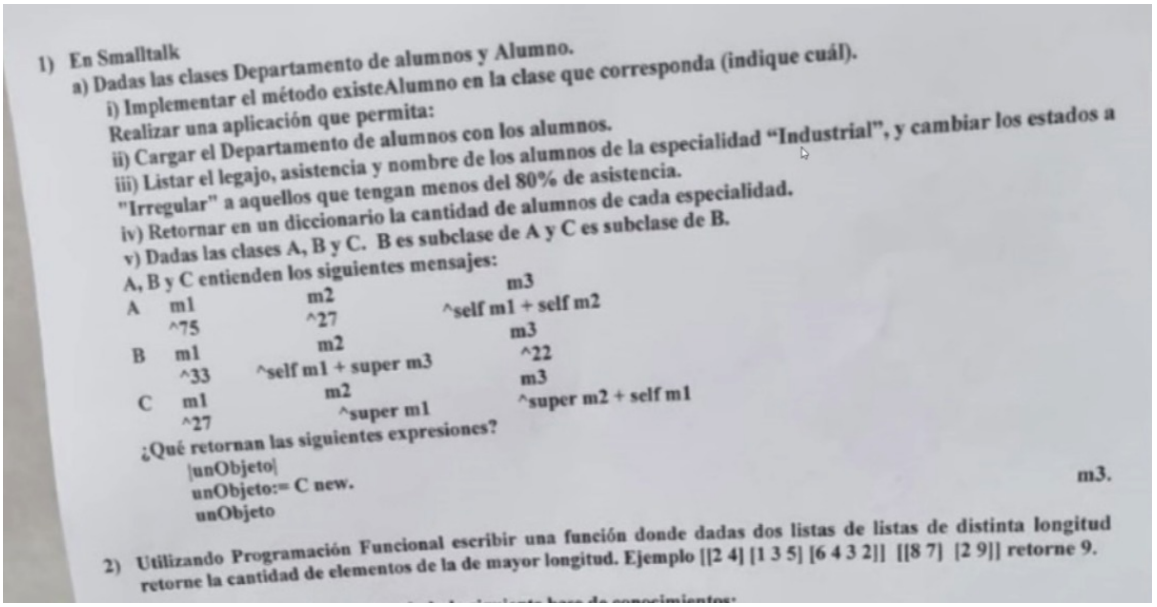
```
cli = Clinica CrearClinica:(Prompter prompt: “Ingres

seguir:= 1.
[seguir=1]whileTrue:[
    "cargar datos de instancias"
    pat:= Paciente CrearPaciente: nom, con:ape, con:
]

"error comun no poner mayuscula cuando se crea una c
emp:= empresa crear: "esta mal"

"dado que agregar es un metodo de instancia se hace
cli agregarPaciente: pat.
```

```
"cuando usamos una instancia el objeto receptor es u
"cuando usamos una clase el objeto receptor es una c
"Tal que tiene la siguiente forma"
OR: selector: arg.
"en el parcial no hace faltar hacer validaciones"
```

2. Diccionarios

```
d:= Dictonary new. "crea el diccionario vacio / {}"
col:= cli verTodos. "crea la coleccion de clientes /
odon:= col collect:[:elem | elem verOdontologo] "odo
odonSR= odon asSet.
odonSR do: [:elem | d at:elem put:(odon ocurrenciasO
to odonSR size do[:i |
    rec:= cli recuperarPac: i.
    d:= at:rec verOdontologo put:(odon ocurrencesOF:
    ]
```

3. Funcional (no toman dos listas de listas)



1) En Smalltalk
a) Dadas las clases Departamento de alumnos y Alumno.
i) Implementar el método existeAlumno en la clase que corresponda (indique cuál).
Realizar una aplicación que permita:
ii) Cargar el Departamento de alumnos con los alumnos.
iii) Listar el legajo, asistencia y nombre de los alumnos de la especialidad "Industrial", y cambiar los estados a "Irregular" a aquellos que tengan menos del 80% de asistencia.
iv) Retornar en un diccionario la cantidad de alumnos de cada especialidad.
v) Dadas las clases A, B y C. B es subclase de A y C es subclase de B.
A, B y C entienden los siguientes mensajes:

A	m1	m2	m3
	^75	^27	^self m1 + self m2
B	m1	m2	m3
	^33	^self m1 + super m3	^22
C	m1	m2	m3
	^27	^super m1	^super m2 + self m1

¿Qué retornan las siguientes expresiones?

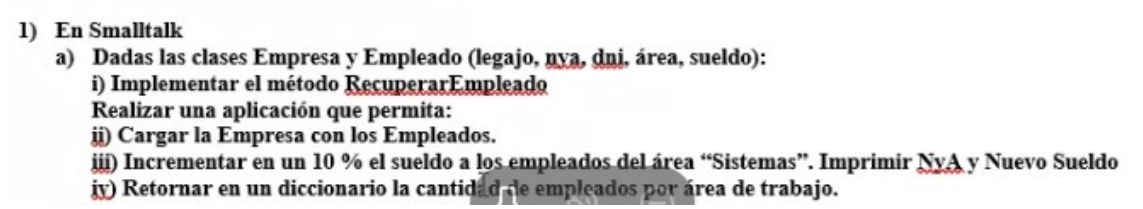
```
|unObjeto|
unObjeto:= C new.
unObjeto
```

m3.

2) Utilizando Programación Funcional escribir una función donde dadas dos listas de listas de distinta longitud retorne la cantidad de elementos de la de mayor longitud. Ejemplo [[2 4] [1 3 5] [6 4 3 2]] [[8 7] [2 9]] retorne 9.

```
%inicializarAlumnos
Departamento>>crearAlumnos: unAlumno
alumno: OrdererColecction new.

%existeAlumno
Departamento>>existeAlumnos: unAlumno
^alumnos include: unAlumno

%2
nom: Prompter prompt: 'Ingrese el nombre del departamento'
dto:= Departamento CrearDepartamento: nom.
seguir=1
[seguir=1]whileTrue:[
    nom:= Prompter prompt: 'Ingrese el nombre del alumno'.
    leg:= Prompter prompt: 'Ingrese el legajo del alumno'.
    asist:= (Prompter prompt: 'ingrese el porcentaje de as
    est:= Prompter prompt: 'Ingrese el estado del alumno'.
    esp:= Prompter prompt: 'Ingrese la especialidad del al
    alu:= Alumno CrearAlumno:nom con:leg con:asist con:est
    dto AgregarEmpleado: alu.
    messageBox notify:'Alumno cargado con exito'.
    seguir:= Prompter prompt 'Desea seguir cargando alumno
]

%3
col= dto verTodos.
col do:[:ele |
    transcript show: 'Legajo: ', ele verLegajo.
    transcript show: 'Asistencia: ', ele verAsist.
    transcript show: 'Nombre: ', ele verNom.
    ((ele verAsist)>80)ifTrue[
        est modAsist: 'irregular'.
    ]
]
%4
d:= Dictionary New.
colAlu:= col do: [:ele | ele verEspecialidad].
```

```
colAluSR:= colAlu asSet.
colAluSR:= do: [:ele | d at:ele put:(colAlu ocurrencesOf:
```



1) **En Smalltalk**
   a) **Dadas las clases Empresa y Empleado (legajo, nya, dni, área, sueldo):**
      i) Implementar el método RecuperarEmpleado
      Realizar una aplicación que permita:
      ii) Cargar la Empresa con los Empleados.
      iii) Incrementar en un 10 % el sueldo a los empleados del área "Sistemas". Imprimir NyA y Nuevo Sueldo
      iv) Retornar en un diccionario la cantidad de empleados por área de trabajo.

```
%1
Empresa>>recEmpleado: i
^empleado at: i

%2
nom:= Prompter prompt 'Ingrese el nombre de la empresa'.
empresa:= Empresa CrearEmpresa: nom.
seguir=1
[seguir=1]whileTrue:[
    nom:= (Prompter prompt: 'Ingrese el legajo del emplead
    nya:= Prompter prompt: 'Ingrese el nombre y apellido d
    dni:= (Prompter prompt: 'Ingrese el dni del empleado')
    area:= Prompter prompt: 'Ingrese el area del empleado'
    sueldo:= (Prompter prompt: 'Ingrese el legajo del empl
    emp:= Empleado CrearEmpleado:nom con:nya con:dni con:a
    empresa AgregarEmpleado: emp.
    seguir:= Prompter promtp: 'Desea seguir cargando emple
]

%3
col:= empresa verTodos.
col do:[:ele |
    ((ele verArea)='Sistemas')ifTrue:[
        sueldo:= ele verSueldo
        nuevoSueldo:= sueldo + sueldo*0.10.
        ele modSueldo: nuevoSueldo.
        transcript Show: 'Nombre y Apellido: ', ele verNyA
        transcript Show: 'Nuevo Sueldo: ', ele verSueldo.
    ]
```

```
]
%4
d:= Dictionary new.
colEmp:= col do:[:ele | ele verArea]
colEmpSR:= colEmp asSet.
colEmepSR:= do:[:ele | d at:ele put:(colEmp ocurrencesOf:
```