# *Programación orientada a objetos. Examen julio 2002.*

*Miércoles 31 de julio de 2002.*

1. Definan en forma breve y precisa qué es una **invariante de clase**. La violación de una precondición "es culpa" del cliente mientras que la violación de una poscondición "es culpa" del servidor. ¿De quién es la "culpa" cuando ocurre una violación de una invariante de clase? Justifiquen su respuesta.

   *5 puntos*

2. [Java] Programen en Java una clase `Pelicula` con atributos `nombre` y `precio`. El atributo nombre es una `string` de sólo lectura y el atributo precio es un `double` de lectura y escritura. Ningún atributo puede estar vacío, es decir, ninguno puede ser `null`, el `nombre` no puede contener "", ni el precio puede ser cero.

   [Smalltalk] Programen en Smalltalk una clase `Pelicula` con atributos `nombre` y `precio`. El atributo nombre es una `String` de sólo lectura y el atributo precio es un `Float` de lectura y escritura. Ningún atributo puede estar vacío, es decir, ninguno puede ser `nil`, el `nombre` no puede contener "", ni el precio puede ser cero.

   *10 puntos*

3. [Java] Sea en Java una clase `Cliente` definida así:

```
public class Cliente {

  private String m_nombre;

  public Cliente(String nombre) {
    m_nombre = nombre;
  }

  public String getNombre {
    return m_nombre;
  }
}
```

   Modifiquen la clase `Cliente` para su comportamiento incluya la posibilidad de alquilar y devolver películas y para que el estado permita conocer las películas alquiladas por el cliente.

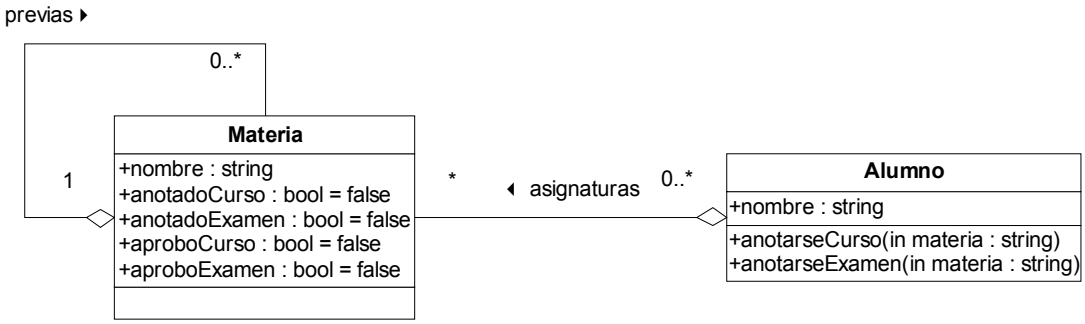
   [Smalltalk] Sea la clase `Cliente` en Smalltalk definida así:

```
Object subclass: #Cliente
  instanceVariableNames: 'nombre'
  classVariableNames: ''
  poolDictionaries: ''!

!Cliente class methods!

new: aString
  ^super new initialize: aString; yourself. !!

!Cliente methods!

initialize: aString
  nombre := aString.!

nombre
  ^nombre. !!
```

   Modifiquen la clase `Cliente` para su comportamiento incluya la posibilidad de alquilar y devolver películas y para que el estado permita conocer las películas alquiladas por el cliente.

   *10 puntos*

4. [Java] Consideren las clases `Alumno` y `Materia` para la inscripción a cursos y exámenes. Cada alumno conoce las materias que debe cursar durante toda la carrera.




La clase `Materia` tiene los siguientes atributos:

- `nombre`: es el nombre de la materia.
- `anotadoCurso`: inicialmente es `false`. Pasa a `true` cuando el alumno se inscribe al curso.
- `anotadoExamen`: inicialmente es `false`. Pasa a `true` cuando el alumno se inscribe al examen.
- `aproboCurso`: inicialmente es `false`. Pasa a `true` cuando el alumno aprueba el curso.
- `aproboExamen`: inicialmente es `false`. Pasa a `true` cuando el alumno aprueba el examen.

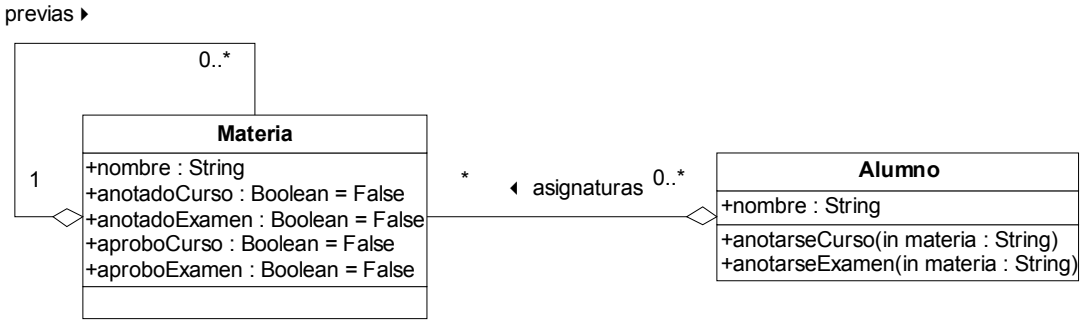
Usando precondiciones y poscondiciones implementen en la clase `Alumno` los métodos `anotarseCurso(String nombre)` y `anotarseExamen(String nombre)` teniendo en cuenta las siguientes consideraciones:

- Si el alumno no está anotado a un curso, no puede tener ni el curso ni el examen aprobado, ni puede estar anotado al examen.
- Si el alumno no tiene aprobado un curso, no puede estar anotado al examen, ni tener el examen aprobado.
- Si el alumno no está anotado al examen, no puede tener el examen aprobado.
- El alumno se puede inscribir a un curso si tiene aprobados los cursos de las materias previas.
- El alumno se puede inscribir a un examen si tiene aprobado el curso y los exámenes de las materias previas.

```
public class Alumno {

  private Set m_materias = new HashSet();

  public Set materias {
    return m_materias;
  }
}

public class Materia {

  private String m_nombre;
  private bool m_anotadoCurso = false;
  private bool m_anotadoExamen = false;
  private bool m_aproboCurso = false;
  private bool m_aproboExamen = false;
  private Set m_previas = new HashSet();

  public Materia(String nombre) {
    m_nombre = aString;
  }

  public bool getAnotadoCurso() {
    return m_anotadoCurso;
  }

  public void setAnotadoCurso(bool value) {
    m_anotadoCurso = value;
  }

  public bool getAnotadoExamen() {
    return m_anotadoExamen;
  }
```

```
  public void setAnotadoExamen(bool value) {
    m_anotadoExamen = value;
  }

  public bool getAproboCurso() {
    return m_aproboCurso;
  }

  public void setAproboCurso(bool value) {
    m_aproboCurso = value;
  }

  public bool getAproboExamen() {
    return m_AproboExamen;
  }

  public void setAproboExamen(bool value) {
    m_AproboExamen = value;
  }

  public String getNombre() {
    return m_nombre;
  }

  public void setNombre(String value) {
    nombre = value;
  }

  public Set getPrevias() {
    return m_previas;
  }
}
```

[Smalltalk] Consideren las clases `Alumno` y `Materia` para la inscripción a cursos y exámenes. Cada alumno conoce las materias que debe cursar durante toda la carrera.




La clase `Materia` tiene los siguientes atributos:

- `nombre`: es el nombre de la materia.
- `anotadoCurso`: inicialmente es `false`. Pasa a `true` cuando el alumno se inscribe al curso.
- `anotadoExamen`: inicialmente es `false`. Pasa a `true` cuando el alumno se inscribe al examen.
- `aproboCurso`: inicialmente es `false`. Pasa a `true` cuando el alumno aprueba el curso.
- `aproboExamen`: inicialmente es `false`. Pasa a `true` cuando el alumno aprueba el examen.

Usando precondiciones y poscondiciones implementen en la clase `Alumno` los métodos `anotarseCurso: nombre` y `anotarseExamen: nombre` teniendo en cuenta las siguientes consideraciones:

- Si el alumno no está anotado a un curso, no puede tener ni el curso ni el examen aprobado, ni puede estar anotado al examen.
- Si el alumno no tiene aprobado un curso, no puede estar anotado al examen, ni tener el examen aprobado.
- Si el alumno no está anotado al examen, no puede tener el examen aprobado.
- El alumno se puede inscribir a un curso si tiene aprobados los cursos de las materias previas.

- El alumno se puede inscribir a un examen si tiene aprobado el curso y los exámenes de las materias previas.

```
Object subclass: #Alumno
  instanceVariableNames:
    'asignaturas '
  classVariableNames: ''
  poolDictionaries: '' !

!Alumno class methods !

new
  ^super new initialize; yourself. !!

asignaturas
  ^asignaturas. !

initialize
  asignaturas := Dictionary new. !

Object subclass: #Asignatura
  instanceVariableNames:
    'nombre aproboCurso aproboExamen anotadoCurso anotadoExamen previas'
  classVariableNames: ''
  poolDictionaries: '' !


!Asignatura class methods!

new
  self invalidMessage. !

new: aString
    ^super new initialize: aString; yourself. !!

!Asignatura methods!

anotadoCurso
  ^anotadoCurso. !

anotadoCurso: aBoolean
  anotadoCurso := aBoolean. !

anotadoExamen
  ^anotadoExamen. !

anotadoExamen: aBoolean
  anotadoExamen := aBoolean. !

aproboCurso
  ^aproboCurso. !

aproboCurso: aBoolean
  aproboCurso := aBoolean. !

aproboExamen
  ^aproboExamen. !

aproboExamen: aBoolean
  aproboExamen := aBoolean. !

initialize: aString
  nombre := aString.
  anotadoCurso := false.
  anotadoExamen := false.
  aproboCurso := false.
  aproboExamen := false.
  previas := Set new. !

nombre
  ^nombre. !
```

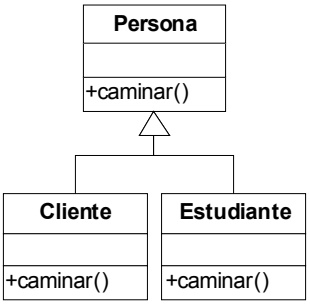
```
nombre: aString
  nombre := aString. !

previas
  ^previas. !!
```

*15 puntos*

5. Supongan un lenguaje de programación orientado a objetos en el que se declara el tipo de las variables.

   a. Expliquen en forma clara y precisa cómo funciona el encadenamiento estático y el encadenamiento dinámico en esos lenguajes.

   b. Describan los dos mecanismos usando el siguiente ejemplo.




*5 puntos*

6. [Java] Supongan sólo por un instante que Java soportara encadenamiento estático pero no así encadenamiento dinámico ¿Qué característica importante de Java se perdería?.

   [Smalltalk] Supongan sólo por un instante que Smalltalk soportara encadenamiento estático en lugar de encadenamiento dinámico ¿Qué característica importante de Smalltalk se perdería?.

*5 puntos*

7. El software es inherentemente complejo. La complejidad excede nuestra capacidad de compresión debido a nuestras limitaciones cognoscitivas. La complejidad es una propiedad esencial de los grandes sistemas de software. La tarea del equipo de desarrollo de software es dar la ilusión de simplicidad. Desde el punto de vista de la complejidad del software:

   a. ¿Qué sucede cuando no se construyen buenas abstracciones?

   b. ¿Qué ocurre si no es posible encapsular las abstracciones de software?

   c. ¿Qué sucede si no es posible ordenar o clasificar las abstracciones?

   d. ¿Qué sucede si los módulos no son cohesivos y están fuertemente acoplados?

*15 puntos*

8. Supongan que existe una veterinaria que desea registrar los perros de los clientes y, dado un perro, saber si tienen algún otro perro candidato para cruzarlo. Las normas para dicha selección son que los perros sean de distinto género e igual raza. Se pide que, tomando en cuenta el problema anterior, programen un ejemplo de una **mala práctica** de los siguientes conceptos: **abstracción**, **encapsulación**, **modularidad**, **jerarquía**.

   En caso que no exista una manera de programar un ejemplo en el lenguaje escogido, describan alguna manera de realizarlo en otro lenguaje o indiquen las características necesarias del lenguaje para ponerlo en práctica.

*15 puntos*

9. [Java] Programen en Java una clase `StringHandler` con los métodos necesarios para que se pueda hacer lo siguiente con un objeto de la clase `String`:

   a. Obtener la cadena al revés: `String reversed(String s).`

   b. Obtener la cantidad de consonantes en la cadena: `int countConsonants(String s).`

   c. Obtener la cantidad de ocurrencias de una letra dada en la cadena: `int countLetter(String s, Char c).`

[Smalltalk] Programen en Smalltalk los métodos necesarios para que un objeto de la clase String pueda responder lo siguiente:

a. Obtener la cadena al revés: `reversed`.

b. Obtener la cantidad de consonantes en la cadena: `countConsonants`.

c. Obtener la cantidad de ocurrencias de una letra dada en la cadena: `countLetter: aChar`.

*15 puntos*

10. Un objeto puede tener más de un tipo y varios objetos de diferentes clases pueden tener el mismo tipo. ¿Porqué?

*5 puntos*