Programación Orientada a Objetos **Septiembre 2005**

NOMBRE Y APELLIDOS:_____

TITULACIÓN: _____

1. a) (0'5 ptos) Explica la siguiente afirmación de B. Meyer "…siempre que se declare una rutina (en C++) hay que especificar su política de ligadura: virtual o no. Esta política va en contra del principio de abierto-cerrado …".

   b) (0'5 ptos) ¿Son equivalentes los conceptos de *encapsulación* y *ocultación de la información*?

2. (0'5 ptos) Escribe en Java la clase equivalente al siguiente código Eiffel manteniendo el modo de exportación de las propiedades.

```
class Posicion feature {Entidad}

   pos_x:REAL;
   pos_y:REAL;

   setX(x:REAL) is do
      pos_x:=x
   end

   setY(y:REAL) is do
      pos_y:=y
    end

end
```

3. Dada la implementación de la clase **`Pila`** a partir de la clase **`ArrayList`** que se muestra abajo.

```
public class Pila extends ArrayList {
    public Pila() {…}

    public void push(Object item) {
      add(item);
    }

    public Object pop() {
      int  len = size();

      return remove(len - 1); //remove devuelve el objeto que ha borrado
    }

    public Object tope() {
      int  len = size();
      if (len == 0) throw new EmptyStackException();

      return get(len - 1);
   }

    public boolean vacia() {
      return size() == 0;
    }
}
```

a) (0'5 ptos) Utiliza este ejemplo para explicar los inconvenientes derivados de la carencia de genericidad en Java (hasta la versión 1.4) a la hora de insertar y extraer los elementos de la pila.

b) (0'5 ptos) Supongamos que un programador va a utilizar la clase **`Pila`** para llevar la cuenta de las personas a las que se les va prestando un libro. Utiliza el siguiente código para explicar las diferencias entre Eiffel y Java, que se derivan del hecho de disponer o no de genericidad.

	Eiffel	Java
(1)	`loTiene: PILA[Persona];`	`Pila loTiene;`
(2)	`P: Persona; quienAhora:String`	`Persona p; String quienAhora;`
(3)	`!!loTiene; !!p.make(…);`	`loTiene = new Pila (); p = new Persona(…);`
(4)	`loTiene.push (p);`	`loTiene.push (p);`
(5)	`loTiene.push("Olivia");`	`loTiene.push("Olivia");`
(6)	`quienAhora:= loTiene.tope();`	`quienAhora = loTiene.tope();`

c) (0'5 ptos) Sea el siguiente código el encargado de imprimir el nombre de la persona que tiene el libro en este momento (`getName` es un método de la clase `Persona`). ¿Es correcto? En el caso de que no lo sea indica la manera de solucionarlo.

```
if ((loTiene.tope()) instanceof Persona)
                        System.out.println(loTiene.pop().getName());
```

d) (0'5 ptos) Indica si la excepción **`EmptyStackException`** debería ser comprobada o no comprobada. Justifica la respuesta.

e) (0'5 ptos) Escribe el método **`tope`**() en Eiffel siguiendo los principios del ***Diseño por Contrato***.

f) (0'5 ptos) Sea el método **`add(int index, Object element)`** un método de la clase **`ArrayList`** que inserta un elemento (`element`) en la posición indicada (`index`). ¿Sería legal el siguiente código? En caso afirmativo ¿es posible ocultar en Java un método al heredar?

```
Pila pila = new Pila();
Persona p = new Persona (…);
...
pila.add(8, p);
```

g) (0'5 ptos) ¿Cómo escribirías en Java la clase **`Pila`** que hereda de **`ArrayList`** para que no sea posible aplicar métodos de **`ArrayList`** sobre instancias de **`Pila`**? Escribe el método **`tope`** de la clase **`Pila`** de acuerdo con la solución propuesta.

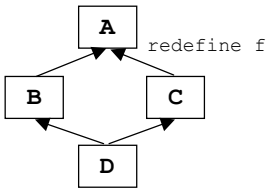
h) (0'5 ptos) En el caso de Eiffel, si **`PILA[G]`** hereda de **`ARRAY[G]`**, ¿es posible ocultar los métodos heredados de **`ARRAY`**?, ¿es posible aplicar métodos de **`ARRAY`** sobre instancias de **`PILA`**? ¿Cómo?

i) (0'5 ptos) ¿Cuál es la solución en C++ para la herencia de implementación? En ese caso, si **`PILA`** hereda de **`ARRAY`**, ¿es posible evitar aplicar métodos de **`ARRAY`** sobre instancias de **`PILA`**? ¿Por qué?

4. (0.5 ptos) Supuesto Java, sea una clase **`ColaImpresion`** la encargada de gestionar los trabajos que van a imprimirse en una impresora predeterminada. Esta clase incluye una **`LinkedList`** en la que se registran los objetos que se van a imprimir (documentos de texto, fotografías, gráficos, …). Dicha **`LinkedList`** es gestionada como una cola, de manera que mientras haya elementos en la cola de impresión, se extrae el primer objeto almacenado y se aplica el método **`print`** sobre dicho objeto. ¿Cuál debería ser el tipo dinámico de los objetos almacenables en la cola? ¿Hay varias soluciones? ¿Cuál sería la más apropiada?

5. (0'5 ptos) Se va a hacer una estadística de las horas semanales dedicadas por una cadena de TV a los programas con contenido del corazón. Para ello, sea `programacion:LIST[Programa]` una variable que almacena los programas de la semana. Éstos pueden ser de distintos tipos (informativos, deporte, corazón, …) y todos disponen de una función `duracion` que devuelve los minutos que abarca el programa. Implementar una rutina en Eiffel **`tiempoRosa(programacion:LIST[Programa]):REAL`** que devuelva el porcentaje de los minutos semanales destinados a los programas del corazón. (Nota: suponga que cada día se retransmite durante 24 horas ininterrumpidas)

6. Sea `A` una clase que contiene un atributo `at` y una rutina `f`.



a) (0'5 ptos) Determina si la jerarquía Eiffel de la figura es correcta. En el caso de que no lo sea indica la manera de solucionar el conflicto.

b) (0'5 ptos) Escribe el esqueleto de las clases de la jerarquía C++ equivalente a la jerarquía Eiffel del apartado anterior.

7. De acuerdo con la implementación en Eiffel de los ***iteradores*** (clase `LINEAL_ITERATOR`) vista en clase responda a las siguientes cuestiones:

a) (0'5 ptos) De acuerdo a la clasificación de iteradores internos y externos ¿a qué tipo se corresponde esta implementación? Justifica la respuesta.

b) (0'5 ptos) ¿Es esta implementación un ejemplo de la aplicación del patrón de diseño del *Método Plantilla*? Justifica la respuesta.

c) (0'5 ptos) ¿Es correcta la implementación del método `forEach` (ejecuta una misma acción sobre todos los elementos de una colección lineal) de la clase LINEAL_ITERATOR que se da a continuación? Justifica la respuesta:

```
forEach is do
      from coleccion.start
      until coleccion.after
      loop
            coleccion.item.action
            coleccion.forth
      end
end;
```

d) (0'5 ptos) El profesor D. Justo Peláez se jubila así que ha decidido, como despedida, modificar al alza las notas de las actas de sus alumnos en distintas proporciones dependiendo del tipo del alumno. Para ello ha hecho una pequeña aplicación donde `alumnos: LIST [Alumno]` es el atributo que almacena todos los alumnos que tiene que calificar e `ITERADOR_NOTAS` el iterador que recorre la lista de todos los alumnos y aplica a cada uno el cambio en la nota. El código cliente de este iterador es el que se muestra a continuación. Se pide la implementación de la clase `ITERADOR_NOTAS`.

```
modificarActas is
local
   iter: ITERADOR_NOTAS
do
   !!iter.make(alumnos)
   iter.forEach
end
```