NOMBRE:.....(Gestión/Sistemas)

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**) Explica cómo contribuye la herencia a alcanzar los criterios de reutilización del software.

2. Dado el siguiente código Java:

`class A{`
`  ? int at;`
`…`
`}`
	`class B{`
	`  m1(A oa){`
	`  oa.at=100;`
	`  }`
	`…`
	`}`

   a) (**0'5 ptos**) ¿Cuál debe ser el nivel de visibilidad del atributo `at` y la relación entre las clases A y B para que el código anterior sea correcto? Indica todas las posibilidades.

   b) (**0'75 ptos**) Responde a la misma pregunta del apartado a) supuesto el código equivalente escrito en C++ y en Eiffel.

   c) (**0'75 ptos**) Sea la definición del método: `m2(){ this.at=100;}` que se encuentra en una clase distinta de `A`. ¿Qué relación debe tener la clase donde se encuentre `m2` con la clase A y cuál debe ser el nivel de visibilidad de `at` para que el código sea correcto?. Especifica todas las posibilidades en Java, C++ y Eiffel.

3. Una cola de prioridades describe una estructura de datos donde **los elementos que se almacenan tienen asociada una prioridad de tipo entero**. Los elementos se ordenan de mayor a menor prioridad y en el caso de tener la misma se colocan por orden de inserción. Las extracciones siempre se hacen por la cabeza de la cola. Las **únicas** operaciones disponibles para una cola de prioridades son: **`add`** (añadir un elemento a la cola); **`get`** (devuelve el elemento de la cabeza de la cola, sin extraerlo), **`remove`** (devuelve el elemento de la cabeza de la cola y lo elimina) y **`size`** (indica el número de elementos de la cola). Sea la clase Eiffel `ColaPrioridades` una implementación de dicha estructura de datos como sigue:

```
class ColaPrioridades[G] inherit List[G] feature
      add(elem:G) is
          local encontrado:BOOLEAN, index:Integer, actual :G
          do
             from index:=1, encontrado := false
             until encontrado=true OR index=size+1
             loop
                actual := get(index);
                if (actual.prioridad<elem.prioridad) then encontrado:=true
                else index:=index+1
             end
             add(elem,index)
      end
      get(elem:G):G is do
             Result:=getFirst;
      end
      remove():G is do
             Result:=removeFirst;
      end
      size():Integer is do
         Result:=size;
      end
end
```

   los métodos `get(int), getFirst, removeFirst, size, add` están definidos en la clase `List`

   a) (**0'5 ptos**) ¿Qué habría que añadir a la definición de la clase `ColaPrioridades` para que sea correcta y se ajuste a la especificación?

b) (**1 pto**) Implementar la **clase `ColaPrioridades` en Java** equivalente a la definición proporcionada en Eiffel comentando todas las diferencias que encuentres entre ambas soluciones. (Nota: la clase `LinkedList` de Java implementa todos los métodos definidos para la clase `List[G]`).

4. (**1 pto**) Sea el método **`boolean intentarTransmitir(LineaTfno l, String msg)`** el encargado de enviar un mensaje de texto por una línea de teléfono. Los requisitos para la correcta ejecución del método son que la línea de teléfono esté conectada y que el mensaje no sea nulo. En caso de que se cumplan estas dos cosas, se llamará al método `transmitir` de la clase `LineaTfno` que puede fallar por diversos motivos (caída de la línea, sobrecarga, etc.). El método `intentarTransmitir` reintentará el envió 50 veces, después de las cuales, si no ha habido éxito devolverá un valor `false` y en caso contrario, notificará que la transmisión se pudo realizar devolviendo un valor `true`. Implementa el método `intentarTansmitir` en Eiffel y en Java

5. (**0'5 ptos**) Sea `exam:LIST[Pregunta]` una variable que almacena las preguntas de un examen. Un examen puede estar formado por distintos tipos de preguntas (test, desarrollo, problemas). Implementar una rutina en Eiffel **`notaTest(exam:LIST[Pregunta):REAL`** que devuelva la nota de las preguntas tipo test sabiendo que se calcula como el número de preguntas acertadas menos el número de preguntas falladas dividido por tres. (Nota: en la clase `Pregunta` existe un método `esCorrecta` que devuelve `true` si la repuesta es correcta y `false` en caso contrario).

6. (**0'5 ptos**) Dadas las clases A, B y C, cuyas declaraciones expresadas en Eiffel aparecen abajo, indica **qué versión se ejecuta** para cada uno de los mensajes que aparecen a la derecha de las declaraciones de las clases. (Utiliza letras griegas para referirte a las distintas implementaciones de las rutinas implicadas).

`class A feature`
`  f is do ... end`
`end`
 	
`class B`
`  inherit A`
`  rename f as g`
`feature`
`  f is do ... end`
`end`
 	
`class C`
`  inherit B`
`  rename f as h`
`  inherit B`
`  redefine f`
`  select f`
`feature`
`  f is do ... end`
`end`
	REPRESENTACIÓN GRÁFICA DE LA JERARQUÍA	`Oa:A; ob:B; oc:C`
		`!!oa; !!ob; !!oc;`
		
		`oa.f`
		
		`oa:=ob`
		`ob.f`
		`oa.f`
		`oa.g`
		
		`ob:=oc`
		`ob.f`
		`ob.g`
		
		`oa:=oc`
		`oa.f`
		`oa.g`
		`oc.f`

7. (**1'5 pto**) Sea la jerarquía de herencia que modela la clase `Pegaso` como descendiente de las clases `Caballo` y `Pajaro`, ambas subclases de la clase `Animal`. La clase `Animal` contiene un atributo `edad`, **compartido** por todas sus subclases y un **método abstracto** `sonido` que harán efectivo las subclases para indicar el sonido de cada animal, es decir, en la clase `Caballo` la salida es "relinchar" y en la clase `Pajaro` es "piar". En el caso de la clase `Pegaso`, el sonido que emite es el mismo que el del caballo (un pegaso no pía). Especifica la ***jerarquía de herencia en C++ y en Eiffel***.

8. A partir de la definición de la clase **`LINEAL_ITERATOR[G]`** vista en clase

   a) (**0'5 ptos**) Implementar un iterador denominado **`ITERADOR_ASCENSO`** que se encargue de subir el sueldo a todos los empleados de una empresa. El conjunto de empleados se guarda en una colección lineal que se le pasará al iterador en el momento de la creación.

   b) (**0'5 ptos**) Explica la importancia de las clases parcialmente diferidas para conseguir ***código genérico***. Utiliza la implementación del método `forEach` de Eiffel para apoyar tu explicación.

9. (**1 pto**) Implementar el **método `forEach` para la clase `LinkedList`** de Java que recorre la colección y ejecuta sobre cada uno de los elementos la acción que se le pasa como parámetro en el momento de invocarlo. Especifica todo lo que sea necesario para utilizarlo para subir el sueldo a todos los empleados de una empresa.