PROGRAMACIÓN ORIENTADA A OBJETOS **13/02/2007**
NOMBRE:_____ **Titulación:**_____

1. Sea la clase `Superheroe` la raíz de una jerarquía que tiene como subclases `Superman` y `Spiderman`. Entre los personajes de la aplicación existe también el representado por la clase `Vengador`, cuya misión es atacar a los superhéroes. Para implementar esta capacidad de los vengadores, nos plantean dos opciones:

Opción A

```
class Vengador {
    …
   public void atacar (Superheroe sh){
        if (sh instanceof Superman)
              //modificar superman
        else if (sh instanceof Spiderman)
             //modificar Spiderman
    }
}
```

Opción B

```
class Vengador {
    …
   public void atacar (Superman s){
             //modificar superman
   }
   public void atacar (Spiderman s){
            //modificar Spiderman
   }
}
```

a) (**0'5 ptos**) ¿Son conceptos equivalentes el polimorfismo puro (*función polimórfica*) y el polimorfismo ad-hoc (*sobrecarga*)? Justifica la respuesta, utiliza (si es posible) el código de la tabla para ejemplificar la explicación.

b) (**0'5 ptos**) De acuerdo con los principios OO ¿Cuál de las dos opciones es la correcta para resolver el problema planteado? Justifica la respuesta.

c) (**0'5 ptos**) ¿La implementación de la Opción A va en contra del *Principio de Elección Única*? Justifica la respuesta.

2. Supongamos que los superhéroes no son inmortales y la clase `Superheroe` tiene un atributo `vidas: Integer` que almacena el número de vidas que le quedan para morir.

a) (**0'5 ptos**) Supuesta la implementación en Eiffel del problema planteado en el ejercicio 1 ¿Cuándo sería correcto el siguiente método?

```
class Vengador feature
    …
  matar (sh: Superheroe) is do
     sh.vidas := sh.vidas -1
  end

end
```

b) (**0'5 ptos**) ¿Y en el caso de C++? Indica todas las posibilidades.

c) (**1pto**) Supongamos que existe una clase `Comic` que mantiene la lista de los superhéroes y por tanto, queremos darle acceso, sólo a esta clase, a las características (atributos y métodos) de la clase `Superheroe`. Compara las soluciones de Eiffel y Java a este problema.

3. La clase **Java** `Buzon` representa un buzón de mensajes, es decir, almacena una lista de mensajes. Por su parte, un mensaje, representado por la clase `Mensaje`, se caracteriza por: el `remitente`, el `destinatario`, `asunto`, `cuerpo` (todos de tipo String) y un valor `boolean` que indica si el mensaje es o no urgente. En la aplicación que gestiona los buzones se quiere proporcionar la funcionalidad de búsqueda de mensajes atendiendo a distintos criterios: buscar los mensajes urgentes, buscar los mensajes enviados por "pepito", buscar los mensajes enviados a "juanito", etc.

a) (**0'5 ptos**) Definir de manera genérica el concepto de criterio de búsqueda aplicable a cualquier tipo de problema.

b) (**0'5 ptos**) Implementar la rutina `buscar` en la clase `Buzon` a la que se le pase como parámetro el criterio de búsqueda y devuelva la lista de mensajes que cumplen dicho criterio.

c) (**0'5 ptos**) Implementar el criterio de búsqueda para buscar todos los mensajes urgentes y la manera de invocar al método buscar de la clase buzón para obtener dichos mensajes.

d) (**0'5 ptos**) Explicar la diferencia entre la genericidad en Java hasta la versión 1.4 frente a la utilizada en la resolución de este ejercicio. Apoya tu explicación con algún ejemplo.

4. Supongamos que cada buzón tiene asociada una cuenta de correo que le permite el envío de los mensajes por correo electrónico. La clase `Cuenta` guardará información acerca del usuario, del servidor de correo entrante, etc. Así, la clase `Buzon` tendrá un método:

```
public void enviar(Mensaje m){
      cuentaCorreo.enviar(m);
}
```

a) (**0'5 ptos**) La precondición del método `enviar` de la clase `Cuenta` es que el destinatario del mensaje no sea la cadena vacía. ¿Cómo se controlaría en Java esta condición?

b) (**0'5 ptos**) El método `enviar` de la clase `Cuenta` debe conectarse al servidor de correo para hacer el envío y lanza una excepción `MailServerException` en el caso de que el servidor de correo no responsa ¿De qué tipo será esta excepción? Justifica la respuesta

c) (**0'5 ptos**) Modifica, si es necesario, el código del método `enviar` de la clase `Buzon` para que trate las situaciones de error de los apartados a y b.

d) (**0'5 ptos**) ¿Proporciona Java el soporte para la técnica del *Diseño por Contrato* tal y como se entiende en el lenguaje Eiffel?

5. Entre los elementos de un proyecto diferenciamos entre tareas y entregables. Un entregable se caracteriza por tener un coste de materiales y un tiempo de producción. Por su parte, una tarea es un elemento de proyecto que puede estar formada por otros elementos (tareas y/o entregables), de manera que su coste de materiales será la suma del coste de materiales de todos los elementos que la componen y el tiempo de producción de una tarea será la suma de los tiempos de todos los elementos que la componen:

a) (**0'5 ptos**) Representa gráficamente la jerarquía en la que se vea la relación entre las clases del problema y la ubicación y tipo de atributos y métodos.

b) (**0'5 ptos**) Implementa los métodos de la clase `Tarea`.

c) (**0'5 ptos**) Utiliza el patrón de diseño del *Método Plantilla* para implementar el método `getCosteEstimado` que devuelve el coste estimado de un elemento de proyecto. Para todos los elementos de proyecto, el coste estimado se calcula como el tiempo requerido multiplicado por el precio de la unidad de tiempo más la suma del coste de los materiales.

d) (**0'5 ptos**) Explica por qué el patrón de diseño del Método Plantilla contribuye a alcanzar los criterios de reutilización planteados en el tema 1.

6. (**0'5 ptos**) Establece gráficamente la relación entre las clases A, B y C y los métodos disponibles en cada clase, de manera que sea correcto el comportamiento asociado al siguiente código Eiffel (la letra griega que aparece a la derecha representa la versión que se ejecuta).

```
oa:A; ob:B; oc:C
!!oa, !!ob, !!oc

oa.f                {α}
oa.h                ERROR tc
ob.g                ERROR tc
ob.f                {β}
oc.f                {δ}
oc.g                {β}
oa := oc            LEGAL
ob := oc            LEGAL
oa.f                {α}
ob.f                {δ}
```