Nombre: _____
Titulación: _____

1. Sea `Properties` una clase Java que representa un conjunto de propiedades persistentes. Cada propiedad está formada por una clave y un valor, ambos de tipo `String`. Sea el método `public void store (OutputStream out)` el encargado de guardar las propiedades en el *stream* que se le pasa como parámetro.

   a) (0'5 ptos) Para que el método `store` se ejecute, `out` tiene que estar conectado a un stream de salida (referencia no nula) y todas las claves almacenadas tienen que ser de tipo `String`. En el caso de no cumplirse alguno de estos requisitos se lanzará una excepción. Justifica en cada caso si se tratará de una **excepción comprobada o no comprobada**.

   b) (0'5 ptos) El método para escribir en un stream lanza una excepción en el caso de que se intente escribir y el stream haya sido cerrado. ¿Se trata de una excepción comprobada o no comprobada? Suponga que `store` no captura esta excepción ¿qué tendrá que hacer para manejarla?

   c) (1 pto) Utiliza el método `store` para explicar la técnica del **Diseño por Contrato** y el **esquema a priori** como técnica de diseño para el manejo de los casos excepcionales.

2. Suponga la definición de la clase **`Cilindro`** a partir de la clase **`Circulo`** para reutilizar la implementación de los métodos que calculan el área y el perímetro de un círculo. El código Java de la clase sería el siguiente:

```
class Cilindro extends Circulo{
      private float altura;

      public float area(){
            return super.perimetro()*(altura+radio);
      }
      public float volumen(){
            return super.area()*altura;
}
```

   a) (0'5 ptos) Puesto que `radio` es un atributo de la clase `Circulo` ¿cuál será el nivel de visibilidad con el que se ha tenido que declarar para que la clase `Cilindro` tenga acceso a dicho atributo y se favorezca el Principio de Ocultación de la Información? ¿Y si el código fuese Eiffel en lugar de Java?

   b) (0'5 ptos) ¿Se trata de un ejemplo de ***herencia de implementación***? Justifica la respuesta.

   c) (0'5 ptos) El cálculo del perímetro no es una característica de la clase `Cilindro`. Por tanto, el método `perimetro` heredado no debe estar disponible para los clientes de la clase `Cilindro`. Modifica el código dado para ocultar el método `perimetro` de la clase `Circulo` en la clase `Cilindro`.

   d) (0'75 ptos) Escribe el código de la clase Eiffel equivalente a la definición de la clase Java `Cilindro` escrita en el apartado b).

   e) (0'5 ptos) Escribe el código de la clase `Cilindro` en C++.

f) (0'5 ptos) Explica la diferencia entre la solución Eiffel y C++ para ocultar el método `perimetro` heredado.

g) (0'75 ptos) Supongamos que una clase `Tanque`, que representa un tanque de agua, tiene un atributo `estructura` de tipo `Cilindro`. Explica cómo se puede especificar en Eiffel, C++ y Java, que las características (atributos y métodos) de la clase `Cilindro` sólo queremos que sean visibles para la clase `Tanque`.

3. a) (1 pto) Aplica el patrón de diseño del ***Método Plantilla*** de manera que el método `imprime` de la clase `Factura` comparta el comportamiento común de los métodos `imprimeASCII` e `imprimeHTML`, y no necesite ningún argumento para su ejecución. A la hora de imprimir una factura el cliente debe conocer qué tipo de factura quiere crear (ASCII o HTML) y a continuación llamará al método `imprime`. La ligadura dinámica se encargará de que se imprima la factura con el formato del tipo creado. Por ejemplo, `new FacturaASCII(compra).imprime();`

```
public class Factura{
      public static int ASCII = 0;
      public static int HTML = 1;
      private Compra compra;

      public Factura(Compra c){
            compra = c;
      }
      public String imprime(int formato){
            if (formato == HTML) return imprimeHTML();
            if (formato == ASCII) return imprimeASCII();
            …
      }
      public String imprimeASCII(){
            //imprime la cabeza de la factura
           String        result        =        "Factura        del        cliente
"+compra.getCliente().getNombre()+"\n";

            //imprime el cuerpo de la factura
           Iterator it = compra.iterator();
           while (it.hasNext())
                 result+=it.next().toString()+"\n";

            //imprime el pie de la factura
           result+="Importe de la compra = "+compra.getPrecio()+"\n";
           return result;
      }
      public String imprimeHTML(){
            //imprime la cabeza de la factura
           String result = "<H1>Factura del cliente<EM> "+
                           compra.getCliente().getNombre()+"</EM></H1><P>\n";

            //imprime el cuerpo de la factura
           Iterator it = compra.iterator();
           while (it.hasNext())
                 result+=it.next().toString()+"<BR>\n";

            //imprime el pie de la factura
           result+="<P>Importe de la compra = <EM>"+
                   compra.getPrecio()()+"</EM><P>\n";
           return result;
      }
}
```

b) (0'5 ptos) Explica el papel de las clases parcialmente diferidas para escribir código genérico.

4. Supongamos que nos encarga una aplicación para la gestión de los menús de un colegio. Los ingredientes que se utilizan para la preparación de los menús pueden ser básicos (pan, tomate, lechuga, …) o platos preparados como combinación de ingredientes básicos (por ejemplo, la salsa boloñesa) y/o otros platos preparados.

   a) (0'5 ptos) Especifica gráficamente una jerarquía basada en herencia múltiple para el diseño de los ingredientes del problema propuesto.
   b) (0'5 ptos) Cada ingrediente básico cuenta con un método `getCalorias` que devuelve las calorías que aporta dicho ingrediente. En el caso de un plato preparado, las calorías que aporta es la suma de las calorías de sus ingredientes. Implementa el método `getCalorías` en la clase que represente los platos preparados.

5. Sea la clase **`ReproductorMultimedia`** la encargada de reproducir cualquier elemento audiovisual (audio y/o video). Esta clase debe mantener una colección de los elementos que se van a reproducir, para ello debe proporcionar métodos para **`añadir`** y **`eliminar`** elementos audiovisuales a la colección. El método **`play`** reproducirá uno a uno los elementos almacenados en la colección. Por ejemplo, si se almacenan objetos de tipo `Animal` (Gato, Perro, …) se reproducirán los sonidos de los animales, si se almacenan objetos de tipo `Reloj` (Pulsera, Cuco, …) se reproducirán los sonidos de cada uno de los relojes, también se pueden almacenar canciones, películas, … en todos estos casos, los objetos almacenados en la colección deben disponer de un método `play` que reproducirá la imagen y/o sonido que corresponda.

   a) (0'5 ptos) Especifica el tipo de la colección y el método `play` de la clase `ReproductorMultimedia` en Java.

   b) (0'5 ptos) Especifica el tipo de la colección y el método `play` de la clase genérica `ReproductorMultimedia` en Eiffel.

6. (0'5 ptos) Encuentra la relación entre las clases A, B y C, y las adaptaciones necesarias en la definición de los métodos, para que la ejecución del código siguiente sea la que se muestra:

```
oa:A, ob:B, oc:C
!!oa, !!ob, !!oc
oa.f      α
ob.f      β
oc.f      δ
oc.h      θ
oa:=oc
oa.f      δ
oc.g      α
ob:=oc
ob.f      θ
oa:=ob    ERROR
ob:=oa    ERROR
```