NOMBRE: _____**Titulación:**_____

1. Dado el siguiente código Java:

```
class EditorGrafico {

      public void dibujarFigura(Figura f) {
            if (f.getTipo()==1)
                  dibujarRectangulo(f);
            else if (f.getTipo()==2)
                  dibujarCirculo(f);
      }

      public void dibujarCirculo(Circulo c) {....}

      public void dibujarRectangulo(Rectangulo r){..}
 }
```

```
class Figura {
      private int tipo;
      public Figura (int tipo){
            this.tipo = tipo;
      }
      public int getTipo(){
            return tipo;
      }
 }
 class Rectangulo extends Figura {
      public Rectangulo() {
            super(1);
      }
 }
 class Circulo extends Figura {
      public Circulo() {
            super(2);
      }

 }
```

EXPLICA si son verdaderas o falsas las siguientes afirmaciones:

a) (**0'5 ptos**) El código de la clase `EditorGrafico` favorece el ***principio de Abierto-Cerrado***.

b) (**0'5 ptos**) La herencia no es un mecanismo esencial para alcanzar los *criterios de reutilización del software*.

2. a) (**1 pto**) Suponiendo una clase `A` con un atributo `at1` y unos métodos `m1` y `m2`, que exporta `at1` a una clase `B`, el método `m1` es exportado a las clases `B` y `C` y el método `m2` es privado. Compara los mecanismos de Eiffel, C++ y Java para poder implementar la clase `A`.

b) (**0'5 ptos**) ¿Existe equivalente en Eiffel al modo de exportación `protected` C++? Razona la respuesta.

c) (**0'5 ptos**) ¿Qué sentido tiene incluir los tipos expandidos en Eiffel? Razona la respuesta.

3. Supuesto Java, se desea crear una clase genérica `SortedList<T>` que representa listas ordenadas de elementos siendo el criterio de ordenación variable; por ejemplo, una misma lista de objetos `Cuenta` podría estar ordenada de menor a mayor saldo o bien alfabéticamente por el nombre del titular. El criterio de comparación entre los elementos de la colección se fija en el momento de la creación de la lista pero es posible cambiarlo en tiempo de ejecución.

a) (**0'5 ptos**) Para implementar la clase `SortedList` podríamos heredar de alguna de las implementaciones de lista o utilizar una relación de clientela. Compara ambas soluciones y elige la que consideres más apropiada.

b) (**0'5 ptos**) Diseña una solución para implementar la clase `SortedList`. Puedes representar las clases implicadas gráficamente pero especificando la signatura completa de los métodos que sean necesarios.

c) (**0'5 ptos**) ¿Es necesario restringir la genericidad de la clase `SortedList`? En caso de ser necesario establece el tipo por el que debe restringirse.

d) (**0'5 ptos**) Escribe el código que implemente los mencionados criterios de comparación.

e) (**0'5 ptos**) Escribe el método `add` para la clase `SortedList` que añada el elemento que se pasa como parámetro a una instancia de la lista manteniendo el orden establecido. Nota: `List<T>` dispone de los métodos: `T get(int index)` que devuelve el elemento que se encuentra en la posición `index` y el método `add(int index, T elem)` que inserta el elemento `elem` en la posición `index` desplazando los elementos hacia la derecha.

f) (**0'5 ptos**) ¿Forma parte la genericidad del resto de lenguajes OO que hemos estudiado (Eiffel, C++ y C#)? ¿Por qué?

4. El movimiento de los *personajes animados* de un juego de ordenador viene caracterizado por los siguientes pasos: 1) determinar la dirección del movimiento según sus objetivos y 2) desplazarse una cantidad (que viene dada por el tipo de personaje) en dicha dirección. Por ejemplo, el `ComePiedras` busca la piedra más cercana y avanza 5 en su dirección y el `Goloso` busca el caramelo más cercano y avanza 2 en su dirección (nótese que `Caramelo` y `Piedra` serían *personajes inanimados*). El controlador de la animación mueve los personajes conforme al siguiente algoritmo, siendo `animar` un método exclusivo de los personaje animados:

```
public void animarJuego(){
      for (Personaje p : personajes)
                  if (p instanteof PersonajeAnimado) p.animar();
}
```

a) (**0'25 ptos**) ¿Es correcto el código del método `animarJuego`? En el caso de que no lo sea indica el modo de solucionarlo.

b) (**0'75 ptos**) Aplica el ***patrón de diseño del Método Plantilla*** en el diseño de una solución para implementar el algoritmo de movimiento de los personajes del juego.

5. Sea la clase `ControladorTanque` la clase encargada de interactuar con el software de la tarjeta hardware que abre y cierra la válvula para la entrada y salida de líquido de un tanque de agua. El software de la tarjeta lanza una excepción cuando encuentra algún error al abrir o cerrar la válvula (por ejemplo, si está atascada). La especificación parcial de esta clases sería:

```
class ControladorTanque feature
     valvula: TarjetaValvula;
     capacidad: REAL;
     volumenActual: REAL;

     quitarAgua (cantidad: REAL) is
     require
              no_vacio: volumenActual >0
              hay_suficiente: cantidad <= volumenActual
     do
       valvula.open(cantidad);
       volumenActual := volumenActual – cantidad;
     ensure
       volumenActual = old volumenActual - cantidad
     end
     …
```

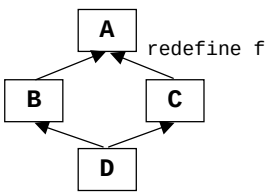
a) (**0'5 ptos**) Utiliza el método `quitarAgua` para explicar la ***técnica del Diseño por Contrato*** propuesta por Bertrand Meyer, especificando las obligaciones y beneficios de cada una de las partes implicadas en el contrato.

b) (**0'5 ptos**) ¿Qué ocurre en **Eiffel** si el método `open` lanza una excepción?

c) (**0'5 ptos**) Supuesta la implementación del problema en **Java** ¿De qué tipo sería la excepción que lanza el método `open` de la clase `TarjetaValvula`?

d) (**0'5 ptos**) Implementar en Java el método `quitarAgua` explicando TODAS las decisiones de diseño tomadas.

6. Sea `A` una clase que contiene un atributo `at` y una rutina efectiva `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**) Especifica la definición de la jerarquía C++ equivalente a la jerarquía Eiffel del apartado anterior.