NOMBRE: _____ **Titulación:** _____

ESTADO DE LAS PRÁCTICAS: _____

1. Las clases A, B, C y D se encuentran implementadas en dos paquetes tal y como se muestra en el siguiente código:

```
package paquete1;

public class A {
   ____________ int at1;

   ____________ int at2;

   ____________ int at3;

…
}



public class B{
   private A at;
   …
   public void reset(){
      at.at1 = 0;              //OK
      at.at2 = 0;              //ERROR
      at.at3 = 0;              //OK
}
```

```
package paquete2;

public class C extends A{
   …
   public void reset(){
      at1 = 0;                 //OK
      at2 = 0;                 //ERROR
      at3 = 0;                 //ERROR
      …
   }
}

public class D{
   private A at;
   …
   public void reset(){
      at.at1 = 0;              //ERROR
      at.at2 = 0;              //ERROR
      at.at3 = 0;              //ERROR
   }
}
```

a) (**0'75 ptos**) Especifica el nivel de visibilidad de los atributos de la clase `A` para que se cumplan las condiciones (`OK` y `ERROR`) mostradas en el código.

b) (**0'75 ptos**) Implementa la clase A en Eiffel con un comportamiento equivalente a la clase Java. Indica que implicaciones tienen en las clases B, C y D los niveles de visibilidad que has establecido en la clase A.

```
class A


…
end

class B
   …
   feature {ALL}
      reset is do
         at.at1 := 0;     --
         at.at2 := 0;     --
         at.at3 := 0;     --
      end
end
```

```
class C inherit A
   …
   feature{ALL}
      reset is do
         at1 := 0;        --
         at2 := 0;        --
         at3 := 0;        --
         …
      end
end



class D
   …
   feature {ALL}
      reset is do
         at.at1 := 0;     --
         at.at2 := 0;     --
         at.at3 := 0;     --
      end
end
```

2. Sea la clase `PizzaMaker` la encargada de gestionar la elaboración automatizada de las pizzas de un pedido, y el atributo `pedido` el que mantiene la colección de pizzas a elaborar:

```
public class PizzaMaker {
    …
    public void cocinarPizzas(){
      for (Pizza pizza : pedido)
           if (pizza instanceof PanPizza)
                pizza.cocinarEnPanCrujiente();
           else if (pizza instanceof Clasica)
                pizza.concinarEnHornoLeña();
      }
}
```

a) (**0'5 ptos**) ¿Por qué el código anterior viola el ***Principio de Abierto-Cerrado***? Justifica la respuesta.

b) (**0'5 ptos**) Explica cómo contribuye la ***ligadura dinámica*** a resolver el problema anterior y vuelve a implementar el método `cocinarPizza` de acuerdo a esta explicación. ¿Qué implicaciones tendría en el código de la clase `PizzaMaker` el que comenzara a elaborarse la `RollingPizza` tal y como se especifica en el método `cocinarEnHornoTradicional()`?.

3. La clase **Java** `Banco` mantiene una lista de empleados en el atributo `plantilla` y utiliza el método auxiliar `crearGrupo` para seleccionar el grupo de empleados que pasan el `Filtro` que se le pasa como parámetro. Esta selección resulta de utilidad para hacer determinadas acciones sólo sobre los empleados que han pasado el filtro establecido.

a) (**0'5 ptos**) Compara las declaraciones (1) y (2) para el atributo `plantilla` que se muestran en el cuadro siguiente, desde el punto de vista de la inserción y extracción de los empleados. Elige finalmente la que consideres más adecuada.

```
(1) List plantilla;
(2) List <Empleado> plantilla;
```

b) (**0'5 ptos**) Explica si es correcta o no la siguiente afirmación: "Las declaraciones (3) y (4) de la tabla son equivalentes puesto que en ambos casos restringen la lista a la clase Empleado".

```
(3) List <Empleado> plantilla;

(4) List <T extends Empleado> plantilla;
```

c) (**0'5 ptos**) ¿Es suficiente con declarar el atributo `plantilla` o es necesaria una rutina de creación explícita? Justifica la respuesta.

d) (**0'75 ptos**) Implementar el método `crearGrupo` y el `Filtro`.

e) (**0'75 ptos**) Utilizando el código del apartado anterior, implementar el método `felicitarCumple` en la clase `Banco` para felicitar a todos los empleados cuyo día y mes de la fecha de nacimiento coincida con el día actual.

NOTA: La clase `Date` se utiliza para representar las fechas. El constructor por defecto de la clase `Date` devuelve la fecha actual. Por su parte, los métodos de instancia `getDay` y `getMonth` de la clase Date devuelven un entero indicando el día y el mes respectivamente. La clase `Empleado` incluye un método `felicitar` y un método `getFechaNacimiento` que devuelve la fecha de nacimiento del empleado.

4. Un código de cuenta de un banco almacena la siguiente información: el banco al que pertenece la cuenta (4 dígitos), la sucursal (4 dígitos), el dígito de control (2 dígitos) y el número de cuenta propiamente dicho (10 dígitos). Sea la clase `CodigoCuenta` la implementación Java de los códigos de cuenta que dispone, entre otros, de los métodos `getBanco`, `getSucursal`, `getControl` y `getCCC`. El método `getCuenta` de la clase `Banco` recibe como parámetro un código de cuenta, que tiene que pertenecer al banco, y se conecta a la base de datos para recuperar la cuenta que se corresponde con dicho código utilizando el método auxiliar `recuperarCta`. El **esquema de la clase** es el siguiente:

```
public class Banco{
     public static final int BANCO_ID = 2043;
     …
     /** Devuelve la cuenta asociada al código de cuenta del banco que se le
      *  pasa como parámetro
      *  @pre:  El código de cuenta se corresponde con una cuenta del banco
      *  @post: Devuelve la cuenta
      */
     public Cuenta getCuenta(CodigoCuenta codigo){
           return recuperarCta(codigo.getCCC());
     }

     /** Se conecta a la base de datos y devuelve un objeto cuenta con la
     *   información recuperada o null si no existe
     *   @throws SQLException si ocurre algún problema en la consulta a la BD
     */
     private Cuenta recuperarCta(int codigo) throws SQLException{
           …
     }
}
```

a) (**1 pto**) Implementa correctamente el método `getCuenta` para que se ajuste al contrato que se ha definido. Especifica y justifica el tipo (comprobada o no comprobada) **de todas** las excepciones que manejes.

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

5. Supuesta la implementación de la clase `LinealIterator` vista en clase para el lenguaje Eiffel, y la implementación de la clase `Banco` que mantiene una lista de todos los empleados del banco:

```
class Banco feature
      plantilla: LIST[Empleado]
      aplicarIncentivo is do
      …
      end
end
```

a) (**0'5 ptos**) Implementa la clase `IteradorIncentivar` que aplica un incentivo a todos los directores del banco que han cumplido con sus objetivos. Suponed que en la clase `Director` (subclase de `Empleado`) existe un método `haCumplido`, que devuelve un valor de tipo `boolean` indicando si se han cumplido o no los objetivos y un método `incentivar`, que aplica el incentivo que le corresponde.

b) (**0'5 ptos**) Implementa el método `aplicarIncentivo` en la clase Banco que haga uso del iterador implementado en el apartado anterior.

6. Dado el siguiente código:

```
public class Empleado {
      private String nombre;
      public String getNombre(){
            return nombre;
      }
...
}
```

```
public class Directivo extends Empleado{
      //Añade V.P. = Vice-Presidente
      public String getNombre(){
           return super.getNombre() + ", V.P.";
      }
      ...
}
```

```
public class Investigador extends Empleado{
      //Añade Dr = Doctor
      public String getNombre(){
            return super.getNombre()+", Dr.";
      }
      ...
}
```

queremos tener una clase que represente objetos que son al mismo tiempo `Directivo` e `Investigador`.

a) (**0'5 ptos**) Vuelve a implementar el método `getNombre` aplicando el patrón de diseño del ***Método Plantilla***. Debes hacer todos los cambios que consideres necesarios.

b) (**0'5 ptos**) Explica cómo contribuyen las clases abstractas a alcanzar los criterios de reutilización del software estudiados en el tema 1.

c) (**0'5 ptos**) Supuesto un lenguaje que soporta herencia múltiple como Eiffel ¿surgiría algún problema al implementar la clase `DirectivoInvestigador` como subclase de `Directivo` e `Investigador`?

d) (**0'5 ptos**) Explica cómo sería posible implementar la clase `DirectivoInvestigador` en Java.