1. a) (0'5 ptos) Explica cómo contribuye la ***sobrecarga de rutinas*** para satisfacer el requerimiento de reutilización de módulos de ***Independencia de la implementación***. Contrasta esta solución con la que proporcionan los lenguajes de programación orientados a objetos.

   b) (0'5 ptos) Si la característica fundamental del ***modelo de ejecución orientado a objetos*** es la ausencia de un programa principal ¿Cómo se justifica que en Java haya que implementar un método `main`?

2. Dado el siguiente código Java:

| | |
|---|---|
| `class A{`<br>`  ? int at;`<br>`…`<br>`}` | `class B{`<br>`  void m1(A oa){`<br>`  oa.at=100;`<br>`  }`<br>`…`<br>`}` |

   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:

```
void 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. Dadas las dos especificaciones siguientes de un método que opera sobre un conjunto que contiene únicamente números naturales (enteros no negativos).

| Especificación A | Especificación B |
|---|---|
| `/**`<br>`* @return si size>0, devuelve el`<br>`*  número más pequeño del conjunto`<br>`* @throws EmptySetException si size==0`<br>`*/`<br><br>`int getSmallest() throws EmptySetException;` | `/**`<br>`* @return si size>0,`<br>`*  devuelve el número más`<br>`*  pequeño del conjunto, si no`<br>`*  devuelve -1.`<br>`*/`<br>`int getSmallest();` |

   a) (0'5 ptos) Justifica cual de las dos especificaciones consideras más apropiada.

   b) (0'5 ptos) ¿Para la especificación A debería ser `EmptySetException` una excepción comprobada o no comprobada? Justifica la respuesta.

4. Una tienda de componentes informáticos (TCI) nos ha encargado una aplicación. Los componentes que TCI vende pueden ser simples (disquetera, DVD, …) o lo que denominan *kits* (por ejemplo, el kit para conexión inalámbrica incluye tarjetas, router, …) que está formado por otros componentes (simples u otros kits). De manera que un kit es un componente formado a partir de otros. Cada uno de los componentes simples tiene un precio y el precio de un kit se calcula sumando el precio de todos sus componentes y haciendo un 5% de descuento sobre el total.
    a. (0'5 ptos) Diseña una jerarquía de clases basada en ***herencia múltiple*** que se ajuste a la especificación dada suponiendo que el lenguaje de programación que vamos a utilizar es Eiffel. Representa la jerarquía gráficamente.
    b. (0'5 ptos) Implementa en Eiffel el método `getPrecio` para la clase que modela el concepto de kit.

5. Sea el método `numMedallas` el encargado de calcular el número de medallas que han conseguido los deportistas de una determinada nacionalidad. El método se ha implementado en Eiffel haciendo uso del diseño de los ***iteradores internos*** visto en clase como sigue:

```
numMedallas(participantes:LIST[Deportista],nacionalidad:String):INTEGER is
local
  it: IteraXNacionalidad;
do
  !!it.make(participantes, nacionalidad);
   it.do_if
   Result:= it.getMedallas();
end
```

a) (1 pto) Implementa en Eiffel la clase `IteraXNacionalidad` (subclase de `Linear_Iterator[G]`) de manera que la ejecución del método `numMedallas` sea la esperada. Considérese que la clase `Deportista` incluye un método `getNacionalidad` que devuelve un `String` con la nacionalidad.

b) (1 pto) Completa el método `numMedallas` escrito en Java que se proporciona a continuación. La funcionalidad es equivalente al método implementado en Eiffel salvo que en este caso se deja abierto el criterio de acuerdo al cual se van a contar las medallas (nacionalidad del deportista, tipo de medalla, deporte, …). El criterio se le pasará al método como parámetro en el momento de invocarlo.

```
public int numMedallas (LinkedList participantes, _____________ criterio){
  int total = 0;
  Iterator it = participantes.iterator();

    ...

  return total;
}
```

c) (0'5 ptos) Implementa todo lo que sea necesario para mostrar cómo habría que utilizar al método `numMedallas` implementado en Java para contar las medallas de los deportistas españoles. (Puedes suponer que el método `numMedallas` es un método `static` de la clase `Olimpiada`).

6. (0'5 ptos) Explica qué dos problemas encontramos en Java al manejar una `LinkedList` en lugar de una lista genérica como en Eiffel para almacenar los deportistas.

7. Supongamos que la clase `Deportista` es la raíz de una jerarquía que representa las distintas categorías deportivas que participan en unas olimpiadas. Uno de sus atributos es `nombre`, que almacena el nombre del deportista. Sea `Tiro` una subclase de `Deportista` que implementa un método `getPlatos` (exclusivo de esta clase) que devuelve el número de platos que un tirador lleva rotos en la competición.

   a) (0'5 ptos) Sea el método `rankingTiro` un método escrito en Java que imprime la tabla con los resultados del tiro al plato. ¿Sería correcto el siguiente fragmento de código? Justifica la respuesta.

```
void rankingTiro(LinkedList participantes){
      ...
      if (participantes.get(i) instanceof Tiro){
            System.out.println(participantes.get(i).getNombre()+"-"+
                                    participantes.get(i).getPlatos());
      ...
}
```

   b) (0'5 ptos) Explica qué forma alternativa proporciona Eiffel para conseguir el mismo efecto que el método `instanceof` de Java.

8. Dado el código C++ que se muestra a continuación:

```
class Vehículo{
      int numBastidor;
      virtual void getInfo(){ cout<<"Vehículo n°"<<numBastidor;}
};
class Terrestre: public Vehículo{
      virtual void getInfo(){ cout<<"Vehículo Terrestre n°"<<numBastidor;}
};
class Acuatico: public Vehículo{
      virtual void getInfo(){ cout<<"Vehículo Acuático n°"<<numBastidor;}
};
class Anfibio: public Terrestre, public Acuatico{
      virtual void getInfo(){ cout<<"Vehículo Anfibio n°"<<numBastidor;}
};
```

   donde el atributo `numBastidor` representa el identificador único de un vehículo sea del tipo que sea.

   a) (0'5 ptos) ¿Sería correcta la definición de las clases anteriores? Justifica la respuesta e indica en el caso de que no sea correcta cuál sería la solución.

   b) (0'5 ptos) Añadir un método `desplazarse` en la jerarquía anterior de manera que:
      i. es abstracto en la clase `Vehículo`
      ii. la implementación la clase `Terrestre` es `//desplazarXTierra`
      iii. la implementación en la clase `Acuatico` es `//desplazarXAgua`.
      iv. la implementación en la clase `Anfibio` coincide con la de la clase `Terrestre`.