Prueba 1 de Evaluación Continua
Análisis y Diseño de Software (2017/2018)

Contesta el ejercicio 1 en esta hoja, y el 2 en hoja separada

Apellidos:
Nombre:

Ejercicio 1: Clases, herencia y polimorfismo (4 puntos)

Dada la definición de las clases en Java,

```
public abstract class Estancia {
	public String f() { return "f de Estancia"; }
}
public class Habitacion extends Estancia {
	public String f() { return  super.f() + " y f de Habitacion"; }
	public String g() { return  "g de Habitacion"; }
}
public class Suite extends Habitacion {
	public String g() { return  super.g() + " y g de Suite"; }
}
public class Oficina extends Estancia {
	public String f() { return  "f de Oficina"; }
	public String m() { return  super.f() + " y m de Oficina"; }
}
```

y asumiendo que ejecutamos secuencialmente las siguientes líneas de código numeradas, indica para cada línea qué valor se imprime en caso de que sea correcta, o bien indica si contiene algún error y explica el motivo del error.

1. `Estancia es = new Suite();` **//Correcta: Suite hereda de Estancia a través de Habitacion**
2. `Estancia ee = new Estancia();` **//Incorrecta: Estancia es abstracta, no instanciable**
3. `Habitacion hs = new Suite();` **// Correcta: Suite hereda de Habitacion**
4. `Oficina os = new Suite();` **// Incorrecta: Suite no hereda de Oficina, no es compatible**
5. `System.out.println( hs.g() );` **// Imprime: g de Habitacion y g de Suite**
6. `System.out.println( es.g() );` **//Incorrecta: la variable es se declaró de tipo Estancia // pero no hay método g() en la clase Estancia**
7. `System.out.println( es.f() );` **// Imprime: f de Estancia y f de Habitacion**
8. `es = new Oficina();` **// Correcta: Oficina hereda de Estancia**
9. `System.out.println( es.m() );` **//Incorrecta: la variable es se declaró de tipo Estancia // pero no hay método m() en la clase Estancia**
10. `System.out.println( es.f() );` **// Imprime: f de Oficina**

Prueba 1 de Evaluación Continua
Análisis y Diseño de Software (2017/2018)

Contesta el ejercicio 1 en esta hoja, y el 2 en hoja separada

Ejercicio 2: Diagramas de clase (6 puntos)

Nos piden realizar una aplicación para gestionar proyectos y equipos. Los proyectos tienen un nombre, un responsable, y estarán compuestos de tareas. El sistema permitirá calcular el esfuerzo del proyecto, en horas de trabajo, y el coste de los equipos necesarios para desarrollarlo.

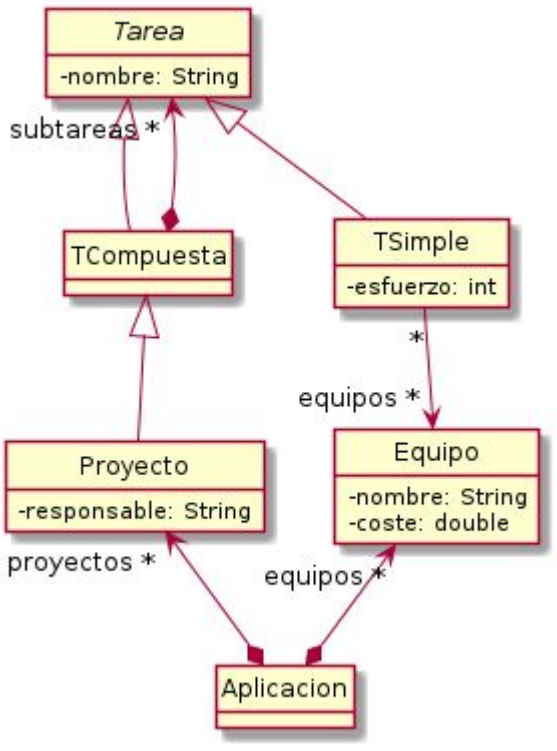
Las tareas a su vez pueden ser simples o compuestas. Las tareas simples tienen un nombre, un esfuerzo (como un número entero de horas), y unos equipos de la empresa asignados. Para cada equipo tendremos el coste por hora y su nombre. Calcularemos el coste que supone el uso de un equipo en una tarea como el coste por hora del equipo multiplicado por el número de horas de la tarea (esfuerzo).

Las tareas compuestas tienen un nombre y contienen subtareas. El esfuerzo y coste son calculados, igual que en el caso de los proyectos. Las subtareas pueden ser tareas simples o compuestas.

Una vez creada una tarea compuesta o proyecto, solo se podrán modificar para añadir tareas. Las tareas simples o los equipos no permitirán ninguna modificación una vez creados.

Se pide:

a. Realiza el diagrama de clases que describe la parte del diseño descrita arriba. **En este diagrama NO se incluirán métodos ni constructores.** (4.5 puntos)
b. Para cada clase del diagrama anterior, enumera los métodos necesarios para implementar la funcionalidad descrita, indicando la clase, el nombre del método, tipos de los argumentos y retorno. **No es necesario incluir constructores, ni *getters* simples** (que únicamente devuelvan el valor de un atributo con el mismo nombre). (1.5 puntos)




```
class TCompuesta {
 +esfuerzo():int
 +coste():double
 +addTarea(t:Tarea):void
}

class Tarea {
 +esfuerzo():int
 +coste():double
}

class TSimple {
 +esfuerzo():int //sobreescribimos esfuerzo()
 +coste():double
}

class Aplicacion { // No se especifican, pero son necesarios
 +addProyecto(p:Proyecto):void
 +addEquipo(e:Equipo):void
}
```