



## Preguntas y ejercicios

1. (0.5 puntos) Teniendo en cuenta el enunciado y el UML, ¿puedes ver el polimorfismo en alguna parte (sin tener en cuenta el método `toString()`)? Justifica la respuesta.
2. (0.5 puntos) ¿Por qué en el diagrama no hay dibujada ninguna relación **explícita** entre la clase `Doctor` y las clases `RevisiónPeriódica`, `VisitaUrgencias` e `IntervenciónProgramada`? ¿Significa eso que dichos tipos de visita no requieren de ningún equipo médico? Justifica tu respuesta.
3. (0.5 puntos) Hay una relación de asociación entre `IntervenciónProgramada` y `Quirofono`. Basándonos en las relaciones de herencia del UML, ¿tienen `RevisiónPeriódica` y `VisitaUrgencias` algún tipo de relación con `Quirofono`? ¿En caso afirmativo, cuál? Justifica tu respuesta.
4. (1 punto) Entre las clases `ControladorHospital` y `Doctor` hay un tipo de relación similar a la existente entre `ControladorHospital` y `Quirófano`. ¿Qué dos relaciones son, y en qué se diferencian? ¿Por qué motivo crees que se ha escogido cada una de ellas en este UML?
5. (0.5 puntos) Siguiendo el principio de encapsulación, prácticamente todos los atributos se han marcado como privados. Sólo hay un atributo que no es privado sino protegido. ¿Qué atributo es este? ¿Qué pasaría si se hubiera marcado también como privado?
6. (0.5 puntos) El método abstracto `requiereObservacion` definido en la clase abstracta `Visita` devuelve `true` si una visita necesita que el paciente se quede en observación, o `false` en caso contrario, según lo descrito en los puntos del enunciado. Teniendo en cuenta que es un método abstracto en la clase `Visita`, ¿dónde crees que debería implementarse el código para que cumpla la función requerida?
7. (3.5 puntos) Escribe la parte de definición del código de todas las clases del UML (empezando por el `public class ...`). Dentro de cada clase, **define solo los atributos y relaciones de éstas**. No es necesario que defines constructores ni métodos.
8. (1.5 punto) Asumiendo que el método `toString()` ya está redefinido y programado en las clases `VisitaUrgencias`, `RevisiónPeriódica` e `IntervenciónProgramada`, escribe el código del método `muestraVisitas()` de la clase `ControladorHospital`, que muestra por pantalla una lista de todas las visitas que están programadas en el hospital.
9. (1.5 puntos) Implementa el código del siguiente método en la clase `ControladorHospital`:  

```
public void creaUrgencia(int hora, int prioridad, Paciente p, Doctor d);
```

Dicho método crea una visita de urgencia, para la hora y prioridad especificada, del paciente `p` con el doctor `d`.

**NOTA 1:** asegúrate de que todas las relaciones entre las clases implicadas quedan completas.

**NOTA 2:** puedes asumir que el constructor `public VisitaUrgencias(int hora, int prioridad)` ya está implementado, así como los *getters* y *setters* que encapsulan los diferentes atributos y relaciones de la clase.

## Solución

1. El método `requiereObservación` es polimórfico, ya que a pesar de estar definido en la clase abstracta `Visita`, debe implementarse en sus diferentes subclases y retornar un valor de acorde a las particularidades de cada tipo de visita.
2. La relación de asociación entre la clase `Doctor` y la clase `Visita` es heredada por las subclases de `Visita`. Por tanto, realmente sí existe una relación entre `Doctor` y `RevisiónPeriódica`, `VisitaUrgencias` e `IntervenciónProgramada`.
3. No tienen ningún tipo de relación, puesto que no hay relación de herencia de `IntervenciónProgramada` a `VisitaUrgencias` y `RevisiónPeriódica`.
4. Entre `ControladorHospital` y `Doctor` existe una relación de agregación, mientras que entre `ControladorHospital` y `Quirofono` existe una relación de composición (o agregación fuerte). Ambas relaciones indican una relación entre un contenedor y un grupo de objetos contenidos. La diferencia es que en una composición, cuando el contenedor es destruido los elementos contenidos en éste también son destruidos. En la agregación no sucede esto.

En el UML del examen se ha escogido una relación de composición entre `ControladorHospital` y `Quirofono` porque si el objeto `ControladorHospital` fuera destruido, los quirófanos contenidos en él también lo serían. La relación entre `ControladorHospital` y `Doctor` es de agregación porque los objetos `Doctor` seguirían teniendo vigencia contenidos en otras clases.

5. El atributo `horaEntrada` de la clase `Visita` ha sido marcado como protegido porque debe poder ser accesible directamente desde de las subclases. Si se hubiera marcado como privado, las subclases no podrían acceder a él directamente. Solo podrían leer y cambiar su valor a través de *getters* y *setters*, si éstos estuvieran definidos.
6. Al ser un método polimórfico, debería ser implementado en las clases `RevisiónPeriódica`, `VisitaUrgencias` e `IntervenciónProgramada`.

```
7. public class ControladorHospital {
    private Set<Doctor> emplea;
    private Set<Paciente> ingresa;
    private Set<Visita> programadas;
    private Set<Quirofono> quirofonos;
}
public class Doctor {
    private String nombre, especialidad;
}
public class Paciente {
    private String nombre, numeroSS;
    private int fechaNacimiento;
    private Visita visita;
}
public abstract class Visita {
    private Paciente paciente;
    protected int horaEntrada;
    private Set<Doctor> equipoMedico;
}
public class RevisiónPeriódica extends Visita { }
public class VisitaUrgencias extends Visita {
    private int prioridad;
}
public class IntervenciónProgramada extends Visita {
    private float duracionEstimada;
    private Quirofono quirofono;
}
public class Quirofono {
    private int numero;
```

```

    }

8. public void muestraVisitas() {
    System.out.println("Las visitas programadas para el hospital son:");
    Iterator<Visita> it = programadas.iterator();
    while(it.hasNext()) {
        Visita v = it.next();
        System.out.println(v.toString());
    }
}

9. public void creaUrgencia(int hora, int prioridad, Paciente p, Doctor d) {
    VisitaUrgencias visita = new VisitaUrgencias(hora, prioridad);
    visita.setPaciente(p);
    p.setVisita(visita);
    visita.setDoctor(d);
    programadas.add(visita);
}

```