

Examen Final

EVALUACIÓN NO CONTINUA

Análisis y Diseño de Software (2011/2012)

Responde a cada pregunta en hojas separadas

Apartado 1. (2.5 puntos)

Realiza un programa para controlar los alquileres de un videoclub. El videoclub presta dos tipos de elementos: Libros y Películas. Ambos se pueden alquilar por un número arbitrario de días. El coste de las películas es por el número de días alquilado, mientras que el coste del alquiler de los libros es por todo el periodo.

Hay tres tipos de tarifas para los socios del videoclub: normal, especial o lector. La tarifa especial implica un descuento de 10 céntimos sobre cada día de alquiler (en caso de películas) o sobre el periodo total (en caso de libros). La tarifa lector implica que el socio puede sacar gratis cualquier libro (pero a precio normal las películas).

Tu programa deberá lanzar una excepción si se trata de alquilar un elemento con días negativos (o cero).

Se pide:

- Realizar las clases necesarias (incluyendo la excepción) para que el siguiente programa produzca la salida de más abajo. Fíjate en que, en la salida final, los elementos están ordenados alfabéticamente.
- Indica cómo completar las dos líneas que faltan en el *main*.

```
public class Main {  
  
    public static void main(String[] args) throws DiasInvalidosException {  
        Alquilerable[] recursos = { new Pelicula("Cyclo", 3), // 3€ al día  
                                     new Libro("La broma infinita", 1), // 1€ en total  
                                     new Libro("Asfixia", 1)}; // 1€ en total  
  
        Socio pedro = new Socio("Pedro", Tarifa.NORMAL);  
        Socio maria = new Socio("Maria", Tarifa.ESPECIAL);  
        Socio felipe= new Socio("Felipe",Tarifa.LECTOR);  
  
        ..... { // linea (1): indica cómo completarla  
            recursos[0].alquilar(maria, 2); // alquila Cyclo por 2 días con tarifa ESPECIAL  
            recursos[1].alquilar(pedro, 0); // error! 0 días  
        }  
        ..... { // linea (2): indica cómo completarla  
            System.out.println("Número de días invalidos: "+e.getDias());  
        }  
        recursos[1].alquilar(pedro, 2); // alquila "La broma..." por 2 días con tarifa NORMAL  
        recursos[1].alquilar(felipe, 2); // devuelve false: ya está alquilada  
        recursos[2].alquilar(felipe, 2); // alquila Asfixia por 2 días con tarifa LECTOR  
  
        TreeSet<Alquilerable> ordenAlfabetico = new TreeSet<Alquilerable>(Arrays.asList(recursos));  
        for (Alquilerable a : ordenAlfabetico )  
            System.out.println("El precio del alquiler de "+a+" es: "+  
                               a.getPrecio()+" alquilada a: "+a.getSocio());  
    }  
}
```

Salida esperada:

Número de días invalidos: 0

El precio del alquiler de Asfixia es: 0.0 alquilada a: Felipe

El precio del alquiler de Cyclo es: 5.8 alquilada a: Maria

El precio del alquiler de La broma infinita es: 1.0 alquilada a: Pedro

Apartado 2. (3.5 puntos)

Se quiere desarrollar una aplicación para que los *vigilantes* puedan establecer umbrales de *alarma* sobre *medidas* (en unidades genéricas: numéricas, temporales, etc.), de forma que los vigilantes sean notificados automáticamente cuando alguna medida se actualice con un valor *igual o superior al umbral* de la(s) alarma(s) que hayan establecido sobre dicha medida. Tras ser notificada al vigilante, la alarma desaparecerá, a menos que se haya indicado que es repetible.

Escribe las clases Usuario, Aviso y Medida, para que el método main dado abajo produzca la siguiente salida, respetando los interfaces dados a continuación:

```
Medida con valor actualizado a 2.3
Medida con valor actualizado a 2.5
***Luis recibe alerta de alarma 2.5***
***Maria recibe alerta de alarma 2.5***
Medida con valor actualizado a 2.7
***Maria recibe alerta de alarma 2.5***
Medida con valor actualizado a 2.2
Medida con valor actualizado a 2.8
***Maria recibe alerta de alarma 2.5***
Medida con valor actualizado a 3.2
***Luis recibe alerta de alarma 3.0***
***Maria recibe alerta de alarma 2.5***
```

```
interface Vigilante<V extends Comparable<V>> {
    void notificacionAlerta(Alarma<V> a);
}

interface Alarma<V extends Comparable<V>> {
    V getUmbral();
    Vigilante<V> getVigilante();
    boolean esRepetible();
}

public class Main {
    public static void main(String[] args) {
        Medida<Double> temperatura = new Medida<Double>(); // temperatura mide Doubles
        Vigilante<Double> us1 = new Usuario<Double>("Luis");
        Vigilante<Double> us2 = new Usuario<Double>("Maria");
        temperatura.setAlarma(new Aviso<Double>(us1, 2.5, false)); // alarma no repetible
        temperatura.setAlarma(new Aviso<Double>(us1, 3.0, true)); // alarma repetible
        temperatura.setAlarma(new Aviso<Double>(us2, 2.5, true)); // alarma repetible
        double[] simulacion = {2.3, 2.5, 2.7, 2.2, 2.8, 3.2};
        for (Double d : simulacion)
            temperatura.update(d);
    }
}
```

Apartado 3. (2 puntos)

La clase “Printer” simula una impresora, y tiene tres métodos que permiten a un usuario iniciar un trabajo, imprimir páginas y terminarlo. Utilizando patrones de diseño, y sin modificar la clase Printer, completa programa para que sea posible crear impresoras de uso exclusivo. Estas impresoras no permiten imprimir a un usuario si ya hay otro que la está utilizando (es decir, si ya ha llamado al método “startJob”). La liberación de la impresora se produce cuando el usuario llama al método “endJob”. ¿Qué patrón (o patrones) de diseño has usado?

```
class PrinterUtil {
    public static IPrinter makeExclusive(Printer p) {

        ..... // Completar

    }
}

interface IPrinter { //Completar
    .....
}

class Printer implements IPrinter{
    public void startJob(String header, String user) {
        System.out.println("User "+user+" Printing Header:: "+header);
    }
    public void printPage(String page, String user) {
        System.out.println("User "+user+" Printing Page:: "+page);
    }
    public void endJob(String footer, String user) {
        System.out.println("User "+user+" Printing Footer:: "+footer);
    }
}

// Completar
...
...

public class Main {
    public static void main(String[] args) {
        Printer p = new Printer();
        IPrinter ap = PrinterUtil.makeExclusive(p);

        ap.startJob("inicio", "juan");
        ap.startJob("otro inicio", "eduardo"); // no imprime, la tiene juan
        ap.printPage("página 1", "juan");
        ap.printPage("otra pagina 1", "eduardo"); // no imprime, la tiene juan
        ap.endJob("final", "juan"); // juan libera la impresora
        ap.startJob("a ver si esta vez...", "eduardo"); // ahora sí
        ap.endJob("fin", "eduardo");
    }
}
```

Salida esperada:

```
User juan Printing Header:: inicio
User juan Printing Page:: página 1
User juan Printing Footer:: final
User eduardo Printing Header:: a ver si esta vez...
User eduardo Printing Footer:: fin
```

Apartado 4. (2 puntos)

Se quiere construir una aplicación para la gestión de los vuelos de un aeropuerto, y de los empleados de las distintas compañías aéreas. Una compañía aérea tiene un nombre y posee aviones, así como empleados. Un avión tiene un nombre, una capacidad máxima de pasajeros, y unos kilómetros de vuelo realizados. Cada avión es de un modelo concreto (por ejemplo *Boeing 747*), que viene descrito por un nombre, una empresa constructora y un volumen de carga.

Hay dos tipos de empleados: el personal de vuelo y el personal de tierra. De todos ellos se quiere guardar el nombre, sueldo base y fecha de contrato. El personal de tierra tiene un tipo de jornada (de entre 4 y 8 horas). Para el personal de vuelo hay que guardar sus horas de vuelo, y el programa debe distinguir entre Pilotos y Auxiliares de Vuelo. Para los pilotos debemos guardar la fecha en que obtuvieron su permiso de vuelo, así como los modelos de avión que tienen permiso para pilotar. Finalmente, hay un tipo de Auxiliar de Vuelo que son los sobrecargos, que tienen una categoría.

Un vuelo está descrito por una ciudad origen y destino, una fecha de inicio, una duración prevista, y un estado (cancelado, enTierra, retrasadoEnTierra, enVuelo, retrasadoEnVuelo, aterrizado). De cada vuelo debemos guardar: el avión que lo realiza, el piloto y el copiloto (un copiloto es también un piloto), y el personal de vuelo (de 1 a 20 personas). Un vuelo puede ser operado por varias compañías aéreas, y tendrá un código (e.g. IB205) para cada una de ellas.

Se pide:

- a) Realizar un diagrama de clases UML que describa el enunciado.
- b) Añade métodos (no hace falta que escribas el código) a las clases para:
 - b.1. Devolver el número de vuelos efectuados por un piloto, como piloto y como copiloto.
 - b.2. Calcular el sueldo de un empleado. El sueldo se calcula descontando el IRPF del sueldo base. Para el personal de tierra se tiene en cuenta su tipo de contrato, y para el de vuelo sus horas de vuelo. Los Pilotos y los sobrecargos tienen una bonificación dependiendo del número total de vuelos efectuados.
 - b.3. Obtener la colección de compañías que operan un vuelo.