Programación II - 2011/2012 - Universitat Jaume I
Evaluación continua - Módulo 2 de teoría - Grupo B

25 de abril de 2012

La duración final de esta prueba ha sido de 1 hora 40 minutos, sin consultar libros ni apuntes.

Ejercicio 1 (3,5 puntos, 0.5 puntos por apartado)

Imagina que hubiésemos implementado así las clases `Hora` y `ParadaTren`. Fíjate bien en lo que hacen.

```
public class Hora {
    private int hora, minuto;
    public Hora(int hora, int minuto) {
        this.hora = hora;
        this.minuto = minuto;
    }
    public Hora(Hora otraHora) {
        this(otraHora.hora, otraHora.minuto);
    }
    public void cambiar(int hora, int minuto) {
        this.hora = hora;
        this.minuto = minuto;
    }
    public Hora sumarMinutos(int minutos) {
        Hora nueva = new Hora(this);
        nueva.minuto += minutos;
        nueva.hora += nueva.minuto / 60;
        nueva.minuto = nueva.minuto % 60;
        return nueva;
    }
    public String toString() {
        return hora + ":" + minuto;
    }
}
public class ParadaTren {
    private Hora hora;
    private String ciudad;
    public ParadaTren(Hora laHora, String laCiudad) {
        hora = laHora;
        ciudad = laCiudad;
    }
    public ParadaTren crearCopiaSuperficial() {
        return new ParadaTren(hora, ciudad);
    }
    public ParadaTren crearCopiaEnProfundidad() {
        return new ParadaTren(new Hora(hora), ciudad);
    }
    public void cambiarHora(int hora, int minuto) {
        this.hora.cambiar(hora, minuto);
    }
    public void sumarMinutos(int minutos) {
        hora = hora.sumarMinutos(minutos);
    }
    public String toString() {
        return hora + " - " + ciudad;
    }
}
```

Indica lo que se escribiría en la salida estándar al ejecutar cada uno de los siguientes programas utilizando esas versiones. No es necesario que expliques por qué.

(a)
```
public class EjercicioA {
    public static void main(String[] args) {
        Hora h1 = new Hora(17, 30);
        Hora h2 = new Hora(h1);
        h1.sumarMinutos(5);
        h2.sumarMinutos(10);
        Hora h3 = h2.sumarMinutos(15);
        System.out.println("h1 = " + h1);
        System.out.println("h2 = " + h2);
        System.out.println("h3 = " + h3);
    }
}
```

(b)
```
public class EjercicioB {
    public static void main(String[] args) {
        ParadaTren p1 = new ParadaTren(new Hora(13, 57), "Teruel"),
                   p2 = p1.crearCopiaSuperficial(),
                   p3 = p2.crearCopiaEnProfundidad();
        p1.cambiarHora(2, 3);
        p2.cambiarHora(4, 5);
        p3.cambiarHora(6, 7);
        System.out.println("p1 = " + p1);
        System.out.println("p2 = " + p2);
        System.out.println("p3 = " + p3);
    }
}
```

(c)
```
public class EjercicioC {
    public static void main(String[] args) {
        ParadaTren p1 = new ParadaTren(new Hora(21, 10), "Lugo"),
                   p2 = p1.crearCopiaSuperficial();
        p1.sumarMinutos(5);
        p2.sumarMinutos(10); // Piensa bien esto
        System.out.println("p1 = " + p1);
        System.out.println("p2 = " + p2);
    }
}
```

(d)
```
import java.util.Arrays;
public class EjercicioD {
    public static void main(String[] args) {
        Hora h = new Hora(11, 22);
        ParadaTren[] v = new ParadaTren[5];
        v[0] = new ParadaTren(h, "Sevilla");
        v[1] = v[0];
        v[2] = v[1].crearCopiaSuperficial();
        v[3] = v[2].crearCopiaEnProfundidad();
        v[4] = v[3].crearCopiaSuperficial();
        h.cambiar(15, 33);
        System.out.println( Arrays.toString(v) );
    }
}
```

(e)
```
import java.util.Arrays;
public class EjercicioE {
    public static void main(String[] args) {
        ParadaTren[] v1 = { new ParadaTren(new Hora(20, 40), "Badajoz"),
                            new ParadaTren(new Hora(20, 50), "Madrid")

        };
        ParadaTren[] v2 = v1.clone();
        v1[0].sumarMinutos(5);
        System.out.println("v1 = " + Arrays.toString(v1) );
        System.out.println("v2 = " + Arrays.toString(v2) );
    }
}
```

(f)
```
import java.util.Arrays;
public class EjercicioF {
    public static void prueba(ParadaTren p1, ParadaTren p2) {
        p1 = p2.crearCopiaSuperficial();
        p2.cambiarHora(9, 45);
    }
    public static void main(String[] args) {
        ParadaTren p1 = new ParadaTren(new Hora(7, 55), "Valencia"),
                   p2 = new ParadaTren(new Hora(8, 50), "Alicante");
        prueba(p1, p2);
        System.out.println("p1 = " + p1);
        System.out.println("p2 = " + p2);
    }
}
```

(g)
```
import java.util.Arrays;
public class EjercicioG {
    public static ParadaTren sumarMinutos(ParadaTren parada, int minutos) {
        ParadaTren resultado = parada;
        resultado.sumarMinutos(minutos);
        return resultado;
    }
    public static void main(String[] args) {
        ParadaTren p1 = new ParadaTren(new Hora(9, 45), "Castellon"),
                   p2 = sumarMinutos(p1, 5);
        System.out.println("p1 = " + p1);
        System.out.println("p2 = " + p2);
    }
}
```

Ejercicio 2 (6,5 puntos)

Supongamos que, haciendo uso de las clases `Hora` y `ParadaTren` del ejercicio anterior, y añadiéndoles los metodos adicionales que necesites, debes implementar una clase `RecorridoTren` para guardar la información de las ciudades en las que tiene parada un tren y las horas de parada previstas. Internamente el recorrido lo debes representar mediante un vector de objetos de la clase `ParadaTren`, no necesariamente ordenado por horas ni por nombres de ciudades.

Define los atributos de la clase `RecorridoTren` e implementa los siguientes métodos además de cualquier otro método que necesites y no esté disponible en las clases `Hora`, `ParadaTren` y `RecorridoTren`.

Puedes suponer que ninguna ciudad aparecerá nunca más de una vez en un recorrido.

Ten en cuenta siempre que puedes tener un recorrido vacío.

Cuando haga falta redimensionar el tamaño del vector puedes utilizar la estrategia que prefieras.

1. Un constructor sin argumentos.

2. Un método `insertarParada` que reciba como parámetros el nombre de una ciudad y dos enteros con la hora y minuto de la parada. Si no existe ya una parada en esa ciudad, se debe añadir una nueva parada con esos datos. En caso contrario, se debe cambiar la hora de la parada.

3. Un método `eliminarParada` que reciba como parámetro el nombre de una ciudad. Si hay una parada en esa ciudad, debe eliminarse la parada del recorrido. En caso contrario, no debe cambiar nada.

4. Un método `eliminarParadas` que reciba como parámetro un vector de nombres de ciudades y elimine del recorrido las paradas en todas ellas, si existe alguna.

5. Un método `buscarHoraParada` que reciba como parámetro el nombre de una ciudad y devuelva la hora a la que el tren para en esa ciudad. Si el tren no tiene parada en esa ciudad, debe devolver `null`.

6. Un método `buscarHorasParadas` que reciba como parámetro un vector de nombres de ciudades y devuelva un vector de horas en el que la posición i-ésima sea la hora a la que el tren para en la ciudad i-ésima del vector de ciudades. Si el tren no tiene parada en alguna ciudad, la correspondiente posición del vector de horas debe contener `null`.

7. Un método `seleccionarParadas` que reciba como parámetro un vector de nombres de ciudades y, sin modificar el recorrido que se tiene, cree y devuelva un nuevo `RecorridoTren` formado por las paradas que haya en esas ciudades. Deben hacerse copias de esas paradas de modo tal que cualquier modificación futura de las mismas en un recorrido no afecte al otro.

8. Un método `equals` que devuelva `true` si y solo si los dos recorridos tienen dos conjuntos de paradas iguales, incluso si las paradas no están en el mismo orden en los dos vectores (y, como caso particular, si ambos conjuntos están vacíos) .