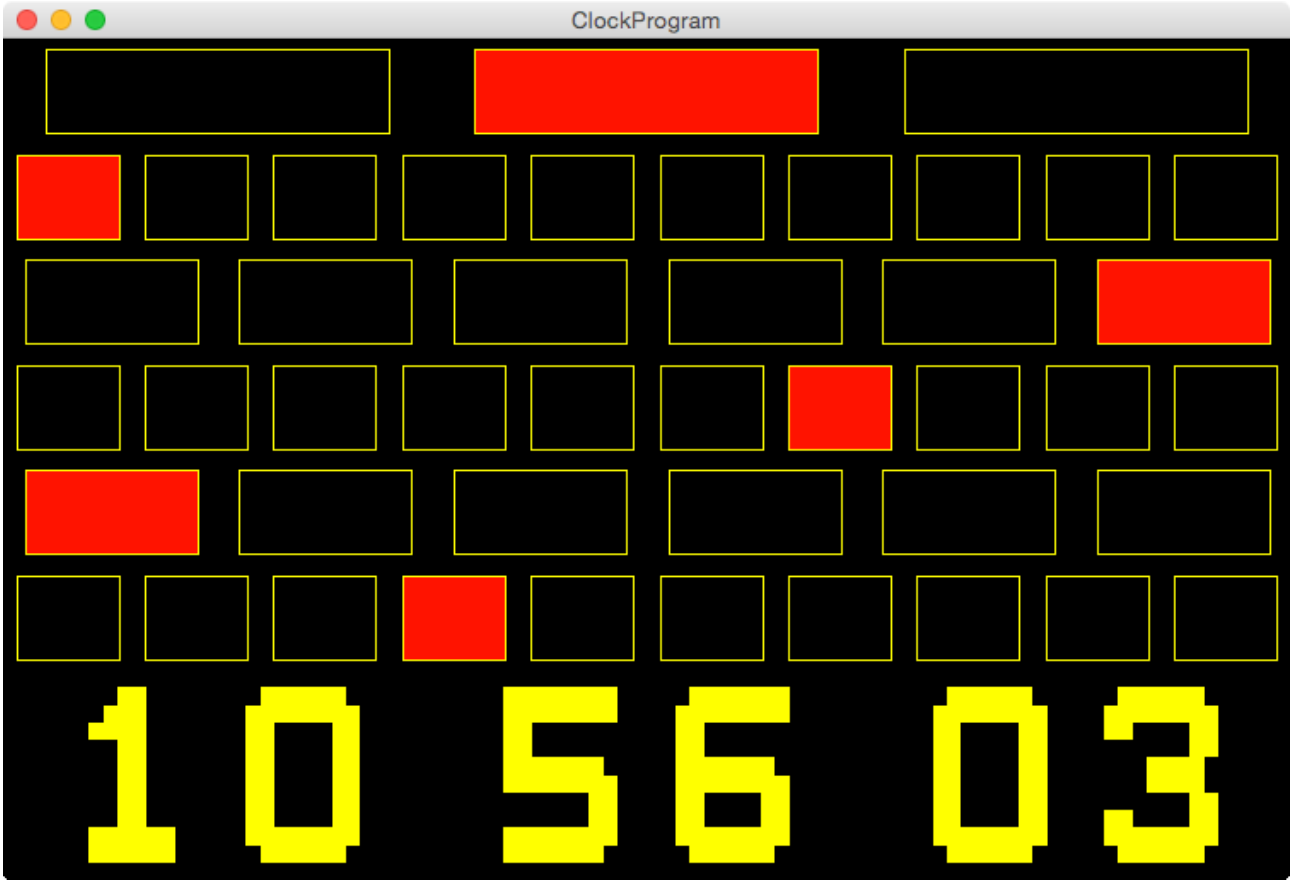
Segunda práctica obligatoria

Programación Orientada a Objetos — Curso 2020-2021

Grau en Tècniques d'Interacció Digital i de Computació

Reloj digital con leds

El objetivo de esta práctica es desarrollar una aplicación gráfica, usando la biblioteca de la ACM, que muestre en la ventana de la aplicación un reloj digital visualizando horas, minutos y segundos. Además de mostrar la hora usando dígitos, de cara a hacer el problema más interesante, también mostraremos la hora usando diferentes elementos gráficos que denominaremos leds. Como una imagen vale más que mil palabras, veamos una captura de la aplicación final:



Encima de la hora pueden verse seis líneas de rectángulos: mirando de arriba abajo, las dos primeras representan la hora, las dos siguientes los minutos y las dos últimas los segundos. Cada línea representa un dígito, empezando por cero. En nuestra captura: el primer dígito es un 1, el segundo un 0 (la hora, como vemos, es 10); el tercero un 5, el cuarto un 6 (los minutos son 56); el quinto un 0 y el sexto un 3 (los segundos son 03).

La aplicación que os proponemos requiere el diseño de varias clases que colaboran entre sí para construir el resultado final que se muestra en la figura. Para facilitaros el trabajo, os damos algunas clases ya hechas y os pedimos que vayáis desarrollando las que faltan siguiendo los pasos que se indican en el enunciado.

Antes de entrar a detallar las diferentes clases que deberéis programar, os explicaremos cómo añadir elementos gráficos en el lienzo desde fuera de la clase que representa el programa principal y cómo comprobar en una función que los parámetros cumplen ciertas propiedades usando aserciones.

Cómo añadir objetos desde fuera de programa principal

Desde el programa principal se tiene acceso al lienzo (canvas) y podéis añadir directamente los objetos gráficos que queréis mostrar usando la operación `add`. Simplemente se puede hacer:

```
public void run() {
      GRect square = new GRect(0.0, 0.0, 20.0, 20.0); // [1]
      add(square);                                    // [2]
}
```

En la instrucción [1] creamos un cuadrado y en la [2] lo añadimos al lienzo.

Esto es así porque el "`extends GraphicsProgram`", que aparece en la definición de la clase que contiene el programa principal (`run`), hace que, desde cualquier método de la clase, se pueda usar la operación `add` para añadir objetos gráficos. ¿Qué pasa si queremos añadir elementos gráficos desde una clase que no es la principal y que, por tanto, no puede usar directamente la operación `add`?

Accediendo al canvas

Para poder añadir objetos gráficos al lienzo desde un método de otra clase, le tendremos que pasar una referencia a él y, para eso, antes debemos obtenerla desde el programa principal. Por suerte, disponemos de la operación `getGCanvas` que cumple este mismo propósito.

Supongamos que tenemos una clase `Figure` (que no es la clase principal) y queremos que sea capaz de añadir objetos gráficos al lienzo. Podemos hacer:

```
public class Example extends GraphicsProgram {
      public void run() {
            Figure f = new Figure();
            f.addToGCanvas(getGCanvas());
      }
}
public class Figure() {
      ...
      public void addToGCanvas(GCanvas canvas) {
            GRect square = new GRect(0.0, 0.0, 20.0, 20.0);
            canvas.add(square);
      }
}
```

Uso de aserciones

Suponed que tenemos una función que requiere que el número entero que se le pase sea siempre positivo (de hecho, si la hemos llamado con un número negativo o cero es que nos hemos equivocado al programar). En este caso, lo mejor que podemos hacer es parar el programa si se detecta esta situación anómala. Eso es justamente lo que hace la sentencia `assert`: comprueba que sea cierta la expresión que contiene y, si no lo es, para el programa indicando el error.

Por ejemplo, la siguiente función sólo tiene sentido si el valor que se le pasa está entre cero y uno:

```
public boolean nextBoolean(double probability) {
      assert 0.0 <= probability && probability <= 1.0;
      ...
}
```

Como las aserciones en Java están deshabilitadas por defecto, y algunas de las clases que os damos las utilizan, debéis pasarle a **`java`** (¡no a `javac`!) la opción **`-ea`** (o `-enableassertions`).

Eligiendo el programa a ejecutar

Entre el código que os damos existen varias clases que representan diferentes programas principales y que van probando las diferentes clases de que consta vuestra solución (son las clases llamadas `Step1`, `Step2`, ..., `Step7` y `ClockProgram`). A medida que avance la realización de la práctica debéis ejecutar cada uno de los programas principales para probar la parte que hayáis solucionado de la práctica.

Clases que os damos completas

Las siguientes clases os las damos **completamente implementadas y no las debéis modificar**:

- `Step1` a `Step7`: estas clases van probando las diferentes clases de vuestra solución y las iréis utilizando una a una.
- `ClockProgram`: esta es la clase que implementa el producto final de la práctica.
- `TimeTeller`: es la clase que utilizaréis para obtener las horas, minutos y segundos de la hora actual.
- `Padding`: clase que representa (en porcentaje) el margen que dejamos en cada una de las cuatro direcciones.
- `Led`: clase que dibuja un led rojo en la zona que se le indica dejando un margen de 0,1 en las cuatro direcciones.
- `DecimalDigit`: clase que dibuja un dígito en forma de número usando rectángulos a modo de píxeles “gordos” (ver Step 4).
- `Clock`: clase que representa el producto final y que se muestra al ejecutar `ClockProgram`.

Clase que os damos parcialmente construida

Las siguientes clases tienen algún método implementado y otros por implementar:

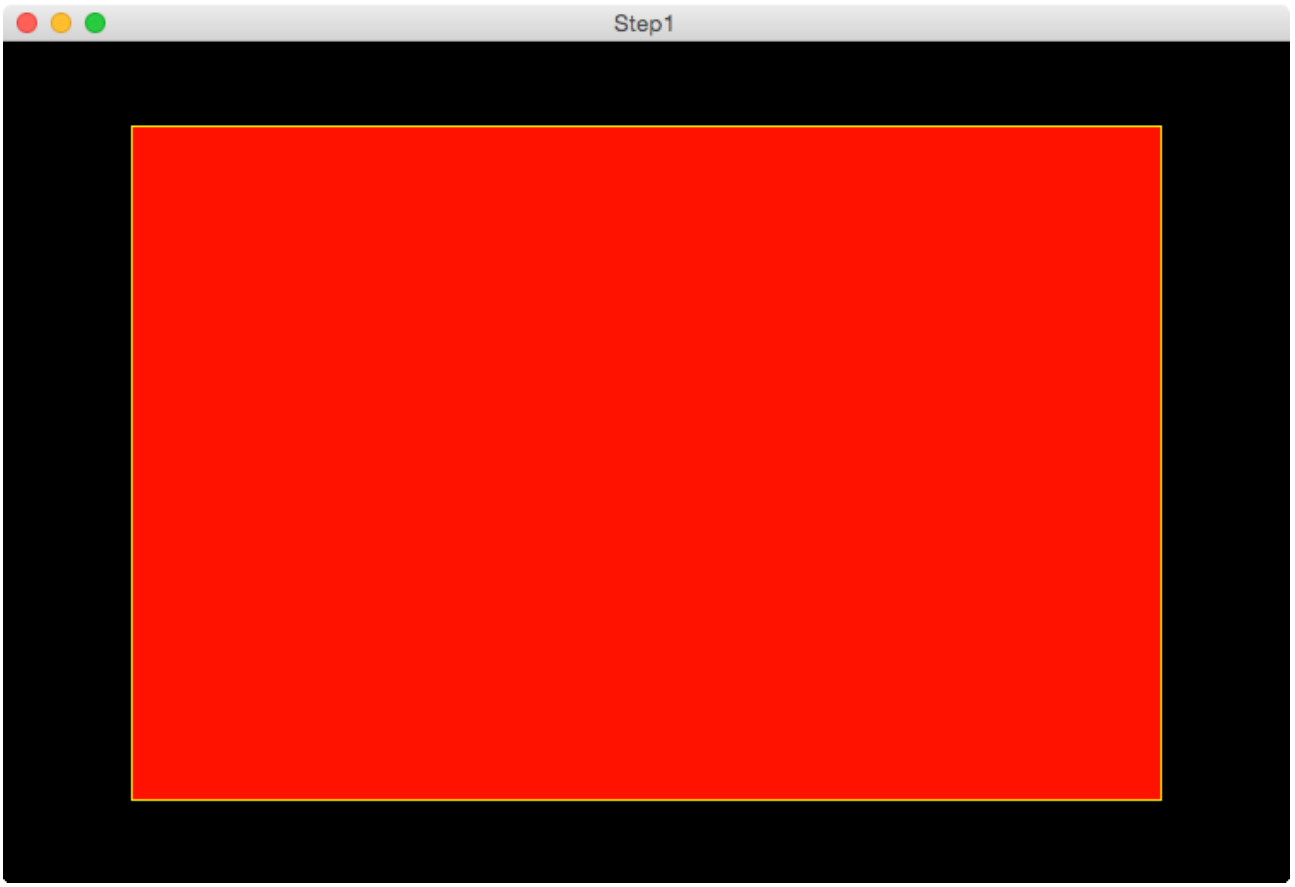
- `Box`: clase que representa una zona de la ventana en la que se dibujan elementos gráficos.

Clases que sólo contienen el esqueleto

Estas clases sólo contienen el esqueleto de las operaciones públicas que debéis implementar:

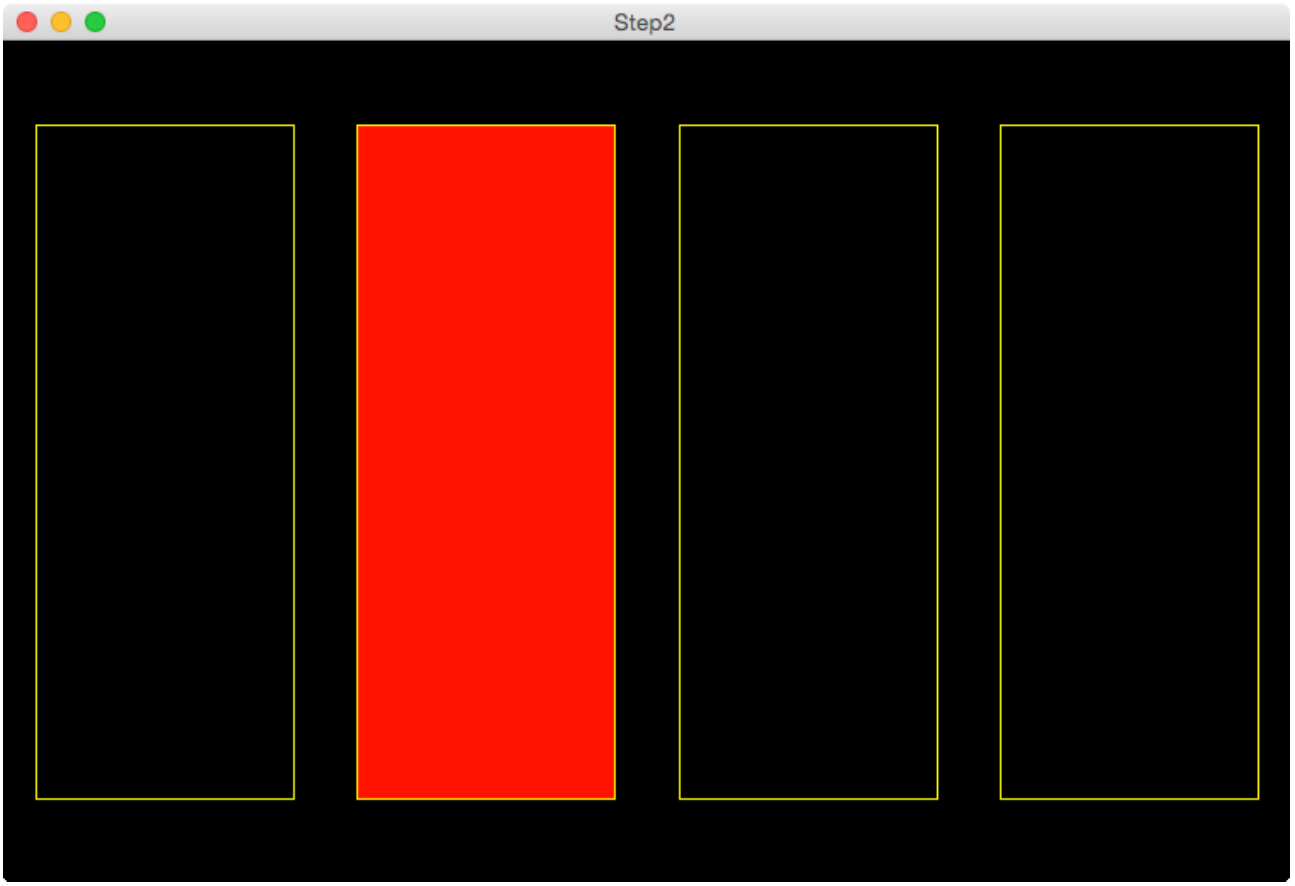
- `Bar`: clase que representa una barra de leds horizontal con los leds dispuestos uno al lado del otro (en Step 2 se muestra una barra formada por 4 leds).
- `Group`: clase que representa un grupo de barras paralelas (en Step 3 se muestra un grupo formado por dos barras, una de 4 leds y otra de 10).
- `DecimalNumber`: clase que representa un número decimal en base a sus dígitos (ver Step 5).
- `DecimalClock`: clase que representa un reloj que muestra la hora usando números decimales (ver Step 6).
- `LedsClock`: reloj que muestra la hora usando diferentes grupos de barras de leds, uno para horas, otro para minutos y otro para segundos (ver Step 7).

Step 1: Probando el led



Las clases que forman el primer paso os las pasamos completamente. La tarea en este paso consiste en entender el código que os damos ya que, como veréis, la mayoría de las clases que tendréis que programar para representar elementos gráficos tendrán una estructura similar. También es importante que comprendáis perfectamente el funcionamiento del método `withPadding` de la clase `Box`.

Step 2: Probando la barra de leds



En este paso debéis programar:

- En la clase `Box`:

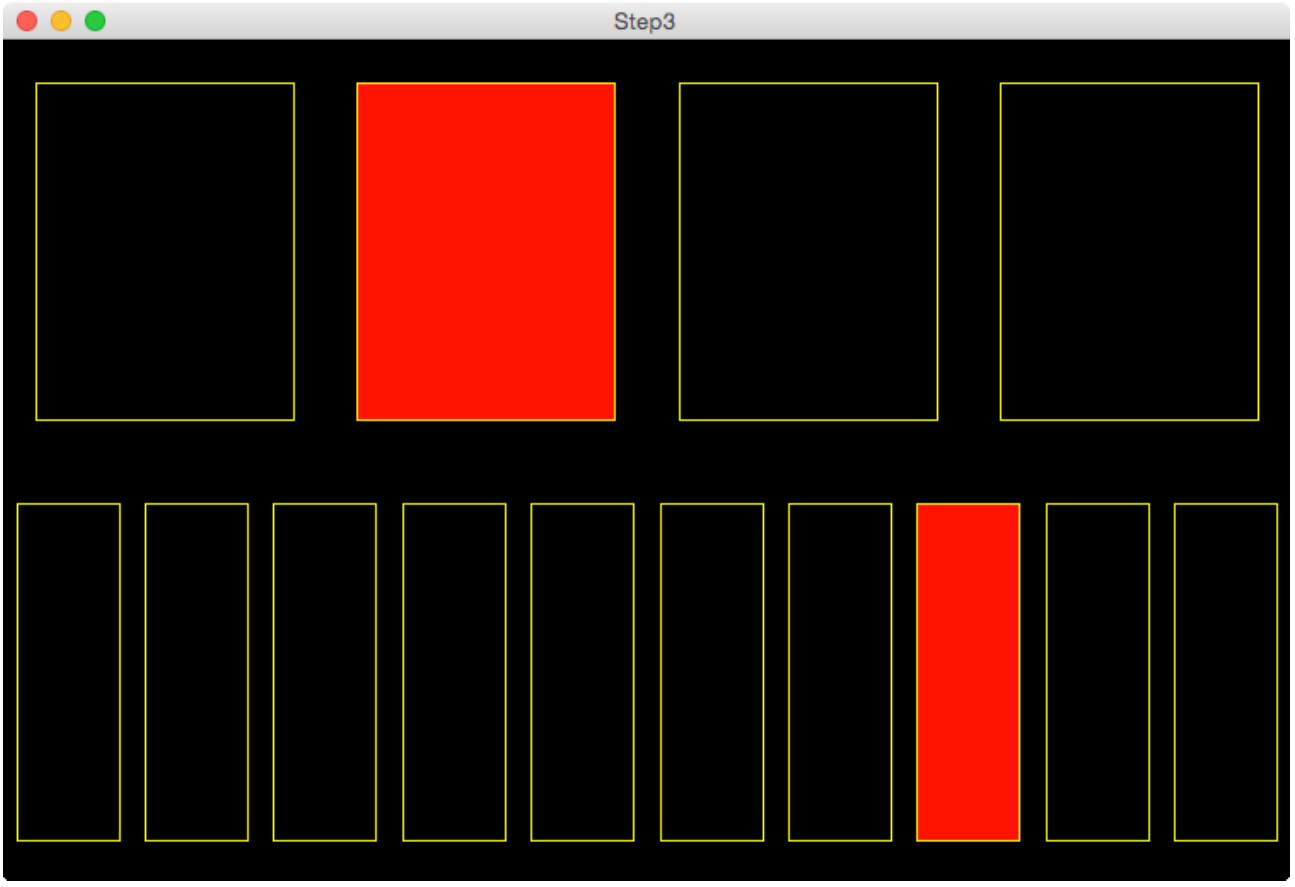
```
/* Distribuye de forma uniforme el espacio indicado en la caja
(Box) del objeto receptor en tantas cajas como indique numColumns.
Las cajas resultantes están colocadas horizontalmente de forma
contigua.
Se devuelve el vector de referencias a esas cajas.
*/
public Box[] distributeHorizontally(int numColumns) {
      assert numColumns > 0;
      ???
}
```

- En la clase `Bar`:

```
/* Construye una barra en la caja indicada por barBox, formada por
size leds del mismo tamaño colocados horizontalmente
*/
public Bar(int size, Box barBox) {
      assert size > 0;
      ???
}
```

```
/* Añade todos los leds que forman la barra al canvas gCanvas
*/
public void addToGCanvas(GCanvas gCanvas) {
      ???
}
/* Ilumina el led indicado por value en la barra, contando desde 0
de izquierda a derecha.
*/
public void render(int value) {
      ???
}
```

Step 3: Probando el grupo de barras



En este paso debéis programar:

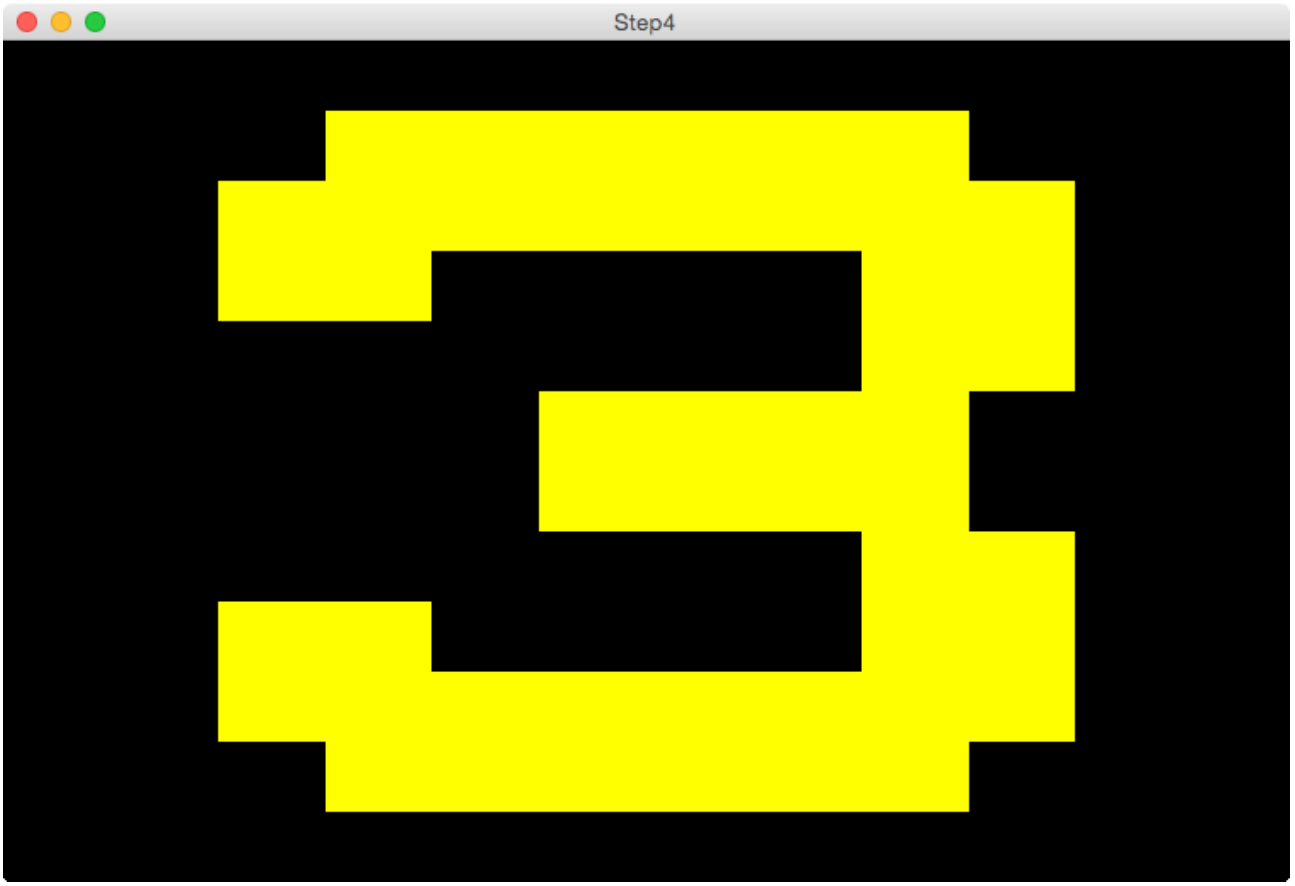
- En la clase Box:

```
/* Distribuye de forma uniforme el espacio indicado en la caja
(Box) del objeto receptor en tantas cajas como indique numRows.
Las cajas resultantes están colocadas verticalmente de forma
contigua. Se devuelve el vector de referencias a esas cajas.
*/
public Box[] distributeVertically(int numRows) {
      assert numRows > 0;
      ???
}
```

- En la clase `Group`:

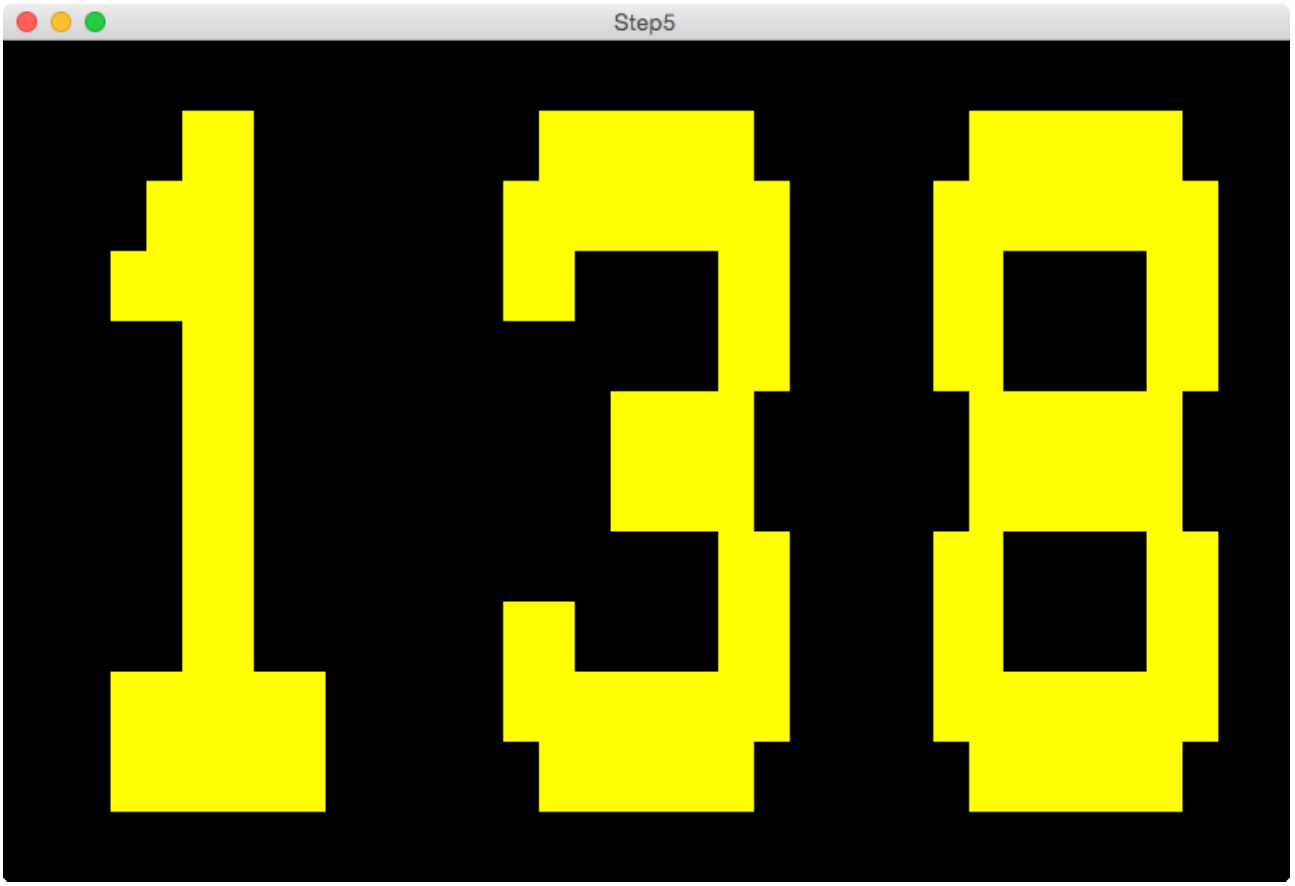
```
/* Construye un grupo en la caja indicada por groupBox, formada por
tantas barras como elementos tenga el array sizes. El array sizes
contiene el número de leds que tienen cada una de las barras.
Las barras están dispuestas, una encima de la otra, verticalmente.
El orden de los sizes se corresponde a los tamaños de las barras de
arriba hacia abajo.
*/
public Group(int[] sizes, Box groupBox) {
     ???
}
/* Añade todas las barras que forman el grupo al canvas gCanvas
*/
public void addToGCanvas(GCanvas gCanvas) {
     ???
}
/* Muestra, en cada una de las barras del grupo, el valor
correspondiente del array values.
El orden de los elementos en el vector values es de abajo hacia
arriba. De esta manera, si queréis mostrar el valor de un número a
partir de sus cifras, la cifra correspondiente a las unidades
estará en la posición 0 del vector values.
*/
public void render(int[] values) {
     ???
}
```

Step 4: Probando el dígito decimal



Este paso os lo damos también hecho.

Step 5: Probando un número decimal



En este paso debéis programar:

- En la clase `DecimalNumber`:

```
/* Construye un número decimal formado por numDigits dígitos. El
número ocupará el espacio indicado por dnBox. Las dígitos están
dispuestos, uno al lado del otro, de forma horizontal.
*/
public DecimalNumber(int numDigits, Box dnBox) {
      assert numDigits > 0;
      ???
}
/* Añade los dígitos al canvas indicado por gCanvas
*/
public void addToGCanvas(GCanvas gCanvas) {
      ???
}
/* Muestra el número indicado por number usando los diferentes
dígitos de que consta el número.
*/
public void render(int number) {
      assert number >= 0;
      ???
}
```

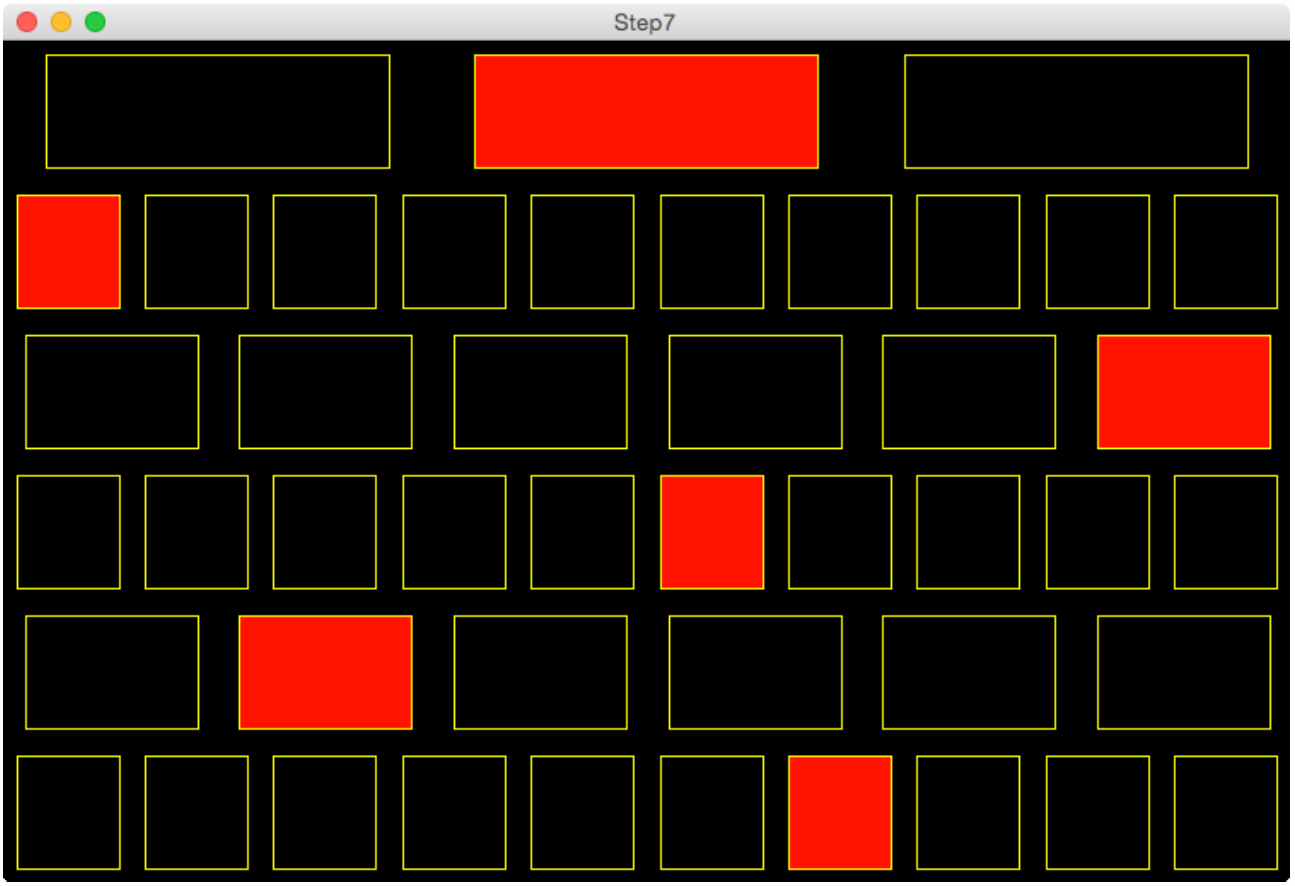
Step 6: Probando la hora decimal

En este paso debéis programar:

- En la clase `DecimalClock`:

```
/* Construye un reloj que usa tres números decimales para indicar
horas, minutos y segundos y lo coloca en la caja indicada por dcBox
*/
public DecimalClock(Box dcBox) {
      ???
}
/* Añade los números al canvas indicado por gCanvas.
*/
public void addToGCanvas(GCanvas gCanvas) {
      ???
}
/* Muestra la hora indicada
*/
public void render(int hours, int minutes, int seconds) {
      assert 0 <= hours   &&   hours <= 23;
      assert 0 <= minutes && minutes <= 59;
      assert 0 <= seconds && seconds <= 59;
      ???
}
```

Step 7: Probando la hora con leds

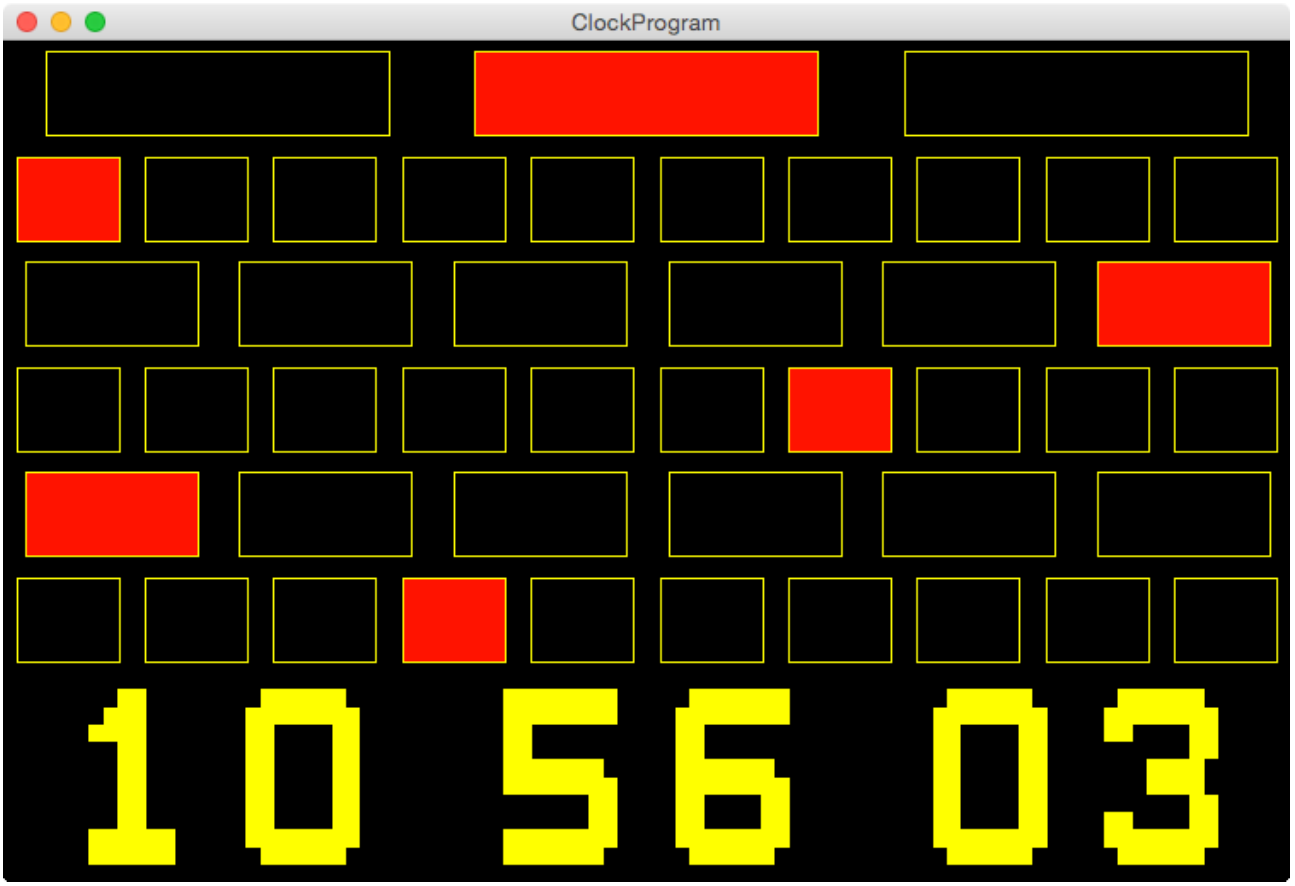


En este paso debéis programar:

- En la clase `LedsClock`:

```
/* Construye un reloj que usa tres grupos de barras de leds para
indicar horas, minutos y segundos y lo coloca en la caja indicada
por lcBox
*/
public LedsClock(Box lcBox) {
      ???
}
/* Añade los grupos de barras de leds al canvas gCanvas.
*/
public void addToGCanvas(GCanvas gCanvas) {
      ???
}
/* Muestra la hora indicada
*/
public void render(int hours, int minutes, int seconds) {
      assert 0 <= hours   &&   hours <= 23;
      assert 0 <= minutes && minutes <= 59;
      assert 0 <= seconds && seconds <= 59;
      ???
}
```

ClockProgram: el producto final



En este paso tan sólo debéis programar:

- En la clase `Box`:

```
/* Parte la caja correspondiente al objeto receptor en dos, una
encima de la otra. La de encima se queda con una altura
correspondiente al ratio y el resto será la altura de la que queda
debajo.
Es decir, si el ratio es 0.25, la caja superior tendrá una cuarta
parte de la altura total y la inferior tres cuartas partes. El
vector que se devuelve como resultado consta de dos referencias: la
de la caja superior seguida de la de la caja inferior.
*/
public Box[] splitTopBottom(double ratio) {
     assert 0.0 <= ratio && ratio <= 1.0;
     ???
}
```

Lo que debéis entregar

Debéis entregar el directorio con el código fuente Java de vuestra solución (borrad todos los ficheros `.class` antes de entregar para que ocupe el mínimo espacio posible) y un informe en PDF que explique (usando diagramas si es necesario) cómo habéis diseñado las clases y los métodos que se os piden. Comprimid ambas cosas en un único archivo `.tar.gz`, `.tar.bz2` o `.zip`.