Problemas de Programación Orientada a Objetos

Problema 1.

Haced un programa usando las bibliotecas gráficas del paquete acm que haga lo siguiente:

- Escriba un rectángulo pintado de rojo en el centro de la pantalla
- Mueva dicho rectángulo de izquierda a derecha de manera que cuando llegue a uno de los extremos de la pantalla, "rebote" y empiece a moverse en el sentido contrario

Podéis hacer varias versiones:

- Cambiando la posición del rectángulo con el método `setLocation`
- Cambiando la posición del rectángulo con el método `move`

Usad constantes para los valores del tamaño del cuadrado, la duración de la pausa que hacéis para que el movimiento sea suave, el número de píxeles que os movéis a cada paso, etc.

Problema 2.

Basado en lo que habéis aprendido en el problema anterior, vamos a simular una carrera de tortugas. Cada tortuga estará representada por un círculo y el número de tortugas estará representado por la constante `NUM_TURTLES`. Las tortugas serán todas verdes y empezarán en la posición de más a la izquierda. El radio de las tortugas será tal que las tortugas quepan verticalmente en la pantalla. Cada tortuga solamente se moverá horizontalmente y, a cada paso, se moverá tantos píxeles como el valor de lanzar un dado. Para ello usaréis el siguiente método:

```
1 public int rollDie() {
2   RandomGenerator rgen = RandomGenerator.getInstance();
3   return rgen.nextInt(1, 6);
4 }
```

Para usarlo deberéis importar la clase `acm.util.RandomGenerator`. Podéis encontrar información sobre esta clase en los apuntes, así como un ejemplo de su uso).
Cuando una tortuga llegue al final de la pantalla por la derecha, cambiará su color a rojo y el programa acabará.

Problema 3.

Hay figuras que pueden describirse recursivamente. Una de las más conocidas es el triángulo de Sierpinski, del que ya hemos hablado en el apartado de recursividad. Lo que haremos en este problema es definir un método recursivo que dibuje en la pantalla dicho triángulo.
Para ello procederemos paso a paso haciendo sucesivas aproximaciones al problema.

Fase 1

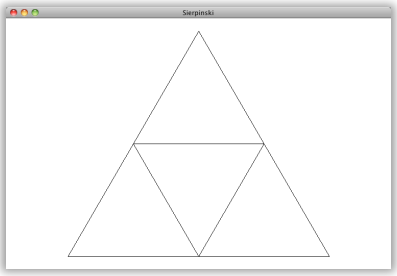
Inicialmente haremos una función tal que dibuje un triángulo equilátero (con el lado inferior alineado horizontalmente) tal que su vértice inferior izquierdo esté en las posición marcada por las coordenadas `x` e `y`, y la longitud del lado sea `side`, es decir:

```
1 public void addTriangle(double x, double y, double side) {
2   ¿?
3 }
```

Haced un programa principal que haga que la altura del triángulo sea el 90% de la altura de la pantalla y que esté centrado.

Fase 2

Utilizando la función anterior haced un método que usando el método anterior (dibujando tres triángulos) realice el siguiente dibujo:



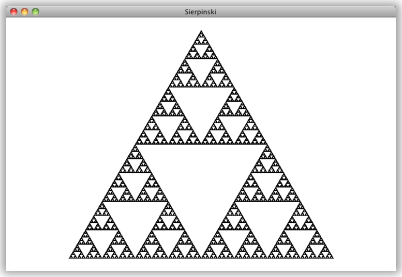
Para posicionar la figura, usaremos los mismos parámetros que en el caso anterior, es decir, las coordenadas del vértice inferior izquierdo y el lado del triángulo total.
Tened en cuenta que la figura parece descompuesta en cuatro triángulos pero solamente hemos de dibujar los tres triángulos externos.
Fijaos en que la descomposición en tres triángulos pequeños ocupa el mismo espacio que el triángulo de la fase 1.

Fase 3

Ahora es cuando haremos recursividad. Para ello definiremos el método:

```
1 public void sierpinski(double x, double y, double side, int level)
  {
2   ¿?
3 }
```

Dónde `x`, `y` y `side` representan lo mismo que en los casos anteriores, y `level` es el parámetro que indicará las llamadas recursivas a hacer. Cuando `level` valga 1, el método sierpinski dibujará un único triángulo. Cuando sea mayor que uno, lo que hará es la misma descomposición que hemos hecho en la fase 2, pero en vez de llamar al método que dibuja un triángulo, llamará recursivamente al método sierpinski decrementando `level` en una unidad.
El resultado de llamar al método con un `level` inicial de 10 es:



Problema 4.

Uno de los más simples sistemas de cifrado es el llamado Cifrado de César, que consiste en remplazar cada letra de un mensaje por otra que está a una distancia fija hacia delante (en orden alfabético) de la letra inicial. Por ejemplo, si la distancia es 3, 'A' se substituiría por 'D', 'B' por 'E', 'C' por 'F', 'X' por 'A', 'Y' por 'B', etc.
Lo que se pide es un método tal que dado una cadena y un entero, devuelva la cadena cifrada correspondiente a cifrar la cadena inicial usando el entero como distancia. Es decir:

```
1 public String encodeCaesar(String message, int distance) {
2   ¿?
3 }
```

Tened en cuenta lo siguiente:

- Los caracteres que no se corresponden con letras se mantienen igual.
- Las letras mayúsculas se traducen en letras mayúsculas.
- Las letras minúsculas en minúsculas.
- Si el entero es negativo, en vez de sustituir por caracteres más adelante en el alfabeto, lo hacemos por caracteres anteriores en el mismo.

- Haced un programa que haga lo siguiente: que pida un valor de distancia, un texto, lo cifre y muestre el resultado y lo vuelva a cifrar (cambiando de signo la distancia) para que se vea de nuevo el mensaje original.

Problema 5.

Implementad un método que dada una línea de texto, devuelva la palabra más larga que contiene. En el texto, además de palabras puede haber espacios, símbolos de puntuación (punto, punto y coma, dos puntos y coma), números (un número no cuenta como palabra), es decir:

```
1 public String longestWord(String line) {
2   ¿?
3 }
```

Pista: usad la clase `StringTokenizer` descrita en los apuntes.

Problema 6.

Un acrónimo consiste en coger las letras iniciales de un texto , convertirlas en mayúscula y separarlas por puntos. Por ejemplo, "Escuela Politécnica Superior" se convierte en "E.P.S.".
Lo que se pide es un método que, dada una línea, la convierta en acrónimo. Es decir:

```
1 public String makeAcronym(String line) {
2   ¿?
3 }
```

Para complicarlo, palabras como "el", "la", "un", "una", "unos", "y", "o", "de" (entre otras que se os pueden ocurrir) no se han de considerar. Por ejemplo, "Estados Unidos de América" quedaría como "E.U.A".
Pista: usad la clase `StringTokenizer` descrita en los apuntes.

Problema 7.

Suponed que la clase `ConsoleProgram` no proporcionara el método readInt y solamente tuviéramos disponible `readLine`. Lo que se pide es que implementéis el método:

```
1 public int myReadInt(String prompt) {
2   ¿?
3 }
```

que simula el funcionamiento de `readInt`, escribiendo el `prompt`, leyendo los caracteres y convirtiendo la cadena leída al número entero que representa.
Tened en cuenta lo siguiente:

- El número puede empezar por el carácter '-' para indicar que es negativo
- Si se encuentran caracteres que no sean dígitos el método muestra un mensaje indicando que hay caracteres ilegales y vuelve a mostrar el `prompt`, esperando una nueva entrada.

Una vez hecha esta versión, los más audaces podéis probar con:

```
1 public double myReadDouble(String prompt) {
2   ¿?
3 }
```

dónde, para simplificar, podéis considerar solamente los valores usando notación decimal, es decir "-23,48".
¿Alguien se atreve con los números expresados en notación científica "-0,2348E+2"?

Problema 8.

Los números racionales. Diseñad e implementad una clase para representar números racionales, es decir, fracciones de números enteros (numerador y denominador). Los métodos que dicha clase deberá tener como mínimo son:

- Un constructor que permita crear un racional a partir del numerador y del denominador.
- Métodos para la suma, resta, multiplicación y división entre racionales. Haced también versiones estáticas que devuelvan siempre un nuevo racional.
- Un método denominado toString() que devuelva una cadena que represente ese racional.

De cara a mantener el número racional lo más simple posible, internamente usaremos el mcd para conseguir que el racional que nos quede sea irreducible, es decir, que numerador y denominador sean primos entre sí.
Por ejemplo, si hacemos:

```
1 ...
2 Rational r = new Rational(4, -8);
3 println(r.toString());
```

Lo que obtendremos en la salida será -1/2. Tened en cuenta que si el denominador es 1, al hacer toString(), solamente deberéis usar el numerador.
Pistas:

- Mantened siempre el signo en el numerador, es decir, si el racional es positivo, tanto numerador como denominador lo serán; y si es negativo, solamente el numerador lo será.
- Para pasar un entero a String, podéis usar el método estático String.valueOf(int x).
- No hace falta que comprobéis que el denominador es diferente de 0.

Problema 9.

Los números complejos[1]. Diseñad e implementad una clase para los números complejos. Dicha clase, además de las operaciones de suma, resta, multiplicación y división, deberá poder obtener tanto la representación del número complejo en coordenadas cartesianas (parte real y parte imaginaria) como en coordenadas polares (módulo y argumento). Para ello, definiréis getters y setters para cada una de estas propiedades. Definid también operaciones para las operaciones aritméticas básicas, así como un método toString() para obtener una cadena que represente al número complejo.

Problema 10.

Tortugas 2.0. Una vez hemos aprendido a hacer clases, vamos a mejorar nuestra solución de la carrera de tortugas. Concretamente querremos encapsular todo lo correspondiente al dibujo y movimiento de la tortuga en la clase Turtle.
En una primera fase, implementaremos la clase tortuga de la siguiente manera:

- La tortuga tendrá un constructor que reciba un double, que representará su coordenada y, y un double que representará su tamaño.
- Tendrá un método move() que lanzará un dado y la moverá tantas posiciones como indique el dado.

[1] Podéis consultar http://es.wikipedia.org/wiki/Número_complejo para refrescar conocimientos básicos sobre los números complejos.

- Tendrá un método getRightExtrem() que devolverá la posición más a la derecha de la tortuga (para que el programa principal pueda saber la ganadora).
- Tendrá un método declareWinner() para que el programa principal indique a la tortuga que ha ganado (y ésta puede celebrarlo cambiando de color).

El método que queda necesita un poco de explicación (a modo de pista) y lo detallaremos a continuación. El problema que tenemos que resolver es cómo añadir al lienzo la tortuga.

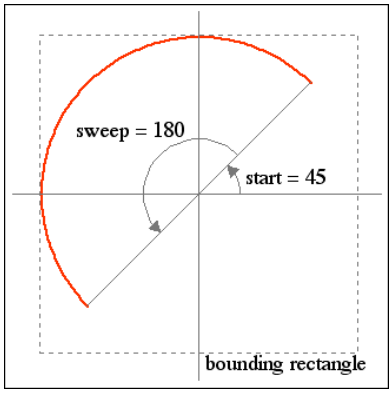
- Antes era fácil, ya que desde la clase del programa principal podemos hacer add y disponíamos del GOval que representaba la tortuga.
- Ahora, como el GOval es interno a la clase Turtle, todo resulta un poco más complicado. Para ello definiremos un método addToCanvas en la clase Turtle que reciba como parámetro un objeto de clase GraphicsProgram y, cuando lo llamemos desde el programa principal, pasaremos this como valor. Dentro del método tendremos todo lo que hace falta para añadir el GOval a la pantalla. Es decir:

```
public void addToCanvas(GraphicsProgram program) {
      program.add(this.oval);
}
```

Una vez completada esta fase inicial, en la que el programa se comportará como en la versión que ya teníamos, añadiremos una etiqueta a cada tortuga para representar su dorsal. Para ello, el constructor de la clase Turtle recibirá un tercer parámetro de tipo entero que será el número de dorsal, el cual permitirá crear un GLabel que moveremos junto con el GOval.

Problema 11.

El PacMan. Es fácil hacer una figura parecida al PacMan utilizando la clase GArc. Por eso lo que pediremos es un pelín más complicado: implementad la clase PacMan de manera que cuando éste se mueva de izquierda a derecha de la pantalla, vaya abriendo y cerrando la boca.



Los parámetros para construir un GArc son:

- **x, y** coordenadas del punto superior izquierdo del rectángulo que contiene al arco
- **w, h** anchura y altura de dicho rectángulo
- **start** ángulo inicial (en grados) desde la horizontal del principio del arco

- **sweep** ángulo que barre el arco (en grados)

Para la boca del pacman, start será 45 y sweep 270 (360-90). Para cambiar start y sweep (y conseguir que parezca que el PacMan abre y cierra la boca) dispondremos de los métodos setStartAngle y setSweepAngle.

Como siempre id aproximando la solución poco a poco:

a) Primero dibujar el PacMan con la boca abierta.
b) Moverlo de izquierda a derecha, hasta llegar al extremo derecho de la ventana.
c) Que abra y cierre la boca mientras se mueve.
d) Que una vez llega al extremo derecho, que empiece a moverse hacia la izquierda (similar al problema del rectángulo que se movía). Obviamente la boca estará en la dirección de movimiento.

Problema 12.

Una vez sabemos crear un PacMan y moverlo ya podemos implementar la carrera de PacMans, modificando nuestra carrera de tortugas (versión 2.0). Como en el caso de las tortugas, añadid un número de dorsal a cada PacMan.

Fijaos en lo fácil que es combinar las cosas si éstas están bien estructuradas.