Ejercicios propuestos del Tema 5

Aquí va la batería de ejercicios propuestos. Como en temas anteriores, sigo recomendándote que primero trates de hacerlos tú. No es necesario que los hagas todos, solo los que te parezcan más interesantes. También puedes construir otros programas que surjan de tus intereses. Recuerda que mantenerse motivado es fundamental.

Para plantear las soluciones, puedes utilizar indistintamente arrays convencionales o colecciones de tipo ArrayList o Vector.

Como en los temas anteriores, podrás encontrar una selección de estos ejercicios resueltos en Moodle Centros, donde iremos subiendo todos los que hagamos en clase y alguno más...

Ejercicio 5.1: Paquete prueba

Si no lo has hecho antes, crea el paquete Utilidades.prueba que se describe en el tema así como un pequeño programa que compruebe que funciona bien.

Ejercicio 5.2: Paquete matemáticas (*)

Crea ahora el paquete Utilidades.matematicas con dos clases llamadas Sumar y Potenciar.

La clase Sumar tendrá un método suma(int, int), sobrecargado como suma(double, double) para poder sumar números reales. La clase Potenciar tendrá un método potencia(int, int) sobrecargado como potencia (double, int), donde el primer parámetro será la base y el segundo el exponente.

Luego escribe una clase que permita probar el paquete para ver si todo resulta como debería. Por cierto, todos los chistes procaces en torno al doble sentido de la palabra "paquete" han sido hechos ya, así que no es necesario que te esfuerces en inventar ninguno nuevo. Dejaron de tener gracia en 1995.

Ejercicio 5.3: Clase Animal (*)

Escribe una clase Animal con los atributos nombre (String) y edad (int), con sus respectivos getters y setters. No, un setter no es una raza de perro en este contexto. Un getter no es una raza de perro en ningún contexto.

Crea un constructor que permita asignar valores a ambos atributos durante la creación del objeto.

Implementa para esta clase el método clone() y el método equals() heredados de Object para poder copiar y comparar en profundidad, y escribe un programa de prueba que lo demuestre.

Ejercicio 5.4: Contador de animales (*)

Para la clase Animal anterior, crea un atributo numAnimales (int) que cuente cuantos objetos de la clase Animal han sido creados, y comprueba que funciona correctamente.

Ejercicio 5.5: Constructor copia (*)

Añade un constructor copia a la clase Animal y (no debería ser necesario decirlo a estas alturas) comprueba que funciona.

Ejercicio 5.6: Herencia animal (*)

Crea dos clases llamadas Perro y Gato que deriven de la clase Animal. Añade a la clase Perro el atributo raza (String) con su getter y su setter. Añade a la clase Gato el atributo vidas (int) con un valor inicial de 7 (¡claro!) y dos métodos: getVidas() y restarVida(). Éste último restará una vida al gato, como es lógico.

Haz un programa que compruebe que todos los métodos de la clase madre (animal) están disponibles en Perro y Gato y funcionan correctamente.

Ejercicio 5.7: Adivinanza

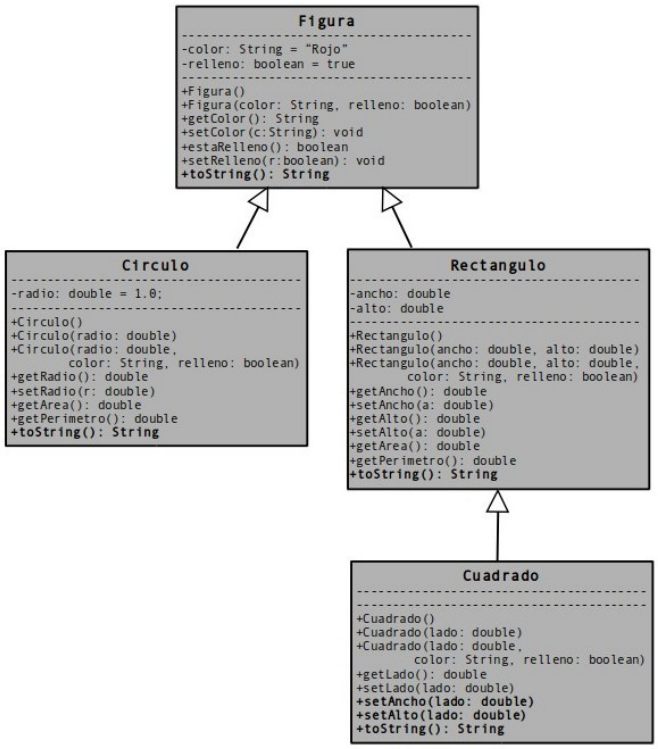
Blanco por dentro, verde por fuera. Que no, que es broma. Se trata de averiguar qué demonios mostrará este programa por pantalla antes de teclearlo y ejecutarlo en la máquina:

```
public class Adivinanza
{
   Adivinanza (int i)
   {
      this("Soy un enigma");
      System.out.println("Mi número es " + i);
   }
   Adivinanza (String s)
   {
      System.out.println(s);
   }
   void saluda()
   {
      System.out.println("Hola.");
   }
   public static void main(String[] args)
   {
      new Adivinanza(8).saluda();
   }
}
```

Ejercicio 5.8: Herencia geométrica (*)

En el tema poníamos un ejemplo clásico de herencia: la clase círculo y la clase rectángulo, que derivaban de una clase general llamada FiguraGeometrica.

Ahora vamos a partir de ese ejemplo para plantear un caso de herencia un poco más complejo de los vistos hasta ahora. Observa el diagrama de aquí abajo. Es un diagrama de clases un poco más complejo que los que hemos mostrado a lo largo del tema, pero no resulta difícil de entender. Simplemente, expresa gráficamente las relaciones de herencia entre las clases y muestra los atributos y métodos de cada una (por cierto: + significa public, y - significa private).



Por supuesto, podíamos haber añadido más figuras, pero con estas ya tienes entretenimiento para un rato. Observa, por cierto, como Cuadrado deriva de Rectángulo, y no directamente de Figura.

Ahora se trata de que hagas lo siguiente:

1) Escribir la superclase Figura (figura geométrica), que contendrá:

- Dos atributos de instancia: color (String) y relleno (boolean).
- Dos constructores: uno sin argumentos que inicializará el color a "rojo" y relleno a true; y otro que recibirá dos argumentos para inicializar los atributos.
- Getters y setters.
- Sobreescritura del método toString() de Object para devolver la cadena: "Soy una figura de color xxx y rellena/no rellena".

2) Escribe una clase que compruebe que los métodos de Figura funcionan.

3) Escribe dos subclases de Figura llamadas Círculo y Rectángulo. La clase Circle contendrá:

- Una variable llamada radio (double).
- Tres constructores, como se ve en el diagrama de clases. El constructor sin argumentos establecerá el radio en 1.
- Getter y setter para el radio.
- Los métodos getArea() y getPerimetro(). Si no recuerdas como calcular el área y el perímetro (o circunferencia) de un círculo… bueno, tal vez necesites tomar algo para mejorar tu memoria. En cualquier caso, ¿para qué está la wikipedia?
- Sobreescribe el método toString() heredado de Figura. Ahora, el método devolverá: "Soy un círculo con radio = x, esta es mi superclase: yyy", donde yyy es la salida del método toString() de la superclase de Círculo.

4) La clase Rectángulo se comportará igual, con las lógicas diferencias en atributos y métodos getters y setters. Mira el diagrama de clases si tienes alguna duda.

5) Escribe una clase llamada Cuadrado como subclase de Rectángulo. Esta clase podía haberse modelado como subclase de Figura, pero es más cómodo hacerlo como subclase de Rectángulo porque podemos aprovechar casi todo el código de su superclase. Basta con crear el siguiente constructor:

```
public Square(double side)
{
   super(side, side);  // Llama a la superclase Rectangle(double, double)
}
```

- Además de crear el constructor, sobreescribe, como en los otros casos, el método toString().
- Atención, pregunta: ¿necesitarás sobreescribir getArea() y getPerimetro(), o funcionarán tal y como han sido heredados de Rectángulo? Haz la prueba a ver qué pasa...
- Sobreescribe los setters setAlto() and setAncho() para evitar que el largo y el ancho del cuadrado puedan tener dimensiones diferentes.

6) Finalmente, escribe una clase de prueba que testee que todo lo anterior funciona como se espera.

Ejercicio 5.9: El rector

Crea una clase Profesor con los atributos nombre y sueldo y los getters y setters correspondientes.

Crea luego una subclase Rector. Los directores cobran un 25% más aunque realicen el mismo trabajo. Sobreescribe el método getSueldo() en Rector y comprueba que funciona.

Ejercicio 5.10: Colores (*)

Escribe dos clases, Blanco y Negro, dentro de una clase Color (clases anidadas).

Crea dos métodos pintaBlanco() y pintaNegro() dentro de Color, que se encarguen de instanciar las clases Blanco y Negro. Estas últimas mostrarán un mensaje en su constructor, para comprobar que funcionan.

Ejercicio 5.11: Calculadora

Utiliza wrappers para escribir una clase que lea por teclado dos números y muestre la suma, la multiplicación, la división y el módulo. En el caso de que el segundo número sea 0, habrá que manejar la excepción aritmética.

Ejercicio 5.12: Formas

Implementa una jerarquía de clases constituida por una clase Forma (abstracta) de la que hereden Circulo, Cuadrado, Triangulo y Rombo.

La clase Forma tendrá un método abstracto toString() y otro identidad(), que mostrará un identificador interno del objeto. Las demás clases heredarán de Forma e implementarán el método abstracto toString()

Después, prueba la jerarquía con este código, pero, cuidado, contiene varios errores. Arréglalo para que funcione.

```
class TestForma {
   public static void main(String[] args) {
      Forma f = new Circulo();
      f.indentidad();
      Circulo c = new Circulo();
      ((Forma) c).identidad();
      ((Circulo) f).identidad();
      Forma f2 = new Forma();
      f2.identidad();
      (Forma)f.identidad();
   }
}
```

Ejercicio 5.13: Hidden class

Crea una clase privada que implemente un interfaz público. Escribe un método que devuelva una referencia a una instancia de la clase privada haciendo upcasting al interfaz. Comprueba entonces que la clase está completamente oculta intentando hacer downcasting hasta ella.