Samsung
TECH INSTITUTE

MÓDULO 2. PROGRAMACIÓN ORIENTADA A OBJETOS CON JAVA

Relación de Problemas Nº 2

Proyecto prTesoro (composición, paquetes)

Un mapa de un tesoro tenía las siguientes indicaciones.

En la playa de la isla Margarita hay tres palmeras, una con una marca amarilla, otra con una marca azul y una tercera con una marca rosa. Las tres palmeras permiten localizar un tesoro escondido en la arena. Para ello, deben seguirse las siguientes instrucciones:

- *Desde la palmera rosa avanzar en línea recta hasta la amarilla contando el número de pasos. Una vez en la palmera amarilla, girar 90 grados en sentido contrario a las agujas del reloj y avanzar en línea recta el mismo número de pasos antes contado. Clavar una estaca (la llamaremos estaca amarilla) en el lugar alcanzado.*
- *Volver a la palmera rosa y repetir el procedimiento caminando ahora hacia la palmera azul pero girando los 90 grados en sentido de las agujas del reloj. Clavar también una estaca (la llamaremos estaca azul) en el lugar alcanzado.*
- *El tesoro se encuentra en la mitad del camino entre la estaca amarilla y la azul.*

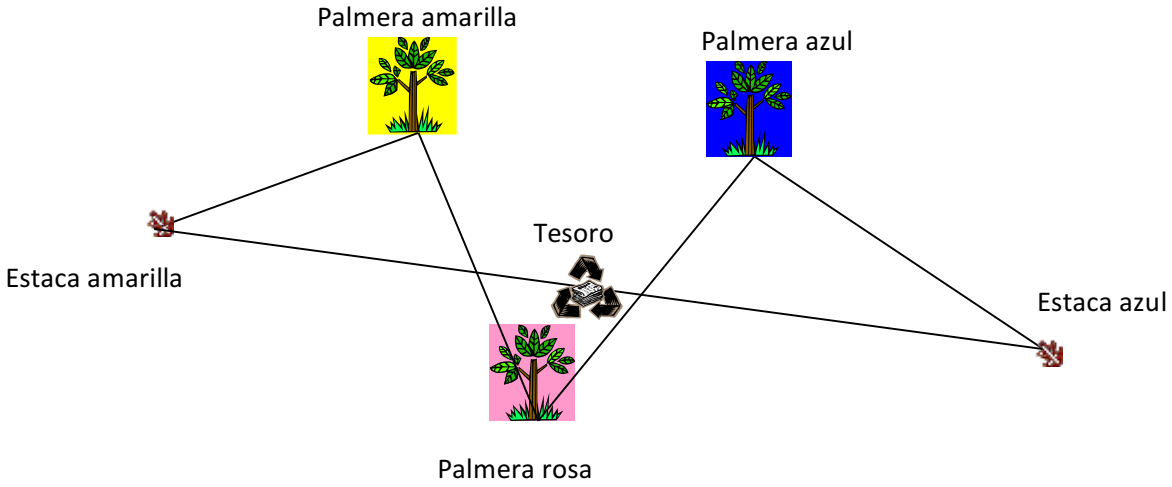
a) Realizar la clase `Tesoro` que almacena información de la posición de las tres palmeras, las dos estacas y la del tesoro. El constructor toma como argumentos tres puntos correspondientes a las posiciones de las palmeras amarilla, azul y rosa.

b) Definir el método `private void calculaPosiciones()` que calcula la posición de las estacas y del tesoro en función de las posiciones de las palmeras.

c) Definir los métodos `public void cambiaPalmeraAmarilla(Punto p)`, `public void cambiaPalmeraAzul(Punto p)`, y `public void cambiaPalmeraRosa(Punto p)` que cambia la posición de la palmeras, amarilla, azul y rosa respectivamente. Una vez cambiada la posición de la palmera, automáticamente se debe recalcular las posiciones de las estacas y del tesoro.

d) Definir los métodos `public Punto estacaAmarilla()` , `public Punto estacaAzul()` y `public Punto tesoro()` que devuelven la posición de la estaca amarilla, de la estaca azul y del tesoro respectivamente.

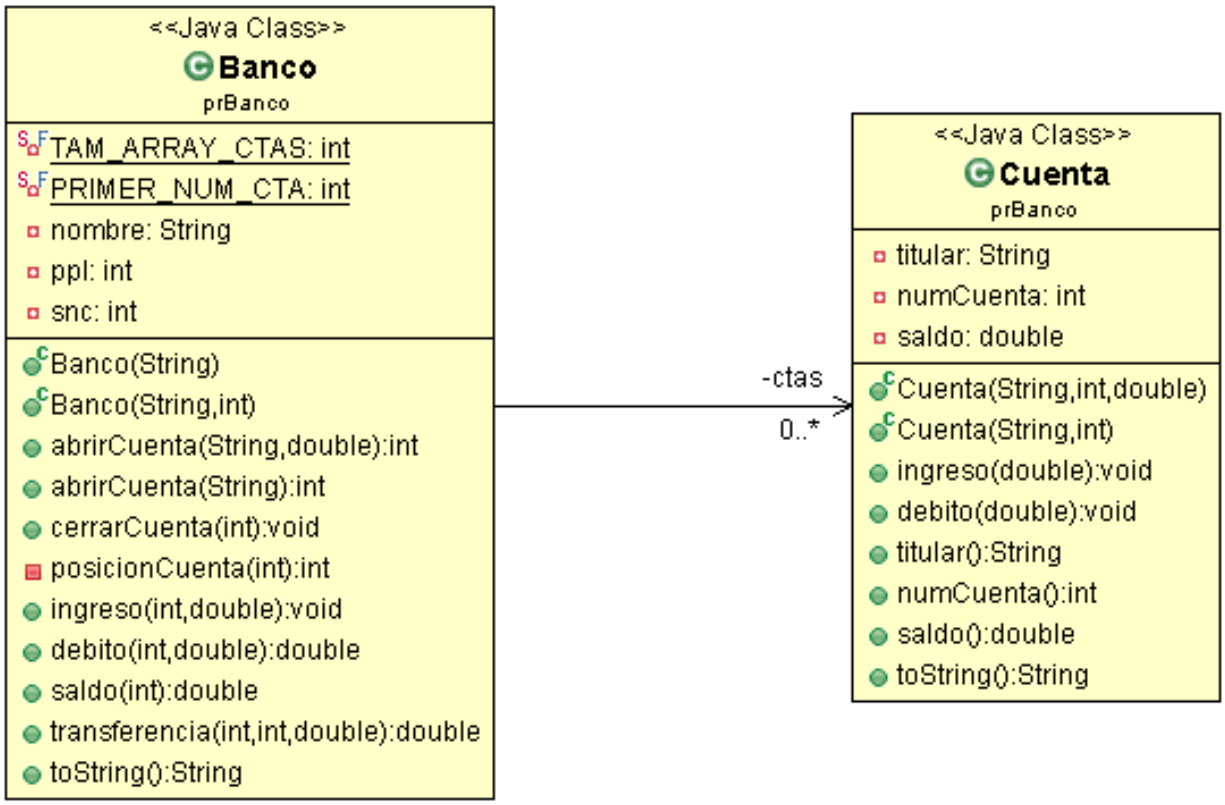
e) Realizar un programa java en el que le proporcionemos seis argumentos, los dos primeros generan la posición en el plano de coordenadas de la palmera amarilla, los dos siguientes la posición de la palmera azul y los dos últimos la de la palmera rosa. El programa debe crear un objeto de la clase `Tesoro` con los datos anteriores y mostrar la posición en la que se encuentra el tesoro.


Palmera amarilla
Palmera azul
Tesoro
Estaca amarilla
Estaca azul
Palmera rosa

Proyecto prBanco (arrays, excepciones)

El objetivo de esta práctica es el de crear clases para representar bancos y cuentas bancarias.

Defínase una clase `Cuenta` de cuentas bancarias. Toda cuenta bancaria vendrá dada por un número de cuenta (de tipo `int`), un titular de la cuenta bancaria (de tipo `String`) y un saldo (de tipo `double`). Sobre una cuenta bancaria se podrán realizar operaciones de ingreso (añadir dinero a la cuenta) y de débito (extraer una cantidad de dinero de la misma, no importando que el saldo quede negativo), además de operaciones de consulta del número de la cuenta, su titular y saldo.



Defínase una clase `Banco` para representar bancos. Un banco tendrá un nombre (dado en el constructor) y dispondrá de una colección de cuentas bancarias (representadas en un `array` de cuentas). Además, la clase `Banco` contendrá las variables de instancia `ppl` y `snc` que representan respectivamente la primera posición libre en el array de cuentas (están en posiciones contiguas) y el siguiente número de cuenta libre, de forma que una vez asignado el número a una nueva cuenta este contador se debe incrementar. El valor inicial de `snc` será el de la constante `PRIMER_NUM_CTA` (`1001`)

La clase Banco debe proporcionar métodos para llevar a cabo operaciones de ingreso, de débito o de consulta de saldo sobre cualquiera de sus cuentas dado su número de cuenta, así como una operación de transferencia entre dos cuentas bancarias indicando los números de cuenta de la cuenta origen y de la cuenta destino y la cantidad a transferir. Los métodos `public int abrirCuenta(String, double)` y `public cerrarCuenta(int)` se encargarán de crear una nueva cuenta bancaria y de cerrar una existente. Para abrir una cuenta bancaria bastará con proporcionar el nombre del titular de la cuenta (suponemos un único titular por cuenta) y un saldo inicial. Si no se indica saldo inicial la cuenta se creará con saldo cero. El banco se encargará de asignar un número de cuenta a la nueva cuenta, que será devuelto como resultado de la operación. A la hora de cerrar una cuenta existente,

después de su supresión las cuentas restantes habrán de ocupar posiciones contiguas en el array de cuentas.

Algunas cuestiones:

- Los constructores de la clase Banco se encargarán de crear el array de cuentas con un tamaño inicial, así como de dar valores iniciales a las variables auxiliares. El segundo argumento del constructor de Banco con dos argumentos indica el tamaño inicial del array de cuentas. En el constructor con un argumento el array se inicializará con un tamaño por defecto (10 definido en la constante `TAM_ARRAY_CTAS`).
- La creación de cuentas no debe fallar porque el array esté lleno; en caso de no quedar espacio en el array se creará un array (de doble capacidad) para habilitar más espacio.
- Un banco no permite que una cuenta quede con saldo negativo. Si se intenta realizar un débito por una cantidad mayor que el saldo, solo se permitirá extraer el saldo. En ese caso, el saldo quedará a 0 y se devolverá el dinero que realmente se ha extraído de la cuenta.
- Para facilitar la implementación de algunos métodos se debe implementar un método llamado `private int posicionCuenta(int)`. Éste es un método auxiliar que devuelve la posición dentro del array de cuentas en la que se encuentra la cuenta con el número de cuenta dado como argumento. Si la cuenta no existe, el método debe lanzar una excepción `RuntimeException` de la siguiente manera:
  `throw new RuntimeException("No existe la cuenta dada");`

- Se debe proporcionar una redefinición del método `toString` para las clases `Cuenta` y `Banco`.

Al ejecutar la siguiente clase de prueba TestBanco, el resultado ha de ser el mostrado más abajo.

```
import prBanco.Banco;

public class TestBancoA {
	public static void main(String[] args) {
		Banco b = new Banco("TubbiesBank", 5);
		int nPo = b.abrirCuenta("Po", 500);
		int nDixy = b.abrirCuenta("Dixy", 500);
		int nTinkyWinky = b.abrirCuenta("Tinky Winky", 500);
		int nLala = b.abrirCuenta("Lala", 500);
		int nNono = b.abrirCuenta("Nono");
		System.out.println(b);
		b.ingreso(nPo, 100);
		b.ingreso(nNono, 300);
		b.debito(nDixy, 100);
		b.transferencia(nTinkyWinky, nLala, 100);
		System.out.println(b);
		b.cerrarCuenta(nNono);
		b.cerrarCuenta(nPo);
		b.ingreso(nDixy,200);
		System.out.println(b);
	}
}
// salida:
// TubbiesBank: [[(Po/1001) -> 500.0] [(Dixy/1002) -> 500.0] [(Tinky Winky/1003) -> 500.0] [(Lala/1004) -> 500.0] [(Nono/1005) -> 0.0] ]
// TubbiesBank: [[(Po/1001) -> 600.0] [(Dixy/1002) -> 400.0] [(Tinky Winky/1003) -> 400.0] [(Lala/1004) -> 600.0] [(Nono/1005) -> 300.0] ]
// TubbiesBank: [[(Dixy/1002) -> 600.0] [(Tinky Winky/1003) -> 400.0] [(Lala/1004) -> 600.0] ]
```

Proyecto prUrna (enum, random, excepciones)

El objetivo de este ejercicio es crear una clase `Urna` cuyos objetos pueden contener bolas blancas y bolas negras, y que nos permita realizar unas operaciones básicas sobre la misma.

- La clase tendrá un par de variables de instancia, `negras` y `blancas`, en las que se almacenará el número de bolas de cada color.
- La clase `Urna` dispondrá de un constructor que permita crear instancias de la clase con el número inicial de bolas blancas y negras pasados como parámetros. El constructor deberá garantizar que no se crean urnas con un número negativo de bolas. En caso de que alguna sea negativa deberá lanzar la excepción `IllegalArgumentException`.
- Además, incluirá métodos para:
  - Consultar el número total de bolas que tiene (`public int totalBolas()`).
  - Extraer una bola aleatoriamente y saber su color (`public ColorBola extraeBola()`). El color vendrá dado por un enumerado `ColorBola` con dos elementos: `Blanca` y `Negra`. El tipo enumerado `ColorBola` debe definirse como una clase interna (anidada) estática a la clase `Urna`. Para extraer una bola aleatoriamente se ha de sumar el número de bolas blancas y negras y tomar un número aleatorio entre 1 y dicha suma. Si ese número es menor o igual que el número de bolas blancas supondremos que la bola que sale es blanca; en otro caso, que es negra. Utilizad la clase `java.util.Random` para la generación de números aleatorios. Consultad la documentación de la API para disponer de información sobre el uso de dicha clase. Si no hay bolas deberá lanzarse una excepción `NoSuchElementException`.
  - Introducir una bola de un color determinado (`public void ponBlanca()`, `public void ponNegra()` y `public void ponBola(ColorBola)`).

Cread una aplicación que cree una urna, tomando el número de bolas blancas y negras como argumentos al ejecutar la aplicación, y realice con ella el siguiente experimento:

- Mientras quede más de una bola en la urna, sacar dos bolas.
- Si ambas son del mismo color introducir una bola blanca en la urna; si son de distinto color introducir una bola negra (se supone que disponemos de suficientes bolas de ambos colores fuera de la urna).
- Por último, cuando quede sólo una bola, sacarla y mostrar su color.

Utilícese el método `parseInt` de la clase `Integer` en `java.lang` para convertir un `String` en un valor de tipo `int`.

Infórmese al usuario por la salida estándar de cualquier error que se pueda producir.

Analizad los resultados obtenidos sobre el color de la bola final dependiendo del número de bolas iniciales y de su color.