EIF207 Estructuras de Datos
Examen parcial

Prof. M.Sc. Georges Alfaro S.

Complete cada uno de los ejercicios solicitados.

Complete los ejercicios en Java, empleando el IDE NetBeans. Guarde las carpetas de proyecto en una sola carpeta con su nombre en el momento de hacer la entrega. Recuerde que los programas no deberán tener errores de sintaxis. Cualquier ejercicio que no compile correctamente, no será calificado. Debe utilizar la biblioteca de colecciones que se estudió en clase o una implementación propia, pero no deberá usar las colecciones de la biblioteca `java.util` (puede sin embargo usar otras clases que no implementen colecciones, como el `Iterator`, por ejemplo).

PARTE 1

Ejercicio #1 (20 pts.)

Defina una **subclase** de la clase `DQueue` implementada en el curso.

Implemente los métodos indicados, para extraer un subconjunto de elementos de la cola.

```
package cr.ac.una.collections;

public class SDQueue<T> extends DQueue<T> {

    // Este método obtiene de la cola todos los elementos para los cuales
    // la función isAcceptable() del fitro especificado es verdadera
    // (regresa un valor true).
    // La función regresa los elementos que cumplen con el criterio indicado
    // SIN ALTERAR la lista original. Es decir, los elementos se agregan
    // al resultado sin eliminarlos.
    //
    public SDQueue<T> select(Filter<T> f) {
        throw new UnsupportedOperationException();
    }

    // Este método extrae de la cola todos los elementos para los cuales
    // la función isAcceptable() del fitro especificado es verdadera
    // (regresa un valor true).
    // La función ELIMINA de la lista original los elementos que cumplen
    // con el criterio indicado.
    //
    public SDQueue<T> extract(Filter<T> f) {
        throw new UnsupportedOperationException();
    }

}
```

```
package cr.ac.una.collections;

@FunctionalInterface
public interface Filter<T> {

    public boolean isAcceptable(T obj);

}
```

Escriba un programa que lea el archivo de prueba '`datos (ejercicio 1).txt`', que contiene información de productos y sus precios, creando una cola de productos y efectúe luego las siguientes operaciones:

- mostrar una lista con todos los productos cuyo código inicie con la letra 'B', **sin borrar** ninguno de la lista original.
- mostrar una lista con todos los productos que tengan un precio mayor a 10.000, extrayéndolos (**eliminándolos**) de la lista original.

Ejercicio #2 (20 pts.)

La función trigonométrica seno se puede calcular por medio de una serie infinita:

$$sen(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots = \sum_{i=0}^{\infty} \frac{(-1)^i}{(2i+1)!} x^{2i+1}$$

que puede aproximarse con una precisión arbitraria calculando solamente una cantidad fijada de términos o hasta que la diferencia relativa entre dos resultados consecutivos de la secuencia sea menor que un valor de error determinado (ε).

$$t_i = \frac{(-1)^i}{(2i+1)!} x^{2i+1}$$

$$sen(x) = \sum_{i=0}^{\infty} t_i \approx f_n(x) = \sum_{i=0}^{n} t_i$$

$$\left| \frac{f_n(x) - f_{n-1}(x)}{f_n(x)} \right| < \varepsilon$$

Escriba una función que calcule el seno de un valor de tipo `double` de manera **recursiva**, con un valor máximo de error de 1.0×10^{-9} (nueve posiciones decimales exactas). **NO utilice** ninguna función para calcular el factorial o las potencias enteras de x (tampoco emplee la función `Math.pow()` para este efecto). Es importante que el tiempo de ejecución de la función de aproximación $f_n(x)$ se calcule en un tiempo $O(n)$, proporcional a la cantidad de términos de la sumatoria. Es decir, la función por implementar es **recursiva lineal**.

Construya un programa para probar la función con varios valores de muestra. Observe el ejemplo de salida de consola, utilizando números en el rango [-2.2, 2.0]. En este ejemplo, se compara el valor de la función obtenido con la función `Math.sin()` y la función solicitada, mostrando además la diferencia entre ambos valores.

```
    public static void main(String[] args) {
        double x0 = -2.2;
        while (x0 <= +2.0) {
            double y0 = Math.sin(x0);
            double y1 = sin(x0); // función de cálculo (recursiva)

            System.out.printf(
                    "f(%1$ 11.9f) = %2$ 11.9f, %3$ 11.9f "
                    + "; error = %4$ 6.4g%n",
                    x0, y0, y1, Math.abs((y1 - y0) / y1));
            x0 += 0.15;
        }
    }
```

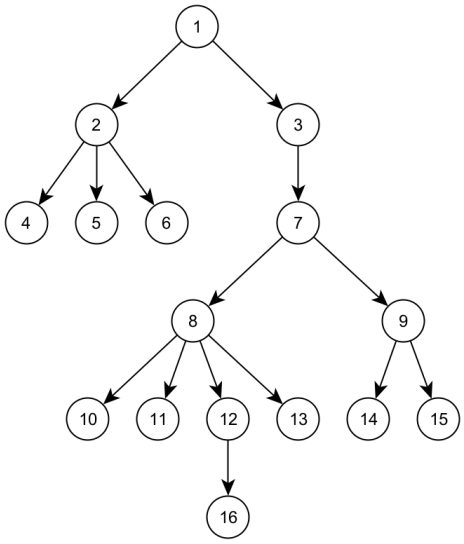
```
f(-2,200000000) = -0,808496404, -0,808496404 ; error =  3,223e-11
f(-2,050000000) = -0,887362369, -0,887362368 ; error =  6,242e-10
f(-1,900000000) = -0,946300088, -0,946300088 ; error =  1,611e-10
f(-1,750000000) = -0,983985947, -0,983985947 ; error =  3,835e-11
f(-1,600000000) = -0,999573603, -0,999573604 ; error =  8,738e-10
f(-1,450000000) = -0,992712991, -0,992712991 ; error =  2,013e-10
f(-1,300000000) = -0,963558185, -0,963558185 ; error =  4,037e-11
f(-1,150000000) = -0,912763940, -0,912763940 ; error =  6,784e-12
f(-1,000000000) = -0,841470985, -0,841470985 ; error =  1,899e-10
f(-0,850000000) = -0,751280405, -0,751280405 ; error =  2,576e-11
f(-0,700000000) = -0,644217687, -0,644217688 ; error =  7,665e-10
f(-0,550000000) = -0,522687229, -0,522687229 ; error =  6,664e-11
f(-0,400000000) = -0,389418342, -0,389418342 ; error =  2,695e-12
f(-0,250000000) = -0,247403959, -0,247403959 ; error =  4,247e-11
f(-0,100000000) = -0,099833417, -0,099833417 ; error =  1,987e-10
f( 0,050000000) =  0,049979169,  0,049979169 ; error =  3,101e-12
f( 0,200000000) =  0,198669331,  0,198669331 ; error =  7,099e-12
f( 0,350000000) =  0,342897807,  0,342897807 ; error =  6,327e-10
f( 0,500000000) =  0,479425539,  0,479425539 ; error =  2,547e-11
f( 0,650000000) =  0,605186406,  0,605186406 ; error =  3,613e-10
f( 0,800000000) =  0,717356091,  0,717356091 ; error =  1,227e-11
f( 0,950000000) =  0,813415505,  0,813415505 ; error =  1,009e-10
f( 1,100000000) =  0,891207360,  0,891207360 ; error =  6,185e-10
f( 1,250000000) =  0,948984619,  0,948984619 ; error =  2,277e-11
f( 1,400000000) =  0,985449730,  0,985449730 ; error =  1,199e-10
f( 1,550000000) =  0,999783764,  0,999783765 ; error =  5,429e-10
f( 1,700000000) =  0,991664810,  0,991664810 ; error =  2,326e-11
f( 1,850000000) =  0,961275203,  0,961275203 ; error =  1,008e-10
f( 2,000000000) =  0,909297427,  0,909297426 ; error =  4,006e-10
```

En este ejemplo, la función se calcula con un error relativo máximo de 1.0×10^{-9}.

Ejercicio #3 (20 pts.)

Para un árbol de orden arbitrario, escriba una función que encuentre el **ancestro común más bajo** (*LCA - Lowest Common Ancestor*) para dos nodos cualesquiera.

El ancestro común más bajo de dos nodos A y B es el nodo más bajo (más lejano a la raíz) que es ancestro de ambos nodos A y B. Cada nodo es un descendiente de sí mismo, de manera que, si el nodo B es accesible desde A, A es el ancestro común.



Por ejemplo:

```
LCA(11, 14) = 7
LCA(5, 6 ) = 2
LCA(3, 9) = 3
```

Observe que siempre es posible calcular el LCA para cualquier par de nodos.

En el programa de prueba, muestre el LCA de **todos** los pares de nodos del árbol de ejemplo.

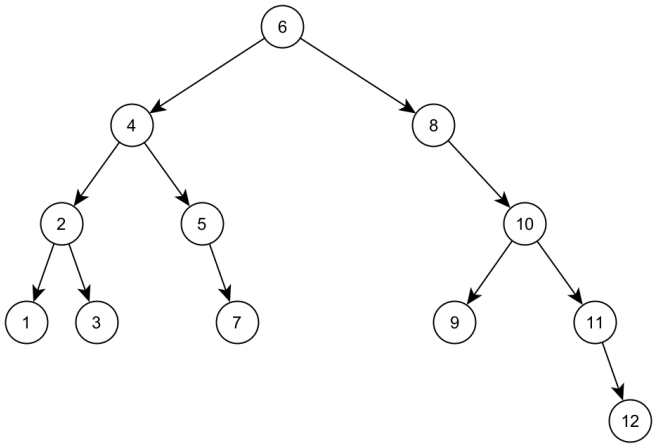
Ejercicio #4 (20 pts.)

Se dice que un árbol binario está balanceado (AVL balanceado) si, para cada nodo, sus dos subárboles están balanceados y la altura del subárbol izquierdo difiere de la altura del subárbol derecho como máximo en 1 (uno). Escriba una función para determinar si un árbol binario dado está o no balanceado.

Escriba un programa de prueba utilizando tanto árboles balanceados como no balanceados.

Ejercicio #5 (20 pts.)

Dado un árbol binario de búsqueda, escriba una función que determine si dos nodos dados son primos entre sí o no. Dos nodos de un árbol binario son **primos** entre sí solo si tienen **diferentes padres** (antecesores), pero están en el **mismo nivel**. Observe que el nivel del nodo está dado por la distancia (número de antecesores hasta la raíz) y no por su altura.



Por ejemplo:

(2, 10), (5, 10), (1, 7), (1,9), (1, 11), (3, 7), (3, 9), (3, 11), (7, 9), (7, 11) **son primos** entre sí.

(1, 3), (4, 8), (2, 5), (9, 11), etc., **no son primos** entre sí.

Escriba un programa de prueba utilizando el árbol mostrado arriba y otro definido por usted, listando todos los pares de nodos primos para cada uno. Si un par ordenado (a, b) corresponde a dos nodos primos, no es necesario (tampoco incorrecto) listar también el par ordenado (b, a). Observe que la relación `primos()` es simétrica, pero no reflexiva ni transitiva.