EDA Recuperación Primer Parcial - ETSInf - 22 de Junio de 2017. Duración: 2 horas.

1. $\boxed{\text{1 punto}}$ Escribe la subinterfaz `ListaConPIPlus` para que amplie la interfaz `ListaConPI` con un método que permita conocer el dato mayor de todos los contenidos en la lista.

Solución:

```
public interface ListaConPIPlus<E extends Comparable <E>> extends ListaConPI<E> {
    E maximo();
}
```

2. $\boxed{\text{3 puntos}}$ Un buscador mantiene una `ListaConPI<String>` con los términos (palabras) relevantes para la búsqueda y representa cada documento que considera como un `Map` que contiene los términos que aparecen en él con su frecuencia de aparición. Por ejemplo, si la lista de términos es: `heroes, day, dolphins, swim, life, sea, wind`; el Documento 1 contendrá los pares `(heroes, 2), (day, 1)` y el Documento 2 `(swim, 2), (dolphins, 2), (heroes, 1), (day, 1)`.

```
Documento 1:
We can be heroes
Just for one day
We can be heroes
```

```
Documento 2:
I, I wish you could swim
Like the dolphins, like dolphins can swim
Oh, we can be heroes, just for one day
```

Se pide:

a) (2 puntos) Escribir un método eficiente que, dados dos documentos `d1` y `d2` (representados con sendos `Maps`), y la lista con los términos, devuelva un array de tres posiciones que contenga en la posición 0 el número de términos que no están ni en d1 ni en d2, en la 1 el número de términos que están en ambos y en la 2 la suma de sus frecuencias. En el ejemplo anterior, el array resultante debería ser (3, 2, 5): hay 3 términos que no están ni en d1 ni en d2, `life`, `sea` y `wind`; hay 2 términos comunes, `heroes` y `day`, y la suma de sus frecuencias es 5 (3 veces aparece `heroes` y 2 `day`).

Solución:

```
public static int[] terminos(Map<String, Integer> d1, Map<String, Integer> d2, ListaConPI<String> terms) {
    int[] res = new int[3];
    for (terms.inicio(); !terms.esFin(); terms.siguiente()) {
        String t = terms.recuperar();
        Integer fd1 = d1.recuperar(t);
        Integer fd2 = d2.recuperar(t);
        if (fd1 == null && fd2 == null) { res[0]++; }
        else if (fd1 != null && fd2 != null) {
            res[1]++;
            res[2] += fd1 + fd2;
        }
    }
    return res;
}
```

b) (1 punto) Indicar la talla del problema, x, y el coste Temporal del método que has diseñado, utilizando la notación asintótica (O y Ω o bien Θ).

Solución: Talla del problema x = terms.talla(). Coste temporal asintótico: $T(x) \in \Theta(x)$.

3. $\boxed{\text{3 puntos}}$ Se desea añadir un nuevo método a la clase `ABB` para comprobar si un `ABB` dado es subárbol del objeto en curso. Supóngase que no hay elementos repetidos en el `ABB`. **Se pide**:

a) (1 punto) Escribir el código del método lanzadera que, utilizando el método protegido que se diseña en el apartado siguiente, resuelva el problema enunciado. Su perfil debe ser:
`public boolean esSubArbol(ABB<E>otro) {...}`

Solución:

```
public boolean esSubArbol(ABB<E> otro) {
    if (otro.raiz == null) { return true; }
    if (this.raiz == null) { return false; }
    return esSubArbol(otro.raiz, this.raiz);
}
```

b) (2 puntos) Escribir el código del método recursivo `protected`, que utiliza el método `esSubArbol` anterior, con el perfil siguiente:
`protected boolean esSubArbol(NodoABB<E>otro, NodoABB<E>actual) {...}`
Este método debe devolver `true` si `otro` es subárbol de `actual`. Se puede suponer que en la clase `ABB` se ha definido el método `iguales`, que devuelve `true` si el `ABB` cuya raíz es `a` es igual al `ABB` cuya raíz es `b`:

```
protected boolean iguales(NodoABB<E> a, NodoABB<E> b) { ... }
```

Solución:

```
// precondición: otro != null && actual != null
//               no hay elementos repetidos
protected boolean esSubArbol(NodoABB<E> otro, NodoABB<E> actual) {
    int x = actual.dato.compareTo(otro.dato);
    if (x == 0) { return iguales(otro, actual); }
    if (x < 0) { return actual.der != null && esSubArbol(otro, actual.der); }
    return actual.izq != null && esSubArbol(otro, actual.izq);
}
```

4. $\boxed{\text{3 puntos}}$ El siguiente método, `contarAprobados`, devuelve el número de alumnos aprobados que hay en el array `a` de tipo base `Alumno`:

```
/** precondición:
 *  a es un Alumno[] tal que:
    - a está ordenado descendentemente
    - el criterio de ordenación es la nota de los alumnos (alumno con la nota mayor
      en la posición 0 del array, alumno con la nota menor en la última posición)
    - a contiene, al menos, un alumno aprobado (cuya nota es mayor o igual a 5.00)
 */
public static int contarAprobados(Alumno[] a) {
   return contarAprobados(a, 0, a.length - 1);
}
```

Siendo la clase `Alumno`, la siguiente:

```
public class Alumno {
    private String nombre;
    private double nota;
    public Alumno(String s, double n) { nombre = s; nota = n; }
    public String getNombre() { return nombre; }
    public double getNota() { return nota; }
}
```

Se pide:

a) (2 puntos) Implementar, siguiendo la estrategia Divide y Vencerás, el método recursivo invocado en el anterior método lanzadera: `contarAprobados(Alumno[], int, int)`.

Solución:

```
private static int contarAprobados(Alumno[] a, int i, int f) {
    int res = 0;
    //Caso base: hay un alumno aprobado en el subarray [i..f]
    if (i == f) { if  (a[i].getNota() >= 5.0) { res = 1; } }
    // usando estrategia DyV, se divide el problema en 2 subproblemas
    if (i < f) {
        int m = (i + f) / 2;
        if (a[m].getNota() >= 5.0) {
            if (a[m + 1].getNota() < 5) { res = m - i + 1; }
            else res = (m - i + 1) + contarAprobados(a, m + 1, f);
        }
        // Si a[m] no es un alumno aprobado, el último aprobado ha de estar antes,
        // i.e. en el subArray a[i..m-1]
        else res = contarAprobados(a, i, m - 1);
    }
    return res;
}
```

b) (1 punto) Estudiar el coste Temporal del método (recursivo) que has diseñado. En concreto:
- Indica la talla del problema, x, en función de los parámetros del método
- Para una talla x dada, indica si existen instancias significativas; si las hubiera, indica cuáles son y por qué.
- Escribe las Relaciones de Recurrencia que requiera tu respuesta en el punto anterior; luego, usa los Teoremas de Coste para resolverlas y acotarlas.
- Finalmente, escribe el coste Temporal Asintótico del método, utilizando la notación asintótica (O y Ω o bien Θ):

Solución: Talla del problema x = f − i + 1
Mejor caso: el alumno en la posición central del subarray v[i, f] es el último de los aprobados.
Peor caso: todos están aprobados o todos suspendidos excepto el primero.
La ecuación de recurrencia del caso peor es: $T^p(x) = 1 * T^p(x/2) + k$
Luego, por Teorema 3 (sobrecarga constante), con a = 1 y c = 2, $T(x) \in \Theta(log2x)$.
Así, $T(x) \in \Omega(1)$ y $T(x) \in O(log2x)$.

ANEXOS

La interfaz `ListaConPI` del paquete `modelos`

```
public interface ListaConPI<E> {
          void insertar(E e);
          /** SII !esFin() */ void eliminar();
          void inicio();
          /** SII !esFin() */ void siguiente();
          void fin();
          /** SII !esFin() */ E recuperar();
          boolean esFin();
          boolean esVacia();
          int talla();
}
```

La interfaz `Map` del paquete `modelos`

```
public interface Map<C, V> {
    V insertar(C c, V v);
    V eliminar(C c);
    V recuperar(C c);
    boolean esVacio();
    int talla();
    ListaConPI<C> claves();
}
```

Las clases `ABB` y `NodoABB` del paquete `jerarquicos`

```
public class ABB<E extends Comparable<E>> {
    protected NodoABB<E> raiz;
    public ABB() { this.raiz = null; }
    ...
}
class NodoABB<E> {
    protected E dato;
    protected NodoABB<E> izq, der;
    int talla;
    NodoABB(E e) {
        this.dato = e;
        this.izq = null; this.der = null;
        talla = 1;
    }
}
```

Teoremas de coste

- **Teorema 1:** sea $f(x) = a * f(x-c) + b$, entonces

  si $a = 1$ $f(x) \in \Theta(x)$
  si $a > 1$ $f(x) \in \Theta(a^{\frac{x}{c}})$
- **Teorema 2:** sea $f(x) = a * f(x-c) + b * x + d$, entonces

  si $a = 1$ $f(x) \in \Theta(x^2)$
  si $a > 1$ $f(x) \in \Theta(a^{\frac{x}{c}})$
- **Teorema 3:** sea $f(x) = a * f(\frac{x}{c}) + b$, entonces

  si $a = 1$ $f(x) \in \Theta(\log_c x)$
  si $a > 1$ $f(x) \in \Theta(x^{\log_c a})$
- **Teorema 4:** sea $f(x) = a * f(\frac{x}{c}) + b * x + d$, entonces

  si $a < c$, $f(x) \in \Theta(x)$
  si $a = c$, $f(x) \in \Theta(x * \log_c x)$
  y si $a > c$, $T_{mR}(x) \in \Theta(x^{\log_c a})$