UNLP. Facultad de Informática.
**Algoritmos y Estructuras de Datos**
**Cursada 2014**

# Práctica 3
## Abstracción y encapsulamiento
## Herencia. Polimorfismo. Tipos Genéricos

**1.** Considere la siguiente especificación de operaciones de una lista de enteros:


a. Importe a Eclipse las clases dadas por la Cátedra. Analice la clase **ListaDeEnteros** y sus subclases. Podría ponerle comportamiento a algún método de la superclase ListaDeEnteros?

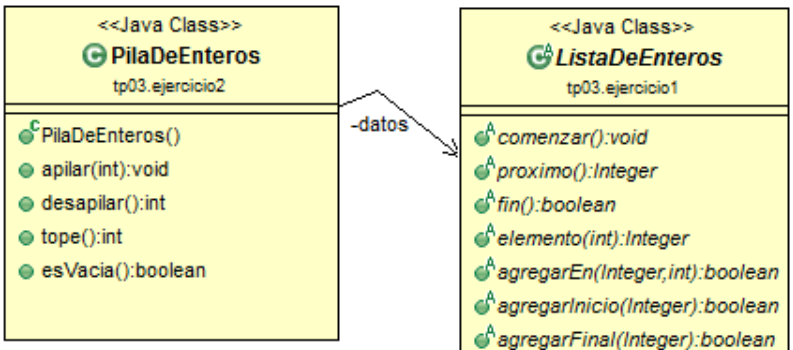
¿Por qué la clase se define como abstracta? Note que una subclase implementa la lista usando un arreglo de tamaño fijo y la otra usando nodos enlazados.

b. ¿Cuál es el motivo por el cual las subclases no compilan? Haga lo necesario para que las dos subclases compilen.

c. Escriba una clase llamada **TestListaDeEnterosConArreglos** que reciba en su método **main** una secuencia de números, los agregue a un objeto de tipo **ListaDeEnterosConArreglos** y luego imprima los elementos de dicha lista.

d. Escriba una clase llamada **TestListaDeEnterosEnlazada** que reciba en su método **main** una secuencia de números, los agregue a un objeto de tipo **ListaDeEnterosEnlazada** y luego imprima los elementos de dicha lista.

**2.** Sea la siguiente especificación de una pila de enteros:




**apilar(int elem):** Agrega elem a la pila.
**desapilar():** Elimina y devuelve el elemento en el tope de la pila.
**tope():** Devuelve el elemento en el tope de la pila sin eliminarlo.
**esVacia():** Retorna true si la pila está vacía, false en caso contrario.

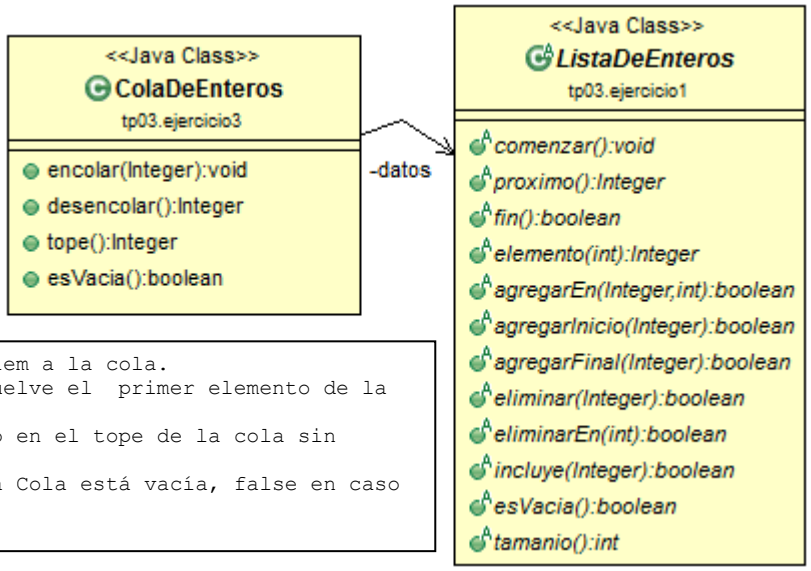
a. Implemente en JAVA (pase por máquina) la clase **PilaDeEnteros,** ubíquela en el paquete tp03.ejercicio2, de acuerdo a la especificación dada en el diagrama de clases. Utilice alguna subclase de la clase **ListaDeEnteros**.

b. Escriba una clase llamada **TestPilaDeEnteros** para ejecutar el siguiente código:

```
PilaDeEnteros p1, p2;
int valor2=0;
p1=new PilaDeEnteros();
p1.apilar(1);
p1.desapilar(2);
p2=p1;
valor2 = p2.desapilar();
System.out.println("El valor del tope de la pila p1 es:  " + p1.desapilar());
```

c. ¿Qué valor imprime? ¿Qué conclusión saca?

**3.** Sea la siguiente especificación de una cola de enteros:




**encolar(int elem)**: agrega elem a la cola.
**desencolar()**: elimina y devuelve el primer elemento de la cola.
**tope()**: devuelve el elemento en el tope de la cola sin eliminarlo.
**esVacia()**: etorna true si la Cola está vacía, false en caso contrario.

a. Implemente en JAVA (pase por máquina) la clase **ColaDeEnteros** de acuerdo a la especificación dada en el diagrama de clases. Defina esta clase adentro del paquete tp03.**ejercicio3** y use alguna de las subclases de ListaDeEnteros.

**4.** El algoritmo conocido como Criba de Eratóstenes permite la obtención de todos los números primos menores que un número dado. Y para conseguir esto se procede del modo siguiente:

- Haga una lista de todos los naturales desde 1 hasta un número n dado:

  *1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 … n*

- Marque el 2 como primer primo y tache de ahí en adelante todos sus múltiplos:

  *1 2 3 ~~4~~ 5 ~~6~~ 7 ~~8~~ 9 ~~10~~ 11 ~~12~~ 13 ~~14~~ 15 ~~16~~ 17 ~~18~~ 19 ~~20~~ 21 … n*

- De los no tachados que siguen a 2, el primero es el siguiente primo. Marque el 3 y tache todos los múltiplos de éste que no haya tachado en el paso anterior:

  *1 2 3 ~~4~~ 5 ~~6~~ 7 ~~8~~ ~~9~~ ~~10~~ 11 ~~12~~ 13 ~~14~~ ~~15~~ ~~16~~ 17 ~~18~~ 19 ~~20~~ ~~21~~ … n*

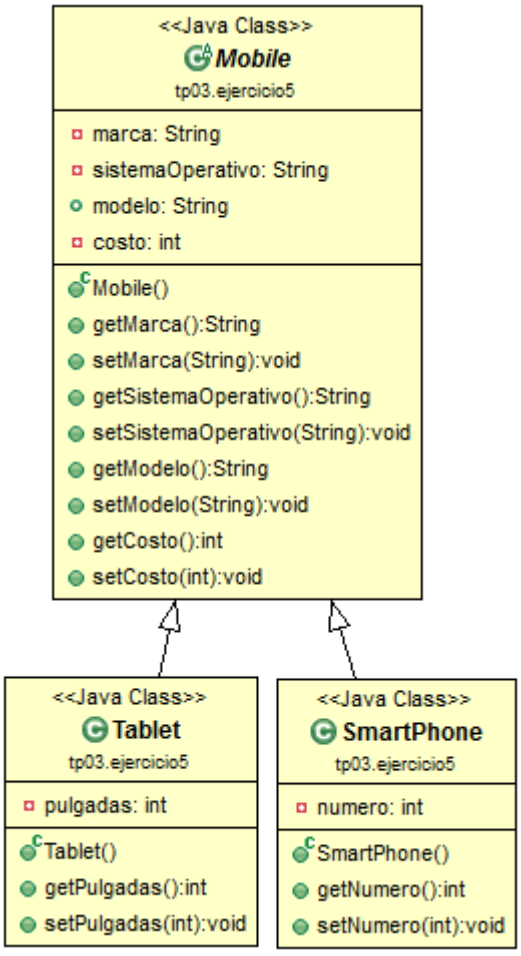
- Continúe de esta manera hasta haber marcado un número mayor que la raíz cuadrada de n y entonces marque todos los números mayores que 1 que hasta ese momento hayan permanecido sin ser tachados:

  *1 2 3 ~~4~~ 5 ~~6~~ 7 ~~8~~ ~~9~~ ~~10~~ 11 ~~12~~ 13 ~~14~~ ~~15~~ ~~16~~ 17 ~~18~~ 19 ~~20~~ ~~21~~ … n*

Los números marcados son todos los primos entre 1 y n.

a. Escriba una clase llamada **tp03.ejercicio4.CribaDeEratostenes** como un método llamado **obtenerPrimos()** que tome como parámetro un objeto de tipo **ListaDeEnteros** que contenga los primeros 1000 números naturales y retorne la lista de los primos correspondientes siguiendo el procedimiento antes descrito.

**5.** Defina la siguiente jerarquía de clases en JAVA.

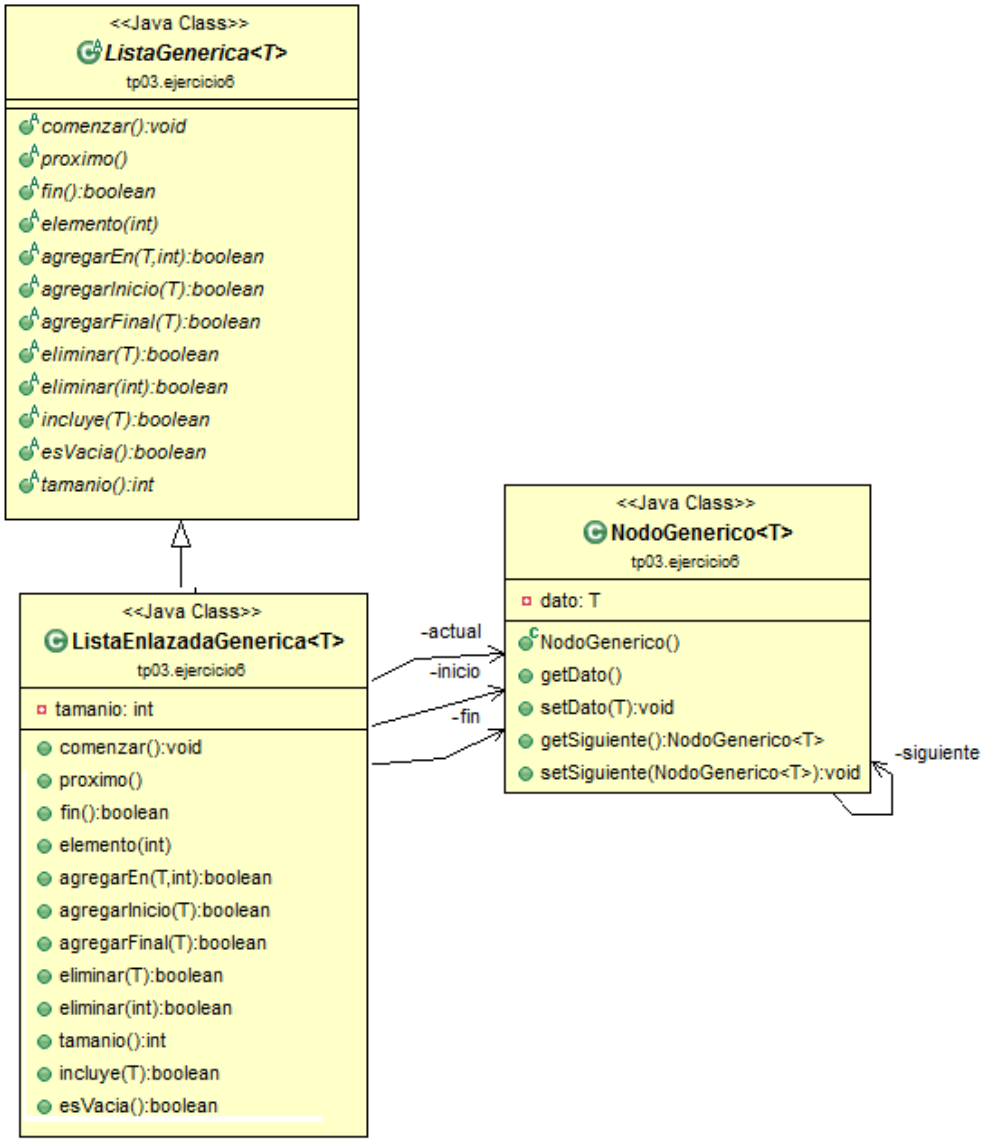



a. Sobrescriba en las clases Tablet y Smart el método **public boolean equals(Object)** de la clase **Object** de manera que las implementaciones sirvan para comparar dos instancias de estos tipos.

b. Sobrescriba también en ambas clases el método **public String toString()** de la clase **Object** de manera que imprima los datos de esos objetos de manera legible.

c. Escriba una clase **tp03.ejercicio4TestSobrescritura** y pruebe los métodos sobrescritos (por ejemplo defina dos objetos de tipo **SmartPhone**, configúrele el mismo número y pruebe el método **equals**, imprima ambos objetos usando el **toString**()).

**6.** Analice el siguiente Diagrama de Clases




a) Implemente en JAVA (pase por máquina) una clase abstracta llamada **ListaGenerica<T>** de acuerdo a la especificación dada, ubíquela en el paquete **tp03**.**ejercicio6**.

b) Escriba una clase llamada **ListaEnlazadaGenerica<T>** como subclase de **ListaGenerica<T>** dentro de mismo paquete anterior, de manera que implemente todos los métodos definidos en la superclase pero haciendo uso de una estructura recursiva.

c) Escriba una clase llamada **TestListaEnlazadaGenerica** que cree 4 objetos de tipo **Mobile** (2 objetos **SmartPhone** y 2 objetos **Tablet**) los agregue a un objeto de tipo **ListaEnlazadaGenerica** usando los diferentes métodos de la lista y luego, imprima los elementos de dicha lista usando el método **toString()**.

**7.** Considere un *string* de caracteres S, el cual comprende únicamente los caracteres: (,),[,],{,}. Decimos que S está balanceado si tiene alguna de las siguientes formas:

S = ‘’ S es el *string* de longitud cero.
S = "(T)"
S = "[T]"
S = "{T}"
S = "TU"

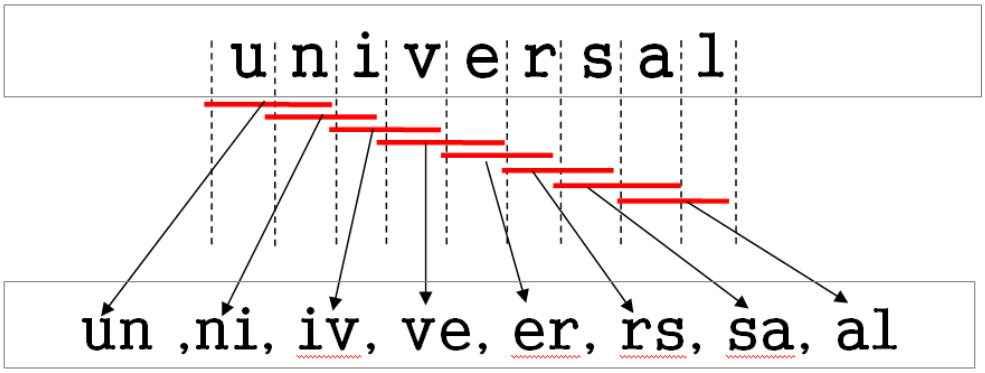
Donde ambos T y U son *strings* balanceados. Por ejemplo, "{( ) [ ( ) ] }" está balanceado, pero "( [ ) ]" no lo está.

a. Implemente una clase llamada **tp03.ejercicio7.TestBalanceo** (pase por máquina), cuyo objetivo es determinar si un *String* dado está balanceado. El *String* a verificar es un parámetro de entrada (no es un dato predefinido).

**Nota: En caso de ser necesario implemente las estructuras de datos que necesite para resolver el ejercicio.**

**EJERCICIOS OPCIONALES**

**8.** Las listas invertidas son una herramienta básica utilizada para la búsqueda por contenido en texto. En este caso particular, una lista invertida almacenará el resultado de la factorización de palabras en n-gramas (un n-grama es una subsecuencia de n caracteres de una palabra) guardando las referencias a las palabras donde se encuentran contenidos.

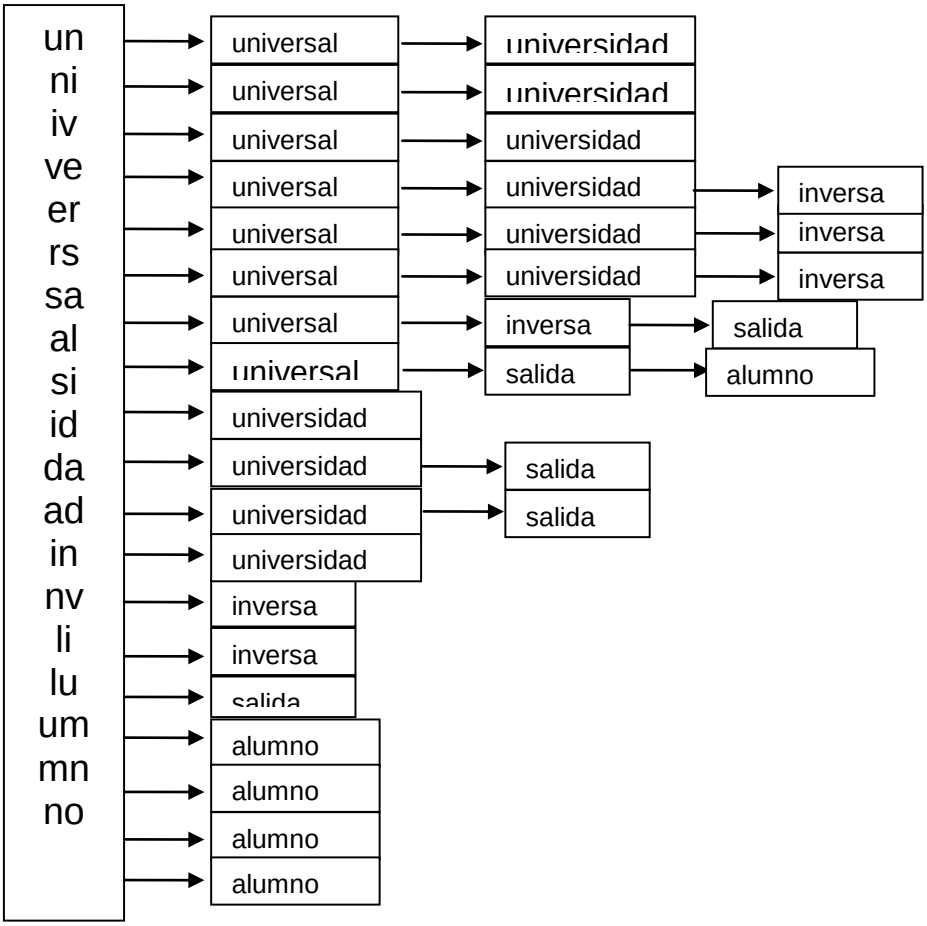



Por Ejemplo, si queremos factorizar la palabra universal en n-gramas de longitud 2 (2-gramas), el resultado será:

Ahora si tenemos la siguiente lista de palabras:

**`universal, universidad, inversa, salida y alumno`**

y deseamos armar una lista invertida con n-gramas de longitud 2, el resultado será:

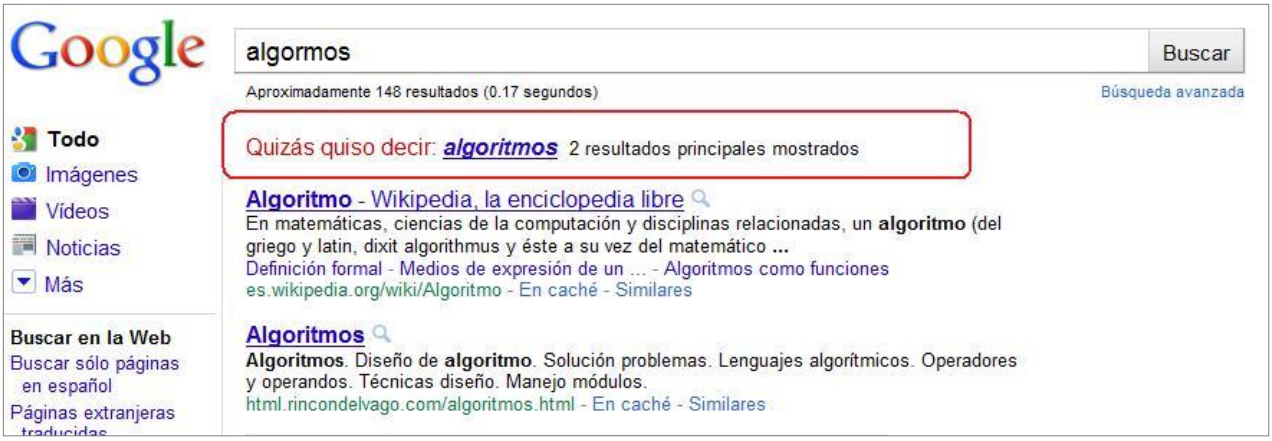



a. Implemente en JAVA una clase llamada **ListaInvertidaDeGramas** ubicada dentro del paquete **ejercicio8**, cumpliendo la siguiente especificación:

```
ListaInvertidaDeGramas (int longGrama)  // instancia una lista invertida, donde
la longitud de los gramas que almacenará esta dada por el parámetro.
agregar (String palabra) // Agrega los gramas del string enviado como parámetro a
la lista y asocia el parámetro a cada uno de los gramas agregados.
recuperarListaDePalabras(String palabra) // Retorna una lista de todas las
palabras en la lista invertida que poseen al menos uno de los gramas del
parámetro.
```

b. Escriba una clase llamada **TestListaInvertidaDeGramas** que reciba en su método **main** una secuencia de strings, los agregue a un objeto de tipo **ListaInvertidaDeGramas** y luego vuelva a recorrer la secuencia de strings invocando el método **recuperarListaDePalabras** con cada uno de ellos e imprima la lista resultado.

**9.** Se desea implementar una herramienta del estilo "Quizás quiso decir" de Google:




a. Implemente en JAVA una clase llamada **ejercicio9.Sugeridor** que contenga un método llamado **quisoDecir()** que tome como parámetro una **ListaInvertidaDeGramas** y un string. El resultado de ejecutar el método **quisoDecir()** será una lista con las palabras sugeridas. Dichas palabras sugeridas se encuentran almacenadas en lista invertida enviada como parámetro y cumplen con la condición que comparadas con el parámetro string devuelven un numero menor o igual a 2.

   **Nota 1:** Para realizar las comparaciones entre las palabras utilice la clase provista por la cátedra llamada **EditDistance** e invoque el método **compararStrings()**, el cual retorna un entero que simboliza el nivel de igualdad entre las palabras comparadas. Si la distancia es 0 -cero-, la palabra buscada es exactamente la encontrada y mientras más grande sea este número más distintas son las palabras.

   **Nota 2:** Las palabras de la lista invertida con las cuales se debe aplicar la comparación son aquellas que comparten gramas con el string enviado como parámetro.

b. Escriba una clase llamada **TestSugeridor** que en su método **main** reciba una secuencia de strings, los agregue a un objeto de tipo **ListaInvertidaDeGramas** a excepción del último e invoque el método **quisoDecir()** de un objeto de tipo **Sugeridor** enviándole como parámetro el objeto de tipo **ListaInvertidaDeGramas** anterior y el último *String* de la secuencia enviada al **main**. Además se debe imprimir la siguiente leyenda:

   **ENTRADA**

   Es como ejecutar el intérprete así: **java TestSugeridor vaca vacaciones playa maya**

   Palabras para armar la lista invertida

   Palabra a buscar

   **SALIDA**

   ```
   Tal vez quiso decir: str1, str2,…., strn
   ```

   Donde str1, str2,…., strn son los strings de la lista resultante de la ejecución del método **quisoDecir()** .

   En el ejemplo anterior, la salida sería: Tal vez quiso decir: playa