Práctica 3: **El mundo Rau**

Programación II

Mayo 2017 (versión 14/04/2017)

DLSI – Universidad de Alicante

Alicia Garrido Alenda

Normas generales

- El plazo de entrega para esta práctica se abrirá el lunes 15 de mayo a las 9:00 horas y se **cerrará el miércoles 24 de mayo a las 23:59 horas**. No se admitirán entregas fuera de plazo.
- Se debe entregar la práctica en un fichero comprimido de la siguiente manera:

  1. Abrir un terminal.
  2. Situarse en el directorio donde se encuentran los ficheros fuente (`.java`) de manera que al ejecutar `ls` se muestren los ficheros que hemos creado para la práctica y ningún directorio.
  3. Ejecutar:

     ```
     tar cfvz  practica3.tgz   *.java
     ```

Normas generales comunes a todas las prácticas

1. La práctica se debe entregar exclusivamente a través del servidor de prácticas del departamento de Lenguajes y Sistemas Informáticos, al que se puede acceder desde la página principal del departamento (`http://www.dlsi.ua.es`, enlace "Entrega de prácticas") o directamente en `http://pracdlsi.dlsi.ua.es`.

   - **Bajo ningún concepto** se admitirán entregas de prácticas por otros medios (correo electrónico, UACloud, etc.).
   - El usuario y contraseña para entregar prácticas es el mismo que se utiliza en UACloud.
   - La práctica se puede entregar varias veces, pero sólo se corregirá la última entrega.

2. El programa debe poder ser compilado sin errores con el compilador de Java existente en la distribución de Linux de los laboratorios de prácticas; si la práctica no se puede compilar su calificación será 0. Se recomienda compilar y probar la práctica con el autocorrector durante su implementación para detectar y corregir errores.

3. La práctica debe ser un trabajo original de la persona que entrega; **en caso de detectarse indicios de copia con una o más prácticas (o compartición de código) la calificación de la práctica será** `0`.

4. Se recomienda que los ficheros fuente estén adecuadamente documentados, con comentarios donde se considere necesario.

5. La primera corrección de la práctica se realizará de forma automática, por lo que es imprescindible respetar estrictamente los formatos de salida que se indican en este enunciado. Además, al principio de todos los ficheros fuente entregados se debe incluir un comentario con el `DNI` de la persona que entrega la práctica, con el siguiente formato:

   // DNI *tuDNI* Nombre

   donde *tuDNI* es el DNI del alumno correspondiente (sin letras), **tal y como aparece en UACloud**, y Nombre es el nombre completo; por ejemplo, el comentario podría ser:

   `// DNI 2737189 CASIS RECOS, ANTONIA`

6. El cálculo de la nota de la práctica y su influencia en la nota final de la asignatura se detallan en las transparencias de la presentación de la asignatura.

7. Para evitar errores de compilación debido a codificación de caracteres que **descuenta** `0.5` **de la nota de la práctica**, se debe seguir una de estas dos opciones:

   a) Utilizar EXCLUSIVAMENTE caracteres ASCII (el alfabeto inglés) en vuestros ficheros, **incluídos los comentarios**. Es decir, no poner acentos, ni ñ, ni ç, etcétera.

   b) Entrar en el menú de Eclipse `Edit > Set Encoding`, aparecerá una ventana en la que hay que seleccionar `UTF-8` como codificación para los caracteres.

8. El tiempo estimado por el corrector para ejecutar cada prueba debe ser suficiente para que finalice su ejecución correctamente, en otro caso se invoca un proceso que obliga a la finalización de la ejecución de esa prueba y se considera que dicha prueba no funciona correctamente.

Descripción del problema

En algún lugar del universo existen unos animales, los *Rau*, que cuando interaccionan con su entorno devuelven energía; si ésta es negativa contaminan su entorno y si es positiva lo depuran.

No existen los *Rau* puros, ya que hay dos tipos bien diferenciados: los *RauC*, que al interaccionar con el entorno lo contaminan y los *RauD*, que lo depuran.

Para esta práctica es necesario implementar las clases que se definen a continuación, a las que se pueden añadir variables de instancia y/o clase (siempre que sean privadas) y métodos cuando se considere necesario.

Todo **Rau** se caracteriza por:

- *sexo*: indica si es macho o hembra (`boolean`).
- *energia*: array de enteros (`int[]`).

No se podrá crear un **Rau**, puesto que esta clase es abstracta, pero sí tendrá constructor, al que se le pasa por parámetro un booleano y un entero. Si el booleano es cierto, el sexo será hembra, en otro caso será macho. El entero determina el tamaño del array de *energia*, que debe ser mayor que `0`, si no por defecto el tamaño será `9`.

Y las acciones que puede realizar son:

- *come*: se le pasan por parámetro dos enteros. El primero indica una posición del array de *energia* y el segundo la cantidad en que ésta se incrementa, si esta cantidad es mayor que `0`. El método devuelve cierto si incrementa la energía de alguna posición, y falso en otro caso.
- *reproduce*: tienen la capacidad de reproducirse de forma asexual, de manera que se clonan y devuelven el nuevo *Rau* creado. En función del tipo de *Rau*, el proceso de clonado se realiza de una manera u otra.
- *reproduce*: se le pasa por parámetro otro *Rau* y un `String`. Si uno es macho y otro es hembra, se pueden llegar a reproducir, creando un nuevo *Rau*, que se devuelve. Dependiendo del tipo de *Rau*, esta acción se realiza de una manera u otra. Si la reproducción no se llega a producir porque ambos son del mismo sexo se lanza y propaga la excepción *IncompatiblesException*, pasándole al constructor de la excepción una cadena que indicará el sexo del rau pasado por parámetro ("macho" o "hembra").
- *interacciona*: se le pasa por parámetro un entero que indica de alguna manera la posición del array de *energia* del que devuelve su contenido, dejando dicha posición a `0`. Dependiendo del tipo de *Rau*, esta acción se realiza de una manera u otra.
- *nivelEnergetico*: devuelve la suma de los contenidos de todas las posiciones de su array de *energia*.
- *getSexo*: devuelve el sexo del rau.
- *getEnergia*: devuelve el array de energia del rau.

Un **RauD** es un *Rau* que además se caracteriza por:

- ⁕ *nombre*: nombre del raud (`String`);
- ⁕ *progenie*: array que contendrá la descendencia del rau (`Rau[]`).

Cuando se crea un **RauD** se le pasan los mismos parámetros que a un *Rau* y un `String`. El `String` indica su nombre[1]. La progenie tendrá un tamaño inicial de `4`.

Las acciones que puede realizar un **RauD** son las mismas que un *Rau*, definiendo el comportamiento de las acciones que dependen del tipo de *Rau*. Por tanto el comportamiento de los **RauD** se define de la siguente manera:

- ⊙ *interacciona*: calcula el resto de dividir el entero pasado por parámetro entre el número de posiciones de su array de *energia*, devolviendo la energia contenida en esta posición, dejándola a `0`. Si el entero pasado por parámetro es negativo, se realiza la acción utilizando su valor absoluto.
- ⊙ *reproduce*: el proceso de clonado en los *RauD* consiste en crear un nuevo raud con el mismo sexo, el mismo array de energía, el mismo nivel energético y el mismo nombre, pero sin la carga de la progenie, es decir, el nuevo raud no tendrá descendencia inicialmente, y el tamaño de su progenie será `4`.
- ⊙ *reproduce*: si el *Rau*[2] pasado por parámetro es también un *RauD*, el nuevo *Rau* que se crea será de tipo *RauD*, y su array de energía tendrá el tamaño de la suma de los de sus padres, su sexo es el del progenitor con un nivel energético más alto (en caso de ser iguales será macho) y el `String` indica el nombre del nuevo rau. Además se comerá el contenido del array de energía de su madre, dejando a ésta agotada ya que deja a `0` las posiciones que se come. Si el *Rau* pasado por parámetro no es de tipo *RauD*, no hay reproducción y se lanza y propaga la excepción *IncompatiblesException* pasándole por parámetro dos cadenas al constructor de la excepción: la primera indicará el sexo del rau pasado por parámetro ("macho" o "hembra") y la segunda su tipo ("RauC" o "RauD"). El método guarda el nuevo rau creado en la progenie del rau receptor del mensaje en la primera posición libre (redimensionando la progenie si es necesario aumentando su tamaño con `3` posiciones más) y devuelve este nuevo rau.

Y además puede realizar las siguientes acciones:

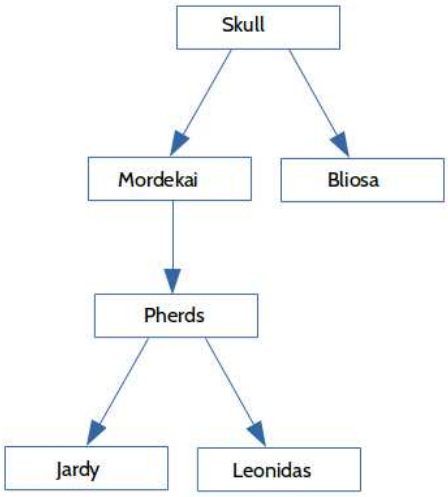
- ⊙ *arbolGenealogico*: devuelve un array de `String` con los nombres de sus hijos, los nombres de los hijos de sus hijos, y así sucesivamente con toda su descencia. El orden de los nombres será:

  1. el nombre del raud;
     a) si tiene hijos, el nombre de su primer hijo;
        1) si éste tiene hijos, el nombre de su primer hijo; y así sucesivamente hasta llegar a un descendiente que no tiene hijos;
        2) cuando se llega a un descendiente que no tiene hijos, se pasa al hermano, es decir, el siguiente hijo de su padre y se repite el proceso.

  Por ejemplo, si tenemos lo siguiente:

[1] Este `String` siempre será distinto de `null`

[2] El rau pasado por parámetro nunca será `null`.



el array de `String` que devuelve será: `[Skull,Mordekai,Pherds,Jardy,Leonidas,Bliosa]`.

⊙ *`getProgenie`*: devuelve la progenie del raud.

⊙ *`getNombre`*: devuelve el nombre del raud.

Un **RauC** es un *`Rau`* que además se caracteriza por:

⁕ *`parejas`*: array que contendrá el nombre de los raud con los que ha intentado reproducirse (`String[]`).

Cuando se crea un **RauC** se le pasan los mismos parámetros que a un *`Rau`* y un entero. El entero indica el tamaño inicial de *`parejas`*, que como mínimo debe ser `3`.

Las acciones que se pueden realizar un **RauD** son las mismas que un *`Rau`*, definiendo el comportamiento de las acciones que dependen del tipo de *`Rau`*. Por tanto el comportamiento de los **RauC** se define de la siguente manera:

⊙ *`interacciona`*: calcula el cociente del entero pasado por parámetro entre el número de posiciones de su array de *`energia`*, devolviendo la energía contenida en esta posición en negativo, dejándola a `0`. Si el entero pasado por parámetro es negativo no se realiza la acción y devuelve `0`.

⊙ *`reproduce`*: el proceso de clonado en los *`RauC`* consiste en crear un nuevo rauc con el mismo sexo, el mismo array de energía, el mismo nivel energético y el mismo tamaño que su array de parejas tenga actualmente, pero sin ninguna pareja.

⊙ *`reproduce`*: si el *`Rau`*[3] pasado por parámetro es de tipo:

- *`RauD`*: se añade su nombre al array `parejas` en la primera posición libre (redimensionando si es necesario aumentando su tamaño con `3` posiciones más). Además si el raud es la hembra, el nuevo *`Rau`* que se crea será de tipo *`RauD`*, su array de energía tendrá el mismo tamaño que el de su madre, su sexo será hembra y su nombre será el `String` pasado por parámetro. Además se come el contenido que puede[4] del array de energía de su padre,

[3] El rau pasado por parámetro nunca será `null`.
[4] Que le cabe en su array de energía.

dejándolo exhausto ya que deja a `0` las posiciones que se come. Si es el macho no hay reproducción y se lanza y propaga la excepción *`IncompatiblesException`* pasándole por parámetro dos cadenas al constructor de la excepción: la primera indicará el sexo del rau pasado por parámetro ("macho" o "hembra") y la segunda su tipo ("RauC" o "RauD").

- *`RauC`*: el nuevo *`Rau`* que se crea será de tipo *`RauC`*, su array de energía tendrá el mismo tamaño que el de su padre, su sexo será el del progenitor con menor nivel energético (en caso de ser iguales será hembra) y el tamaño de su array de parejas será el de su madre. Además se come el contenido del array de energía de su madre, dejándola exhausta ya que deja a `0` las posiciones que se come.

Y además puede realizar las siguientes acciones:

⊙ *`getParejas`*: devuelve el array de parejas del rauc.

La clase **IncompatiblesException** hereda de la clase *`Exception`* de java y tendrá dos constructores:

- con un parámetro de tipo `String`: el mensaje que tiene que devolver la excepción al invocarse su método *`getMessage`* es esta cadena;
- con dos parámetros de tipo `String`: el mensaje que tiene que devolver la excepción al invocarse su método *`getMessage`* será la cadena formada por la primera cadena, seguida de un guión, seguida por la segunda cadena. Por ejemplo:
  `macho-RauC`

La clase **Entorno** es una aplicación en la que se simulará un entorno con el que interaccionan diferentes raus. La información para crear los diferentes objetos se leerá de un fichero de texto, y los resultados de la simulación se escribirán en otro fichero de texto. Para ello esta clase debe tener un método `main` en el que:

⊙ compruebe el número de parámetros de la aplicación; si este número es distinto de `2`, muestre por pantalla el siguiente mensaje y termine la ejecución del programa:

```
Error se necesitan dos nombres de fichero
```

⊙ se abra para lectura el fichero de texto que se le pasa como primer parámetro a la aplicación y para escritura el fichero de texto que se le pasa como segundo parámetro; el formato del fichero de texto de lectura:

- dos enteros separados por un espacio en blanco que indican el tamaño del entorno, que será una matriz de enteros; el primero indica el número de filas y el segundo el número de columnas;
- líneas con secuencias de enteros separados por un espacio en blanco; cada entero indica el contenido de las posiciones del entorno, de manera que la primera línea se corresponde con la fila `0` del entorno, y el primer entero de esa línea se corresponderá con la posición `[0][0]` del entorno;
- un entero que indica el número de raus que se van a crear;

- a continuación vendrá la información sobre cada rau, uno por línea. El formato de cada línea será el siguiente, separando cada dato por un espacio en blanco:
  - un carácter que indica si se trata de un raud ("`D`") o un rauc ("`C`");
  - un carácter que indica si se trata de un macho ("`M`") o una hembra ("`H`");
  - un entero que indica el tamaño de su array de energía;
  - una cadena si se trata de un raud, para su nombre; un entero si se trata de un rauc, para el tamaño de sus parejas;
- a continuación vendrá la información sobre las acciones a realizar sobre el entorno. Para ello se recorre la matriz creada para el entorno por filas (empezando en la posición `[0][0]`), y para cada posición de la matriz leerá una línea del fichero, mientras queden líneas por leer, que contendrá la acción a realizar. El formato de cada acción será:
  1. entero que indica el rau que realiza la acción; este entero indica el orden de creación del rau (`0` es el primer rau creado, `1` es el segundo, y así sucesivamente);
  2. cadena con el nombre de la acción (método) a realizar. En función de la acción, los parámetros serán:
     - *`come`*: se le pasa como primer parámetro la fila de la casilla actual de la matriz y como segundo parámetro el contenido de dicha casilla, escribiendo en el fichero de escritura la cadena "`come`", un espacio en blanco, el resultado de la acción y un retorno de línea;
     - *`interacciona`*: se le pasa por parámetro la columna de la casilla actual de la matriz y se modifica el contenido de dicha casilla con el valor devuelto por el método, escribiendo en el fichero de escritura la cadena "`interacciona`", un espacio en blanco, el valor devuelto por el método y un retorno de línea;
     - *`reproduce`*: se le pasa por parámetro el rau anterior al actual en el orden de creación y la cadena que aparece en la línea de acción a continuación del nombre del método. Si se reproducen, el nuevo rau se añade a continuación de los rau existentes hasta el momento. Si el rau actual es el primero que se ha creado no se realiza la acción.

⊙ implemente el manejador de la excepción *`IncompatiblesException`* de manera que la escriba en el fichero de escritura y continue con el recorrido de la matriz. El formato para escribir la excepción será el nombre de la excepción, dos puntos y su mensaje asociado. Por ejemplo:

```
IncompatiblesException: macho-rauC
```

⊙ escriba en el fichero de escritura el tipo ("`RauC`" o "`RauD`") y sexo ("`macho`" o "`hembra`"), separados por un espacio en blanco, de todos los rau que existan, uno por línea.

⊙ por último la aplicación debe cerrar ambos ficheros, el de lectura y el de escritura;

⊙ el tratamiento para todas las excepciones comprobadas de los ficheros será invocar el método `printStackTrace()` del objeto excepción capturado y acabar la ejecución del programa.

Un ejemplo de fichero de lectura para la aplicación sería el siguiente:

```
10 3
2 1 5
-5 2 1
1 -3 4
8 7 -6
-2 11 3
-4 5 -1
9 9 9
7 -3 15
-12 14 7
3 2 1
4
D H 8 Jesabel
C M 9 5
D M 7 Morme
D H 4 Mali
3 come
1 come
2 come
3 interacciona
1 reproduce Korma
3 come
2 reproduce Malic
0 come
2 come
3 come
1 interacciona
3 reproduce Clam
```

Y el fichero de salida que se generaría sería:

```
come true
come true
come true
interacciona 2
come true
IncompatiblesException: macho
come false
come true
come true
interacciona 0
RauD hembra
RauC macho
RauD macho
RauD hembra
RauD hembra
RauD macho
```

El contenido de los ficheros de lectura siempre será correcto en el sentido de que donde debe aparecer un entero, habrá un entero, donde debe aparecer una cadena, habrá una cadena y donde debe haber un carácter, habrá un carácter. Además si en el fichero se indica que la matriz tiene `m` filas y `n` columnas, estos son los datos que aparecerán en el fichero (ni más ni menos), al igual que si se indica que hay `k` raus, en el fichero aparecerá la información necesaria para cada uno de ellos.

Segmentación de una cadena en distintos elementos

Una vez tenemos una línea del fichero en una variable de tipo `String`, por ejemplo la variable `linea`, hay que segmentarla para obtener de ella los distintos elementos que necesitamos para procesar la información contenida de manera conveniente.

Esta segmentación la podemos hacer usando el método `split` de la clase String. Este método devuelve un array de `Strings` a partir de uno dado usando el separador que se especifique. Por ejemplo, supongamos que tenemos la variable `linea` de tipo `String` que contiene la cadena "`5 4`" de la cual queremos obtener por separado los dos datos que contiene para, por ejemplo, crear la matriz del entorno. Para ello primero debemos indicar el separador y después segmentar usando el método `split` de la siguiente manera:

```
//indicamos el separador de campos: un espacio en blanco
 String separador = " ";
//segmentamos
 String[] elems = linea.split(separador);
//convertimos a entero cada cadena contenida en elems
 int nf = Integer.parseInt(elems[0]);
 int nc = Integer.parseInt(elems[1]);
// con estos datos ya se puede construir un tablero
 int[] entorno uno=new int[nf][nc];
```

Restricciones en la implementación

⊛ Todas las variables de instancia de las clases deben ser privadas (no accesibles desde cualquier otra clase).

⊛ Algunos métodos deben ser públicos y tener una ***signatura*** concreta:

- En **Rau**
  - `public Rau(boolean b,int i)`
  - `public boolean come(int i,int j)`
  - `public abstract Rau reproduce();`
  - `public abstract Rau reproduce(Rau r,String n) throws IncompatiblesException;`
  - `public abstract int interacciona(int i);`
  - `public int nivelEnergetico()`
  - `public boolean getSexo()`
  - `public int[] getEnergia()`
- En **RauD**
  - `public RauD(boolean b,int i,String n)`
  - `public int interacciona(int i)`
  - `public Rau reproduce()`
  - `public Rau reproduce(Rau r,String n) throws IncompatiblesException`
  - `public String[] arbolGenealogico()`
  - `public Rau[] getProgenie()`
  - `public String getNombre()`
- En **RauC**
  - `public RauC(boolean b,int i,int j)`

- `public int interacciona(int i)`
- `public Rau reproduce()`
- `public Rau reproduce(Rau r,String n) throws IncompatiblesException`
- `public String[] getParejas()`

- En **IncompatiblesException**
  - `public IncompatiblesException(String n)`
  - `public IncompatiblesException(String n,String m)`

⊛ Únicamente el fichero `Entorno.java` de los ficheros entregados en esta práctica puede contener un método `public static void main(String[] args)`.

⊛ Todas las variables de clase e instancia deben ser privadas. En otro caso se restará un `0.5` de la nota total de la práctica.

Probar la práctica

- En UACloud se publicará un corrector de la práctica con un conjunto mínimo de pruebas (se recomienda realizar pruebas más exhaustivas de la práctica).
- Podéis enviar vuestras pruebas (fichero con extensión `java`, con un método `main` y sin errores de compilación) a los profesores de la asignatura mediante tutorías de UACloud, para obtener la salida correcta a esa prueba. En ningún caso se modificará/corregirá el código de las pruebas. Los profesores contestarán a vuestra tutoría adjuntando la salida de vuestro `main`, si no da errores.
- El corrector viene en un archivo comprimido llamado `correctorP3.tgz`. Para descomprimirlo se debe copiar este archivo donde queramos realizar las pruebas de nuestra práctica y ejecutar:

  ```
  tar xfvz correctorP3.tgz
  ```

  De esta manera se extraerán de este archivo:

  - El fichero `corrige.sh`: *shell-script* que tenéis que ejecutar.
  - El directorio `practica3-prueba`: dentro de este directorio están los ficheros con extensión `.java`, programas en Java con un método `main` que realizan una serie de pruebas sobre la práctica, y los ficheros con el mismo nombre pero con extensión `.txt` con la salida correcta para la prueba correspondiente.
- Una vez que tenemos esto, debemos copiar nuestros ficheros fuente (sólo aquellos con extensión `.java`) al mismo directorio donde está el fichero `corrige.sh`.
- Sólo nos queda ejecutar el *shell-script*. Primero debemos darle permisos de ejecución si no los tiene. Para ello ejecutar:

  ```
  chmod +x corrige.sh
  corrige.sh
  ```
- Una vez se ejecuta `corrige.sh` los ficheros que se generarán son:

- `errores.compilacion`: este fichero sólo se genera si el corrector emite por pantalla el mensaje "`Error de compilacion: 0`" y contiene los errores de compilación resultado de compilar los fuentes de una práctica particular. Para consultar el contenido de este fichero se puede abrir con cualquier editor de textos (gedit, kate, etc.).
- Fichero con extensión `.tmp.err`: este fichero debe estar vacío por regla general. Sólo contendrá información si el corrector emite el mensaje "`Prueba p01: Error de ejecucion`", por ejemplo para la prueba `p01`, y contendrá los errores de ejecución producidos al ejecutar el fuente `p01` con los ficheros de una práctica particular.
- Fichero con extensión `.tmp`: fichero de texto con la salida generada por pantalla al ejecutar el fuente correspondiente, por ejemplo `p01.tmp` contendrá la salida generada al ejecutar el fuente `p01` con los ficheros de una práctica particular.

Si en la prueba no sale el mensaje "`Prueba p01: ok`", por ejemplo para la prueba `p01`, o algún otro de los comentados anteriormente, significa que hay diferencias en las salidas para esa prueba, por tanto se debe comprobar que diferencias puede haber entre los ficheros `p01.txt` y `p01.tmp`. Para ello ejecutar en línea de comando, dentro del directorio `practica3-prueba`, la orden: `diff -w p01.txt p01.tmp`