Práctica 2: **La evolución**

Programación II

Abril 2018 (versión 22/03/2018)

DLSI – Universidad de Alicante

Alicia Garrido Alenda

Normas generales

- El plazo de entrega para esta práctica se abrirá el lunes 23 de abril a las 9:00 horas y se **cerrará el viernes 27 de abril 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  practica2.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

El tiempo ha transcurrido y tanto los habitantes como los místicos han evolucionado. Entre los habitantes se ha desarrollado una civilización en la que un *Habitante* puede ser *Plebeyo* o *Guerrero*, mientras que los místicos se pueden haber decantado por hacer el bien o el mal, de manera que un *Mistico* ahora puede ser *Blanco* si favorece a los habitantes en sus demandas, o bien *Oscuro* si es malvado.

Se crean los clanes; un *Clan* tiene un nombre y a él pertenecerán los habitantes que tienen el mismo clan, un único místico para todos los habitantes del clan al que agasajar, y terrenos en los que recolectar. Además alguna que otra vez se producen enfrentamientos entre clanes rivales.

También han aparecido en escena unas bestias, que los plebeyos pueden domesticar, ya que son animales que, en estado salvaje, simplemente pasturean por los terrenos que se encuentran tomando lo que necesitan, pero que si son domesticadas ofrecen lo que encuentran a sus amos. Una *Bestia* sólo puede pertenecer a un plebeyo, al que le será fiel y al que ayudará cuando éste se lo pida, siempre que no sea hechizada, ya que cuando una *Bestia* es hechizada por un *Oscuro* se convierte en un *Demoledor*, que arrasa con todo lo edificado en un terrreno, hasta que un *Blanco* es capaz de anular el hechizo.

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. También se pueden añadir métodos a las superclases cuando sea necesario.

Aplicable en todo el enunciado de la práctica: **Una cadena nunca será una cadena vacía ni una referencia vacía.**

Una **Bestia** se caracteriza por:

- *fuerza*: fuerza de la bestia (`double`);
- *amo*: habitante que la ha domesticado (`Plebeyo`);
- *nombre*: nombre de la bestia una vez domesticada (`String`);
- *amuleto*: producto de tipo `Filosofal` (`Producto`);

Cuando se crea una **Bestia** se le pasa por parámetro su fuerza que debe ser mayor que `0`, en otro caso valdrá `10.5`. Inicialmente no tiene amo ni nombre ni amuleto.

Una **Bestia** puede realizar las siguientes acciones:

- *busca*: se le pasa por parámetro un terreno en el que busca, recorriéndolo por filas empezando en la posición `(0,0)`, el primer producto de tipo `Filosofal` que encuentre para recogerlo y quedárselo como amuleto, si no tenía amuleto previamente. El método devuelve el nombre del amuleto de la bestia, o `null` por defecto.
- *domestica*: se le pasa por parámetro un habitante que quiere domesticar a la bestia. Si la bestia no tiene ya un amo, el habitante no tiene ya una bestia y es un plebeyo, la bestia se deja domesticar, de manera que toma al habitante como amo y devuelve cierto. En cualquier otro caso devuelve falso.

- ⊙ *alimenta*: se le pasa por parámetro un producto. Si es de tipo `Vegetal` suma a su fuerza el 10 % del peso del producto y devuelve cierto. En cualquier otro caso, devuelve falso.
- ⊙ *pasturea*: se le pasa por parámetro un terreno y un entero. La bestia recorre la fila del terreno indicada por el segundo parámetro, empezando por la posición `0`, recogiendo del terreno todos los productos que encuentra que no sean de tipo `Edificado`. Una vez ha terminado de recoger, si dispone de amuleto, los convierte todos al tipo `Vegetal`. A continuación se alimenta únicamente con los productos de tipo `Vegetal`: si tiene amo únicamente con el primer producto recogido, si no con todos. Finalmente si tiene amo devuelve todos los productos recogidos menos el primero, si no tiene amo no devuelve ningún producto.
- ⊙ *ayuda*: devuelve el 50 % del su fuerza, decreméntandola en dicha cantidad.
- ⊙ *hechiza*: la bestia ha recibido un hechizo y se convierte en un demoledor, para ello devuelve un demoledor con todas sus características iguales a las propias.
- ⊙ *setNombre*: se le pasa por parámetro una cadena para actualizar su nombre.
- ⊙ *getNombre*: devuelve su nombre.
- ⊙ *getFuerza*: devuelve su fuerza.
- ⊙ *getAmuleto*: devuelve su amuleto.
- ⊙ *getAmo*: devuelve su amo.

Un **Demoledor** es una *Bestia* que tiene sus mismas características pero que realiza alguna de sus acciones de forma diferente:

- ⊙ *domestica*: se le pasa por parámetro un habitante que quiere domesticar a la bestia pero como es un demoledor no se deja domesticar y lo que hace es robar. Por tanto comprueba si ya tiene amo por alguna circunstancia, y si lo tiene lo abandona, no antes de obtener su cesta y dejarla vacía. Si no tiene amo, obtiene la cesta del habitante pasado por parámetro y la deja vacía. Con el resultado de su rapiña incrementa su fuerza, sumándole el número de productos robados. Si ha conseguido robar algún producto devuelve cierto. En cualquier otro caso devuelve falso.
- ⊙ *alimenta*: se le pasa por parámetro un producto. Si es de tipo `Animal` suma a su fuerza el `15 %` del peso del producto y devuelve cierto. En cualquier otro caso devuelve falso.
- ⊙ *pasturea*: se le pasa por parámetro un terreno y un entero. El demoledor recorre la fila del terreno indicada por el segundo parámetro, empezando por la posición `0`, recogiendo del terreno todos los productos que encuentra y destruyendo todos los de tipo `Edificado`. Una vez ha terminado de recoger, si dispone de amuleto, los convierte todos al tipo `Animal`. A continuación se alimenta con los productos que son de tipo `Animal` y finalmente devuelve los productos recogidos no consumidos.
- ⊙ *ayuda*: devuelve el `50 %` del su fuerza con valor negativo, sin modificar su fuerza.

Y además puede:

⊙ *arrasa*: se le pasa por parámetro un terreno que recorre destruyendo cualquier edificación que encuentra. Este ataque destructor decrementa su fuerza en un 25 % en cualquier caso. El método devuelve el peso total de las edificaciones derruídas.

⊙ *exorciza*: el demoledor ha sido liberado de su hechizo y vuelve a ser la bestia pacífica que era. Para ello devuelve una bestia con todas sus características iguales a las propias.

Como consecuencia de la evolución y la aparición de los clanes ya no habrán habitantes como tales, tan sólo plebeyos y guerreros que pueden realizar nuevas acciones comunes a ambos relativas al clan, y que por tanto deben formar parte de su interfaz como *Habitante*. Para reflejar esta situación la clase **Habitante** se convierte en *clase abstracta*, y las nuevas acciones que debe incluir son:

⊙ *esAcogido*: se le pasa por parámetro un clan. Si el nombre del clan coincide, ignorando diferencias de mayúsculas y minúsculas, con el clan que aparece en su propio nombre, y no pertenece ya a otra tribu, toma el clan pasado por parámetro como su tribu y devuelve cierto. En cualquier otro caso devuelve falso.

⊙ *esDesterrado*: si es miembro de un clan significa que ha sido desterrado del mismo, por tanto elimina la referencia de su tribu dejándola a `null` y devuelve cierto. En cualquier otro caso devuelve falso.

⊙ *cambiaClan*: se le pasa una cadena por parámetro que indica el nombre de su nuevo clan. El habitante cambia en su nombre el clan anterior por este nuevo.

⊙ *getTribu*: devuelve su tribu.

Estas acciones no se pueden implementar en la clase *Habitante*, puesto que un habitante no tiene tribu, por tanto se tienen que implementar en las clases que heredan de ella.

Un **Plebeyo** es un *Habitante* que además se caracteriza por:

⁕ *bestiola*: bestia domesticada por el plebeyo (`Bestia`).

⁕ *patrono*: guerrero al que presta sus servicios el plebeyo (`Guerrero`).

⁕ *tribu*: clan del que es miembro (`Clan`).

Cuando se crea un **Plebeyo** se le pasan los mismos parámetros que a un *Habitante*, todavía no ha domesticado ninguna bestia, no tiene patrono y no es miembro de ningún clan.

Y las acciones que puede realizar son las mismas que un *Habitante*, aunque algunas de forma diferente:

⊙ *trueque*: se le pasa por parámetro un habitante con el que realizar el trueque, de manera que:

1. si tanto el plebeyo como el habitante pasado por parámetro no son miembros de ningún clan, se realizará el trueque de la forma habitual;
2. si uno de ellos es miembro de algún clan pero el otro no:

- el plebeyo intentará hacer el trueque con el primer producto de su cesta, y el primer tipo de producto distinto a éste que tenga en la cesta. Si no tiene distintos tipos de producto en su cesta no realiza el trueque;

3. si ambos son miembros de un clan:
   - si pertenecen a clanes con el mismo nombre exactamente y ambos son plebeyos, no hay trueque, sino que se intercambian sus bestiolas;
   - si son clanes distintos o el habitante pasado por parámetro es un guerrero, realizará el trueque utilizando el último producto de su cesta y el primer tipo de producto distinto a éste que tenga en la cesta. Si no tiene distintos tipos de producto en su cesta no realiza el trueque.

En cualquier caso en el que el trueque es posible le pasa un mensaje al otro habitante con el tipo de producto y el producto seleccionados con los que hacer el trueque, que si lo acepta le devolverá un producto para almacenar al final de su cesta. El método devuelve un array con los nombres de los productos intercambiados, primero el devuelto por el otro habitante y segundo por el que se le ha cambiado, menos en el caso de que intercambien bestiolas: en este caso se devuelve un array con los nombres de las bestiolas, primero el de la bestiola que tenía anteriormente y segundo el de la que tiene actualmente. Si no se puede obtener el nombre de alguna de las bestiolas por algún motivo, en la posición correspondiente del array se incluye la cadena "`Ignoto`". Si por cualquier circunstancia no fuera posible el trueque, el método devuelve `null`.

⊙ ***`tributa`***: si el místico pasado por parámetro es el de su tribu, el plebeyo recorre el feudo de la tribu, empezando por el primer terreno, recoge sólo el primer producto que no sea de tipo `Edificado` que encuentra en su recorrido de los terrenos y se lo ofrece como culto al místico junto con su nombre. Si la tribu no dispone de terrenos, el plebeyo no podrá tributar. Si el místico no es el de su tribu, primero comprueba si el místico es blanco u oscuro, de manera que si es blanco le ofrecerá como culto el primer producto de su cesta junto con su nombre, mientras que si es oscuro le ofrecerá como culto su bestiola como sacrificio quedándose sin ella. En cualquier caso el plebeyo sumará a su vigor lo que el místico tenga a bien otorgarle. El método devuelve la cantidad en la que se incrementa el vigor del plebeyo.

Y además puede:

⊙ ***`plegaria`***: si es miembro de un clan pide ayuda al místico de su tribu para incrementar su vigor, ofreciéndole a cambio el amuleto de su bestiola, sumando a su vigor lo que el místico le conceda. Si incrementa su vigor devolverá cierto, y en cualquier otro caso falso.

⊙ ***`domestica`***: se le pasa por parámetro una bestia a la que quiere domesticar y una cadena. Para ello le pasa a la bestia el mensaje ***`domestica`*** pasándose a sí mismo por parámetro. Si la bestia se deja, es decir, devuelve cierto, el plebeyo la adopta como su bestiola, le da como nombre la cadena pasada por parámetro y devuelve cierto. En cualquier otro caso devuelve falso.

⊙ ***`recolecta`***: se le pasan por parámetro dos enteros. El primer entero indica la posición del terreno que pertenece a su tribu sobre el que se va a realizar la recolecta, mientras que el segundo entero indica el número de productos a recolectar. Si el vigor del plebeyo es mayor que `0`, debe recorrer el terreno por filas, empezando en la posición `(0,0)`, recogiendo y almacenando

al final de su cesta los productos que encuentra que no sean de tipo `Edificado` hasta alcanzar el número pasado como segundo parámetro. El método devuelve el número de productos recolectados y decrementa su vigor en un `5%`, o `-1` si no ha conseguido recolectar ninguno por algún motivo, sin decrementar su vigor.

- ⊙ *`vasallaje`*: se le pasa por parámetro un terreno. Si el vigor del plebeyo es mayor que `0`, recorrerá el terreno recogiendo todos los productos que pueda que no sean de tipo `Edificado` y los devolverá. Esta acción decrementará en un `10%` su vigor aunque no consiga recoger ningún producto.

- ⊙ *`alimenta`*: se le pasa por parámetro un entero. Si es miembro de un clan indica el terreno de la tribu por el que lleva a pasturar a su bestiola, si no es miembro de un clan indica la posición del producto de su cesta con el que la alimenta. Si ha llevado a pasturar a la bestiola, lo ha hecho por todas las filas del terreno, guardando al final de su cesta lo que su bestiola le ha ido devolviendo por cada fila y el método devuelve `1`. Si por el contrario la ha alimentado con un producto de su cesta, quita dicho producto de su cesta y devuelve `2`. En cualquier otro caso, el método devuelve `0`.

- ⊙ *`tutela`*: se le pasa por parámetro un guerrero, que si no tiene sirviente y el plebeyo no tiene patrono, se convierte en el patrono del plebeyo y devuelve cierto. En cualquier otro caso se devuelve falso.

- ⊙ *`libera`*: si el plebeyo tenía patrono, ha sido liberado de su yugo, por tanto deja de tener patrono y devuelve el nombre de su expatrono. En cualquier otro caso devuelve la cadena vacía (“”).

- ⊙ *`getBestiola`*: devuelve su bestia.

- ⊙ *`getPatrono`*: devuelve su patrono.

Un **Guerrero** es un *`Habitante`* que además se caracteriza por:

- ⁕ *`sirviente`*: plebeyo que se encarga de las labores (`Plebeyo`).

- ⁕ *`armamento`*: armas de las que dispone para la lucha (`ArrayList<Producto>`).

- ⁕ *`tribu`*: clan del que es miembro (`Clan`).

Cuando se crea un **Guerrero** se le pasan los mismos parámetros que a un *`Habitante`*, todavía no tiene sirviente, crea su armamento, aunque todavía no tiene ningún arma, y no es miembro de ningún clan.

Y las acciones que puede realizar son las mismas que un *`Habitante`*, aunque algunas de forma diferente:

- ⊙ *`trueque`*: solo realiza trueques con otros guerreros, por tanto si el habitante pasado por parámetro no es un guerrero no se realiza el trueque. Si el habitante pasado por parámetro es otro guerrero intenta realizar el trueque con el primer producto de su armamento y le pasa un mensaje de intercambio al otro guerrero con el producto, que le devolverá un producto para almacenar al final de su armamento. El método devuelve un array con los nombres de los productos intercambiados, primero el devuelto por el otro habitante y segundo por el que se le ha cambiado. Si por cualquier circunstancia no fuera posible el trueque, el método devuelve `null`.

⊙ *tributa*: si el místico pasado por parámetro es el de su tribu, tributa como habitante. En cualquier otro caso tributa como guerrero, ofreciéndole al místico como culto sus dos armas más poderosas, es decir, aquellas cuyo valor del producto sea mayor[1] ordenadas de mayor a menor valor junto con su nombre, sumando a su vigor lo que el místico tenga a bien otorgarle. Si el guerrero no dispone de suficiente armamento para realizar la ofrenda, no puede tributar. El método devuelve la cantidad en la que se incrementa el vigor del guerrero.

Y además puede:

⊙ *plegaria*: si es miembro de un clan, y el místico de su tribu es *Blanco*, el guerrero le pide ayuda al místico del clan para incrementar su vigor, ofreciéndole a cambio el primer producto de tipo `Filosofal` que haya en su cesta, sumando a su vigor lo que el místico le otorgue. Si incrementa su vigor devolverá cierto, y en cualquier otro caso falso.

⊙ *intercambio*: se le pasa por parámetro un producto. Si es de tipo:

- `Madera`: saca de su armamento el primer producto que tiene de tipo `Piedra`
- `Piedra`: saca de su armamento el primer producto que tiene de tipo `Madera`

almacenando al final de su armamento el producto pasado por parámetro y devolviendo el que se ha sacado y eliminado de su armamento. Si no tuviera ningún producto del tipo requerido el método devuelve `null`.

⊙ *acoge*: se le pasa por parámetro un habitante, de manera que si es un plebeyo que no tiene patrono, lo intenta acoger bajo su mando y le pasa un mensaje de tutela, de manera que si es aceptado convierte al plebeyo en su sirviente y devuelve cierto. En cualquier otro caso se devuelve falso.

⊙ *recolecta*: se le pasa por parámetro un entero que indica en qué terreno de su tribu, si la tiene, tiene que recolectar. El guerrero recoge del terreno todos los productos de tipo `Piedra` o `Madera` que encuentre y pueda, ya que sólo estos dos tipos de productos se usan como arma, añadiéndolos al final de su armamento. Además le pide a su sirviente vasallaje, indicándole ese mismo terreno, de manera que añade al principio de su cesta los productos obtenidos por el vasallaje de su sirviente. Por ejemplo, si en la cesta del guerrero tenemos los productos:
`<brocoli,rodronita>`
y el sirviente con el vasallaje le devuelve:
`<pato,espinaca,cerdo>`
la cesta del guerrero quedará:
`<pato,espinaca,cerdo,brocoli,rodronita>`
El método devuelve el número total de productos nuevos obtenidos, para el armamento y para la cesta.

⊙ *lucha*: se le pasa por parámetro un habitante. Si dicho habitante es un guerrero y no es miembro del mismo clan o directamente es miembro de ninguno, se entabla una lucha entre ellos. El ganador se decide en función de su poder, que se calcula en base al armamento de cada guerrero y su vigor, aplicando la fórmula:

$$poder = vigor * (nm * 0{,}4 + np * 0{,}6)$$

[1] A igual valor, se queda con la primera que encuentra en el armamento.

donde *nm* es el número de productos de tipo `Madera` y *np* el de tipo `Piedra` que tiene el guerrero en su armamento.

El ganador es aquel que tiene mayor poder, en caso de empate no hay lucha. El ganador le quita al perdedor su armamento añadiéndolo al final del suyo en el orden en el que se lo encuentra y libera al sirviente del perdedor. Si el ganador es el guerrero pasado por parámetro se devuelve 1, si es el propio guerrero se devuelve 2 y si por cualquier motivo no ha habido lucha, se devuelve 0.

⊙ *`getSirviente`*: devuelve su sirviente.

⊙ *`getArmamento`*: devuelve su armamento.

Un **Clan** se caracteriza por:

⁕ *`nombre`*: nombre del clan (`String`).

⁕ *`miembros`*: habitantes que pertenecen al clan (`ArrayList<Habitante>`).

⁕ *`feudo`*: terrenos que pertenecen al clan (`ArrayList<Terreno>`).

⁕ *`deidad`*: místico al que todos los habitantes que pertenecen al clan adoran (`Mistico`).

Cuando se crea un **Clan** se le pasa por parámetro un `String`. El `String` se corresponde con el nombre del clan, que se utilizará para formar el mismo a partir de los habitantes; es decir, se recorrerá la población de habitantes, y aquellos que contengan en su nombre el mismo clan que el nombre del clan serán añadidos a sus miembros y eliminados de la población general de habitantes. Además crea su feudo sin ningún terreno y no tiene deidad.

Y las acciones que puede realizar son:

⊙ *`conquista`*: se le pasa por parámetro un terreno, que si no pertenece ya a otro clan[2], pasa a formar parte del feudo de este clan añadiéndolo en la última posición y se devuelve cierto. En cualquier otro caso se devuelve falso.

⊙ *`enfrenta`*: se le pasa por parámetro un clan. Si son distintos clanes[3] se entabla una batalla en la que los guerreros del clan luchan contra los del otro clan, uno contra uno según el orden en el que aparezcan como miembros de su clan. El clan que gane más luchas es el vencedor, y ante un empate, gana el que mayor número de guerreros tiene. Si el empate persiste, no hay vencedor y se devuelve `null`. El ganador evangeliza al perdedor obligándole a adoptar como deidad a su místico abandonando al anterior, de manera que si su deidad anterior era un blanco, el clan perdedor deja de estar entre sus acólitos. El método devuelve el nombre del clan ganador.

⊙ *`laboro`*: se le pasa por parámetro un entero. Es día de trabajo y es necesario que los miembros del clan recolecten productos de los terrenos del feudo. Para ello se recorren los miembros del clan indicándoles que recolecten en el terreno correspondiente a su posición en el clan[4]. Si además se trata de un plebeyo se le indica que recoja tantos productos como indica el entero pasado por parámetro. El método devuelve la cantidad total de productos recolectados.

[2]Un terreno no puede ser compartido por dos clanes

[3]Un clan no puede luchar contra sí mismo.

[4]Posición que ocupan como miembros del clan.

- ⊙ *destierra*: se le pasa por parámetro una cadena. Se busca entre sus miembros el primer habitante cuyo nombre coincida exactamente con la cadena pasada por parámetro. Si lo encuentra, dicho habitante es expulsado del clan, por tanto deja de formar parte de él y es devuelto por el método. En cualquier otro caso el método devuelve `null`.

- ⊙ *acoge*: se le pasa por parámetro un habitante que es acogido como nuevo miembro del clan, de manera que es añadido al mismo en el último lugar y se le pasa un mensaje para que cambie su nombre, a fin de que sea reconocido como miembro de este clan. También deberá abandonar la tribu anterior en la que estuviera y actualizar su tribu. El método devuelve el número actual de miembros del clan.

- ⊙ *adoptaDeidad*: se le pasa por parámetro un místico, de manera que si el clan no tenía anteriormente una deidad, adopta al místico como deidad y devuelve cierto. En cualquier otro caso devuelve falso. Además si se trata de un blanco se le pasa un mensaje de fervor pasándose a sí mismo por parámetro.

- ⊙ *getNombre*: devuelve el nombre del clan.

- ⊙ *getMiembros*: devuelve los miembros del clan.

- ⊙ *getDeidad*: devuelve el místico del clan.

- ⊙ *getFeudo*: devuelve los terrenos del clan.

Un **Blanco** es un *Mistico* que además se caracteriza por:

- ⁕ *acolitos*: clanes que tienen al blanco como deidad (`ArrayList<Clan>`).

- ⁕ *nombre*: nombre del blanco (`String`).

Cuando se crea un **Blanco** se le pasa por parámetro una cadena para su nombre y crea sus acólitos.

Las acciones que puede realizar son las mismas que un *Mistico*, pero además puede:

- ⊙ *explora*: se le pasa por parámetro una cadena y debe buscar entre los terrenos de sus acólitos, empezando siempre por el primero, un producto cuyo nombre coincida exactamente con la cadena pasada por parámetro y que sea de tipo `Filosofal`. Si lo encuentra lo recoge del terreno, lo guarda al principio de sus tributos y devuelve cierto. En cualquier otro caso devuelve falso.

- ⊙ *exorcismo*: se le pasa un entero por parámetro que indica qué clan entre sus acólitos va a exorcizar. Para ello recorre los miembros de dicho clan y cuando encuentra un plebeyo mira su bestiola. Si la bestiola ha sido convertida en demoledor por un oscuro y el blanco cuenta entre sus tributos con al menos dos productos de tipo `Filosofal`, aplica el exorcismo y vuelve a convertir la bestiola en bestia, que debe reconocer a su dueño. El método devuelve el número de exorcismos realizados por el blanco.

- ⊙ *ayuda*: se le pasa por parámetro un producto. Si dicho producto es de tipo `Filosofal`, el blanco lo incorpora al final de sus tributos y devuelve el peso del producto.

- ⊙ *fervor*: se le pasa un clan por parámetro. Si dicho clan no consta todavía entre sus acólitos y no tiene deidad, pasa a formar parte de ellos, colocándolo en último lugar.

⊙ *getNombre*: devuelve su nombre.

⊙ *getAcolitos*: devuelve sus acólitos.

Un **Oscuro** es un *Mistico* que además se caracteriza por:

⁕ *caterva*: bestias que han sido ofrecidas como culto (`ArrayList<Demoledor>`).

Cuando se crea un **Oscuro** no se le pasan parámetros y crea su caterva aunque todavía no tiene ninguna bestia a su servicio.

Las acciones que puede realizar son las mismas que un *Mistico*, pero además puede:

⊙ *culto*: se le pasa por parámetro una bestia, que si no pertenece aún a su caterva, lo transforma en demoledor y guarda al final de su caterva, devolviendo el `20 %` de la fuerza de la bestia, decrementando la fuerza de la bestia en dicho valor.

⊙ *malogra*: se le pasa por parámetro un habitante. Si dicho habitante es un plebeyo hechiza a su bestia, convirtiéndola en un demoledor y devuelve cierto. En cualquier otro caso devuelve falso.

⊙ *calamidades*: se le pasa por parámetro un clan y el oscuro manda a su caterva contra ellos. Para ello hace que la primera bestia de la caterva arrase el primer terreno del clan, la segunda el segundo, y así sucesivamente. El método devuelve el número de terrenos arrasados.

⊙ *ayuda*: se le pasa por parámetro un producto. El oscuro castiga esta acción devolviendo el valor opuesto al valor del producto y guarda al principio de sus tributos dicho producto.

⊙ *getCaterva*: devuelve su caterva.

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:

- Modificación de **Habitante**
  - `public abstract boolean esAcogido(Clan c);`
  - `public abstract boolean esDesterrado();`
  - `public abstract void cambiaClan(String s);`
  - `public abstract Clan getTribu();`
- En **Plebeyo**
  - `public Plebeyo(String n,char c)`
  - `public ArrayList<String> trueque(Habitante h)`
  - `public double tributa(Mistico m)`
  - `public boolean plegaria()`
  - `public boolean domestica(Bestia b,String s)`
  - `public int recolecta(int i,int j)`
  - `public ArrayList<Producto> vasallaje(Terreno t)`

- `public int alimenta(int i)`
- `public boolean tutela(Guerrero g)`
- `public String libera()`
- `public Bestia getBestiola()`
- `public Guerrero getPatrono()`
- `public boolean esAcogido(Clan c)`
- `public boolean esDesterrado()`
- `public void cambiaClan(String s)`
- `public Clan getTribu()`

- En **Guerrero**
  - `public Guerrero(String n,char c)`
  - `public ArrayList<String> trueque(Habitante h)`
  - `public double tributa(Mistico m)`
  - `public boolean plegaria()`
  - `public Producto intercambio(Producto p)`
  - `public boolean acoge(Habitante h)`
  - `public int recolecta(int i)`
  - `public int lucha(Habitante h)`
  - `public Plebeyo getSirviente()`
  - `public ArrayList<Producto> getArmamento()`
  - `public boolean esAcogido(Clan c)`
  - `public boolean esDesterrado()`
  - `public void cambiaClan(String s)`
  - `public Clan getTribu()`
- En **Bestia**
  - `public Bestia(double p)`
  - `public String busca(Terreno r)`
  - `public boolean domestica(Habitante h)`
  - `public boolean alimenta(Producto p)`
  - `public ArrayList<Producto> pasturea(Terreno r,int k)`
  - `public double ayuda()`
  - `public Demoledor hechiza()`
  - `public void setNombre(String s)`
  - `public String getNombre()`
  - `public double getFuerza()`
  - `public Producto getAmuleto()`
  - `public Plebeyo getAmo()`
- En **Demoledor**
  - `public Demoledor(double p)`
  - `public boolean domestica(Habitante h)`
  - `public boolean alimenta(Producto p)`
  - `public ArrayList<Producto> pasturea(Terreno r,int k)`

    - `public double ayuda()`
    - `public double arrasa(Terreno r)`
    - `public Bestia exorciza()`
- En **Clan**
    - `public Clan(String n)`
    - `public String enfrenta(Clan c)`
    - `public boolean conquista(Terreno t)`
    - `public int laboro(int i)`
    - `public Habitante destierra(String s)`
    - `public int acoge(Habitante h)`
    - `public boolean adoptaDeidad(Mistico n)`
    - `public String getNombre()`
    - `public ArrayList<Habitante> getMiembros()`
    - `public Mistico getDeidad()`
    - `public ArrayList<Terreno> getFeudo()`
- En **Blanco**
    - `public Blanco(String n)`
    - `public boolean explora(String c)`
    - `public int exorcismo(int i)`
    - `public double ayuda(Producto p)`
    - `public void fervor(Clan c)`
    - `public String getNombre()`
    - `public ArrayList<Clan> getAcolitos()`
- En **Oscuro**
    - `public Oscuro()`
    - `public double culto(Bestia c)`
    - `public boolean malogra(Habitante h)`
    - `public int calamidades(Clan c)`
    - `public double ayuda(Producto p)`
    - `public ArrayList<Demoledor> getCaterva()`

⊛ Ninguno de los ficheros entregados en esta práctica debe 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 `correctorP2.tgz`. Para descomprimirlo se debe copiar este archivo donde queramos realizar las pruebas de nuestra práctica y ejecutar:

  ```
  tar xfvz correctorP2.tgz
  ```

  De esta manera se extraerán de este archivo:

  - El fichero `corrige.sh`: *shell-script* que tenéis que ejecutar.
  - El directorio `practica2-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 `practica1-prueba`, la orden: `diff -w p01.txt p01.tmp`