Práctica 2: **El cambio climático**

Programación II

Abril 2019 (versión 19/3/2019)

DLSI – Universidad de Alicante

Alicia Garrido Alenda

Normas generales

- El plazo de entrega para esta práctica se abrirá el lunes 8 de abril a las 9:00 horas y se **cerrará el viernes 12 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` y se enviará un informe al respecto tanto a la dirección del departamento como de la titulación.

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.

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

Debido a la creciente contaminación, los fenómenos meteorológicos y el aumento del tamaño del agujero en la capa de ozono, la mayor incidencia de los rayos gamma sobre las plantas ha tenido el efecto de crear una mutación. Así ha sido como han aparecido los trífidos. Un **Trifido** es una planta que tiene la capacidad de desplazarse y atacar.

A su vez las personas también han evolucionado, de manera que ahora hay algunas personas que son inmunes a los ataques de los trífidos, y son los únicos capaces de arrancar este tipo de plantas de las huertas. Además disponen de vehículos que les permiten trasladarse de una huerta a otra para intentar erradicar esta plaga.

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 (que pueden ser públicos o privados).

Un **Trifido** es una **Planta** que además se caracteriza por:

- ⁕ *`posicion`*: array de dos enteros que contiene la posición (fila,columna) que ocupa el trífido dentro del huerto.

Por tanto cuando se crea un **Trifido** se le pasa por parámetro una cadena para el nombre de la planta, el nombre de los frutos que produce y la cantidad máxima de ellos que puede tener. Inicialmente no tiene posición asignada.

Un **Trifido** puede realizar las mismas acciones que una **Planta**, pero algunas de ellas las realiza de forma diferente:

- ⊙ *`abona`*: se le pasa por parámetro un entero. Si está plantado en una huerta, buscará a su cuidador para atacarlo. Si el cuidador no es inmune y su estado es "`adulta`", transforma todos los frutos que tiene hasta ahora con la cantidad indicada en el parámetro y, a continuación, genera dos nuevos frutos si tiene capacidad para ello. Si su estado no es "`adulta`", cambia su estado a "`adulta`" y genera dos nuevos frutos. En cualquiera de los dos casos, si dispone de espacio para un solo fruto, lo generará, y los nuevos frutos se añadirán en las primeras posiciones libres que haya. Una vez es atacado el cuidador, es convertido en zombie y devuelve cierto.

  Si la huerta no tiene cuidador, o bien éste es inmune, y el trífido ha producido frutos, los utiliza, eliminándolos[1], para desplazarse:

  - si la huerta en la que está plantado está localizada a la huerta localizada más cercana, es decir la que tenga una menor distancia entre sus coordenadas[2];
  - si la huerta en la que está plantado no está localizada se desplaza a la primera huerta encontrada recorriendo el array `localizadas` desde la posición `0`;

  y que tenga cuidador. Entonces se planta en la primera posición libre del huerto, ignorando el hecho de que haya otras plantas en esa fila del huerto. Devuelve cierto si consigue desplazarse. Si no consigue desplazarse, se queda donde está y devuelve falso.

[1] Elimina sus frutos tanto si consigue desplazarse como si no.

[2] Si hay dos a la misma distancia se quedará en la primera encontrada recorriendo el array de localizadas desde la posición `0`.

En cualquier otro caso, devuelve falso.

- ⊙ *recolecta*: cuando intentan recolectar sus frutos, si los tiene (sean comestibles o no), el trífido intenta huir a la primera huerta localizada más próxima, con una menor distancia entre sus coordenadas[3] siempre que la huerta en la que está plantado esté localizada, situándose en la primera posición del huerto al que se ha desplazado. Si dicha posición tenía una planta, el trífido ocupa su lugar y devuelve un array dinámico con todos los frutos de dicha planta ordenados por su peso de mayor a menor[4]. Si la posición estaba libre, no devuelve ningún fruto. Si la huerta en la que está plantado actualmente el trífido no está localizada o bien no está plantado en ninguna, la huída no es posible y se recolecta como una planta normal para pasar desapercibido. Si el trífido no tiene frutos, no tiene motivos para huir y no devuelve ningún fruto.

- ⊙ *arranca*: cuando se intenta arrancar un trífido, éste se defiende atacando al cuidador de la huerta. Si el cuidador es inmune el trífido es arrancado de la huerta en la que está, por tanto deja de estar plantado en dicha huerta y de tener una posición dentro de ella. Si el cuidador es zombie la acción no tiene ningún efecto. En otro caso el cuidador es convertido en zombie y el trífido sigue plantado.

- ⊙ *setPlantada*: se le pasa por parámetro la huerta en la que ha sido plantado, de manera que el trífido se busca dentro de su huerto para poder determinar su posición dentro de él.

Además puede realizar las siguientes acciones:

- ⊙ *otea*: si no está plantado en ningún lugar, busca la primera huerta localizada que no tenga cuidador[5], y se coloca en la primera posición libre de su huerto, actualiza su huerta y su posición, cambia su estado a "`adulta`" y devuelve cierto. Si ya está plantado en alguna huerta, o bien la huerta a la que pretende desplazarse no dispone de posiciones libres, continúa sin estar plantado y devuelve falso.

- ⊙ *abona*: el trífido elimina la competencia devorando las plantas que se encuentran en las posiciones adyacentes a la suya dentro del huerto siempre que no sea otro trífido. Por ejemplo, si la imagen siguiente es un huerto, y el trífido ocupa la posición marcada con una "`t`", devora las plantas marcadas con una "`d`":

p	p	p	p	p
d	d	d	p	p
d	t	d	p	p
d	d	d	p	p
p	p	p	p	p

  Con la primera planta devorada el trífido cambia su propio estado a "`adulta`" si no lo era, y genera un fruto nuevo por cada planta devorada que no cambie el estado del trífido. El método devuelve el número de plantas devoradas.

- ⊙ *getPosicion*: devuelve la posición que ocupa el trífido en el huerto actualmente.

[3]Si hay dos a la misma distancia se quedará en la primera encontrada recorriendo el array `localizadas` desde la posición `0`.

[4]Cuando hay dos valores iguales, el segundo encontrado se coloca detrás del primero.

[5]Recorriendo el array `localizadas` desde la posición `0`.

Un **Inmune** es una **Persona** que además se caracteriza por:

- ⁕ *vehiculo*: vehículo con el que se puede desplazar de un punto a otro en un momento dado (`Vehiculo`);

Un **Inmune** por tanto se crea pasándole por parámetro una cadena con su nombre, no está al cuidado de ninguna huerta ni tiene vehículo inicialmente.

Un **Inmune** puede realizar las mismas acciones que una **Persona**, pero algunas de forma distinta:

- ⊙ *planta*: se le pasan por parámetro una planta y una huerta. Si la planta resulta ser un trífido, el inmune lo convierte en una planta normal e intenta plantar esta planta inocua en la huerta. Si la planta no es un trífido se intenta plantar directamente en la huerta. El método devuelve cierto si la acción se realiza con éxito, y falso en cualquier otro caso.

- ⊙ *malasHierbas*: recorre la huerta buscando los posibles trífidos que haya y los arranca cuando los encuentra. El método devuelve un array dinámico con las plantas arrancadas.

- ⊙ *abona*: se le pasa por parámetro un entero y un `String`, de manera que se abonan con la cantidad indicada por el primer parámetro, siempre que ésta sea mayor que `0`, las plantas de la huerta que producen frutos cuyo nombre coincida con la cadena pasada por parámetro. El método devuelve la cantidad de plantas que han sido abonadas, cambien o no su estado.

Y además puede realizar las siguientes acciones:

- ⊙ *apropia*: se le pasa por parámetro un vehículo. Si el inmune no tenía ya un vehículo asignado y el vehículo pasado por parámetro no tiene ocupante, el inmune se apropia del vehículo haciéndolo suyo, siendo el ocupante del mismo a partir de ese momento y devuelve cierto. En cualquier otro caso, devuelve falso.

- ⊙ *abandona*: el inmune abandona el vehículo que ha usado hasta ahora para sus traslados. El método devuelve cierto si abandona el vehículo, dejando de ser el ocupante del vehículo, y falso en cualquier otro caso.

- ⊙ *repostaje*: recolecta de todas las plantas de la huerta sus frutos comestibles y se los pasa a su vehículo para repostar combustible. Devuelve un array dinámico con los nombres de los frutos usados en el repostaje ordenados lexicográficamente.

- ⊙ *paseo*: se le pasa por parámetro una coordenada. Si el inmune es el cuidador de una huerta y dispone de vehículo, se traslada temporalmente a cada huerta localizada en dicha coordenada, en el orden en el que se encuentran en el array `localizadas`, para erradicar los posibles trífidos que pudiera haber y expulsar al cuidador si se trata de un zombie. Una vez hecha la limpieza, el inmune vuelve a su huerta. El método devuelve el número de trífidos erradicados en total.

- ⊙ *getVehiculo*: devuelve el vehículo que está usando actualmente.

Un **Zombie** es una **Persona** sin ninguna característica adicional pero que realiza las acciones de forma diferente, por tanto se crea pasándole por parámetro una persona de la que coge todas sus características.

Un **Zombie** puede realizar las mismas acciones que una **Persona**, pero de forma distinta:

⊙ *planta*: se le pasan por parámetro una planta y una huerta. Si la huerta no es la misma que la que cuida no puede realizar la acción, puesto que solo puede cuidar su propia huerta. Si la huerta es la que está a su cuidado, el zombie está al servicio de los trífidos, por tanto si la planta es un trífido intenta plantarlo en la huerta. Si la planta no es un trífido, la convierte en trífido y a continuación se intenta plantar en la huerta. El método devuelve cierto si la acción se realiza con éxito, y falso en cualquier otro caso.

⊙ *paseo*: un zombie no puede cambiar de la huerta que cuida, por tanto devuelve la coordenada de su huerta si la tiene. En cualquier otro caso devuelve `null`.

⊙ *malasHierbas*: recorre la huerta buscando las plantas que sean adultas y no sean trífidos y las arranca cuando las encuentra. El método devuelve un array dinámico con las plantas arrancadas.

⊙ *abona*: se le pasa por parámetro un entero y un `String`, el zombie ignora la cadena pasada por parámetro de manera que se abona todos los trífidos que encuentra con la cantidad indicada por el primer parámetro. El método devuelve la cantidad de plantas que han sido abonadas.

Además puede:

⊙ *abona*: el zombie busca los trífidos de la huerta y le pasa un mensaje para que se abonen devorando las plantas adyacentes. El método devuelve el número de plantas total devoradas por todos los trífidos.

Un **Vehiculo** se caracteriza por:

⁕ *ocupante*: persona que transporta (`Persona`);

⁕ *combustible*: depósito de barras con el combustible no contaminante disponible (cada posición cierta es una barra de combustible y estarán en posiciones consecutivas en el array) (`boolean[]`);

Cuando se crea un **Vehiculo** se le pasa por parámetro el tamaño de su combustible, que como mínimo debe poder tener dos barras de combustible; se crea sin ocupante inicialmente y con el depósito de combustible vacío.

Un **Vehiculo** puede realizar las siguientes acciones:

⊙ *traslada*: traslada al ocupante desde la posición que ocupe actualmente, determinada por la coordenada en la que esté localizada su huerta, hasta la coordenada pasada por parámetro.

Si realiza el trayecto devuelve cierto, de manera que:

- se busca la primera huerta en el array `localizadas` con la coordenada pasada por parámetro, empezando por la posición `0`; si esa huerta es distinta a la del ocupante, se actualiza la huerta del ocupante para que sea a la que ha sido trasladado; si se trata de la misma huerta, se busca en el array `localizadas` la siguiente huerta localizada en la misma coordenada, y si no la encuentra devuelve falso.
- si la coordenada pasada por parámetro no se corresponde con la de ninguna huerta localizada, traslada la huerta que cuida el ocupante a la nueva coordenada, de manera que es necesaria una actualización de datos.

En cualquier otro caso devuelve falso.

Para poder realizar el trayecto es necesario que la distancia sea mayor o igual que `0` y disponer de suficiente combustible, esto es que el número de barras de combustible sea superior o igual a la distancia existente entre las dos coordenadas, eliminando del combustible las consumidas al realizar el trayecto (parte entera de la distancia), empezando por la barra que ocupa la mayor posición en el array.

⊙ ***`repostaje`***: se le pasa por parámetro un array dinámico de frutos, que es de donde obtiene el combustible. El combustible generado por cada fruto es su valor calórico, por tanto el combustible extraído es la suma del combustible generado por todos los frutos pasados por parámetro. El número de barras que se añaden al combustible del depósito es la parte entera de la suma calculada, y se añaden empezando por la menor posición vacía del array. El método devuelve el número de barras añadidas al combustible.

⊙ ***`sube`***: se le pasa por parámetro la persona que desea subir al vehículo; esto solo es posible si en el vehículo no hay ocupante actualmente y dicha persona es un inmune que no tiene asignado otro vehículo, en cuyo caso pasa a ser el nuevo ocupante, que a partir de ahora tendrá este vehículo asignado, y devuelve cierto. En cualquier otro caso devuelve falso.

⊙ ***`baja`***: la persona que ha viajado en el vehículo baja del mismo. Esto solo es posible si en el vehículo había efectivamente un ocupante, en cuyo caso devuelve cierto, y el ocupante a su vez deja de tener un vehículo asignado. En cualquier otro caso se devuelve falso.

⊙ getOcupante: devuelve su ocupante.

⊙ getCombustible: devuelve su combustible.

Restricciones en la implementación

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

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

- En **Trifido**
  - `public Trifido(String,String,int)`
  - `public boolean abona(int)`
  - `public ArrayList<Fruto> recolecta()`
  - `public void arranca()`
  - `public void setPlantada(Huerta)`
  - `public boolean otea()`
  - `public int abona()`
  - `public int[] getPosicion()`

- En **Inmune**
  - `public Inmune(String)`
  - `public boolean planta(Planta,Huerta)`
  - `public ArrayList<Planta> malasHierbas()`
  - `public int abona(int,String)`
  - `public boolean apropia(Vehiculo)`
  - `public boolean abandona()`
  - `public ArrayList<String> repostaje()`
  - `public int paseo(Coordenada)`
  - `public Vehiculo getVehiculo()`
- En **Zombie**
  - `public Zombie(Persona)`
  - `public boolean planta(Planta,Huerta)`
  - `public Coordenada paseo()`
  - `public ArrayList<Planta> malasHierbas()`
  - `public int abona(int,String)`
  - `public int abona()`
- En **Vehiculo**
  - `public Vehiculo(int)`
  - `public boolean traslada(Coordenada)`
  - `public int repostaje(ArrayList<Fruto>)`
  - `public boolean sube(Persona)`
  - `public boolean baja()`
  - `public Persona getOcupante()`
  - `public boolean[] getCombustible()`

⊛ 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`