Práctica 1: **Transporte galáctico**

Programación II

Marzo 2017 (versión 12/2/2017)

DLSI – Universidad de Alicante

Alicia Garrido Alenda

Normas generales

- El plazo de entrega para esta práctica se abrirá el lunes 6 de marzo a las 9:00 horas y se **cerrará el viernes 10 de marzo 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  practica1.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

Los navegantes son una raza nómada que viaja a través de la *Galaxia*, que está formada por sectores. Cada *Sector* está situado en una *Coordenada* y dispone de su propio almacén, donde se pueden almacenar y recoger objetos de tipo *Mercancia*. En concreto, disponen de un tipo de nave espacial básico, capaz de realizar viajes interestelares de forma autotripulada. Dichas naves son las *Romskip*, a las que se le pasan las coordenadas de destino para poder calcular la viabilidad del viaje y realizarlo. Además cada *Romskip* dispone de su bodega de carga donde transporta las mercancías.

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.

Una **Mercancia** se caracteriza por:

- *peso*: indica el peso de la mercancía (`double`);
- *dimension*: array de reales que contendrá las distintas dimensiones de la mercancía (`double[]`);
- *etiqueta*: identificación de la mercancía (`String`);
- *almacenada*: indicador de carga para saber si una mercancía está almacenada en algún sitio (`boolean`);

Cuando se crea una **Mercancia** se le pasan por parámetro su peso y sus dimensiones. El peso debe ser mayor o igual a `0`, en otro caso se le asigna un peso estándar de `1.5`. Para las dimensiones se le pasa un array del cual debe realizar una copia en sus dimensiones, si el array pasado por parámetro es válido; en otro caso se genera un array de dimension `1` que contendrá la dimensión por defecto `0.5`. La etiqueta inicialmente será `null` y la mercancía inicialmente no está almacenada en ningún sitio.

Las acciones que se pueden realizar con una **Mercancia** son:

- *almacena*: se le pasa por parámetro una cadena[1]. Esta acción se realiza cuando se almacena la mercancía en algún sitio. Si no tiene ya una etiqueta generada, con dicha cadena y los datos relativos a su peso y dimensiones se genera la etiqueta de la mercancía, ya que ésta se genera cuando una mercancía es almacenada en algún sitio por primera vez. Para ello se concatena a la cadena pasada por parámetro el peso, un guión (`-`) y la suma de sus dimensiones con un solo decimal. Si ya tiene una etiqueta, ésta no se vuelve a generar.

  Si la mercancía no está almacenada en ningún sitio se actualiza el valor de su indicador de carga y devuelve cierto. En cualquier otro caso devuelve falso.

- *recogida*: si su indicador de carga está a cierto, se actualiza su valor a falso y el método devuelve cierto. En cualquier otro caso el método devuelve falso.

- *iguales*: se le pasa por parámetro un objeto de tipo *Mercancia*. El método devuelve cierto si el parámetro no es `null` y su etiqueta[2], peso y dimensiones tienen exactamente los mismos valores que esta mercancía (sobre la que se invoca el método).

[1] Esta cadena nunca será `null`.

[2] Se deben ignorar diferencias por mayúsculas/minúsculas, es decir, la etiqueta "agua1.5-2.0" es la misma que la etiqueta "AGUA1.5-2.0", por ejemplo.

- ⊙ *duplicado*: crea y devuelve un objeto de tipo *Mercancia* con los mismos valores que las características de esta mercancía (sobre la que se invoca el método).
- ⊙ *getDimension*: devuelve la suma de sus dimensiones.
- ⊙ *getAlmacenada*: devuelve el indicador de carga.
- ⊙ *getPeso*: devuelve el peso.
- ⊙ *getEtiqueta*: devuelve la etiqueta si esta ha sido generada. En otro caso devuelve una referencia vacía (`null`).

Los sectores de una galaxia se ubican en unas coordenadas determinadas. Una **Coordenada** se caracteriza por:

- ⋇ *i*: indica la fila (`int`);
- ⋇ *j*: indica la columna (`int`);
- ⋇ *k*: indica la posición de la tercera dimensión (`int`);

Una **Coordenada** se crea pasándole por parámetro tres enteros; el primero para la fila, el segundo para la columna y el tercero para la tercera dimensión.

Las acciones que se pueden realizar con una **Coordenada** son:

- ⊙ *getCoordenadas*: devuelve un array de 3 enteros que contiene las coordenadas `i`, `j` y `k`, en este orden.

Un **Sector** de la *Galaxia* se caracteriza por:

- ⋇ *coordenada*: objeto que contiene la ubicación del sector dentro de la galaxia (`Coordenada`);
- ⋇ *almacen*: array que contendrá los objetos de mercancías disponibles en este sector (`Mercancia[]`);
- ⋇ *existencias*: array de enteros que contendrá la cantidad disponible de una determinada mercancía(`int[]`);

Un **Sector** se crea pasándole por parámetro un entero que indica el número máximo de objetos de mercancías que puede contener este sector, con el que debe crear su almacen y su array de existencias si el número es mayor que `0`. En otro caso el sector no dispondrá de almacén ni de existencias.

Las acciones que se pueden realizar con un **Sector** son:

- ⊙ *situa*: se le pasan por parámetro tres enteros correspondientes a las coordenadas que ocupa este sector dentro de la galaxia. Si el sector no tenía coordenadas asignadas previamente, debe crear sus propias coordenadas con estos datos y devuelve cierto. En cualquier otro caso devuelve falso.
- ⊙ *almacena*: se le pasa por parámetro una mercancía para ser almacenada. Para ello se deben cumplir una serie de condiciones:
  - La mercancía no puede estar almacenada en otro sitio actualmente.
  - El sector debe estar situado en alguna coordenada.

Ahora se pueden dar dos circunstancias:

1. Si el sector no tiene almacenada una mercancía con la misma etiqueta (ignorando diferencias entre mayúsculas/minúsculas), el mismo peso y las mismas dimensiones, debe disponer de espacio libre en su almacén para la nueva mercancía, que se almacenará en la primera posición libre de su *`almacen`* y devuelve cierto. Si es la primera vez que dicha mercancía es almacenada, es necesario generar su etiqueta, para esto se utiliza como cadena la concatenación de las coordenadas `i`, `j`, `k` que ocupa el sector.

   Por ejemplo, si se almacena por primera vez una mercancía con peso `2.3` y dimensiones `[2.2,3.3]` en el sector situado en las coordenadas `[3,7,4]`, la etiqueta que se generaría sería `3742.3-5.5`.

2. Si el sector ya tiene una mercancía almacenada con las mismas características, no ocupa una posición nueva del almacén, sino que incrementa el número de existencias de dicha mercancía y devuelve cierto.

En cualquier otro caso la mercancía no se almacena y se devuelve falso.

⊙ *`recoge`*: se le pasa por parámetro una cadena. Busca entre sus mercancías el primer objeto cuya etiqueta coincida con la cadena pasada por parámetro, ignorando diferencias entre mayúsculas/minúsculas. Si no encuentra ninguno devolverá una referencia vacía. Si lo encuentra devolverá el objeto encontrado, decrementando el número de unidades de dicha mercancía y dejando una posición vacía en el almacén si sólo había una unidad de dicha mercancía, en otro caso se devuelve una copia de dicho objeto.

⊙ *`recuento`*: se le pasa una cadena y devuelve el número de unidades disponibles de las mercancías que tengan como etiqueta dicha cadena, ignorando diferencias entre mayúsculas/-minúsculas.

⊙ *`disponibilidad`*: devuelve un array de `String` del mismo tamaño que el almacén, que contendrá en cada posición la etiqueta de la mercancía que ocupa la correspondiente posición en el almacén.

⊙ *`capacidadDisponible`*: devuelve el número de posiciones libres en el almacén.

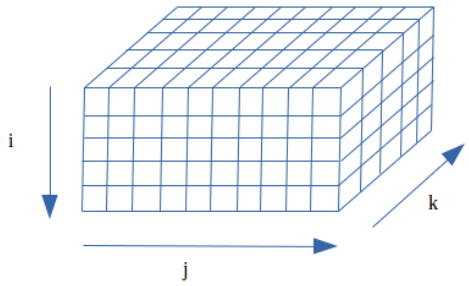
⊙ *`getCoordenada`*: devuelve la coordenada del sector.

Una **Galaxia** es un espacio tridimensional que se compone de sectores, por tanto se caracteriza por:

⋇ *`nombre`*: nombre de la galaxia (`String`);

⋇ *`espacio`*: matriz tridimensional de sectores (`Sector[][][]`);

Cuando se crea una **Galaxia** se le pasa por parámetro una cadena para su nombre y tres enteros, correspondientes a las tres dimensiones para poder crear su espacio. El primero se corresponde con el número de filas de la matriz (`i`), el segundo con el de las columnas (`j`) y el tercero para la dimensión de profundidad (`k`).



Cada entero debe ser mayor que `0`, si alguno no es mayor que `0` dicho entero (sólo ese, los demás mantendrán su valor) tomará por defecto el valor `2`. Inicialmente una galaxia no tiene ningún sector en su espacio, por tanto una galaxia puede quedar con sectores sin crear que se consideran como agujeros negros que no se pueden visitar.

Las acciones que se pueden realizar con una **Galaxia** son:

- ⊙ *`coloca`*: se le pasa por parámetro un sector, de manera que tiene que colocarlo en las coordenadas indicadas por el propio sector, si dichas coordenadas pertenecen al espacio de la galaxia y dicha ubicación no está ya ocupada por otro sector, devolviendo cierto. En cualquier otro caso devuelve falso. En todos los casos las coordenadas deben estar dentro del rango `[0,n-1]`, siendo `n` el tamaño de la dimensión correspondiente.

- ⊙ *`existencias`*: se le pasa una cadena y busca en qué sectores hay disponibles mercancías cuya etiqueta coincida esta cadena, ignorando diferencias entre mayúsculas/minúsculas, y devuelve un array con todas las coordenadas de estos sectores, ordenadas en primer lugar por la fila, en segundo por la columna y en tercero por la profundidad que ocupan dentro de la galaxia. Si no hay sectores con esa mercancía devuelve una referencia vacía.

- ⊙ *`getNombre`*: devuelve el nombre de la galaxia.

- ⊙ *`getAgujerosNegros`*: devuelve el número de agujeros negros que tiene la galaxia.

Las **Romskip** son las naves que utilizan los navegantes para transportar las mercancías de un sector a otro de una misma galaxia. Una **Romskip** se caracteriza por:

- ⁕ *`nombre`*: nombre de la nave (`String`);
- ⁕ *`bodega`*: su bodega de carga, es decir, donde se transportan las mercancías (`Mercancia[]`);
- ⁕ *`carga`*: el peso máximo que puede transportar la nave (`double`);
- ⁕ *`mapa`*: galaxia donde se situan las naves, compartida por todas ellas (`static Galaxia`);

Cuando se crea una **Romskip** se le pasan por parámetro un `String`, un entero y un real. El `String` es el nombre[3] de la nave. El entero indica la capacidad de la bodega de la nave, que debe ser mayor que `0`, en otro caso toma por defecto el valor `3`. El número real indica la carga máxima de la nave, que debe ser mayor que `0`, en otro caso toma por defecto el valor `8.5`.

Las acciones que se pueden realizar con una **Romskip** son:

[3]El nombre pasado por parámetro al constructor siempre será distinto de `null`.

- ⊙ *embarque*: se le pasa por parámetro una cadena y una coordenada. Se comprueba si la coordenada se corresponde con un sector de la galaxia, ya que debe recoger la primera mercancía que encuentre en ese sector cuya etiqueta coincida con la cadena pasada por parámetro, ignorando diferencias entre mayúsculas y minúsculas. Si encuentra una mercancía con esa etiqueta, tiene que almacenarla en la bodega de carga de la nave, pero antes de almacenarla se deben realizar una serie de comprobaciones, de manera que si alguna no se cumple no se realiza el embarque y se devuelve una cadena con el motivo del fallo en el embarque. Las comprobaciones a realizar son:

  1. si queda espacio en la bodega para almacenar la nueva mercancía; si no hay espacio se devuelve una cadena en la que se concatena la etiqueta seguida de un mensaje, de la siguiente manera:

     ```
     agua1.5-2.0: bodega de carga completa
     ```

  2. si no se sobrepasa el peso máximo que puede transportar la nave. De ser así, no se realiza el embarque de la mercancía y se devuelve una cadena en la que se concatena la etiqueta seguida de un mensaje, de la siguiente manera:

     ```
     agua1.5-2.0: sobrecarga
     ```

  donde `agua1.5-2.0` es un posible ejemplo de etiqueta de la mercancía que no se puede almacenar. Si se da más de un motivo por el que no se puede realizar el embarque, el orden de comprobación para la generación de la cadena es el orden en el que aparecen aquí. Si no se almacena la mercancía debe permanecer en el sector.

  Si se cumplen las condiciones necesarias se procede al almacenamiento de la mercancía. En la bodega las mercancías que deben estar ordenadas en función de la suma de sus dimensiones, de manera que los objetos cuya suma sea mayor queden al fondo (final) de la bodega y aquellos cuya suma sea menor queden delante (principio), dejando todas las posiciones libres juntas al final, y se devuelve una referencia vacía. Si en la bodega de carga ya hay una mercancía con la misma dimensión, la nueva se almacena a continuación de la existente.

- ⊙ *desembarque*: se le pasa por parámetro una cadena. Busca entre sus mercancías el primer objeto cuya etiqueta coincida con la cadena pasada por parámetro, ignorando diferencias entre mayúsculas/minúsculas. Si no encuentra ninguno devolverá una referencia vacía. Si lo encuentra devolverá el objeto encontrado, dejando una posición vacía en la bodega de carga, que debe compactarse de manera que queden todas las posiciones vacías al final.

- ⊙ *conteo*: este método se utiliza para saber cuántos objetos hay en la bodega de carga de la nave en un momento determinado.

- ⊙ *getPeso*: devuelve el peso total que la nave transporta en ese momento en la bodega de carga.

- ⊙ *teletransporte*: se le pasa por parámetro la nave a la que se va a teletransportar la mercancía que ocupa la posición indicada por el entero que se le pasa por parámetro, de manera que si la teletransportación se lleva a cabo con éxito la mercancía teletransportada deja de estar en la bodega de la nave, que debe compactarse de manera que queden todas las posiciones vacías al final, y se devuelve cierto. En cualquier otro caso se devuelve falso.

- ⊙ *verifica*: se le pasa por parámetro una mercancía que está siendo teletransportada. Si no supera la carga máxima de la nave, la almacena en la bodega en función de la suma de sus

dimensiones para mantener el orden de las mercancías. Si no hay posiciones vacías disponibles para almacenar esta mercancía, se tiene que redimensionar la bodega para dar cabida a esta nueva mercancía. El método devuelve cierto si finalmente almacena la mercancía y falso en cualquier otro caso.

⊙ *getBodega*: devuelve la bodega de la nave.

⊙ *getNombre*: devuelve el nombre de la nave.

⊙ *getNombreMapa*: devuelve el nombre de la galaxia en la que están actualmente situadas las naves.

⊙ *setMapa*: se le pasa por parámetro una galaxia para actualizar el mapa en el que se situan las naves.

⊙ *hayAgujerosNegros*: devuelve cierto si el mapa contiene agujeros negros.

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 **Coordenada**
  - `public Coordenada(int i,int j,int k)`
  - `public int[] getCoordenadas()`
- En **Mercancia**
  - `public Mercancia(double p,double[] d)`
  - `public boolean almacena(String s)`
  - `public boolean recogida()`
  - `public boolean iguales(Mercancia m)`
  - `public Mercancia duplicado()`
  - `public double getDimension()`
  - `public boolean getAlmacenada()`
  - `public double getPeso()`
  - `public String getEtiqueta()`
- En **Sector**
  - `public Sector(int n)`
  - `public boolean situa(int i,int j,int k)`
  - `public boolean almacena(Mercancia o)`
  - `public Mercancia recoge(String s)`
  - `public int recuento(String s)`
  - `public String[] disponibilidad()`
  - `public int capacidadDisponible()`
  - `public Coordenada getCoordenada()`

- En **Galaxia**
  - `public Galaxia(String s,int i,int j,int k)`
  - `public boolean coloca(Sector s)`
  - `public Coordenada[] existencias(String s)`
  - `public String getNombre()`
  - `public int getAgujerosNegros()`
- En **Romskip**
  - `public Romskip(String n,int i,double x)`
  - `public String embarque(String s,Coordenada c)`
  - `public Mercancia desembarque(String s)`
  - `public int conteo()`
  - `public double getPeso()`
  - `public boolean teletransporte(Romskip r,int i)`
  - `public boolean verifica(Mercancia m)`
  - `public Mercancia[] getBodega()`
  - `public String getNombre()`
  - `public static String getNombreMapa()`
  - `public static void setMapa(Galaxia g)`
  - `public static boolean hayAgujerosNegros()`

⊛ 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 `correctorP1.tgz`. Para descomprimirlo se debe copiar este archivo donde queramos realizar las pruebas de nuestra práctica y ejecutar:

  ```
  tar xfvz correctorP1.tgz
  ```

  De esta manera se extraerán de este archivo:

  - El fichero `corrige.sh`: *shell-script* que tenéis que ejecutar.
  - El directorio `practica1-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`