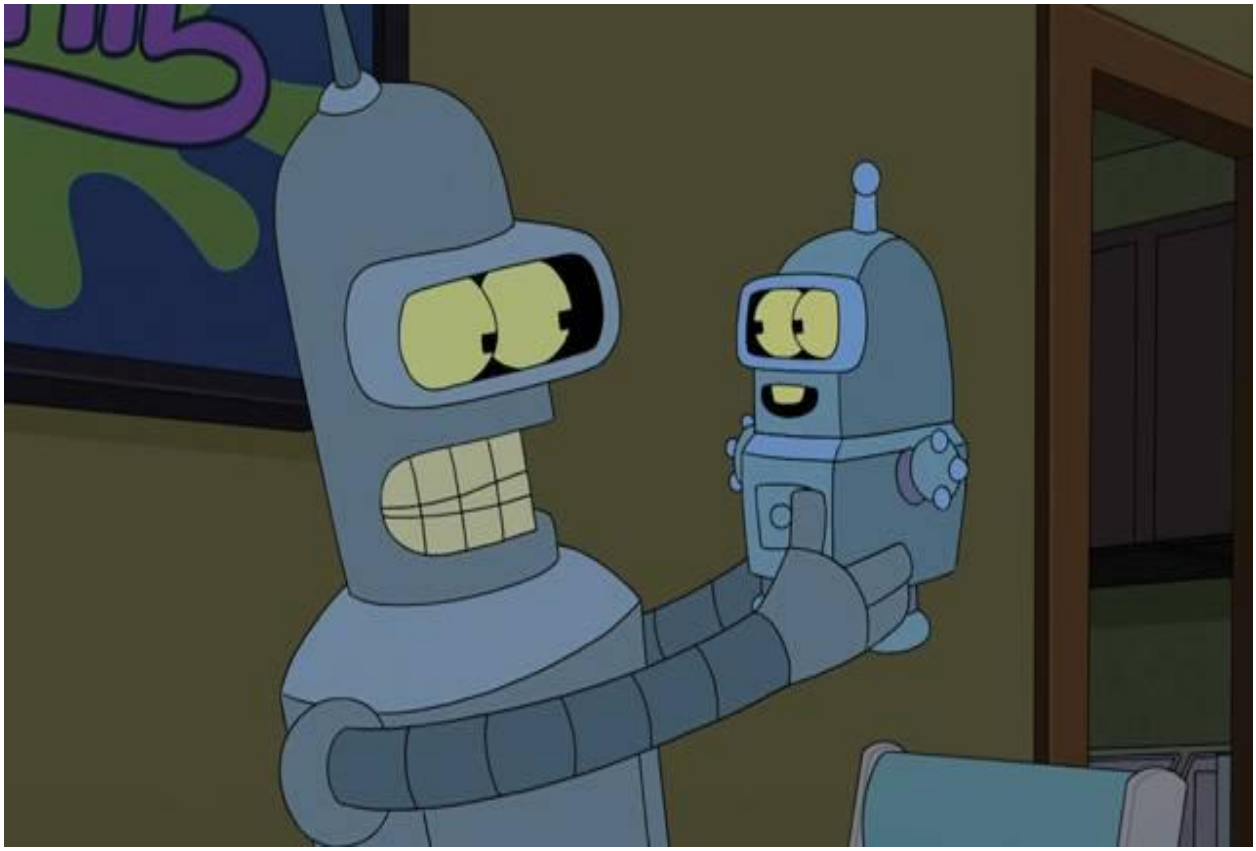
BENDER



INDICE

MAPA

En esta práctica se nos pide programar un robot, al que llamaremos Bender, para que a partir de un mapa generado por los tests sea capaz de llegar desde el punto de partida (representado con una X) hasta su objetivo (representado con un "$").

Este es un ejemplo de un mapa:

```
"" +
                    "##########\n" +
                    "######    #\n" +
                    "#$   #    #\n" +
                    "#### #    #\n" +
                    "#  # #    #\n" +
                    "#  #X#    #\n" +
                    "#  #I#    #\n" +
                    "#  ###    #\n" +
                    "#         #\n" +
                    "##########";
```

Hay una serie de condiciones a tener en cuenta:

Las paredes están representadas en el mapa con el símbolo "#".

El robot tiene una orden de prioridades en cuanto a la dirección que tiene que seguir para llegar al destino. Este orden es "Sur, Este, Norte, Oeste". Pero estas prioridades cambian en el momento en que el robot cae en la posición del mapa que contenga una "I" (inversores). En ese caso, el orden pasa a ser "Norte, Oeste, Sur, Este".

También debemos considerar que hay unas casillas con la letra T. Estas casillas son teletransportadores que se encargan de mandar al robot al teletransportador más

cercano. En caso de que haya dos teletransportadores a la misma distancia, el robot saldrá por el primero guiándonos por el sentido de las agujas del reloj.

Lo primero que debemos hacer es declarar diferentes variables del tipo char que guardarán los caracteres clave para el desarrollo de la práctica. Esto nos servirá para identificar los puntos de partida, de objetivo y de las casillas especiales mientras vayamos iterando el mapa

Lo segundo que tenemos que hacer es declarar una String que se encargue de generar el mapa que le pasará el test. Deberemos utilizar el método de las Strings "split" con un bucle para que, cada vez que la función encuentre un salto de línea ("/n"), el contador se incremente y se actualice el número de filas de forma que el número de filas siempre será igual al número de saltos de línea más 1. De esta manera, generamos el mapa siempre con el número de filas correcto de acuerdo a lo que nos proporcione el test.

De igual manera tenemos que crear otro bucle que se encargue de contar cuantos caracteres hay en cada fila y de esta forma sabremos el número de columnas que contiene nuestro mapa.

Para ello también será necesario crear una variable que guarde el número máximo de caracteres que hay en cada fila del mapa, ya que la anchura del mismo puede variar según avanzan las filas y con ello, varía el número de columnas.

El siguiente paso por el que pasará nuestro programa una vez generado el mapa será crear una función que itere posición por posición todo el mapa a través de las coordenadas X e Y. Crearemos una serie de condiciones que, mientras iteramos el mapa, comprobarán mediante el método charAt si la posición en cuestión contiene algún carácter clave (#, T, I, X o $), ya sea la posición inicial del robot, el objetivo o alguna de las casillas especiales.

Estos datos serán necesarios también para guardar las coordenadas en las que se encuentran los puntos clave del mapa, así como para recrear el mapa posición por posición. Tendremos dos variables, X e Y, las cuales utilizando el constructor, mediante el método this, y utilizando un setter and getter vamos actualizando en todo momento mientras iteramos el mapa. Al mismo tiempo guardaremos el carácter que corresponda a cada coordenada para generar nuestro mapa.

RECORRIDO

Para el movimiento de Bender hemos declarado una serie de variables importantes:

```
private int xActual;
private int yActual;
private int xObjetivo;
private int yObjetivo;

private int indiceDireccionActual;

private char ordenPrioridadDireccionNormal[] = new char[]{
        DIRECCION_SUR, DIRECCION_ESTE, DIRECCION_NORTE,
DIRECCION_OESTE};
private char ordenPrioridadDireccionInverso[] = new char[]{
        DIRECCION_NORTE, DIRECCION_OESTE, DIRECCION_SUR,
DIRECCION_ESTE};
private char ordenPrioridadDireccionActual[] =
ordenPrioridadDireccionNormal;
```

Necesitaremos saber en todo momento las coordenadas en las que nos encontramos y las coordenadas en las que se encuentra nuestro objetivo. Tenemos un boolean que las comparará y cuando coincidan se dejará de generar el mapa y se habrá pasado el test.

He generado tambien tres arrays de chars: Uno con el orden de prioridades predeterminado, otro con el orden invertido para cuando el robot cae en la casilla de la “I” y otro que guarda el orden actual, que empieza siendo el normal. También creamos un int que guardará el indice de estos arrays de las direcciones (0, 1, 2, 3).

Estas son las constantes que he utilizado para las posibles casillas del mapa:

```
final char POSICION_VACIA = ' ';
    final char POSICION_MURO = '#';
    final char POSICION_PORTAL = 'T';
    final char POSICION_INICIO = 'X';
    final char POSICION_OBJETIVO = '$';
    final char POSICION_INVERSOR = 'I';

    final char DIRECCION_SUR = 'S';
```

```
    final char DIRECCION_NORTE  = 'N';
    final char DIRECCION_OESTE = 'W';
    final char DIRECCION_ESTE = 'E';
```

Una vez hemos generado el mapa de acuerdo a lo que nos pasa el test, debemos identificar desde que casilla parte el robot. Hemos guardado en el char `POSICION_INICIO` el caracter “X” y tenemos dos variables (xActual, yActual) que guardarán las coordenadas del punto de partida y de los siguientes movimientos que hagamos en el mapa.

Para movernos de coordenadas siempre tendremos que avanzar en función de la prioridad establecida. Cambiaremos las coordenadas siguiendo este patrón:

```
private Posicion obtenerSiguientePosicion(char direccion) {
    Posicion p = new Posicion(xActual, yActual);

    switch(direccion) {
        case DIRECCION_ESTE:
            p.setX(p.getX() + 1);
            break;
        case DIRECCION_OESTE:
            p.setX(p.getX() - 1);
            break;
        case DIRECCION_NORTE:
            p.setY(p.getY() - 1);
            break;
        case DIRECCION_SUR:
            p.setY(p.getY() + 1);
            break;
    }
    return p;
}
```

Así modificaremos los ejes de coordenada para que se correspondan con las coordenadas nuevas en función de la dirección que tengamos que seguir.

Para avanzar de casilla siguiendo la prioridad primero tenemos que comprobar si la casilla es válida. Siempre será válida a no ser que se trate de un muro (#). Cuando topemos con un muro tendremos que cambiar la dirección siguiendo la prioridad.

El siguiente fragmento de código sirve para, a través del indice del array de chars de las direcciones, establecer la siguiente dirección hacia la que avanzará el robot si se topa con un muro.

```
private void cambiarDireccionSegunPrioridad() {
        indiceDireccionActual = (indiceDireccionActual + 1) %
ordenPrioridadDireccionActual.length;
    }
```

Cabe recordar que cuando encontramos un inversor se invierte el orden de las direcciones.

Este fragmento de código se encarga de cambiar un array de chars por el otro, es decir, el normal por el invertido y el invertido por el normal cuando nos encontramos un inversor:

```
 private void invertirPrioridadMovimiento() {
        if (ordenPrioridadDireccionActual ==
ordenPrioridadDireccionNormal) {
            ordenPrioridadDireccionActual =
ordenPrioridadDireccionInverso;
        } else {
            ordenPrioridadDireccionActual =
ordenPrioridadDireccionNormal;
        }
    }
```

Si la posición siguiente basándonos en la dirección predeterminada no es válida, volveremos a poner a 0 el indice del array de chars para que empiece desde el principio con las direcciones.

Debemos ir actualizando la posición actual de X e Y con la siguiente posición válida para avanzar.

Por último, basta con resolver los teletransportadores. Se supone que el robot cuando cae en una casilla cuyas coordenadas corresponden con una “T”, este tiene que salir por la “T” más cercana.

Para llevar a cabo esto debemos realizar un cálculo conocido como distancia euclidiana.

He creado una función para realizar este cálculo:

```
 public static double distanciaEuclidea(Posicion p1, Posicion p2) {
        return Math.sqrt(Math.pow(p1.x - p2.x,2) + Math.pow(p2.y - p1.y,
2));
    }
```

En el siguiente fragmento del código básicamente guardamos las coordenadas de las casillas que contienen una T y, partiendo desde las coordenadas actuales, utilizamos ese cálculo de arriba para elegir siempre el Teletransportador que se encuentre a menor distancia, es decir, el más cercano.

```
private Posicion obtenerPortalMasProximo() {
        Posicion portalActual = new Posicion(xActual, yActual);

        Posicion masProximo = null;
        double distanciaMinima = Double.MAX_VALUE;

        for (int p = 0; p < portales.length; p++) {
            Posicion portal = portales[p];
            if (portal.equals(portalActual)) {
                continue;
            }

            double dist = Posicion.distanciaEuclidea(portal,
portalActual);
            if (dist < distanciaMinima) {
                masProximo = portal;
                distanciaMinima = dist;
            }
        }

        return masProximo;



    }
```

Hemos creado el objeto posición para guardar las coordenadas.

Guardamos en una String vacía las direcciones que hemos ido tomando y cada vez que tomamos una nueva la acumulamos en el String. Esta String es lo que le devolvemos al test.