SOLUCIÓN EJERCICIOS DE UTILIZACIÓN DE TIPOS ABSTRACTOS DE DATOS

1. El juego *pasiegos vs zombies*.

(1) Especificar que atributos contendrían las clases *Zombie* y *Pasiego* resaltando los invariantes que deben cumplir sus atributos.

```
CLASE ABSTRACTA Ser ES
    // Inv: id != 0
    // Inv: s1, s2: Ser; (s1.getId() ==  s1.getId() ) iff (s1.igual(z2))
    HEREDABLE id: NATURAL;
    // Inv: (vida <= 100)
    HEREDABLE vida : Natural;
    // Inv: nombre != NULO
    HEREDABLE nombre:  CadenaCaracteres;
    // Inv: primerApellido!= NULO
    HEREDABLE primerApellido:  CadenaCaracteres;
    // Inv: segundoApellido!= NULO
    HEREDABLE segundoApellido:  CadenaCaracteres;

    // Asumimos la existencia de getters para todos estos atributos
    // Asumimos la existencia de un setters para el atributo vida.
FINCLASE

CLASE Zombie HEREDA Ser ES
FINCLASE

CLASE Pasiego HEREDA Ser ES
     sobaos : NATURAL;
    // Inv: (muertos != NULO)
    muertos : Lista<Zombie>
    // Inv: p1, p2: Pasiego; (p1.getMote() ==  p1.getMote() ) iff [1](p1.igual(p2))
    // Inv: mote != NULO
    HEREDABLE mote :  CadenaCaracteres;

    // Asumimos la existencia de getters para todos estos atributos
    // Asumimos la existencia de un setters para el atributo sobaos.
FINCLASE
```

(2) Especificar los atributos de una clase *Juego*, indicando que TADs son los más adecuados para almacenar tanto los pasiegos como los zombies.

Debemos almacenar pasiegos y zombies. La operación *onPixelClicked* del componente *GameZone* deberá recuperar la información de un pasiego o de un zombie a partir de su identificador. Por tanto, una tabla o mapa que asocie identificadores con sus correspondientes objetos pasiegos o zombies parece una buena elección. La operación *onPasiegoSelected*

[1] iff significa si y sólo si

necesita recuperar un objeto pasiego a partir de su mote. Por tanto, crearemos otra tabla o mapa usando el mote como llave y asociando a cada mote su correspondiente pasiego.

Las operaciones lanzamiento de sobao y lanzamiento de bolo recuperan zombies y personas a partir de sus identificadores. Por tanto, no hace falta crear un nuevo mapa o tabla.

La operación *onPixelClicked* es de suponer que tendrá que recuperar los datos de un pasiego o un zombie a partir de su identificador, el cual se encuentra almacenado en la matriz de la pantalla. En principio no sabemos si se tratará de un pasiego o zombie. Podríamos pensar en usar una sola tabla de seres, pero dado que buscar en una tabla puede ser bastante eficiente, para mejorar la comprobación estática de tipos, mantendremos dos tablas separadas, una para zombies y otra para pasiegos.

Por tanto, los TASDs a usar serían:

HEREDABLE mapaZombies : Mapa<Natural,Zombie>;
HEREDABLE mapaPasiegos : Mapa<Natural,Pasiego>;
HEREDABLE pasiegosPorMote : Mapa<CadenaCaracteres,Pasiego>;

Con los siguientes invariantes:
// Inv: El mapa de zombies existe
mapaZombies != NULO
// Inv: El mapa de pasiegos existe
mapaPasiegos != NULO
// Inv: El mapa de pasiegos por existe
pasiegosPorMote != NULO
// Inv: Si una entrada existe en mapaZombies, no está vacía
id: Natural; SI mapaZombies.contains(id) ENTONCES mapaZombies.get(id) != NULO;
// Inv: Si una entrada existe en mapaZombies, la asociación es correcta
id: Natural; SI mapaZombies.contains(id) ENTONCES (mapaZombies.get(id) == id);
// Inv: Si una entrada existe en mapaPasiegos, no está vacía
id: Natural; SI mapaPasiegos.contains(id) ENTONCES (mapaPasiegos.get(id) != NULO);
// Inv: Si una entrada existe en mapaPasiegos, la asociación es correcta
id: Natural; SI mapaPasiegos.contains(id) ENTONCES (mapaPasiegos.get(id) == id);
// Inv: Si una entrada existe en pasiegosPorMote, no está vacía
mote:CadenaCaracteres;
SI pasiegosPorMote.contains(mote) ENTONCES (pasiegosPorMote.get(mote) != NULO);
// Inv: Si una entrada existe pasiegosPorMote, la asociación es correcta
mote:CadenaCaracteres; SI pasiegosPorMote.contains(mote) ENTONCES
(pasiegosPorMote.get (mote).getMote() == id);
// Inv: Si un pasiego está dado de alta el mapa de pasiegos por mote también lo está en el
// mapa de pasiegos
mote:CadenaCaracteres; SI pasiegosPorMote.contains(mote) ENTONCES
mapaPasiegos.contains(pasiegosPorMote.get(mote).getId());

```
// Inv: Si un pasiego está dado de alta el mapa de pasiegos también lo está en el mapa de
//      pasiegos por mote
id: NATURAL;
    SI mapaPasiegos.contains(id) ENTONCES
      pasiegosPorMote.contains(mapaPasiegos.get(id).getMote());
```

(3) Para cada TAD identificado en el punto 2, indicar que implementación resultaría la más adecuada, justificándolo adecuadamente.

Para el mapa *mapaZombies*, su número no crece y no existe necesidad alguna de mantenerlos ordenados. Por tanto, una tabla de dispersión parece adecuada, ofreciendo buenos tiempos de inserción, búsqueda y borrado si somos capaces de diseñar una buena función de dispersión.

Para el mapa *mapaPasiegos* se aplica el mismo razonamiento que para los zombies.

Para el mapa *pasiegosPorNombre* hay que tener en cuenta que una operación frecuente será obtener la lista de pasiegos ordenados por mote, con objeto de poder visualizarlos en el componente *PasiegoGraphicalList*. Por tanto, un árbol binario de búsqueda autobalanceado parece una estructura adecuada, usando el mote de los pasiegos como criterio de comparación a la hora de insertar los pasiegos en dicho árbol. Como el mote es la llave de dicho mapa, no hay ningún problema.

Ahora debemos elegir entre un árbol roji-negro o un árbol AVL. Teniendo en cuenta que los pasiegos pueden nacer y morir durante el juego, es de suponer que haya un número decente de inserciones y eliminaciones de dicho árbol, pero en cualquier caso parece que el número de consultas será superior. Por tanto, al primar las consultas sobre las inserciones y borrados, un árbol AVL parece más adecuado. Si no fuera así, y el número de consultas no fuese muy superior, merecería entonces la pena cambiar a un árbol roji-negro.

(4) Indicar que modificaciones habría que realizar, si hubiese que realizar alguna, sobre las clases *Pasiego* y *Zombie*, para poder almacenarlos en los TADs elegidos en el punto 2 y usando las implementaciones escogidas en el punto 3.

Dado que las llaves de los mapas *Pasiego* y *Zombie* son Naturales, habría que asegurarse de que dicho tipo de datos es dispersable, es decir, tiene una función de dispersión predefinada y son comparables por igualdad.

En el caso del mapa *pasiegosPorMote*, al ser la llave una CadenaCaracteres, habría que asegurarse de que dicho tipo es Comparable y que la función de orden que implementa es la deseada.

(5) Añadir a la clase Juego el procedimiento *addPasiego(p : Pasiego)* que sirva para dar de alta un nuevo pasiego. Calcular su complejidad.

```
// PRE: p != NULO
PROC addPasiego(p : Pasiego) ES
   mapaPasiegos.set(p.getId(),p); // O(1)
   pasiegosPorMote.set(p.getMote(),p); // O(log(n))
   Lista(Pasiego) listaPasiegosPorMote = pasiegosPorMote.getValuesOrderedPerKey(); // O(n)
   lista. displayList(listaPasiegosPorMote); // O(n)
FINPROC
```

Complejidad: O(n) siendo *n* el número de pasiegos vivos

(6) *addZombie(z : Zombie)* que sirvan para dar de alta un nuevo zombie en el juego. Calcular su complejidad.

```
// PRE: z != NULO
PROC addZombie(z : Zombie) ES
      mapaZombies.set(z.getId(),z); // O(1)
FINPROC
```

Complejidad: O(1)

(7) Implementar la operación *onPixelClicked* añadiendo para ello las operaciones que se consideren necesarias en la clase *Juego*.

```
CLASE GameZone ES
        …
// Pre: dimensionesCorrectas(fila, columna)
PROC onPixelClicked(fila, columna : NATURAL) ES // O(1)
      SI pantalla[fila, columna] != 0 ENTONCES     // O(1)
         controlJuego.serSeleccionado(pantalla[fila][columna]); // O(1)
      FINSI
FINPROC
FINCLASE

CLASE Juego ES

// PRE: mapaPasiegos.contains(id) OR mapaZombies.contains(id)
PROC serSeleccionado(id : Natural) ES // O(1)
      SI mapaPasiegos.contains(id) ENTONCES // O(1)
         pasiegoSeleccionado(id); // O(1)
      SINO
         zombieSeleccionado(id); // O(1)
      FINSI
FINPROC
```

```
// PRE: mapaPasiegos.contains(id)
PROC pasiegoSeleccionado(id : Natural) ES // O(1)
        Pasiego p = mapaPasiegos.get(id);     // O(1)
        pasiegoActivo := p;                   // O(1)
        pInfo.display(p);                     // O(1)
FINPROC

// PRE: mapaZombies.contains(id)
PROC zombieSeleccionado(id : Natural) ES // O(1)
        Zombie z = mapaZombies.get(id);   // O(1)
        zInfo.display(z);                 // O(1)
FINPROC

FINCLASE
```

(8) Implementar la operación *onPasiegoSelected* añadiendo para ello las operaciones que se consideren necesarias en la clase *Juego*.

```
CLASE PasiegoGraphicalList ES
…
PROC onPasiegoSelected(mote : CadenaCaracteres) ES
        controlJuego.pasiegoSeleccionado(mote);
FINPROC

FINCLASE

CLASE Juego ES

// PRE: pasiegosPorMote.contains(mote)
PROC pasiegoSeleccionado(mote : CadenaCaracteres) ES // O(1)
         pasiegoSeleccionado(pasiegosPorMote.get(mote).getId()); // O(1)
FINPROC

FINCLASE
```

(9) Calcular la complejidad de todas las operaciones creadas en los puntos (7) y (8).

Ver en las propias operaciones.

(10)Implementar la operación *lanzamientoSobao(idZombie: Natural, idPasiego : Natural)* en la clase *Juego*. Se puede asumir que ambos identificadores son correctos, pero nada más. Dicha operación debe ejecutar el lanzamiento de un sobao del pasiego que se pasa como parámetro al zombie que se pasa como parámetro.

```
CLASE Juego ES

CONSTANTE PenalizacionPorSobao ES NATURAL CON VALOR 25;

// PRE: mapaPasiegos.contains(idPasiego) AND mapaZombies.contains(idZombie)
PUBLICO PROC lanzamientoSobao(idZombie: Natural, idPasiego : Natural) ES
        Pasiego p = mapaPasiegos.get(id) ; // O(1)
        Zombie z  = mapaZombies.get(id);  // O(1)
        SI p.getSobaos() > 0 ENTONCES      //  O(1)
           p.setSobaos(p.getSobaos()-1);    //  O(1)
           vidaRemanente : ENTERO;
           vidaRemanente := (ENTERO) (z.getVida() – PenalizacionPorSobao); //  O(1)
           actualizaVidaZombie(z, p, vidaRemanente); // O(1)
       FINSI
FINPROC

HEREDABLE PROC actualizaVidaZombie(z : Zombie; killer:  Pasiego; vida : ENTERO) ES // O(1)
   SI vida > 0 ENTONCES //  O(1)
      z.setVida((NATURAL) vida);      //  O(1)
   SINO
      mapaZombies.remove(z.getId()); //  O(1)
      killer.getMuertos().add(z);   //  O(1)
      zone.removeZombie(z); //  O(1)
   FINSI
FINCLASE
```

(11) Calcular la complejidad de la operación del punto (10).

Ver apartado anterior

(12)Implementar la operación *lanzamientoBolo(idZombie: Natural, idPasiego : Natural)* en la clase *Juego*. Se puede asumir que ambos identificadores son correctos, pero nada más. Dicha operación debe ejecutar el lanzamiento de un sobao del pasiego que se pasa como parámetro al zombie que se pasa como parámetro.

```
CLASE Juego ES

CONSTANTE PenalizacionPorBolo ES NATURAL CON VALOR 50;

// PRE: mapaPasiegos.contains(idPasiego) AND mapaZombies.contains(idZombie)
PROC lanzamientoSobao(idZombie: Natural, idPasiego : Natural) ES
        Pasiego p = mapaPasiegos.get(id);  // O(1)
        Zombie z  = mapaZombies.get(id);  // O(1)
        vidaRemanente := (ENTERO) (z.getVida() – PenalizacionPorBolo); //  O(1)
        actualizaVidaZombie(z,p, vidaRemanente);
FINPROC
```

(13)Calcular la complejidad de la operación del punto (12).

Ver apartado anterior

(14)Diseñar una función de dispersión para la clase *Zombie*.

Lo más fácil es usar el identificador, usándolo tal como es.

(15)Diseñar una función de dispersión para la clase *Pasiego* que no involucre al identificador del mismo.

Dado que los nombres y apellidos son muy similares entre pasiegos, usarlos para una función de dispersión no es útil, puesto que favorecerá las colisiones. Por tanto, usaremos como valor de dispersión la suma de los valores numéricos correspondiente a los caracteres del mote de cada pasiego.

Pablo Sánchez Barreiro.