Departamento de TIC
Algoritmos y Estructuras de Datos
Unidad 2 Laboratorio

Nicolás Martinez, Cristian Molina y Juan Manuel Castillo.

Paso 1. Identificación del problema

Identificación de necesidades y síntomas

- Los jugadores presentan dificultades para encontrar una partida que posea jugadores con un nivel de juego similar.
- Los jugadores presentan dificultades al encontrar una partida debido a que estas son multiplataforma, lo que genera un retraso para los jugadores que poseen un ping muy alto.
- Los jugadores presentan preocupaciones al no contar con una celebración particular para una fecha especial (festividad).
- Necesita de un sistema de clasificación que permita a los usuarios ser rankeados, que no afecte el tiempo promedio de búsqueda de una nueva partida entre n usuarios con ranking similar.
- Necesita cerciorarse de que todas las personas en una partida tenga un latencia muy cercana. Es decir, que la desviación estándar del ping de los jugadores de una partida sea la mínima posible (para asegurar la igualdad de condiciones en la partida).
- Necesita implementación para el modo plataforma.
- Necesita implementación del modo de juego que será lanzado para San Valentín.

Definición del problema

El equipo de Epic Games requiere de una serie de funcionalidades adicionales, las cuales serán estudiadas para la futura implementación en una actualización, con base a las recomendaciones más comunes presentadas por los usuarios del juego multiplataforma Fortnite. Todas aquellas funcionalidades adicionales se han planteado con el objetivo de mejorar la experiencia de los usuarios en el juego y la eficiencia de la estructura en la que está diseñada el juego.

Requerimientos funcionales

Nombre	**R.F. # 1 Clasificar jugadores por experticia para una partida.**
Resumen	**Proponer un sistema de clasificación que permita a los usuarios ser rankeados según su experticia en el juego.**
Entrada	**Usuarios**
Salida	**Se han clasificado los jugadores por experticia al buscar una partida.**

Nombre	R.F. # 2 Clasificar jugadores por plataforma para una partida.
Resumen	Proponer un sistema de clasificación que permita a los usuarios ser clasificados según la plataforma en el juego.
Entrada	Usuarios
Salida	Se han clasificado los jugadores por plataforma al buscar una partida.

Nombre	R.F. # 3 Clasificar jugadores por latencia (ping) para una partida.
Resumen	Proponer un sistema de clasificación que permita a los usuarios ser clasificados según su latencia en el juego.
Entrada	Usuarios
Salida	Se han clasificado los jugadores por su latencia al buscar una partida.

Nombre	R.F. # 4 Clasificar jugadores por región para una partida.
Resumen	Proponer un sistema de clasificación que permita a los usuarios ser clasificados según su región en el juego para la igualdad de condiciones en una partida.
Entrada	Usuarios
Salida	Se han clasificado los jugadores por su región al buscar una partida.

Nombre	R.F. # 5 Establecer un modo de juego especial para Saint Valentines Day.
Resumen	Crear un nuevo modo de juego para Saint Valentines Day que tenga en cuenta el sistema de clasificación anteriormente mencionado.
Entrada	Usuarios
Salida	Se ha implementado un nuevo nuevo modo de juego para Saint Valentines Day.

Nombre	R.F. # 6 Establecer como única arma la última recogida en la partida (Para Saint Valentines Day mod)
Resumen	Cambiar de arma solo es posible encontrando una nueva o agotando las municiones del arma actual, lo cual hará desaparecer el arma actual y se cambiará con la última arma que se levantó antes que esa.
Entrada	Jugador
Salida	Se ha implementado como única arma la última recogida en la partida.

Nombre	R.F. # 7 Establecer como arma inicial el hacha al empezar la partida.(Para Saint Valentines Day mod)
Resumen	Todos los jugadores de Fornite por defecto salen con un Hacha al iniciar la partida, la cual es un arma de corto alcance sin municiones.
Entrada	Jugadores
Salida	Se ha implementado como arma inicial el hacha.

Requerimientos no funcionales

Nombre	R.N.F# 1 Implementar el sistema de clasificación de jugadores para una partida en el menor tiempo posible.
Resumen	**Con base al sistema implementado para la clasificación de jugadores al buscar una partida, se debe tener en cuenta el menor tiempo posible.**

Fase 2. Recopilación de la información necesaria

Definiciones

Fuente:

https://es.wikipedia.org/

https://definicion.de/

https://www.aprenderaprogramar.com/

https://es.slideshare.net/

http://www.it.uc3m.es/

http://lineadecodigo.com

http://www.forosdelweb.com/

Epic Games

Es una empresa de desarrollo de videojuegos estadounidense con sede en Cary, Carolina del Norte, ahora asociado con la compañía china Tencent Holdings.1 Ellos son bien conocidos por su tecnología Unreal Engine, que ha impulsado su popular serie de shooters en primera persona Unreal y la saga Gears of War, así como el mundialmente famoso Fortnite. (Fuente: Wikipedia)

Fortnite: Battle Royale

Es un modo de juego derivado del Fornite el cual cuenta con hasta 100 jugadores, solo, en dúos o en escuadrones de hasta cuatro jugadores, intentando ser el último jugador con vida matando a otros jugadores o evadiéndolos, al tiempo que se mantiene dentro de una zona segura que se encoge constantemente para evitar recibir daños letales por estar fuera de ella. Los jugadores deben buscar armas para ganar ventaja sobre sus oponentes. El juego

además cuenta como juego multiplataforma limitado entre PlayStation 4, Xbox One, versiones para ordenador y versiones para móviles. (Fuente: Wikipedia)

Ping

A partir del comando ping se envían series de información a otro equipo remoto y se aguarda la respuesta. En breve, ping muestra las estadísticas de su trabajo, indicando qué cantidad de paquetes fue respondida y cuánto demoró la respuesta. Ping, por lo tanto, brinda la posibilidad de conocer la condición, la velocidad y la calidad de una red. (Fuente: Definición.de)

Desviación Estándar

La desviación típica es una medida del grado de dispersión de los datos con respecto al valor promedio. Dicho de otra manera, la desviación estándar es simplemente el "promedio" o variación esperada con respecto a la media aritmética. (Fuente: Wikipedia)

Stack

La clase Stack es una clase de las llamadas de tipo LIFO (Last In - First Out, o último en entrar - primero en salir). Esta clase hereda de la clase que ya hemos estudiado anteriormente en el curso Vector y con 5 operaciones permite tratar un vector a modo de pila o stack.Las operaciones básicas son push (que introduce un elemento en la pila), pop (que saca un elemento de la pila), peek (consulta el primer elemento de la cima de la pila), empty (que comprueba si la pila está vacía) y search (que busca un determinado elemento dentro de la pila y devuelve su posición dentro de ella). (Fuente: Aprende a Programar)

Queue

Una cola (también llamada fila) es otro TDA, es una estructura de datos, caracterizada por ser una secuencia de elementos en la que la operación de inserción push se realiza por un extremo y la operación de extracción pop por el otro. También se le llama estructura FIFO (del inglés First In First Out), debido a que el primer elemento en entrar será también el primero en salir. (Fuente: SlideShare)

LinkedList

Una lista enlazada simple es una estructura de datos en la que cada elemento apunta al siguiente. La lista enlazada se compone de nodos (objetos instanciados pertenecientes a la clase Node), cada uno de los cuales tiene dos únicas tareas: guardar la información de la posición i y ofrecer una referencia a la posición i+1. (Fuente: uc3m)

HashTable

Una Hashtable Java es una estructura de datos que utiliza una función hash para identificar datos mediante una llave o clave (ej. Nombre de una persona). La función hash transforma una llave a un valor índice de un arreglo de elementos. (Fuente lineadecodigo)

TAD

Un tipo de dato abstracto (TDA) o Tipo abstracto de datos (TAD) es un modelo matemático compuesto por una colección de operaciones definidas sobre un conjunto de datos para el modelo.
Algunos ejemplos de utilización de TDA´s en programación son:

Conjuntos: Implementación de conjuntos con sus operaciones básicas (unión, intersección y diferencia), operaciones de inserción, borrado, búsqueda...
Árboles Binarios de Búsqueda: Implementación de árboles de elementos, utilizados para la representación interna de datos complejos. Aunque siempre se los toma como un TDA separado son parte de la familia de los grafos.
Pilas y Colas: Implementación de los algoritmos FIFO y LIFO.
Grafos: Implementación de grafos; una serie de vértices unidos mediante una serie de arcos o aristas. (Fuente: forosdelweb)

PSP

El proceso personal de software, PSP, es un conjunto de prácticas disciplinadas para la gestión del tiempo y mejora de la productividad personal de los programadores o ingenieros de software, en tareas de desarrollo y mantenimiento de sistemas, mediante el seguimiento del desempeño predicho frente al desempeño real. Está alineado y diseñado para emplearse en organizaciones con modelos de procesos CMMI o ISO 15504. (Fuente: Wikipedia)

Fase 3. Búsqueda de soluciones creativas.

El grupo ha acordado entender el entorno donde hay jugadores como un gran servidor global, que contiene a su vez servidores (En nuestro caso 4: América, Europa, África y Asia) por áreas geográficas diferentes. Las formas de almacenar todos los jugadores pueden ser:

- Utilizando una lista genérica
- Utilizando una cola por Lista enlazada.
- Utilizando una cola por referencias.x
- Utilizando una pila.x
- Utilizando una cola de prioridad.x

Ideas para modelar el ranking:

El ranking es un conjunto de elementos tales que, para uno o varios criterios, el primero de ellos presenta un valor superior al segundo, este a su vez mayor que el tercero y así sucesivamente. por lo cual se podría modelar como:

- ❖ Cola por lista enlazada.
- ❖ Cola por referencia.
- ❖ Cola de prioridad.x

Ideas para modelar la creación de partida:

A la hora de buscar una partida, los usuarios presentan problemas para encontrar contrincantes que tengan un nivel similar de juego, o que su latencia no varíe mucho para que su experiencia en el juego sea más estable, por lo tanto esto se puede modelar como:

- ❏ Utilizando una lista ordenada según criterios requeridos.
- ❏ Haciendo uso de una cola de prioridad.x

Ideas para modelar el modo plataforma:

Para innovar, el equipo de Fornite quiere establecer partidas específicas en la cuales puedan participar usuarios de una sola plataforma y que se pueda limitar el número de personas en la partida, esto se puede modelar como:

- ➔ *Una lista conformada por jugadores que cumplan las restricciones.*
- ➔ *Una cola conformada por jugadores que cumplan las restricciones.*
- ➔ *Una pila conformada por jugadores que cumplan las restricciones.x*

Ideas para modelar el modo San Valentín:

Para que el nuevo modo san valentín sea un éxito se debe cumplir el objetivo de su funcionalidad, por lo tanto el equipo ha acordado que las formas de implementar este modo son:

- ➢ Una pila de armas y un hashtable que guarde la información de ellas.
- ➢ Una cola de armas y una lista que guarde las posibles armas que puede encontrar en el camino.x

Fase 4. Transición de la formulación de ideas a los diseños preliminares

Ideas descartadas del servidor global:

- Utilizando una cola por referencias:
- Utilizando una pila:
- Utilizando una cola de prioridad.:

Las anteriores ideas han sido descartadas por la razón de que un servidor global cuenta con un conjunto de servidores (En nuestro caso 4: America, Europa, Africa y Asia) que no necesariamente tenga que ser implementado por tales estructuras, una lista podría almacenarlos prácticamente.

ideas descartadas para el ranking:

- ❖ Cola de prioridad: Se descarta ya que si se ordenan los datos con las restricciones específicas no es necesario usar colas de prioridad, una cola normal podría perfectamente representar el ranking.

Ideas descartadas para la creación de partida:

- ❏ Haciendo uso de una cola de prioridad: Ya que se va a hacer uso del ordenamiento, una cola podría representar mayor dificultad a la hora de la creación de partida.

Ideas descartadas para el modo plataforma:

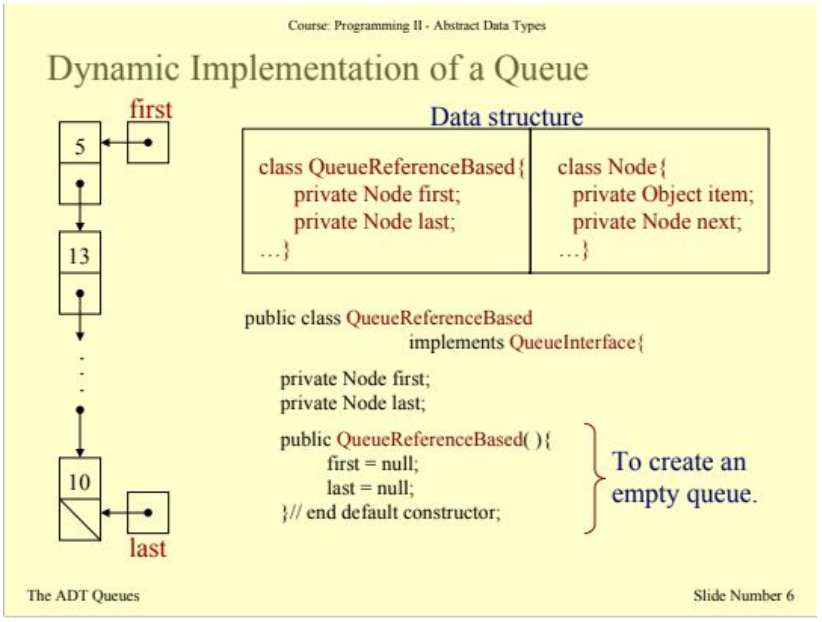
➔ *Una pila conformada por jugadores que cumplan las restricciones: Al utilizar una pila no podría disponer de la modificación de los jugadores en la partida, solo podría saber el primero de ellos.*

Ideas descartadas para el modo San Valentín:

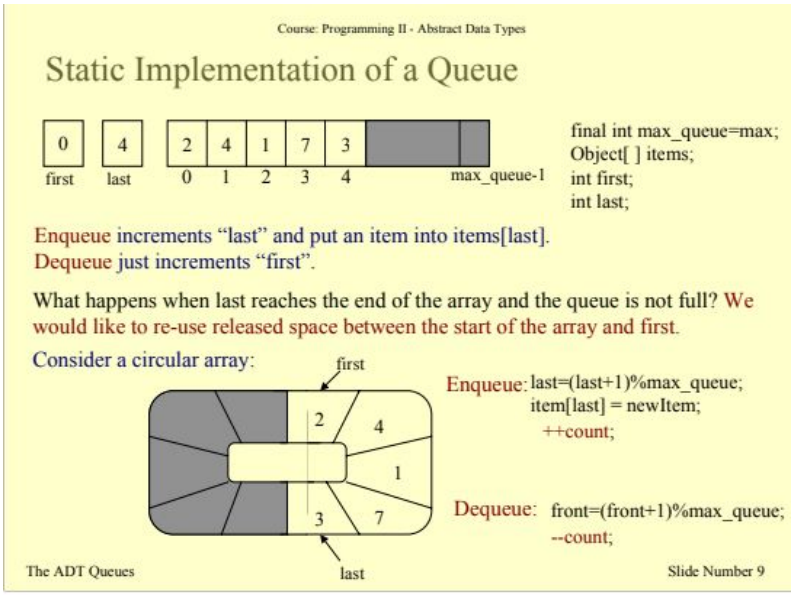
➢ Una cola de armas y una lista que guarde las posibles armas que puede encontrar en el camino: se descarta ya que según el contexto del problema no se asemeja mucho a las necesidades descritas por el usuario, quizás si se crean muchas armas diferentes sería más eficiente archivarlas en un diccionario de datos. Y la necesidad de usar siempre la última arma encontrada puede ser representada por una pila.

DISEÑOS PRELIMINARES (Recopilación de fuentes para crear nuestras propias estructuras)

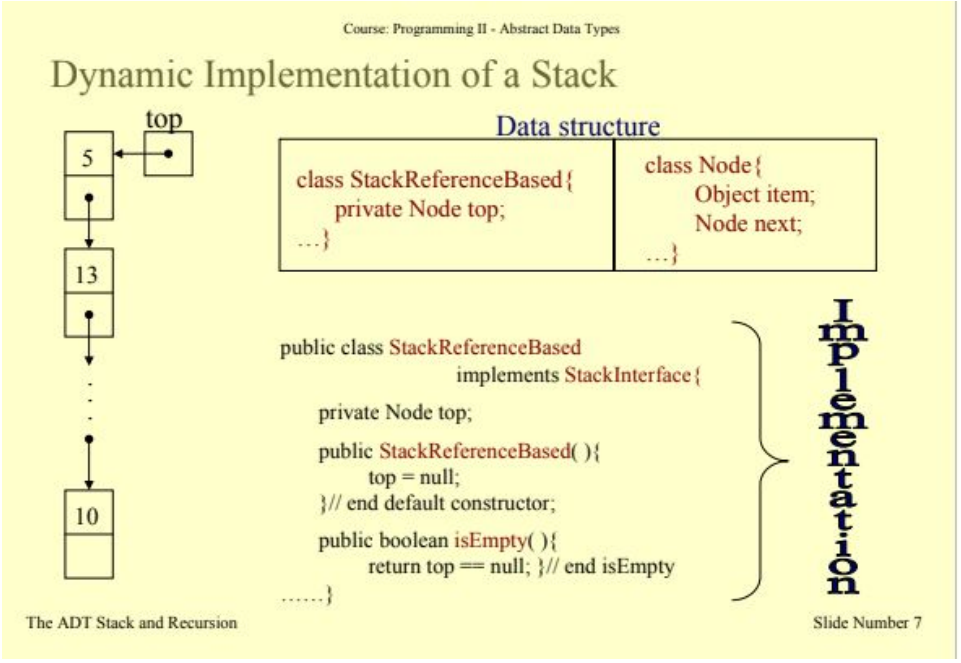
Cola basada en lista enlazada:



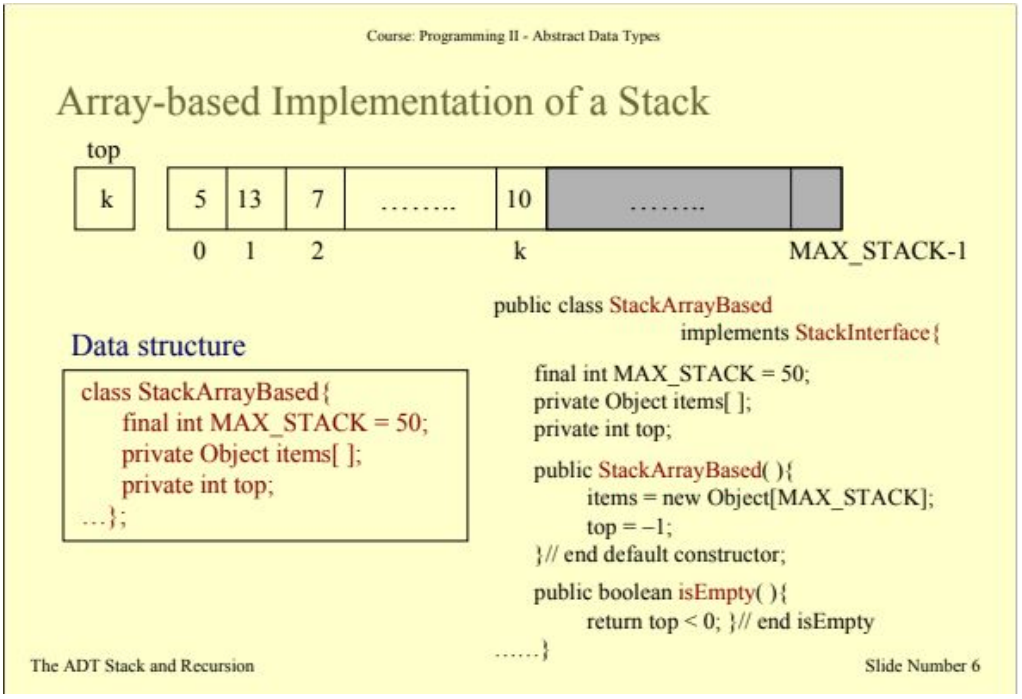
Cola basada en referencias estáticas:



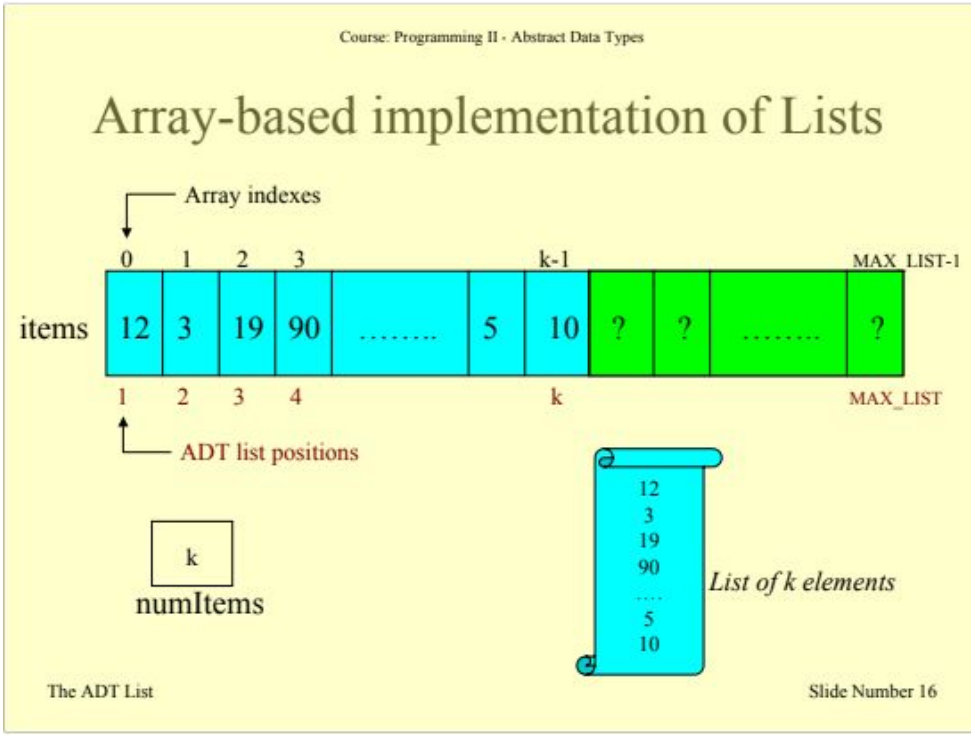
Pila basada en lista enlazada:



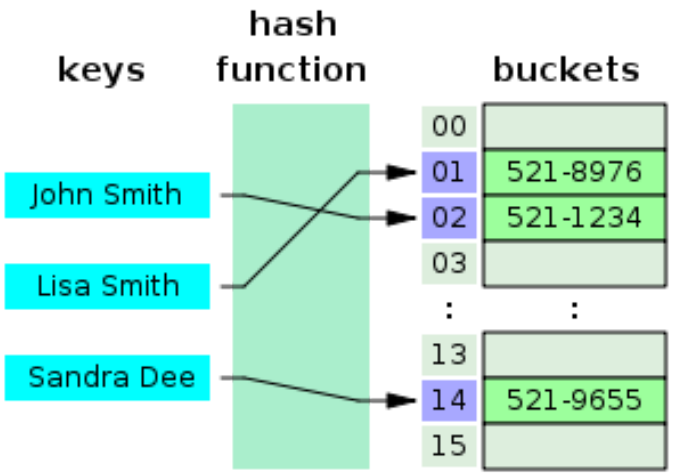
Pila basada en referencias estáticas:



Lista basada en referencias estáticas:



Hashtable basada en referencias estáticas:



Fase 5. Evaluación y selección de la mejor solución.

Mejor solución para el Servidor global:

Al evaluar las opciones que quedaban, la que se tomó como la mejor opción para solucionar el problema del Servidor global fue la lista genérica, ya que el único objetivo era almacenar a todos los jugadores y la solución más simple fue una lista genérica.

Mejor solución para el Ranking:

La solución tomada para resolver este problema fue la de una cola por lista enlazada, puesto que en el Ranking solo sirve tenerlo ordenado, no se necesita referencias para acceder a un elemento que no sea el consecutivo de donde se encuentra el apuntador.

Mejor solución para crear una partida:

Se seleccionó la lista ordenada como mejor solución debido a las condiciones impuestas por los usuarios. Los criterios para la creación de partidas implican tener un orden, ya que hay distintos criterios, la lista es una herramienta dinámica que se puede acoplar muy bien a las necesidades.

Mejor solución para el modo plataforma:

El método seleccionado para resolver el problema fue la lista. Dado que el momento de ingreso del usuario no es significativo a la hora de implementar el modo plataforma, una cola complicaría las cosas, mientras una lista sería muy útil ya que la agrupación de elementos con una característica en común se hace mucho más fácil.

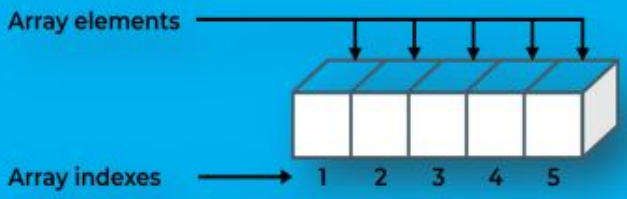
Mejor solución para el modo San Valentín:

La pila de armas fue la herramienta que mejor se acomodó a las necesidades. Era necesario obtener el último elemento en entrar al almacenamiento. Para tener la información de todas las armas y sus municiones se usó una HashTable con el nombre del arma como clave y su munición como valor.

Fase 6. Preparación de informes y especificaciones. Diseño de diagrama de clases.

Diseño de TADs

TAD FortniteList



{inv: first = null | ∀ next ≠ null}

- Constructor: → ForniteList(Constructor)
- add: FortniteList x Elem → Elem(Modificador)
- remove: FortniteList x Entero → ForniteList(Modificador)
- get: ForniteList x Entero → Elem(Analizador)
- size: FortniteList → Entero (Analizador

add(T o)

“Adiciona un elemento a la lista T”
p: Hay posición disponible.
q: Hay elemento a adicionar.
v: El elemento puede ser adicionado.
{pre:((p∧ q) v)}
p: El elemento se adicionó.
q: La posición del índice se ocupó.
{post: (pq)}

remove(int i)

“Elimina un elemento a la lista T”
p: Existe el índice recibido.
q: Hay elemento a eliminar.
v: El elemento puede ser eliminado.
{pre:((p∧ q) v)}
p: El elemento se eliminó.
q: La posición del índice está disponible.
{post: (pq)}

get(int i)

“Obtiene un elemento de la lista”
p: El array esta vacío.
q: Existe el índice buscado.
v: El elemento puede ser obtenido.
{pre:((¬p∧ q) v)}
p: El elemento se eliminó.
q: La posición del índice está disponible.
{post: (pq)}

size()

“Obtiene el tamaño de la lista”
p: La array existe
q: La array está declarada
v: El tamaño de la lista puede ser obtenida.
{pre:((p∧ q) v)}
p: El tamaño de la lista se obtuvo
q: La lista existe
{post: (pq)}

TAD ForniteHashTable

{inv: T(a_1, a_2, a_3 ,.., a_n) | ∀ data length > 0 }

- hash: → ForniteHashTable(Constructor)
- put: FortniteHashTable x key → Elem(Modificador)
- get: FortniteHashTable x key → ForniteHashTable(Modificador)
- remove: FortniteHashTable x key → Elem(Analizador)
- size: FortniteHashTable → Entero (Analizador
- isEmpty: FortniteHashTable → boolean (Analizador

hash(K key)

"Devuelve el valor del código hash para este mapa según la definición en la interfaz del mapa."
{pre: Se reciben parámetros en términos de <K, V>}
{post: Con base a la entrada <K, V> se retorna el código hash.}

put(K key, E ítem)

"Asigna la clave especificada al valor especificado en la tabla hash."
{pre: Se reciben parámetros en términos de <K, V> , E ítem}
{post: Con base a la entrada <K, V> se ha asignado una clave a un valor especifico.}

remove(K key)

“Elimina la clave (y su valor correspondiente) de la tabla hash.”

{pre: Se reciben parámetros en términos de <K, V>}

{post:Con base a la entrada <K, V> se ha eliminado la clave y valor específico de la tabla hash.}

get(*K key)*

“Devuelve el valor al que se asigna la clave especificada, o nulo si la la tabla hash no contiene ninguna asignación para la clave.”
{pre: Se reciben parámetros en términos de <K, V>}
{post: Con base a la entrada <K, V> se ha devuelto el valor el valor asignado a una clave especifica ∀ elem ≠ null.}

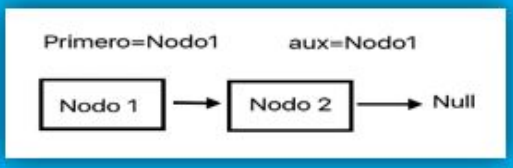
size()

“Devuelve el número de claves en la tabla hash.”
{pre: Debe existir la tabla hash para la implementación del método.}
{post: Se retorna un entero con el tamaño de la tabla hash.}

isEmpty()

“Comprueba si la tabla hash no asigna claves a los valores”
{pre: Debe existir la tabla hash para la implementación del método.}
{post: Se retorna un booleano si la tabla hash ha/ no ha asignado claves a los valores.}

TAD ForniteQueue



{inv: next ≠ null ∀ next - last}

- Constructor: → ForniteQueue(Constructor)
- enqueue: FortniteQueue x Elem → Elem(Modificador)
- dequeue: FortniteQueue → ForniteQueue(Modificador)
- front: ForniteQueue → Elem(Analizador)
- size: FortniteQueue → Entero (Analizador

enqueue(*Object item*)

"Inserta el elemento especificado en la cola si es posible hacerlo inmediatamente sin violar las restricciones de capacidad."
{pre: Se reciben parámetros en términos de Object item}
{post: Se ha insertado un elemento a la cola dependiendo de las condiciones en que se de su inserción.}

dequeue()

"Recupera y quita el elemento a la cabeza de la cola."

{pre: Debe existir la cola para la implementación del método.}

{post: Se ha retirado el cabezal de la cola para luego ser tomado.}

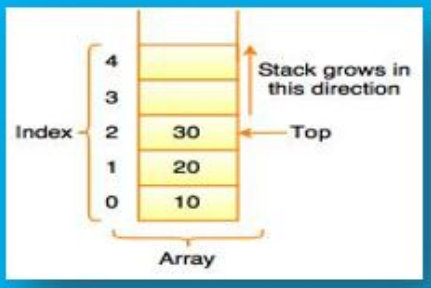
front()

“Recupera, pero no elimina, el elemento a la cabeza de la cola, o devuelve nulo si la cola está vacía.”
{pre: Debe existir la cola para la implementación del método.}
{post: Se ha tomado el cabezal de la cola ∀ elem ≠ null.}

size()

“Devuelve el número de ítems en la cola.”
{pre: Debe existir la cola para la implementación del método.}
{post: Se retorna un entero con el tamaño de la cola.}

TAD FortniteStack



{inv: next ≠ null ∀ next - last}

- Constructor: → ForniteStack(Constructor)
- push: FortniteStack x Elem → Elem(Modificador)
- pop: FortniteStack → ForniteStack(Modificador)
- isEmpty: ForniteStack → boolean(Analizador)
- top: FortniteStack → Entero (Analizador

push(*T o*)

“Empuja un elemento en la parte superior de la pila.
{pre: Se reciben parámetros en términos de T o}
{post: Con base a la entrada T o se ha empujado un elemento al top de la pila.}

pop()

“Elimina el objeto en la parte superior de la pila y devuelve ese objeto como el valor de la función.”

{pre: Debe existir la pila para la implementación del método.}

{post: Se ha eliminado el objeto del top de la pila para ser obtenido.}

isEmpty()

“Prueba si la pila está vacía.”
{pre: Debe existir la pila para la implementación del método.}
{post: Se retorna un booleano si la pila se encuentra o no, vacia.}

top()

“Entrega el objeto en la parte superior de la pila sin quitarlo de esta.”
{pre: Debe existir la cola para la implementación del método.}
{post: Se ha obtenido el elemento en el top de la pila.}

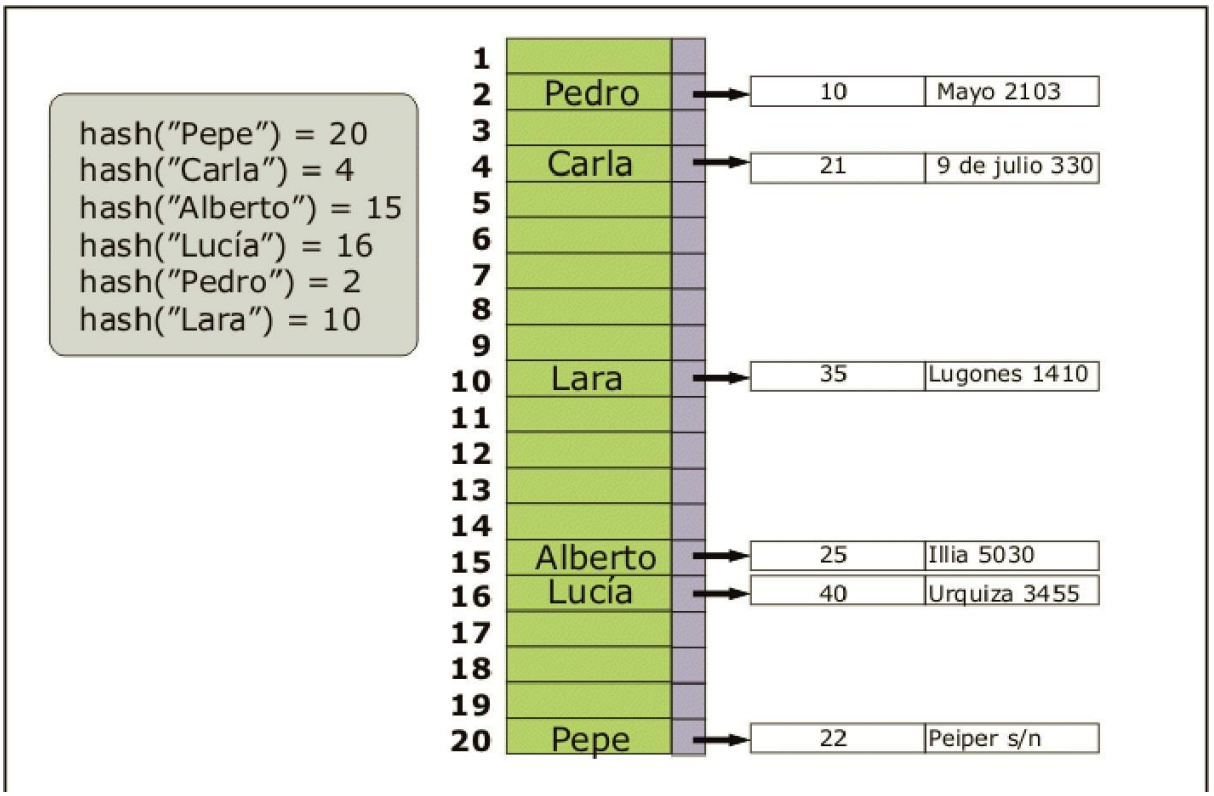
Diseño de pruebas

FortniteHashtableTest

Scene1(): *Una tabla hash de tamaño 40, iniciando desde 0 asignando la clave con base al índice y el valor multiplicado 2 veces por el índice.*
Scene2(): *Una tabla hash de tamaño 40 iniciando desde 1 asignando la clave con base al índice multiplicado por 7 y el valor multiplicado 2 veces por el índice.*
Scene3(): *Una tabla hash vacía.*

Método	Escenario	Entrada	Salida
getTest()	*Scene1()* *Scene2()* *Scene3()*	Ninguna.	Se prueba que la tabla hash si devuelve el valor al que se asigna la clave especificada, o nulo si esta no contiene ninguna asignación para la clave.
putTest()	*Scene1()* *Scene2()* *Scene3()*	Ninguna	Se prueba que la tabla hash asigne la clave especificada al valor especificado.
removeTest()	*Scene1()* *Scene2()*	Ninguna	Se prueba que la tabla hash elimina la clave (y su valor correspondiente).
isEmptyTest()	*Scene1()* *Scene2()* *Scene3()*	Ninguna	Se prueba que la tabla hash comprueba si esta no asigna claves a los valores.

Representación gráfica Caso 1 Caso 2 - Método put():



Fuente: *https://www.alipso.com/monografias4/hashing/*

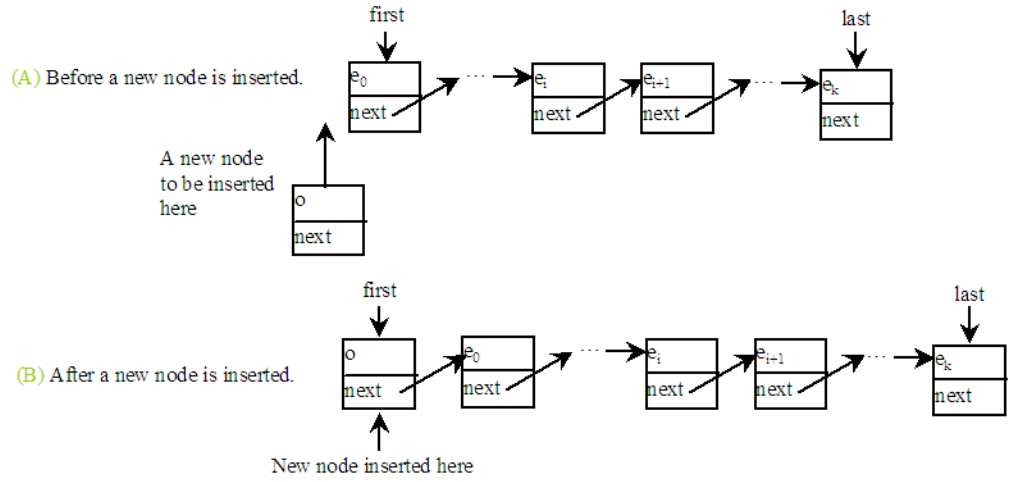
FortniteListTest

Scene1(): *Una lista de tamaño 40 iniciando en 0 con elementos multiplicados 2 veces por el valor del índice.*
Scene2():*Una lista de tamaño 15 iniciando en 0 agregando elementos generados aleatoriamente hasta el 15 sin incluir el 15 y el 0.*
Scene3(): *Una lista vacía.*

Método	Escenario	Entrada	Salida
addTest()	*Scene1()* *Scene3()*	Ninguna.	Se prueba que la lista agregue el elemento especificado al final de esta.
removeTest()	*Scene1()* *Scene2()* *Scene3()*	Ninguna	Se prueba que la lista elimine un elemento

getTest()	*Scene1()* *Scene2()* *Scene3()*	Ninguna	Se prueba que la lista obtiene un elemento de ésta, en caso de ser nula; una excepción.
sizeTest()	*Scene1()* *Scene3()*	Ninguna	Se prueba que la lista obtiene el tamaño de ésta.

Representación gráfica Caso 1 Caso 2 - Método add():



Fuente: *http://cim.mcgill.ca/~gamboa/cs202/Material/class18/*

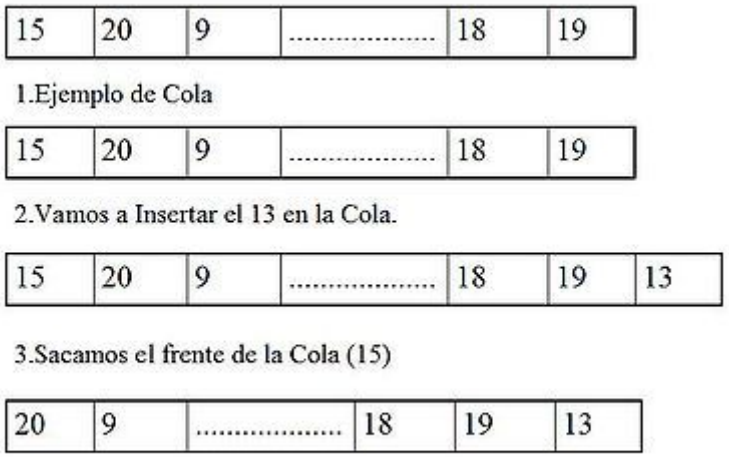
FortniteQueueTest

Scene1(): *Una cola de tamaño 40 iniciando en 0 agregando los valores con base al valor del índice.*
Scene2(): *Una cola de tamaño 40 iniciando en 0 agregando valores aleatorios hasta el 15 sin incluir el 15 y el 0.*
Scene3(): *Una cola vacia.*

Método	Una cola de Escenario	Entrada	Salida

isEmptyTest()	*Scene1()* *Scene3()*	Ninguna.	Prueba que la cola devuelve verdadero si no contiene elementos/ falso si contiene.
frontTest()	*Scene1()* *Scene3()*	Ninguna	Prueba que la cola recupera, pero no elimina, el elemento a la cabeza de la cola, o devuelve nulo si la cola está vacía
dequeueTest()	*Scene1()* *Scene3()*	Ninguna	Prueba que la cola recupere y quite el elemento a la cabeza de esta.
enqueueTest()	*Scene1()* *Scene2()* *Scene3()*	Ninguna	Prueba que la cola inserte el elemento especificado si es posible hacerlo inmediatamente sin violar las restricciones de capacidad

Representación gráfica Caso 1 Caso 2 - Método enqueue():



Fuente: *https://karenalduncin.wordpress.com/2011/07/08/pilas-y-colas/*

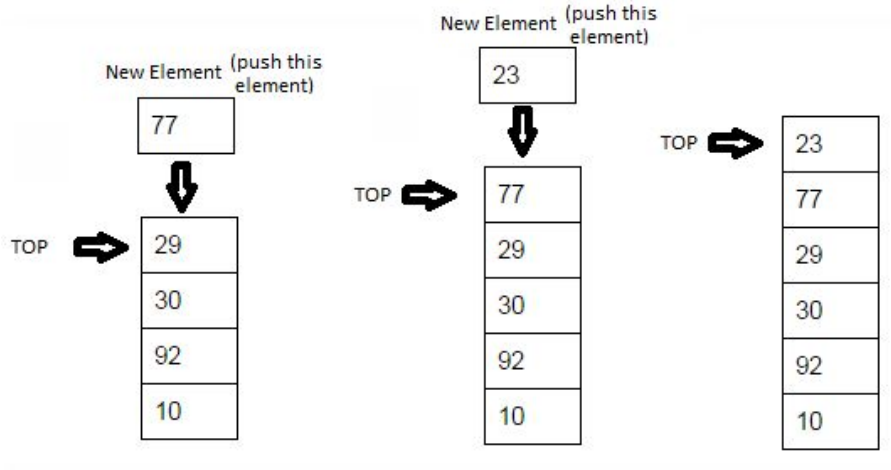
FortniteStackTest

Scene1(): *Una pila de tamaño 40 que inicia en 0 y que agrega elementos con base al valor del índice.*
Scene2(): *Una pila de tamaño 15 que inicia en 0, agregando elementos aleatoriamente hasta el 15 sin incluir el 15 y el 0.*
Scene3(): *Una pila vacía.*

Método	Escenario	Entrada	Salida
pushTest()	*Scene1()* *Scene2()* *Scene3()*	Ninguna.	Prueba que la pila empuje un elemento en la parte superior de esta pila.
popTest()	*Scene1()* *Scene2()* *Scene3()*	Ninguna	Prueba que la pila elimine el objeto en la parte superior de esta y devuelve ese objeto como el valor.

topTest()	*Scene1()* *Scene2()* *Scene3()*	Ninguna	Comprueba que la pila obtiene el objeto en la parte superior de esta sin retirarlo.
isEmptyTest()	*Scene1()* *Scene3()*	Ninguna	Prueba que la pila de un boolean verdadero si y solo si no tiene componentes, es decir, su tamaño es cero; falso de lo contrario.
sizeTest()	*Scene1()* *Scene3()*	Ninguna	Prueba que la pila devuelve un entero con la cantidad de elementos que posee.

Representación gráfica Caso 1 Caso 2 - Método psuh():



Fuente: *https://www.javamadesoeasy.com/2015/01/stacks.html*

Diagrama de pruebas unitarias:

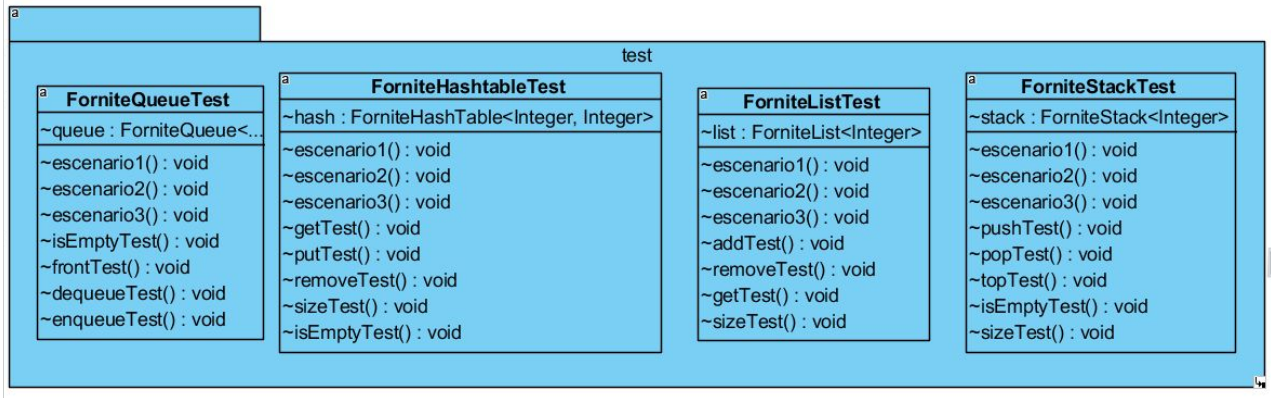


Diagrama del paquete Fornite.Util