

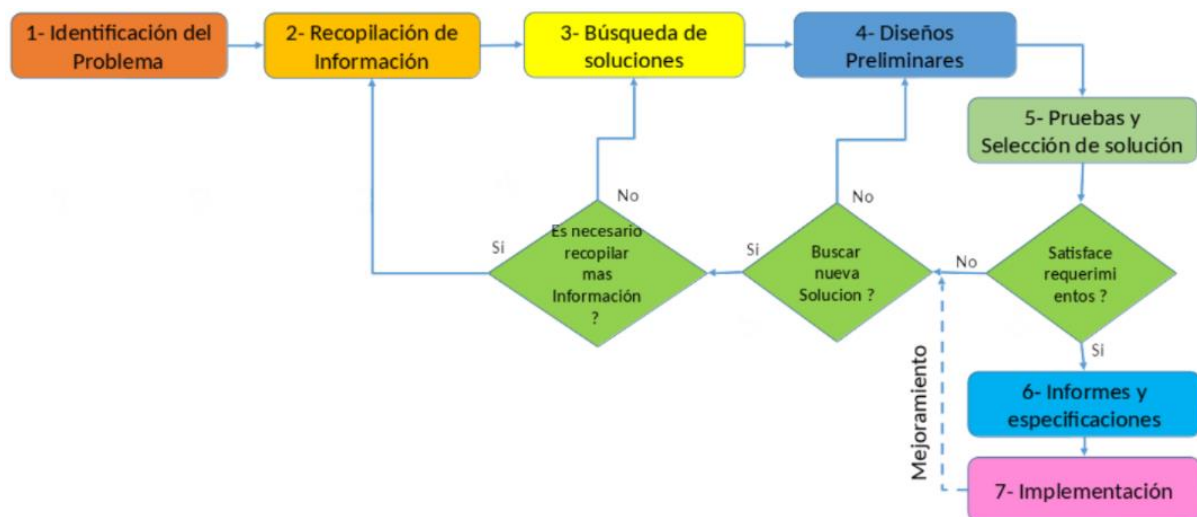
Contexto problemático

Un excéntrico millonario quiere abrir una tienda¹ que preste servicios de manera innovadora para la venta de videojuegos, donde se tiene un sistema de compra definido. Se requiere un sistema el cual simula el proceso que tendría que realizar cada cliente en la tienda, para así, solucionar el problema que se podría generar en los clientes, el cual es, identificar cómo funciona la tienda.

Desarrollo de la solución

Para sobrellevar el problema anteriormente mencionado se seleccionó el Método de la Ingeniería para desarrollar una solución, mediante un enfoque sistemático y acorde a la situación problemática planteada.

Para esto, se implementó la descripción del Método de la Ingeniería del libro “Introduction to Engineering” de Paul Wright, se define el siguiente diagrama de flujo, para seguir los pasos en el desarrollo de la solución.



Fase 1. Identificación del problema

Se reconocen de forma concreta las necesidades del cliente para llevar a cabo la solución del problema, reconociendo sus causas y síntomas.

Identificación de causas y síntomas

- Los clientes de la tienda requieren entender el proceso de compra de esta.
- No existe un programa de simulación que presente a los usuarios el funcionamiento de la tienda.
- La simulación debe ser clara y explícita.

¹ Tienda ficticia, inspirada del siguiente [enunciado](#)

- La simulación debe ser eficiente para que el servicio prestado por la tienda sea rápido y efectivo.

Definición del problema

La tienda de videojuegos requiere un software que simule el proceso de compra de la tienda, para que así, sus clientes puedan identificar la forma en que funciona el establecimiento.

Requerimientos funcionales

El software debe de estar en las capacidades de:

Req 1. Registrar la cantidad de casos de prueba que serán evaluados.

Req 2. Registrar la cantidad de cajeros disponibles en la tienda de videojuegos.

Req 3. Registrar la cantidad de estanterías disponibles en la tienda de videojuegos, cada una de ellas con un identificador y una cantidad determinada de juegos.

Req 4. Crear las estanterías de videojuegos, las cuales contienen los videojuegos que se encuentran disponibles a la venta.

Req 5. Agregar un nuevo juego. Se podrá agregar un juego con un código, un precio, la estantería donde se encuentra y la cantidad de unidades disponibles.

Req 6. Registrar el número de clientes que ingresan en una jornada.

Req 7. Agregar información de cada uno de los clientes, como lo es su código o cédula y los códigos de los juegos que adquirió.

Req 8. Crear lista que ordene los videojuegos que adquieren los clientes de manera que sea eficiente su recolección de los videojuegos de las estanterías.

Req 9. Mostrar información de los clientes en el orden de salida, en donde está su número de cédula, su valor total a pagar y los códigos de los videojuegos adquiridos.

Fase 2: Recopilación de información

Con el objetivo de abarcar el problema con una mayor claridad, se buscarán las definiciones relacionadas con el problema previamente definido.

Definiciones:

Fuente:

<https://es.wikipedia.org/>
<https://conceptodefinicion.de>
<https://definicion.de/>
<https://www.rae.es>

Simulación: La simulación es el proceso de diseñar un modelo de un sistema real y llevar a término experiencias con él, con la finalidad de comprender el comportamiento del sistema o evaluar nuevas estrategias -dentro de los límites impuestos por un cierto criterio o un conjunto de ellos- para el funcionamiento del sistema.

Estantería: Una estantería, estante, librería o librero es un mueble con tablas horizontales que sirve para almacenar libros, mapas cuentos y en general otro tipo de objetos.

Fila: Las filas suelen ser columnas de personas. Se trata de un método habitual de organización cuando tienen que establecerse turnos o cuando hay que lograr un orden para realizar algo.

Ordenamiento: Colocar algo o a alguien de acuerdo con un plan o de modo conveniente.

Tiempo: El tiempo es una magnitud física que hace posible ordenar la continuidad de los hechos, dando lugar a un presente, pasado y futuro.

Fase 3: Búsqueda de soluciones creativas:

Se realiza la búsqueda de posibles soluciones que se pueden dar para que se realice una simulación de forma rápida, clara e intuitiva.

Alternativa 1. Simulación por consola:

Es el método más sencillo de implementar, ya que sólo se requiere imprimir los datos obtenidos de la simulación y representar el proceso de esta. En la representación los objetos utilizados son variables como letras, números o caracteres.

```

Problems Javadoc Declaration Console x
Main (18) [Java Application] C:\Program Files\Java\jdk1.8.0_281\bin\javaw.exe (17/09/2021, 10:03:54 a. m.)
=====
BIENVENIDOS
=====

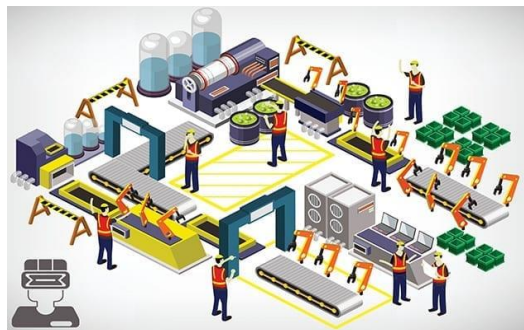
-----
MENU DE OPCIONES
-----

1. Iniciar juego
2. Puntuaciones
3. Salir de la aplicación
4. Creditos
-----

Ingrese un numero para elegir la opcion
  
```

Alternativa 2. Simulación por interfaz gráfica:

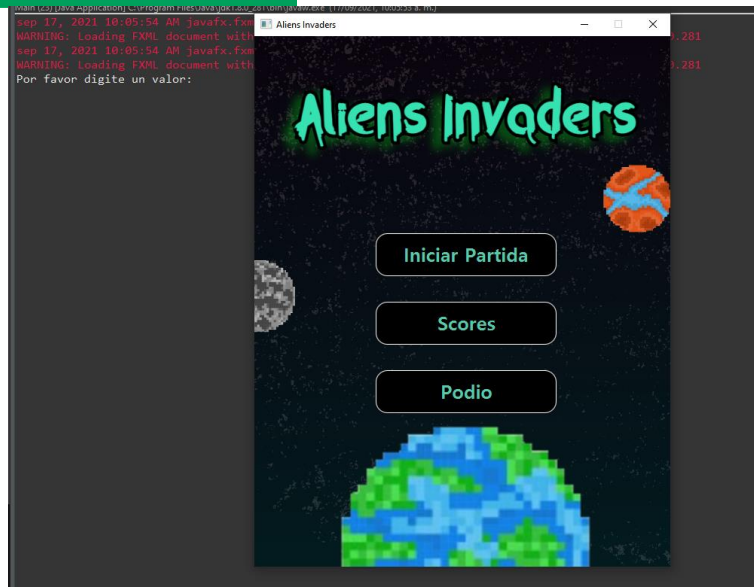
Este método supone una complejidad mayor a la de consola, esto debido a la necesidad de unos escenarios para la representación de los eventos dentro de la simulación.



Alternativa 3. Simulación por consola e interfaz gráfica:

Este método, es la unión de los dos métodos anteriores, permitiéndonos observar las características y atributos de cada uno. dando una perspectiva completa del objetivo del proyecto desde la parte gráfica como interactúan los diversos factores en el problema y de la simulación por consola lo cual sólo muestra el resultado a obtener.

² <https://bit.ly/3EshBZL>



Fase 4. Transición de las Ideas a los Diseños Preliminares.

La alternativa 1 (Simulación por consola) queda descartada al no cumplir con la necesidad de ser una simulación intuitiva y clara a la vista del cliente, ya que, como se dijo anteriormente, esta simulación sólo puede representar variables como letras y números.

Luego de descartar las alternativas no factibles, realizaremos una amplia revisión de las opciones restantes

Alternativa 2. Simulación por interfaz gráfica.

- La claridad de la interfaz depende de la habilidad de diseño de la persona o grupo que realiza la simulación.
- Este método supone una mayor carga para el sistema donde se lleva a cabo la simulación, debido a que se necesitan más recursos para una interfaz gráfica.
- Requiere de una mayor complejidad debido a la necesidad de unas ventanas previamente diseñadas.

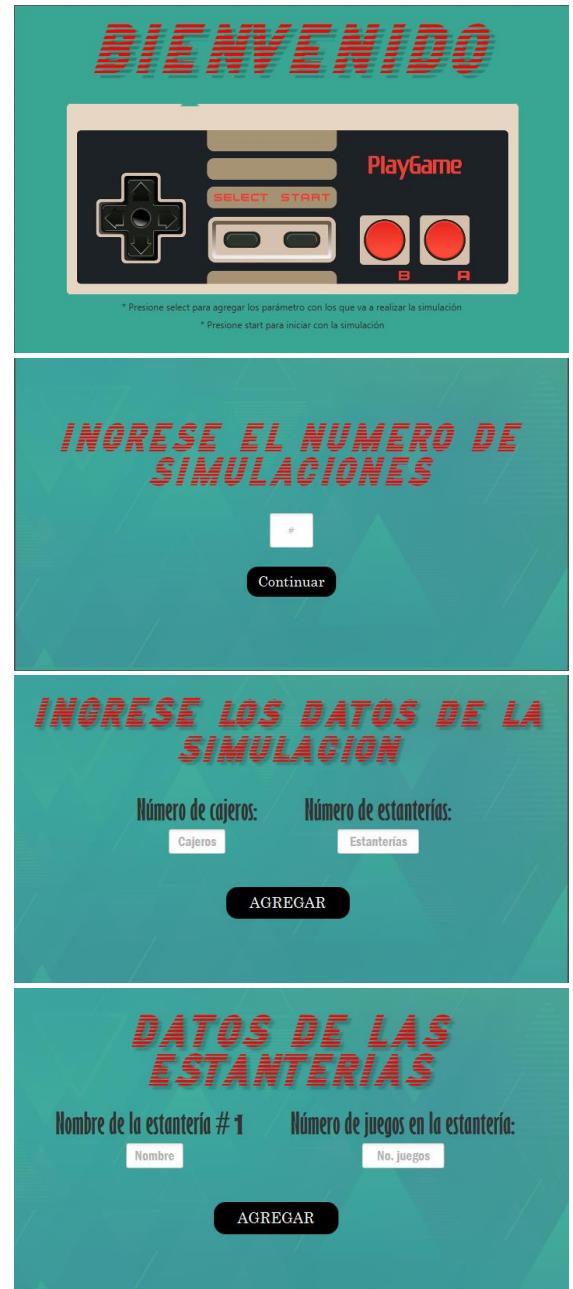
Mockups:

En este Mockup iniciamos la aplicación dándonos las funciones de inicio y personalización de la simulación. por medio de dos botones llamados Select y Start

Aquí se define la cantidad de repeticiones que va tener la simulación.

Aquí personalizamos los atributos de la simulación los cuales son la cantidad de cajeros y número de estantes disponibles en esa simulación.

Podemos asignar un nombre a cada una de las estanterías antes definidas y el número de juegos con el que contará.



A cada uno de los juegos se le asigna un nombre, un precio y las unidades disponibles, cada vez que se agrega un juego el label de código será modificado, por la suma de uno.



Aquí se ingresa la cantidad de clientes que van a ingresar a la tienda.



Aquí se ingresan los datos que tiene cada cliente, como lo es su cédula o código y los códigos de los juegos que comprará



Los clientes eligen el ordenamiento con el que se ordenará la lista de juegos que estos poseen.



Este hará alusión al ordenamiento de la lista de videojuegos.



Los clientes seleccionan de las estanterías los juegos que están presentes en la lista de juegos.



Los clientes seleccionan de las estanterías los juegos que están presentes en la lista de juegos. Muestra la cantidad de cajeros disponibles y la cantidad de clientes que no han sido atendidos, el número de clientes irá disminuyendo cada vez que sea atendido uno de ellos. El número de los cajeros irá aumentando dependiendo de si el cajero está ocupado.



Se muestra en una lista a todos los clientes en el orden de salida al momento de realizar todas las ventas.



Alternativa 3. Simulación por consola e interfaz gráfica.

- Esta alternativa puede hacer que se pierda el hilo de la simulación, ya que está se estará ejecutando en dos “ambientes” diferentes y esto puede generar que la simulación pierda claridad.
- La claridad de la interfaz depende de la habilidad de diseño de la persona o grupo que realiza la simulación.

En este Mockup iniciamos la aplicación dándonos las funciones de inicio y personalización de la simulación. por medio de dos botones llamados Select y Start



Aquí se define la cantidad de repeticiones que va tener la simulación.



Aquí personalizamos los atributos de la simulación los cuales son la cantidad de cajeros y número de estantes disponibles en esa simulación



Podemos asignar un nombre a cada una de las estanterías antes definidas y el número de juegos con el que contará.



DATOS DE LAS ESTANTERIAS

Nombre de la estantería # 1 Número de juegos en la estantería:

AGREGAR

A cada uno de los juegos se le asigna un nombre, un precio y las unidades disponibles, cada vez que se agrega un juego el label de código será modificado, por la suma de uno.




DATOS DE LOS JUEGOS

Código del juego # 1 Precio del juego: Unidades disponibles:

AGREGAR JUEGO

Aquí se ingresa la cantidad de clientes que van a ingresar a la tienda.



NUMERO DE CLIENTES

Digite el número de clientes que entrarán a la tienda

CONTINUAR

Aquí se ingresan los datos que tiene cada cliente, como lo es su cédula o código y los códigos de los juegos que comprará



DATOS DE LOS CLIENTES

Cédula o código del cliente # 1

Introduzca los códigos de juegos que adquiere el cliente, cada uno separado por ;

AGREGAR CLIENTE

Este hará alusión al ordenamiento de la lista de videojuegos.

Los clientes seleccionan de las estanterías los juegos que están presentes en la lista de juegos.

Muestra la cantidad de cajeros disponibles y la cantidad de clientes que no han sido atendidos, el número de clientes irá disminuyendo cada vez que sea atendido uno de ellos. El número de los cajeros irá aumentando dependiendo de si el cajero está ocupado.

Se muestran los clientes en el orden que van saliendo de cada uno de los cajeros. Cada cliente con su número de cédula, su total a pagar y los códigos de los juegos que adquirió. Dos líneas representan a un cliente.



```
Cientes en el orden de salida
50141 18560
451 487
95545 86000
451 213
|
```

Fase 5. Evaluación y Selección de la Mejor Solución

Criterios

Se van a definir criterios para así evaluar las alternativas de solución al problema. Cada uno de los criterios nos permitirá definir cuáles son las características fundamentales que debe tener la solución. Cada característica representada en los criterios tendrá un valor que significa qué tanto peso tiene esta en la solución del problema.

-Criterio A. Claridad de la simulación. La alternativa entrega una solución:

- [3] Completa y clara.
- [2] Completa y fragmentada.
- [1] Sin final.

-Criterio B. Eficiencia. Se prefiere una solución con mejor eficiencia que las demás. La eficiencia puede ser:

- [4] Constante.
- [3] Logarítmica.
- [2] Lineal.
- [1] Cuadrática.

-Criterio C. Dinámica. Se refiere a una solución sea fácil y agradable de comprender, como puede ser una solución:

- [2] Intuitiva e interactiva.
- [1] Intuitiva.

-Criterio D. Óptimo uso de recursos para el funcionamiento de la solución:

- [2] Compatible para cualquier equipo de cómputo con bajos recursos.
- [1] Compatible para cualquier equipo de cómputo con recursos medios.

Evaluación

Evaluando los criterios anteriores de las dos alternativas definidas, se obtiene la siguiente tabla:

	Criterio A	Criterio B	Criterio C	Criterio D	Total
Alternativa 2. Simulación por interfaz gráfica	Completa y clara. 3	Cuadrática 1	Intuitiva e interactiva. 2	Compatible con bajos recursos. 2	8
Alternativa 3. Simulación por consola e interfaz gráfica	Completa y fragmentada 2	Cuadrática 1	Intuitiva e interactiva 2	Compatible con bajos recursos. 2	7

Selección

A partir de los resultados obtenidos en la evaluación, podemos concluir que la alternativa 2 (Simulación por interfaz gráfica) es la mejor solución, ya que obtuvo el mayor resultado, por ende, es la solución seleccionada.

Fase 6. Preparación de informes y especificaciones

Especificación del problema: (en términos de entrada y salidas)

Problema: Simular el proceso de compra de la tienda de videojuegos.

Entradas: Número de casos de prueba a realizar, número de cajeros en la tienda, número de estanterías, nombre de las estanterías, número de juegos en cada estantería, código de cada juego, precio de cada juego, número de unidades disponibles de cada juego, número de clientes que ingresan a la tienda por día, cédula o código de cada cliente, códigos de juegos que va a comprar cada cliente.

Salidas: Lista de clientes en orden de salida, en donde cada cliente tiene una cédula, precio total a pagar y código de cada videojuego comprado.

Consideraciones:

Se deben tener en cuenta los siguientes casos para poder realizar la simulación del proceso de compra de la tienda.

1. El número de casos de prueba a examinar tiene que ser mayor o igual a 1.
2. El número de cajeros y estanterías debe ser mayor o igual a 1.
3. El número de videojuegos que contiene cada estantería debe ser mayor o igual a cero.

4. El número de clientes que entran a la tienda debe ser mayor o igual a uno.

Fase 7. Implementación del diseño.

Diseño de las tablas de tipos abstractos de datos:

TAD De Cliente			
Cliente = {Cédula = ⟨cedula⟩, Lista de códigos = ⟨códigos⟩}			
{ inv: cliente.Código ≠ nulo }			
Operaciones primitivas			
CrearCliente:	Texto x Lista Enteros		Constructora
BuscarCliente	Texto	→ Cliente	Analizadora
ObtenerCodigos		→ Lista entero	Analizadora
ObtenerCedula		→ Texto	Analizadora

CrearCliente(código, n)

“Se ingresa un cliente nuevo que va a realizar una compra”

{pre: código ≠ vacío, n es una lista ∧ los componentes de n ∈ Entero +}

{post: Si buscarCliente(código) ≠ nulo, entonces TRUE, de lo contrario FALSE}

BuscarCliente(t)

“Se encarga de buscar a un cliente a partir de su cédula”

{pre: cliente = (cedula : ⟨t⟩, códigos : ⟨códigos⟩) ∧ t ≠ vacío}

{post: cliente = (cedula : ⟨t⟩, códigos : ⟨códigos⟩)}

ObtenerCodigos()

“Muestra el código de un cliente”

{pre: TRUE}

{post: ⟨código⟩}

ObtenerCedula()

“Se encarga de obtener la cédula de un cliente”

{pre: cliente = (cedula : ⟨cédula⟩, códigos : ⟨códigos⟩) ∧ t ≠ vacío}

{post: ⟨cédula⟩}

TAD De Juegos

Juego = {Código = <código>, Precio = <precio>, Unidades = <unidades>, Estantería = <estantería>}			
{ inv: juego.Código ≠ nulo ∧ juego.Precio ≠ nulo ∧ juego.Unidades ≠ nulo ∧ juego.Estantería ≠ nulo }			
Operaciones primitivas			
CrearJuego:	Entero x Entero x Entero x Texto		Constructora
BuscarJuego:	Entero	→ Juego	Analizadora
ObtenerCodigo:		→ Entero	Analizadora
ObtenerPrecio:		→ Entero	Analizadora
ObtenerUnidades:		→ Entero	Analizadora
ObtenerEstanterías:		→ Texto	Analizadora

CrearJuego(Código, Precio, Unidades, Estantería)

“Se crea un nuevo juego y se añade a una estantería”

{pre: código ∈ Entero ∧ Precio ∈ Entero ∧ Unidades ∈ Entero ∧ Estantería = {nombre: <nombre>, juegos: {juegos}}}
{post: Si BuscarJuego(Código) ≠ nulo entonces TRUE, de lo contrario FALSE}

BuscarJuego(código)

“Se encarga de buscar un juego a partir de su código”

{pre: juego = (código: <código>, precio: <precio>, unidades: <unidades>, estantería: <estanterías>) ∧ código ∈ Entero}
{post: juego = (código: <código>, precio: <precio>, unidades: <unidades>, estantería: <estanterías>))}

ObtenerCodigo()

“Se encarga de obtener el código de un juego”

{pre: juego = (código: <código>, precio: <precio>, unidades: <unidades>, estantería: <estanterías>)}
{post: <código>}

ObtenerPrecio()

“Se encarga de obtener el precio de un juego”

{pre: juego = (precio: <precio>, código: <codigo>, unidades: <unidades>, estantería: <estanterías>)}
{post: <Precio>}

ObtenerUnidades()

“Se encarga de obtener las unidades disponibles de un juego”

{pre: juego = (código: <código>, precio: <precio>, unidades: <unidades>, estantería: <estanterías>)}
{post: <unidades>}

ObtenerEstanterías()

“Se encarga de obtener el nombre de la estantería a la que pertenece un juego”

{pre: juego = (código: <código>, precio: <precio>, unidades: <unidades>, estantería: <estanterías>)}
{post: <estanterías>}}

TAD De HashTable

HashTable= {Lista de llaves: <Llaves>, Lista de valores: <valores>}

{ inv: llave \neq nulo \wedge valor \neq nulo }

Operaciones primitivas

CrearTabla:			Constructora
VerificarEspacio:		→ Booleano	Analizadora
LongitudDeTabla:		→ Entero	Analizadora
InsertarElemento:	Llave x Valor	→ Booleano	Modificadora
EliminarElemento:	Llave	→ Booleano	Modificadora
BuscarElemento:	Llave	→ Elemento	Analizadora

CrearTabla()

“Se crea una nueva tabla hash”

{pre: True}
{post: HashTable<> \wedge Hashtable.VerificarEspacio = True (Significa que la tabla está vacía)}

VerificarEspacio()

“Determina si la tabla se encuentra vacía”

{pre: TRUE}
{post: Si el hashTable está vacía entonces TRUE, de lo contrario FALSE}

LongitudDeTabla()

“Informa cuál es la cantidad de elementos que están presentes en la tabla”

{pre: TRUE}
{post: <Entero>}

InsertarElemento(k, v)

“Inserta un nuevo elemento con su respectiva llave en la hashTable”

{pre: k = llave \wedge v = valor}
{post: Si buscarElemento(k) \neq nulo entonces TRUE, de lo contrario FALSE}

EliminarElemento(k)

“Elimina un elemento de la hash table”

{pre: Llave}

{post: si elimino el elemento entonces TRUE, de lo contrario FALSE}

BuscarElemento(k)

“Busca un elemento con la llave dentro de la tabla hash”

{pre: k = llave}

{post: valor}

TAD De Stack

$Stack = \{ \langle e_1, e_2, e_3, e_4, e_n \rangle, top \}$

{ inv: $0 \leq n \wedge \text{tamaño del stack} = n \wedge top = e_n$ }

Operaciones primitivas

CrearStack:			Constructora
InsertarElemento:	Elemento	→ Booleano	Modificadora
EliminarElemento:	Elemento	→ Booleano	Modificadora
ObtenerTop:	Elemento	→ Elemento	Analizadora
EstáVacio:		→ Booleano	Analizadora

CrearStack()

“Crear una stack vacía”

{pre: TRUE}

{post: stack = vacío}

InsertarElemento(e)

“Insertar un elemento e al final del stack”

{pre: $stack = \langle e_1, e_2, e_3, e_4, e_n \rangle \wedge e \vee \text{stack son vacíos}$ }

{post: $Stack\ s = \langle e_1, e_2, e_3, e_n, e_{n+1} \rangle \vee \text{stack} \neq \text{vacío}$ }

EliminarElemento()

“Eliminar y obtener el elemento top del stack”

{pre: $stack = \langle e_1, e_2, e_3, e_4, e_n \rangle \vee \text{stack} \neq \text{vacíos}$ }

{post: $Stack\ s = \langle e_1, e_2, e_3, e_4, e_{n-1} \rangle$ }



ObtenerTop()

“Obtiene el elemento en el top del stack”

{pre: stack = $\langle e_1, e_2, e_3, e_4, e_n \rangle$ }

{post: $\langle e_n \rangle$ }

EstáVacío()

“Informa si la pila stack está vacía o no”

{pre: stack = $\langle \rangle$ }

{post: TRUE si está vacía o FALSE si posee elementos}

TAD De Queue

Queue = $\{ \langle \langle e_1, e_2, e_3, e_4, e_n, \text{front}, \text{back} \rangle \rangle \}$

{ inv: $0 \leq n \wedge \text{tamaño del queue} = n \wedge \text{front} = e_1 \wedge \text{back} = e_n$ }

Operaciones primitivas

CrearQueue:			Constructora
InsertarElemento:	Elemento	→ Booleano	Modificadora
EliminarElemento:	Elemento	→ Booleano	Modificadora
ObtenerPrimero:	Elemento	→ Elemento	Analizadora
EstáVacío		→ Booleano	Analizadora

CrearQueue():

“Crear una queue vacía”

{pre: TRUE}

{post: queue = vacío}

InsertarElemento(e):

“Insertar un elemento e al final del queue”

{pre: queue = $\langle e_1, e_2, e_3, e_4, e_n \rangle \wedge e \vee \text{queue son vacíos}$ }

{post: Queue queue = $\langle e_1, e_2, e_3, e_n, e_{n+1} \rangle \vee \text{queue} \neq \text{vacío}$ }

EliminarElemento()

“Eliminar y obtener el primer elemento del Queue”

{pre: queue = $\langle e_1, e_2, e_3, e_4, e_n \rangle \vee \text{queue} \neq \text{vacíos}$ }

{post: Queue q = $\langle e_2, e_3, e_4, e_n \rangle$ }

ObtenerPrimero()

“Obtener el primer elemento del Queue”

$\{pre: queue = \langle e_1, e_2, e_3, e_4, e_n \rangle\}$
 $\{post: \langle e_1 \rangle\}$

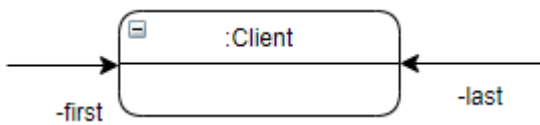
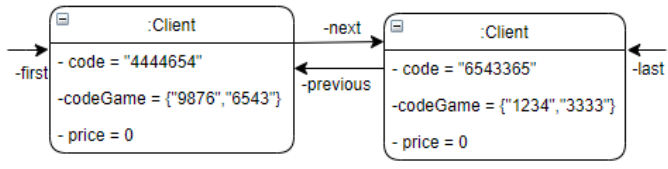
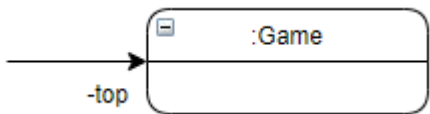
EstáVacio()

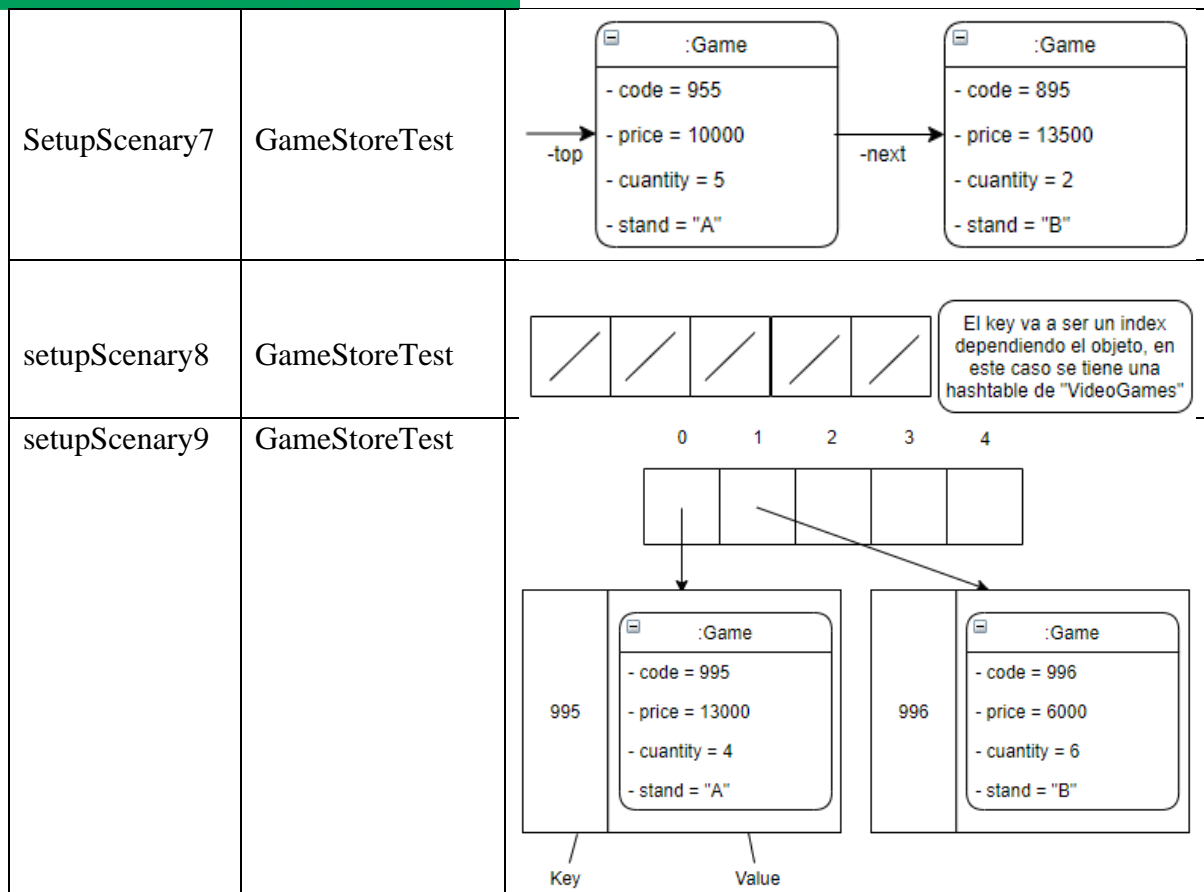
"Informa si la cola queue está vacía o no"

$\{pre: queue = \langle \rangle\}$
 $\{post: TRUE \text{ si está vacía o } FALSE \text{ si posee elementos}\}$

CASOS DE PRUEBA

Configuración:

Nombre	Clase	Escenario
setupScenary1	GameStoreTest	
setupScenary2	GameStoreTest	
setupScenary3	GameStoreTest	
setupScenary4	GameStoreTest	
SetupScenary5	GameStoreTest	
setupScenary6	GameStoreTest	



Diseño de casos de prueba:

Objetivo de la prueba: Verificar que se crea una nueva cola.				
Clase	Método	Escenario	Valores de entrada	Resultado
		setupScenary1		Se creó correctamente una cola con un objeto first y un last nulo.

Objetivo de la prueba: Verificar que se creó una nueva pila.				
Clase	Método	Escenario	Valores de entrada	Resultado
		setupScenary2		Se creó correctamente una pila con un objeto top nulo.

Objetivo de la prueba: Verificar que se creó una hashTable.				
---	--	--	--	--

Clase	Método	Escenario	Valores de entrada	Resultado
		setupScenary3	arraySize = 25	Se creó correctamente una hashTable (En este caso un array vacío de tamaño 25).
		setupScenary3	arraySize = 10	Se creó correctamente una hashTable (En este caso un array vacío de tamaño 10).

Objetivo de la prueba: Validar que se agregó un objeto a la fila.

Clase	Método	Escenario	Valores de entrada	Resultado
		setupScenary4	code = "6998987" codeGame = "1234;6543"	true Se agregó un objeto a la fila, ahora es first y last.
		setupScenary4	code = "65459876" codeGame = "6669;33354"	true Se agregó un objeto a la fila, ahora es first y last.

Objetivo de la prueba: Validar que se eliminó un objeto de la cola.

Clase	Método	Escenario	Valores de entrada	Resultado
		setupScenary5		true Se eliminó el objeto con la referencia first, ahora la cola sólo posee un objeto.

Objetivo de la prueba: Validar que hay un objeto en la posición first de la cola.

Clase	Método	Escenario	Valores de entrada	Resultado
-------	--------	-----------	--------------------	-----------



		setupScenary5		<pre>:Client - code = "4444654" -codeGame = ("9876","6543") - price = 0</pre>
--	--	---------------	--	---

Objetivo de la prueba: Validar que la cola se encuentra vacía.

Clase	Método	Escenario	Valores de entrada	Resultado
		setupScenary4		true La cola se encuentra vacía, por lo que retorna verdadero.

Objetivo de la prueba: Validar que se agrega un objeto en la pila.

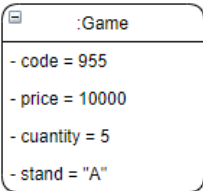
Clase	Método	Escenario	Valores de entrada	Resultado
		setupScenary6		true Se agregó un objeto a la pila por lo que la referencia de top ahora es el objeto nuevo.

Objetivo de la prueba: Validar que se elimina el objeto en el top de la pila.

Clase	Método	Escenario	Valores de entrada	Resultado
		setupScenary7		<pre>:Game - code = 955 - price = 10000 - quantity = 5 - stand = "A"</pre> Se elimina el objeto y se retorna al mismo tiempo, las referencias del objeto top cambian al

				objeto que le sigue.
--	--	--	--	----------------------

Objetivo de la prueba: Validar el objeto que se encuentra en el top de la pila.

Clase	Método	Escenario	Valores de entrada	Resultado
		setupScenary7		 <p>Se retorna el objeto en la referencia top.</p>

Objetivo de la prueba: Validar si la pila se encuentra vacía.

Clase	Método	Escenario	Valores de entrada	Resultado
		setupScenary6		<p>true</p> <p>La pila se encuentra vacía, por lo que retorna verdadero.</p>

Objetivo de la prueba: Validar si la hashTable se encuentra vacía.

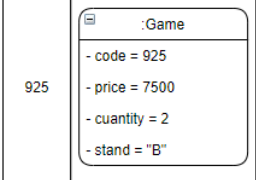
Clase	Método	Escenario	Valores de entrada	Resultado
		setupScenary8		<p>true</p> <p>La hashTable se encuentra vacía, por lo que retorna verdadero.</p>

Objetivo de la prueba: Validar el tamaño de la hashTable.

Clase	Método	Escenario	Valores de entrada	Resultado
		setupScenary8		<p>5</p> <p>La hashTable tiene un tamaño de 5.</p>

Objetivo de la prueba: Validar que se agregue un objeto en la hashTable.

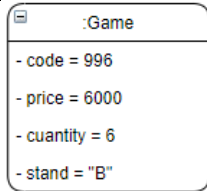
Clase	Método	Escenario	Valores de entrada	Resultado
-------	--------	-----------	--------------------	-----------

		setupScenary8		true Se agregó el objeto con la llave en la hashTable.
--	--	---------------	--	---

Objetivo de la prueba: Verificar que se puede eliminar un elemento de la hashTable.

Clase	Método	Escenario	Valores de entrada	Resultado
		setupScenary9	Key = 996	true Se eliminó correctamente el objeto con la llave 996.

Objetivo de la prueba: Verificar que se puede encontrar un objeto gracias a su llave.

Clase	Método	Escenario	Valores de entrada	Resultado
		setupScenary9	Key = 996	 Se encontró el objeto con la llave 996.