/\*

#ifndef HASH\_TABLE\_ENUMERATOR

#define HASH\_TABLE\_ENUMERATOR

#include "Enumerator.h"

#include "Exceptions.h"

#include "HashTable.h"

template <typename T>

class HashTableEnumerator : public Enumerator<T>

{

private:

unsigned long bucket = 0; // the current bucket during the enumeration process

OULinkedListEnumerator<T> chainEnumerator = NULL; // used to move through the linked list of the current bucket

HashTable<T>\* hashTable = NULL; // pointer to the HashTable being enumerated

// you may add additional member variables and functions here to support the operation of your code

public:

HashTableEnumerator(HashTable<T>\* hashTable); // constructor needs a pointer to the HashTable to be enumerated

virtual ~HashTableEnumerator();

bool hasNext() const; // true if there are elements that have not yet been returned via next()

T next(); // throws ExceptionEnumerationBeyondEnd if no next item is available

T peek() const; // throws ExceptionEnumerationBeyondEnd if no next item is available

unsigned long getBucket() {

return bucket;

}

};

// put implementation for HashTableEnumerator here

//constructor

template <typename T>

HashTableEnumerator<T>::HashTableEnumerator(HashTable<T>\* hashTable) {

this->hashTable = hashTable;

while (hashTable->table[bucket]->getSize() == 0) {

bucket++;

if (bucket >= hashTable->baseCapacity) {

throw new ExceptionHashTableAccess;

}

}

OULinkedListEnumerator<T> tempEnumerator = hashTable->table[bucket]->enumerator();

chainEnumerator = tempEnumerator;

}

//destructor

template <typename T>

HashTableEnumerator<T>::~HashTableEnumerator() {

chainEnumerator = NULL;

hashTable = NULL;

}

//determines if there is an item at next

template <typename T>

bool HashTableEnumerator<T>::hasNext() const {

unsigned long tempBucket = bucket;

if (chainEnumerator.hasNext()) {

return true;

}

else {

tempBucket++;

}

while (tempBucket < hashTable->getBaseCapacity()) {

OULinkedListEnumerator<T> tempEnumerator = this->hashTable->table[tempBucket]->enumerator();

if (!tempEnumerator.hasNext()) {

tempBucket++;

}

else {

return true;

}

}

return false;

}

//goes to next item, returns current item

template <typename T>

T HashTableEnumerator<T>::next() {

T temp;

if (!hasNext()) {

throw new ExceptionEnumerationBeyondEnd;

}

else {

if (chainEnumerator.hasNext()) {

temp = chainEnumerator.next();

return temp;

}

else {

bucket++;

}

while (bucket < hashTable->baseCapacity) {

OULinkedListEnumerator<T> tempEnumerator = hashTable->table[bucket]->enumerator();

chainEnumerator = tempEnumerator;

if (chainEnumerator.hasNext()) {

temp = chainEnumerator.next();

return temp;

}

else {

bucket++;

}

}

}

return temp;

}

//returns next item without moving to the next item

template <typename T>

T HashTableEnumerator<T>::peek() const

{

unsigned long tempBucket = bucket;

if (chainEnumerator.hasNext()) {

return chainEnumerator.peek();

}

else {

tempBucket++;

}

while (tempBucket < hashTable->getBaseCapacity()) {

OULinkedListEnumerator<T> tempEnumerator = this->hashTable->table[tempBucket]->enumerator();

if (!tempEnumerator.hasNext()) {

tempBucket++;

}

else {

return tempEnumerator.peek();

}

}

throw new ExceptionEnumerationBeyondEnd;

}

#endif // !HASH\_TABLE\_ENUMERATOR

\*/

#ifndef HASH\_TABLE\_ENUMERATOR

#define HASH\_TABLE\_ENUMERATOR

#include "Enumerator.h"

#include "Exceptions.h"

#include "HashTable.h"

template <typename T>

class HashTableEnumerator : public Enumerator<T>

{

private:

unsigned long bucket = 0; // the current bucket during the enumeration process

OULinkedListEnumerator<T> chainEnumerator = NULL; // used to move through the linked list of the current bucket

HashTable<T>\* hashTable = NULL; // pointer to the HashTable being enumerated

// you may add additional member variables and functions here to support the operation of your code

public:

HashTableEnumerator(HashTable<T>\* hashTable); // constructor needs a pointer to the HashTable to be enumerated

virtual ~HashTableEnumerator();

bool hasNext() const; // true if there are elements that have not yet been returned via next()

T next(); // throws ExceptionEnumerationBeyondEnd if no next item is available

T peek() const; // throws ExceptionEnumerationBeyondEnd if no next item is available

unsigned long getBucket()

{

return bucket;

}

};

// put implementation for HashTableEnumerator here

//constructor

template <typename T>

HashTableEnumerator<T>::HashTableEnumerator(HashTable<T>\* hashTable)

{

this->hashTable = hashTable;

// Check for first bucket with items

while (this->hashTable->table[bucket]->getSize() == 0)

{

bucket++;

if (bucket >= this->hashTable->baseCapacity)

{

throw new ExceptionHashTableAccess;

}

}

OULinkedListEnumerator<T> tempEnum = this->hashTable->table[bucket]->enumerator();

this->chainEnumerator = tempEnum;

//this->chainEnumerator = &(this->hashTable->table[bucket]->enumerator());

}

//destructor

template <typename T>

HashTableEnumerator<T>::~HashTableEnumerator()

{

chainEnumerator = NULL;

hashTable = NULL;

}

//determines if there is an item at next

template <typename T>

bool HashTableEnumerator<T>::hasNext() const

{

unsigned long tempBucket = bucket;

if (chainEnumerator.hasNext())

{

return true;

}

else

{

tempBucket++;

}

while (tempBucket < hashTable->getBaseCapacity())

{

OULinkedListEnumerator<T> tempEnum = this->hashTable->table[tempBucket]->enumerator();

//this->chainEnumerator = tempEnum;

if (!tempEnum.hasNext())

{

tempBucket++;

}

else

{

return true;

}

}

return false;

}

//goes to next item, returns current item

template <typename T>

T HashTableEnumerator<T>::next()

{

T tempReturn;

if (!hasNext())

{

throw new ExceptionEnumerationBeyondEnd;

}

else

{

if (chainEnumerator.hasNext())

{

tempReturn = chainEnumerator.next();

return tempReturn;

}

else

{

bucket++;

}

while (bucket < hashTable->getBaseCapacity())

{

OULinkedListEnumerator<T> tempEnum = this->hashTable->table[bucket]->enumerator();

this->chainEnumerator = tempEnum;

if (chainEnumerator.hasNext())

{

tempReturn = chainEnumerator.next();

return tempReturn;

}

else

{

bucket++;

}

}

}

return tempReturn;

}

//returns next item without moving to the next item

template <typename T>

T HashTableEnumerator<T>::peek() const

{

//return chainEnumerator.peek();

unsigned long tempBucket = bucket;

if (chainEnumerator.hasNext())

{

return chainEnumerator.peek();

}

else

{

tempBucket++;

}

while (tempBucket < hashTable->getBaseCapacity())

{

OULinkedListEnumerator<T> tempEnum = this->hashTable->table[tempBucket]->enumerator();

//this->chainEnumerator = tempEnum;

if (!tempEnum.hasNext())

{

tempBucket++;

}

else

{

return chainEnumerator.peek();

}

}

throw new ExceptionEnumerationBeyondEnd;

}

#endif // !HASH\_TABLE\_ENUMERATOR