# CS302 - Data Structures

## *using C++*

Topic: Designing an ADT

Kostas Alexis

AUTONOMOUS ROBOTS LAB

# Abstract Data Type

- **Abstract Data Type (ADT)**
  - A specification for a group of values and operations on those values
    - Defined (conceptually) independently of any programming language
- **Data Structure**
  - Implementation of an ADT within a programming language
- **Collection**
  - Object that groups other objects together
  - Provides various services to clients
    - Add
    - Remove
    - Query

# Abstract Data Type

- **Abstract Data Type (ADT)**
  - A specification for a group of values and operations on those values
    - Defined (conceptually) independently of any programming language
- **Data Structure**
  - Implementation of an ADT within a programming language
- **Collection**
  - Object that groups other objects together
  - Provides various services to clients
    - Add
    - Remove
    - Query

Example of collection & data type: a Bag

# A Bag's Behaviors

- Let's list a bag's expected behaviors

# A Bag's Behaviors

- Get the number of items currently in the bag
- See whether the bag is empty
- Add a given object to the bag
- Remove an occurrence of a particular object from the bag, if possible
- Remove all objects from the bag
  - (empty or clear the bag)
- Count the number of times an object occurs in the bag
- Test whether the bag contains a particular object
- Look at all objects in the bag

# A Bag's Behaviors

- Get the number of items currently in the bag
- See whether the bag is empty
- Add a given object to the bag
- Remove an occurrence of a particular object from the bag, if possible
- Remove all objects from the bag
  - (empty or clear the bag)
- Count the number of times an object occurs in the bag
- Test whether the bag contains a particular object
- Look at all objects in the bag

CRC Card

**Bag**

**Responsibilities**

- Get the number of items currently in the bag
- See whether the bag is empty
- Add a given object to the bag
- Remove an occurrence of a specific object from the bag, if possible
- Remove all objects from the bag
- Count the number of times a certain object occurs in the bag
- Test whether the bag contains a particular object
- Look at all objects that are in the bag

**Collaborations**

- The class of objects that the bag can contain

# A Bag's Behaviors

- Get the number of items currently in the bag
- See whether the bag is empty
- Add a given object to the bag
- Remove an occurrence of a particular object from the bag, if possible
- Remove all objects from the bag
  - (empty or clear the bag)
- Count the number of times an object occurs in the bag
- Test whether the bag contains a particular object
- Look at all objects in the bag

**Class Responsibility Collaboration Card**

CRC Card

**Bag**

**Responsibilities**

- Get the number of items currently in the bag
- See whether the bag is empty
- Add a given object to the bag
- Remove an occurrence of a specific object from the bag, if possible
- Remove all objects from the bag
- Count the number of times a certain object occurs in the bag
- Test whether the bag contains a particular object
- Look at all objects that are in the bag

**Collaborations**

- The class of objects that the bag can contain

# A Bag's Behaviors

## CRC Card

**Bag**

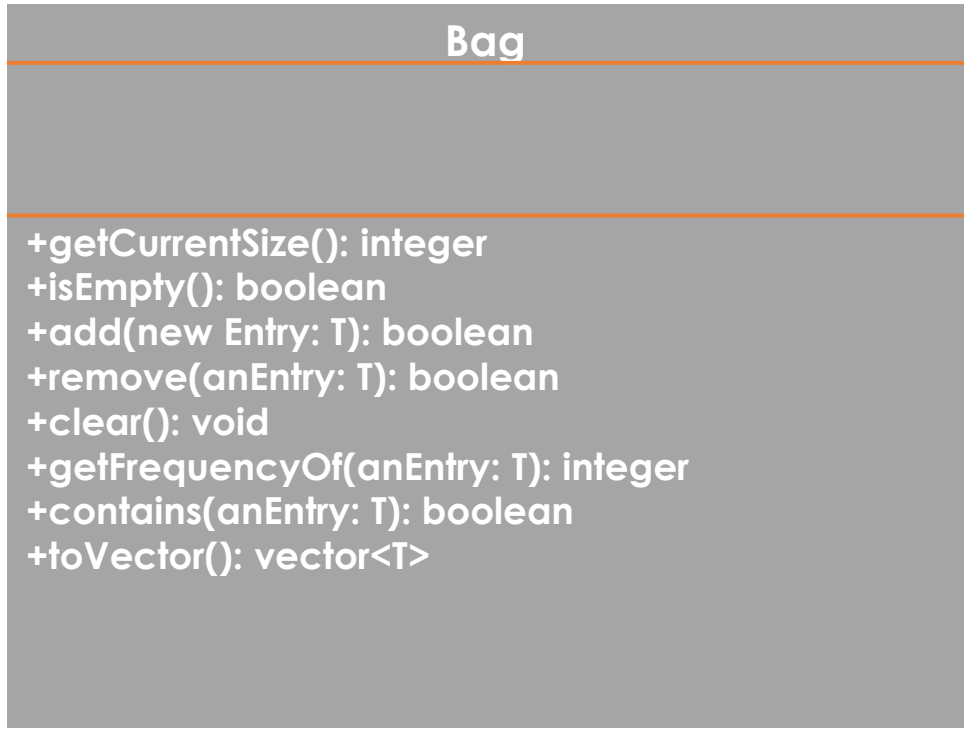
**Responsibilities**

- Get the number of items currently in the bag
- See whether the bag is empty
- Add a given object to the bag
- Remove an occurrence of a specific object from the bag, if possible
- Remove all objects from the bag
- Count the number of times a certain object occurs in the bag
- Test whether the bag contains a particular object
- Look at all objects that are in the bag

**Collaborations**

- The class of objects that the bag can contain

## UML Notation



# A Bag's Behaviors

## CRC Card

**Bag**

**Responsibilities**

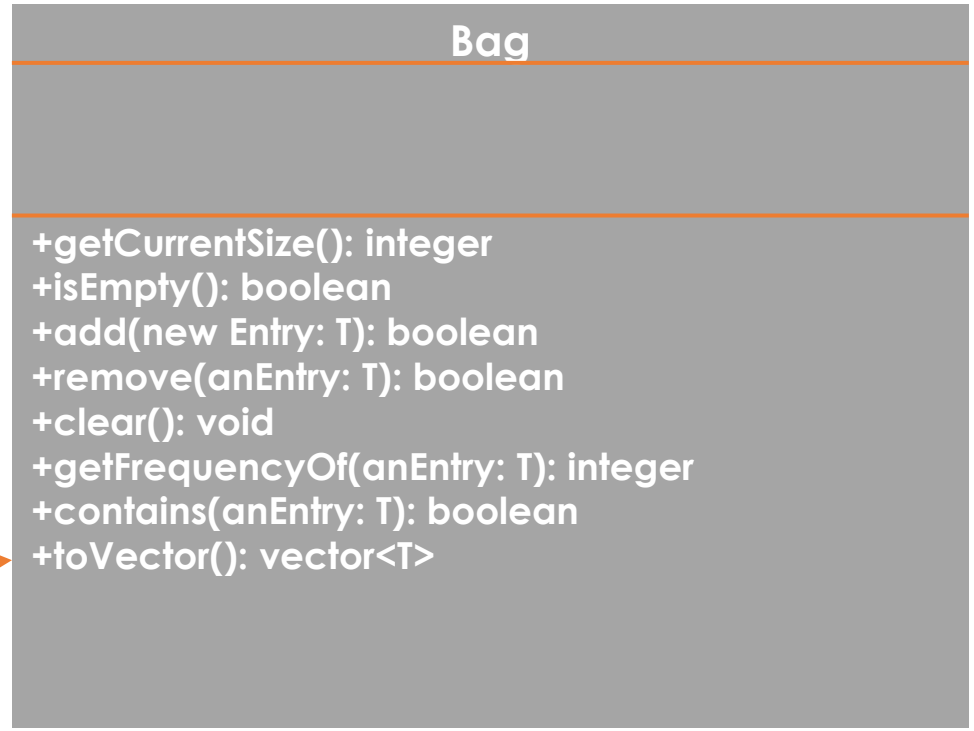
- Get the number of items currently in the bag
- See whether the bag is empty
- Add a given object to the bag
- Remove an occurrence of a specific object from the bag, if possible
- Remove all objects from the bag
- Count the number of times a certain object occurs in the bag
- Test whether the bag contains a particular object
- Look at all objects that are in the bag

**Collaborations**

- The class of objects that the bag can contain

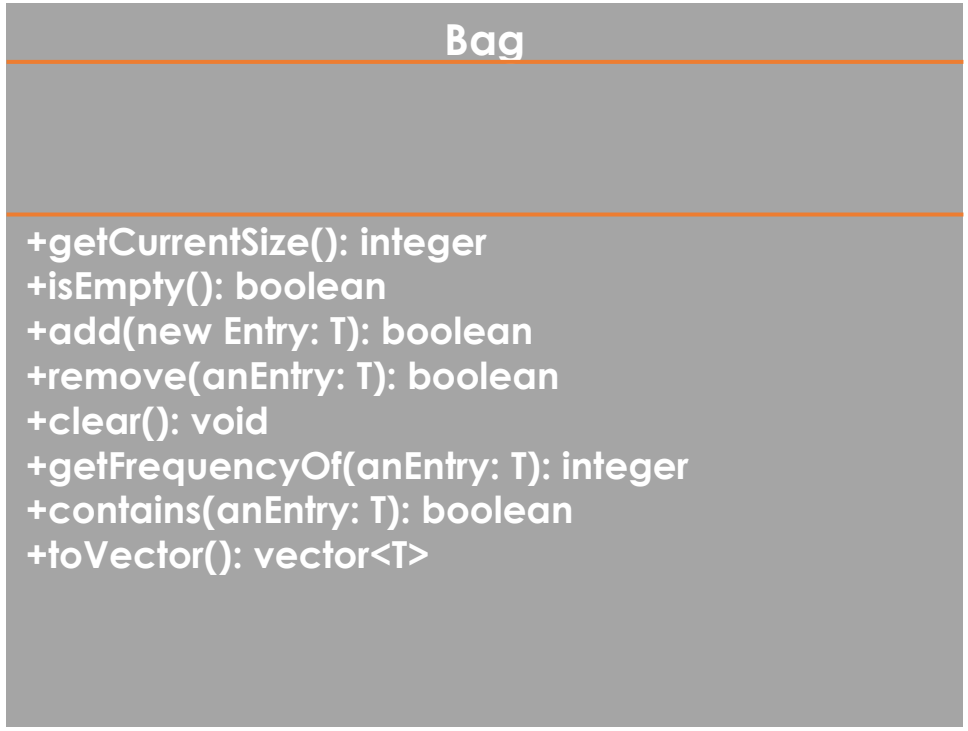
Returns a vector of these entries – makes class more portable and independent of display methods

## UML Notation



# Interface of the ADT Bag

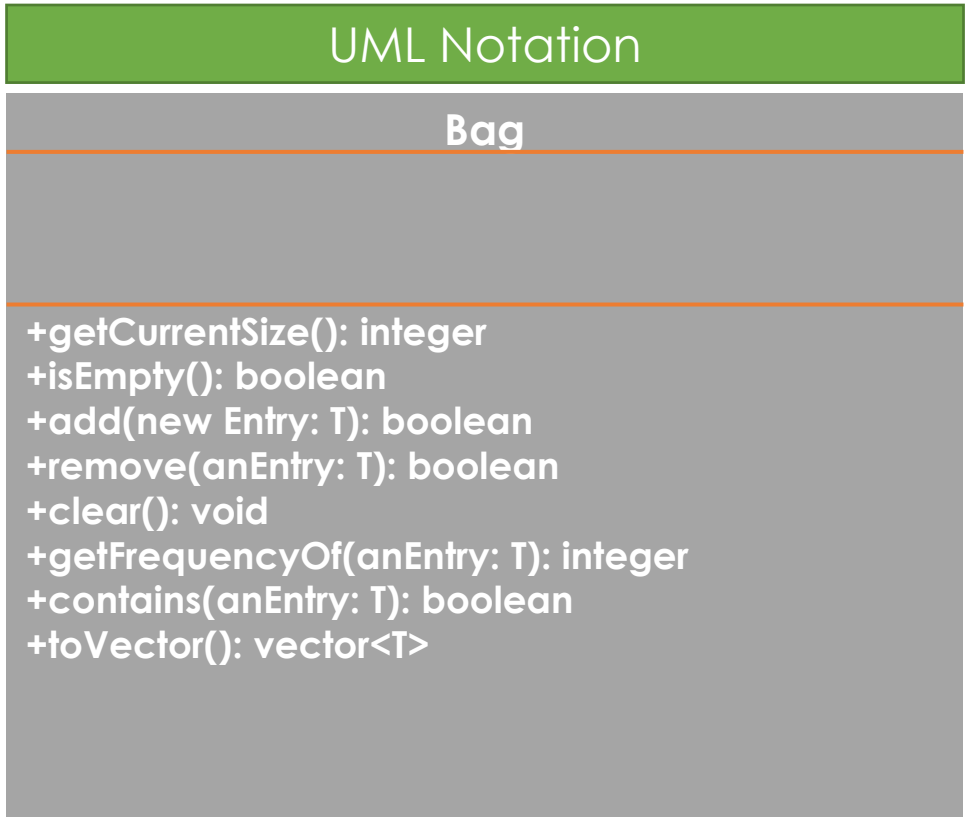
UML Notation




BagInterface.h

```
#ifndef _BagInterface_h
#define _BagInterface_h

#include <vector>

template<class ItemType>
class BagInterface
{
public:
     virtual int getCurrentSize() const = 0;
     virtual bool isEmpty() const = 0;
     virtual bool add(const ItemType& newEntry) = 0;
     virtual bool remove(const ItemType& anEntry) = 0;
     virtual void clear() = 0;
     virtual int getFrequencyOf(const ItemType& anEntry) const = 0;
     virtual bool contains(const ItemType& anEntry) const = 0;
     virtual std::vector<ItemType> toVector() const = 0;
     virtual ~BagInterface() { }
};
#endif
```

# Interface of the ADT Bag

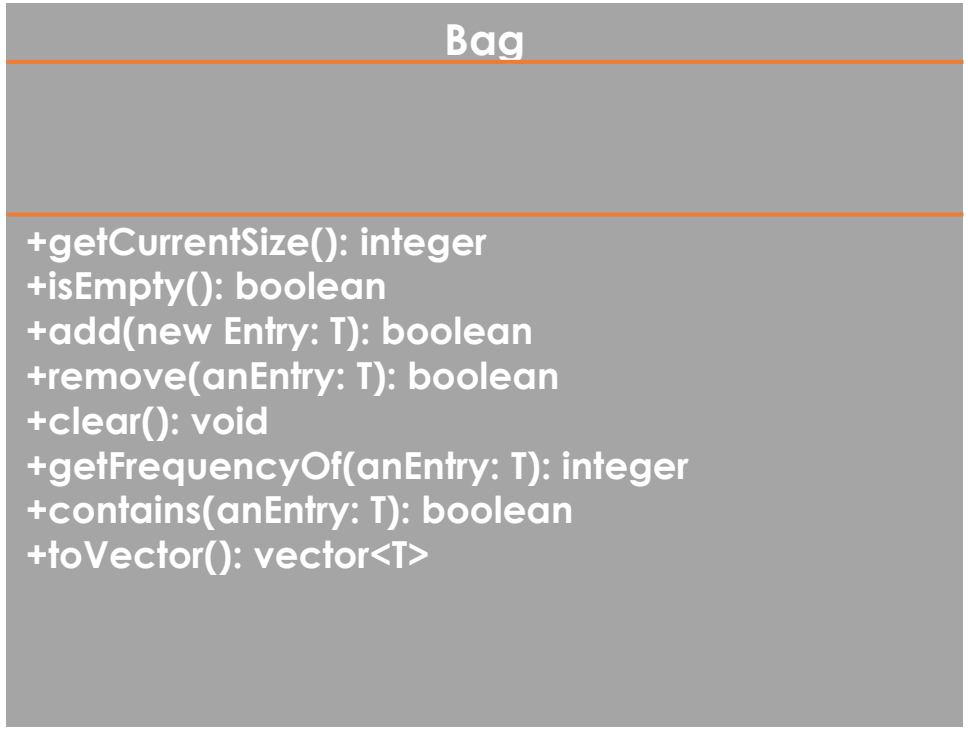



```
#ifndef _BagInterface_h
#define _BagInterface_h

#include <vector>

template<class ItemType>
class BagInterface
{
public:
     virtual int getCurrentSize() const = 0;
     virtual bool isEmpty() const = 0;
     virtual bool add(const ItemType& newEntry) = 0;
     virtual bool remove(const ItemType& anEntry) = 0;
     virtual void clear() = 0;
     virtual int getFrequencyOf(const ItemType& anEntry) const = 0;
     virtual bool contains(const ItemType& anEntry) const = 0;
     virtual std::vector<ItemType> toVector() const = 0;
     virtual ~BagInterface() { }
};
#endif
```

ensures this header file is only included once when app is built

# Interface of the ADT Bag

UML Notation

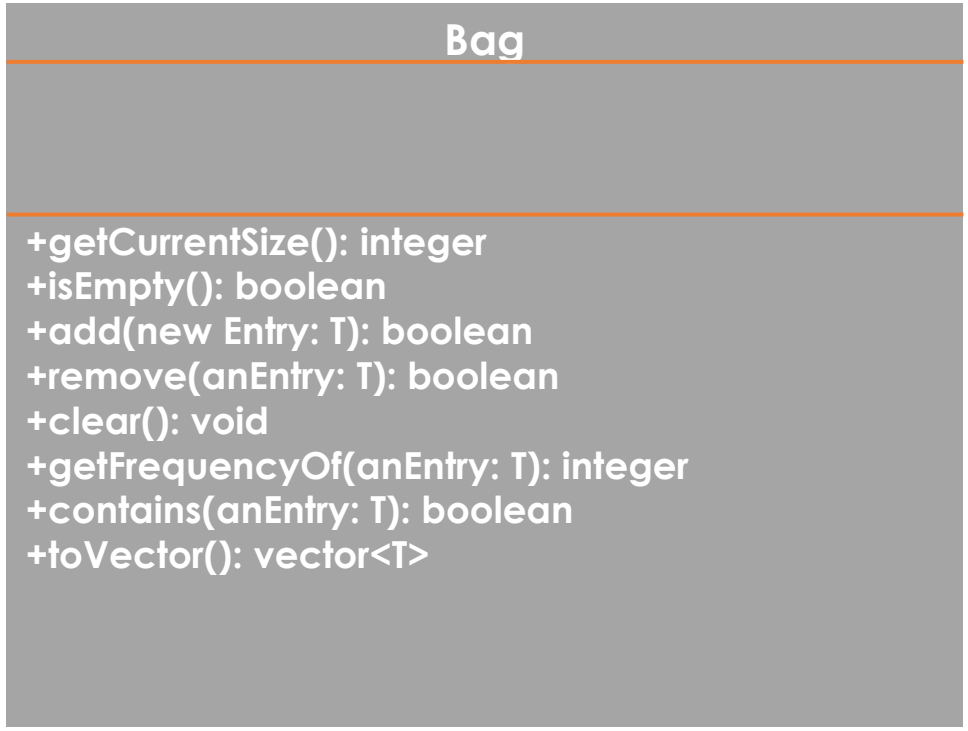



templated class

```
#ifndef _BagInterface_h
#define _BagInterface_h

#include <vector>

template<class ItemType>
class BagInterface
{
public:
     virtual int getCurrentSize() const = 0;
     virtual bool isEmpty() const = 0;
     virtual bool add(const ItemType& newEntry) = 0;
     virtual bool remove(const ItemType& anEntry) = 0;
     virtual void clear() = 0;
     virtual int getFrequencyOf(const ItemType& anEntry) const = 0;
     virtual bool contains(const ItemType& anEntry) const = 0;
     virtual std::vector<ItemType> toVector() const = 0;
     virtual ~BagInterface() { }
};
#endif
```

# Interface of the ADT Bag

UML Notation



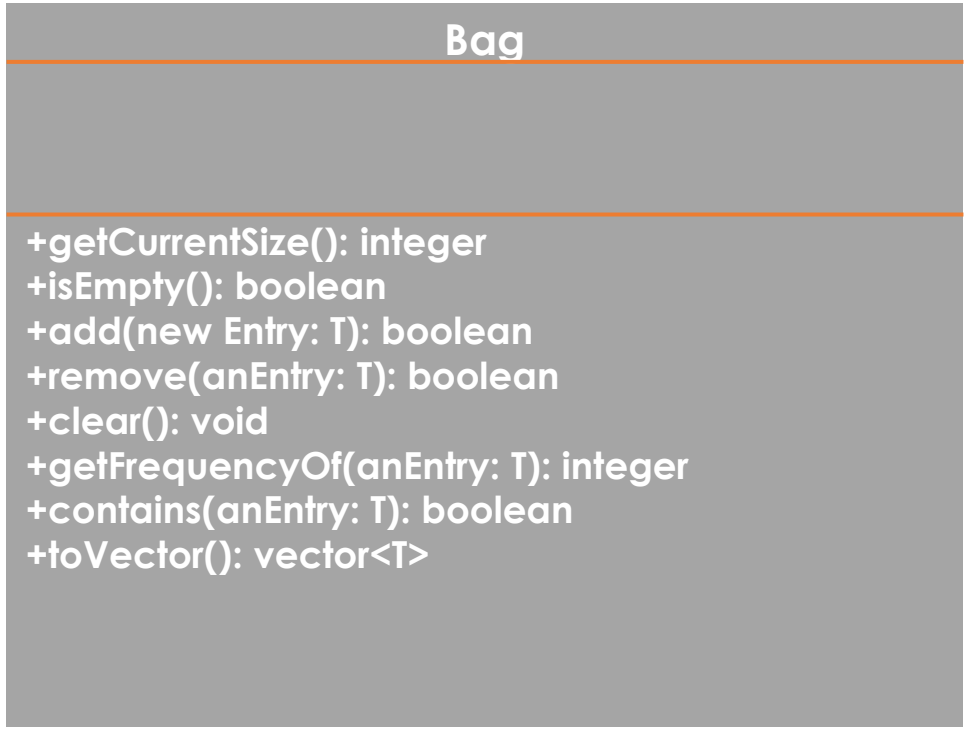

```
#ifndef _BagInterface_h
#define _BagInterface_h

#include <vector>

template<class ItemType>
class BagInterface
{
public:
     virtual int getCurrentSize() const = 0;
     virtual bool isEmpty() const = 0;
     virtual bool add(const ItemType& newEntry) = 0;
     virtual bool remove(const ItemType& anEntry) = 0;
     virtual void clear() = 0;
     virtual int getFrequencyOf(const ItemType& anEntry) const = 0;
     virtual bool contains(const ItemType& anEntry) const = 0;
     virtual std::vector<ItemType> toVector() const = 0;
     virtual ~BagInterface() { }
};
#endif
```

templated class

allows polymorphism

# Interface of the ADT Bag

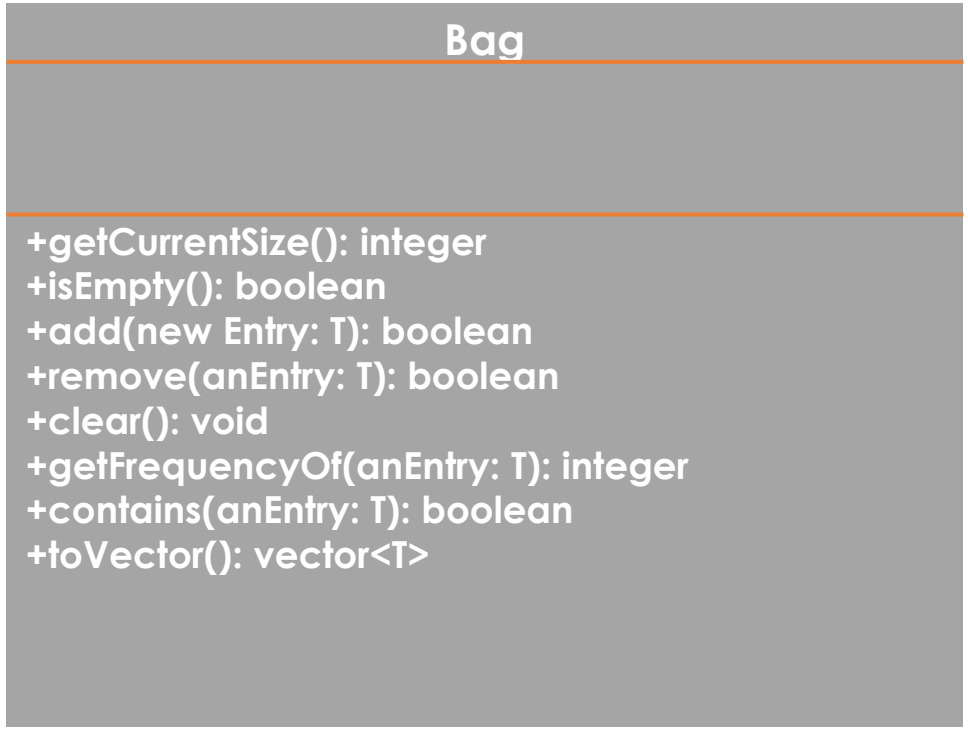
UML Notation




```
/** #file BagInterface.h */
#ifndef _BagInterface_h
#define _BagInterface_h

#include <vector>

template<class ItemType>
class BagInterface
{
public:
     virtual int getCurrentSize() const = 0;
     virtual bool isEmpty() const = 0;
     virtual bool add(const ItemType& newEntry) = 0;
     virtual bool remove(const ItemType& anEntry) = 0;
     virtual void clear() = 0;
     virtual int getFrequencyOf(const ItemType& anEntry) const = 0;
     virtual bool contains(const ItemType& anEntry) const = 0;
     virtual std::vector<ItemType> toVector() const = 0;
     virtual ~BagInterface() { }
};
#endif
```

# Interface of the ADT Bag

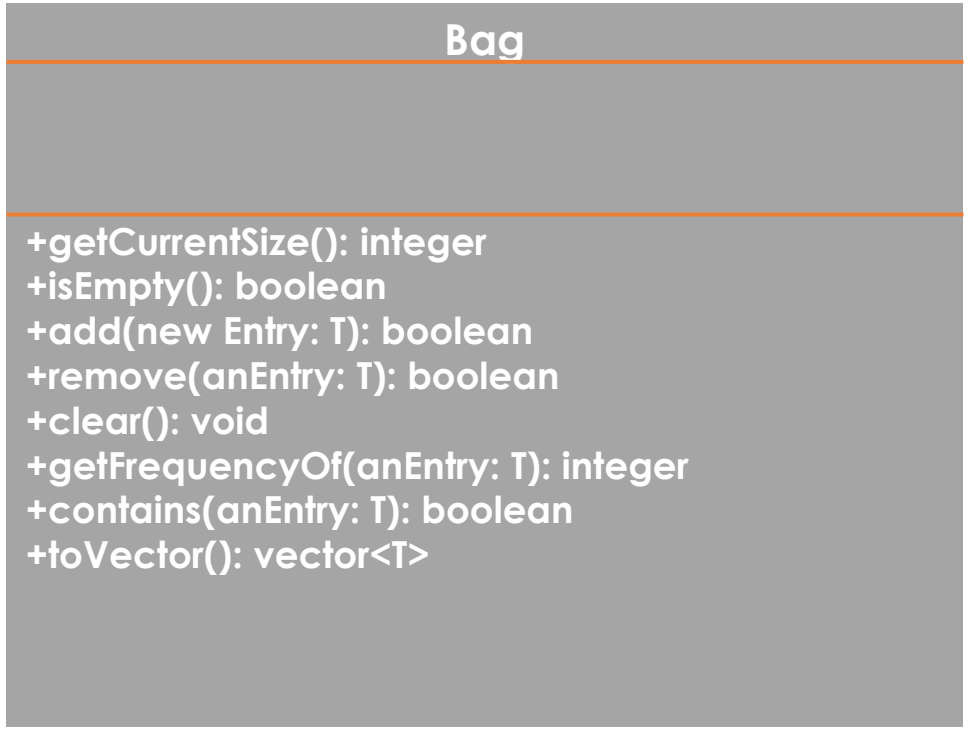
UML Notation




```
#ifndef _BagInterface_h
#define _BagInterface_h

#include <vector>

template<class ItemType>
class BagInterface
{
public:
    /** Gets the current number of entries in this bag.
    @return the integer number of entries currently in the bag */
    virtual int getCurrentSize() const = 0;
    virtual bool isEmpty() const = 0;
    virtual bool add(const ItemType& newEntry) = 0;
    virtual bool remove(const ItemType& anEntry) = 0;
    virtual void clear() = 0;
    virtual int getFrequencyOf(const ItemType& anEntry) const = 0;
    virtual bool contains(const ItemType& anEntry) const = 0;
    virtual std::vector<ItemType> toVector() const = 0;
    virtual ~BagInterface() { }
};
#endif
```

# Interface of the ADT Bag

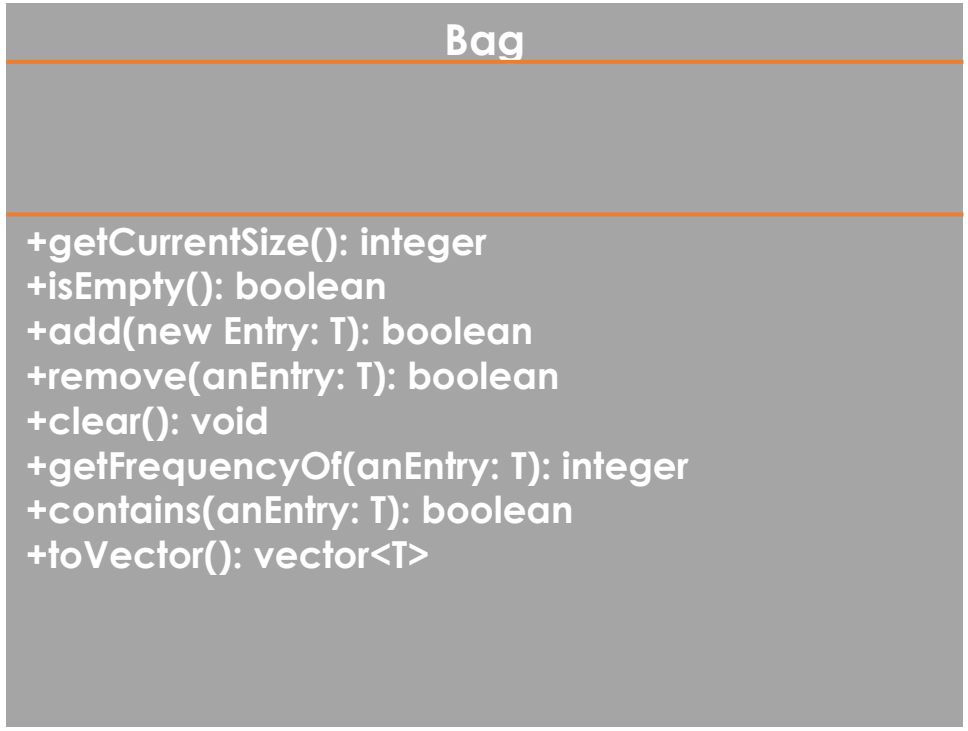
UML Notation




```
#ifndef _BagInterface_h
#define _BagInterface_h

#include <vector>

template<class ItemType>
class BagInterface
{
public:
     virtual int getCurrentSize() const = 0;
     /** Sees whether this bag is empty.
     @return true if the bag is empty, or false if not. */
     virtual bool isEmpty() const = 0;
     virtual bool add(const ItemType& newEntry) = 0;
     virtual bool remove(const ItemType& anEntry) = 0;
     virtual void clear() = 0;
     virtual int getFrequencyOf(const ItemType& anEntry) const = 0;
     virtual bool contains(const ItemType& anEntry) const = 0;
     virtual std::vector<ItemType> toVector() const = 0;
     virtual ~BagInterface() { }
};
#endif
```

# Interface of the ADT Bag

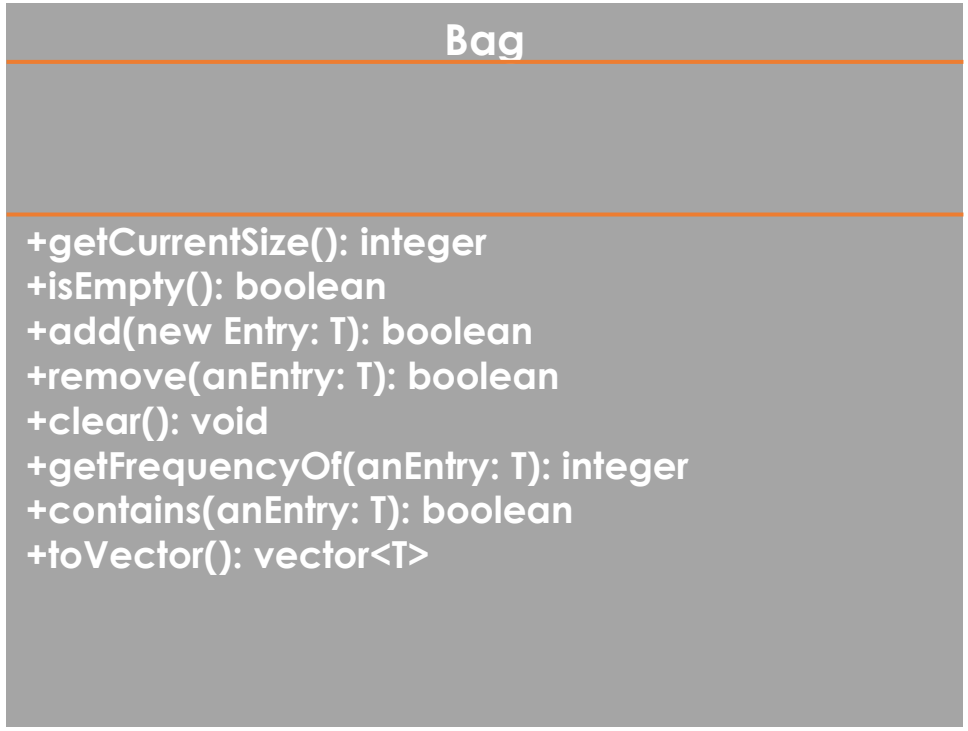
UML Notation




```
#ifndef _BagInterface_h
#define _BagInterface_h

#include <vector>

template<class ItemType>
class BagInterface
{
public:
     virtual int getCurrentSize() const = 0;
     virtual bool isEmpty() const = 0;
     /** Adds a new entry to this bag.
     @post if successful, newEntry is stored in the bag and the
     count of items in the bag is increased by 1.
     @param newEntry the object to be added as a new entry.
     @return true if addition was successful, or false if not. */
     virtual bool add(const ItemType& newEntry) = 0;
     virtual bool remove(const ItemType& anEntry) = 0;
     virtual void clear() = 0;
     virtual int getFrequencyOf(const ItemType& anEntry) const = 0;
     virtual bool contains(const ItemType& anEntry) const = 0;
     virtual std::vector<ItemType> toVector() const = 0;
     virtual ~BagInterface() { }
};
#endif
```

# Interface of the ADT Bag

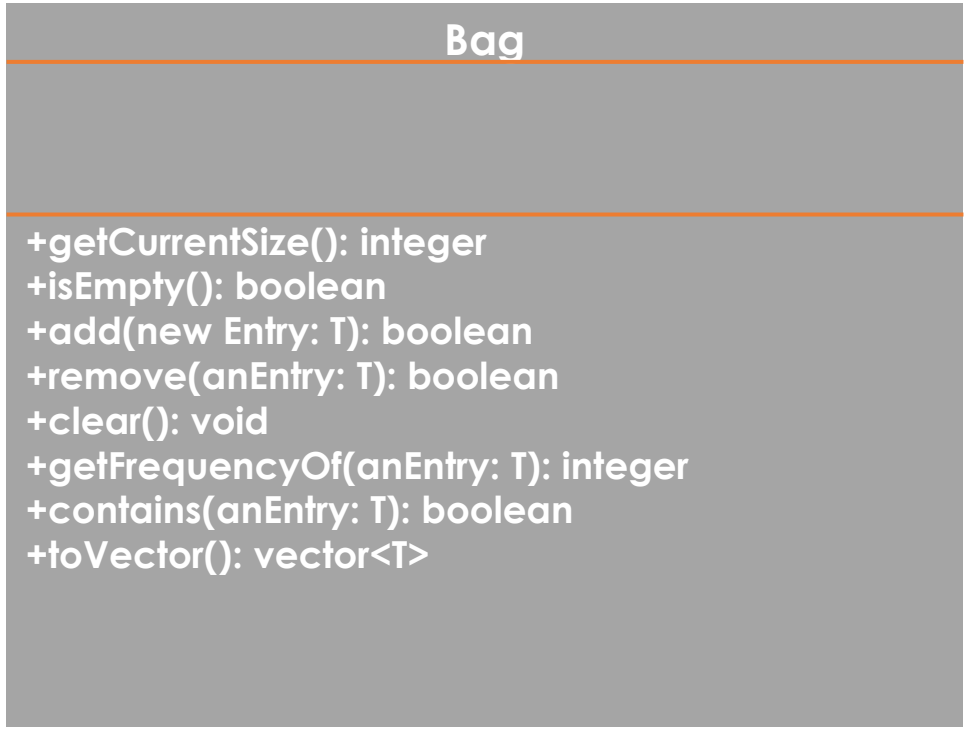
UML Notation




```
#ifndef _BagInterface_h
#define _BagInterface_h

#include <vector>

template<class ItemType>
class BagInterface
{
public:
     virtual int getCurrentSize() const = 0;
     virtual bool isEmpty() const = 0;
     virtual bool add(const ItemType& newEntry) = 0;
     /** Removes one occurrence of a given entry from bag, if
     possible.
     @post if successful, anEntry is removed from the bag, count -1
     @param anEntry the entry to be removed
     @return true if removal was successful, or false if not. */
     virtual bool remove(const ItemType& anEntry) = 0;
     virtual void clear() = 0;
     virtual int getFrequencyOf(const ItemType& anEntry) const = 0;
     virtual bool contains(const ItemType& anEntry) const = 0;
     virtual std::vector<ItemType> toVector() const = 0;
     virtual ~BagInterface() { }
};
#endif
```

# Interface of the ADT Bag

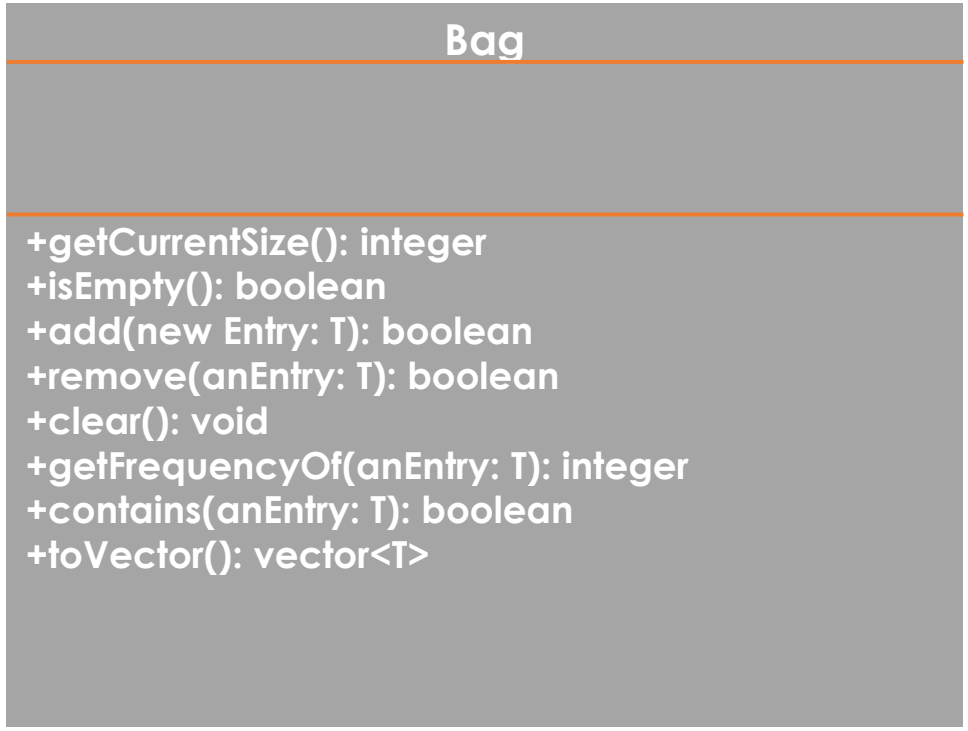
UML Notation




```
/** #file BagInterface.h */
#ifndef _BagInterface_h
#define _BagInterface_h

#include <vector>

template<class ItemType>
class BagInterface
{
public:
     virtual int getCurrentSize() const = 0;
     virtual bool isEmpty() const = 0;
     virtual bool add(const ItemType& newEntry) = 0;
     virtual bool remove(const ItemType& anEntry) = 0;
     /** Removes all entries from this bag.
     @post Bag contains no items, and the count of items is 0. */
     virtual void clear() = 0;
     virtual int getFrequencyOf(const ItemType& anEntry) const = 0;
     virtual bool contains(const ItemType& anEntry) const = 0;
     virtual std::vector<ItemType> toVector() const = 0;
     virtual ~BagInterface() { }
};
#endif
```

# Interface of the ADT Bag

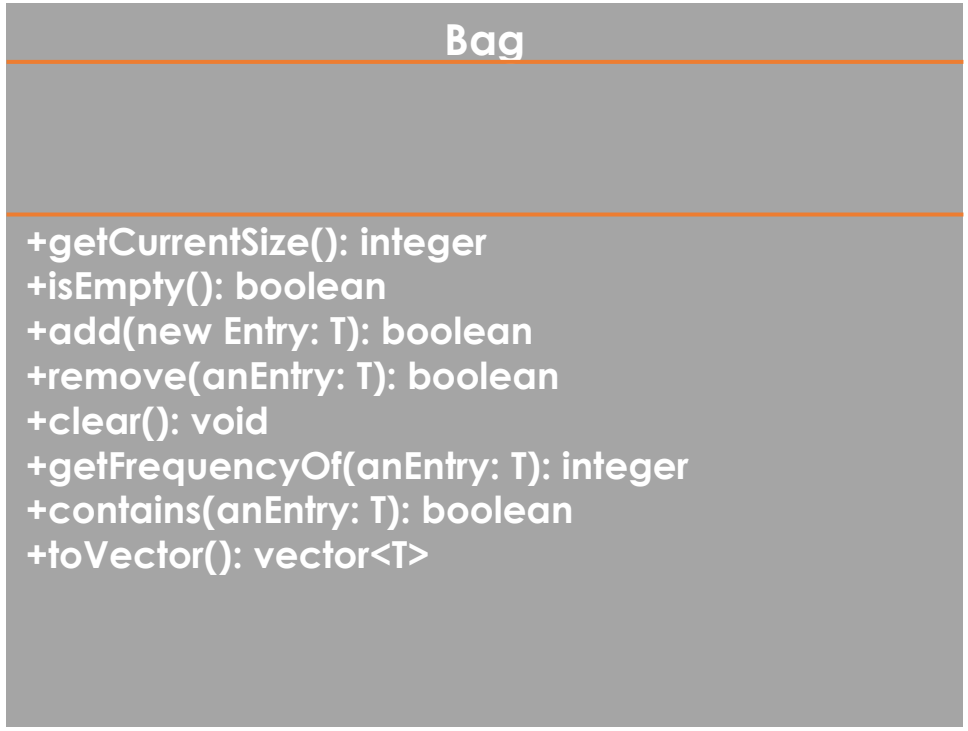
UML Notation




```
/** #file BagInterface.h */
#ifndef _BagInterface_h
#define _BagInterface_h

#include <vector>

template<class ItemType>
class BagInterface
{
public:
     virtual int getCurrentSize() const = 0;
     virtual bool isEmpty() const = 0;
     virtual bool add(const ItemType& newEntry) = 0;
     virtual bool remove(const ItemType& anEntry) = 0;
     virtual void clear() = 0;
     /** Counts the number of times a given entry appears in this
     bag.
     @param anEntry the entry to be counted.
     @return the number of items anEntry appears in the bag. */
     virtual int getFrequencyOf(const ItemType& anEntry) const = 0;
     virtual bool contains(const ItemType& anEntry) const = 0;
     virtual std::vector<ItemType> toVector() const = 0;
     virtual ~BagInterface() { }
};
#endif
```

# Interface of the ADT Bag

UML Notation




```
/** #file BagInterface.h */
#ifndef _BagInterface_h
#define _BagInterface_h

#include <vector>

template<class ItemType>
class BagInterface
{
public:
     virtual int getCurrentSize() const = 0;
     virtual bool isEmpty() const = 0;
     virtual bool add(const ItemType& newEntry) = 0;
     virtual bool remove(const ItemType& anEntry) = 0;
     virtual void clear() = 0;
     virtual int getFrequencyOf(const ItemType& anEntry) const = 0;
     /** Tests whether this bag contains a given entry.
     @param anEntry the entry to locate
     @return true if bag contains anEntry, false otherwise */
     virtual bool contains(const ItemType& anEntry) const = 0;
     virtual std::vector<ItemType> toVector() const = 0;
     virtual ~BagInterface() { }
};
#endif
```

# Interface of the ADT Bag

UML Notation

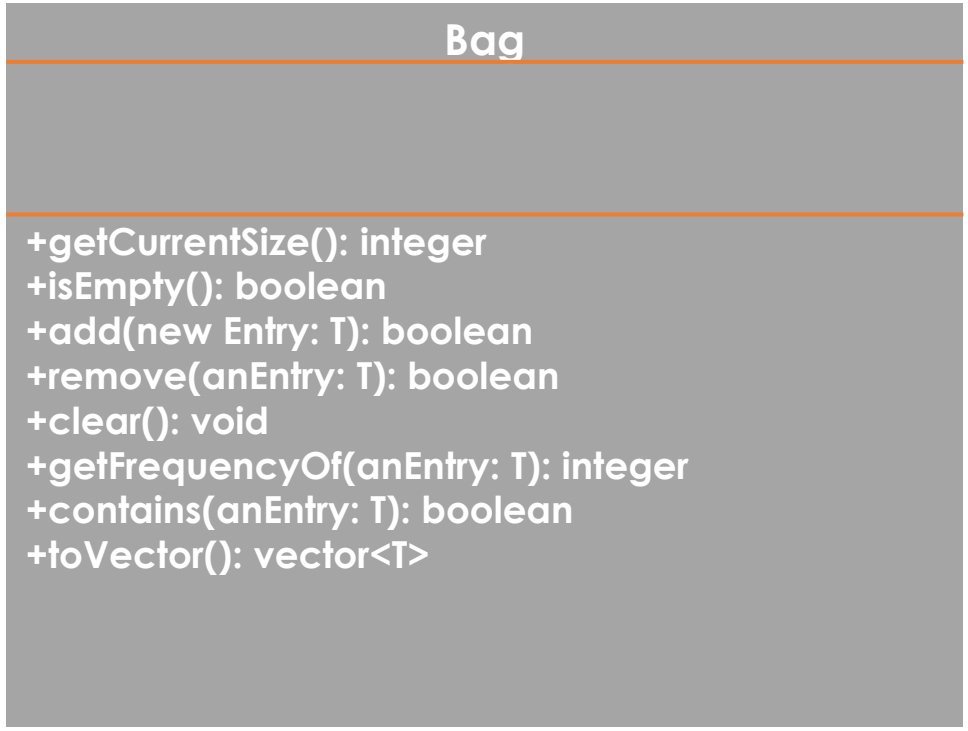
**Bag**

+getCurrentSize(): integer
+isEmpty(): boolean
+add(new Entry: T): boolean
+remove(anEntry: T): boolean
+clear(): void
+getFrequencyOf(anEntry: T): integer
+contains(anEntry: T): boolean
+toVector(): vector<T>

```
/** #file BagInterface.h */
#ifndef _BagInterface_h
#define _BagInterface_h

#include <vector>

template<class ItemType>
class BagInterface
{
public:
     virtual int getCurrentSize() const = 0;
     virtual bool isEmpty() const = 0;
     virtual bool add(const ItemType& newEntry) = 0;
     virtual bool remove(const ItemType& anEntry) = 0;
     virtual void clear() = 0;
     virtual int getFrequencyOf(const ItemType& anEntry) const = 0;
     virtual bool contains(const ItemType& anEntry) const = 0;
     /** Empties and then fills a given vector with all entries
     that are in this bag.
     @return a vector containing copies of all the entries in the
     bag. */
     virtual std::vector<ItemType> toVector() const = 0;
     virtual ~BagInterface() { }
};
#endif
```

# Interface of the ADT Bag

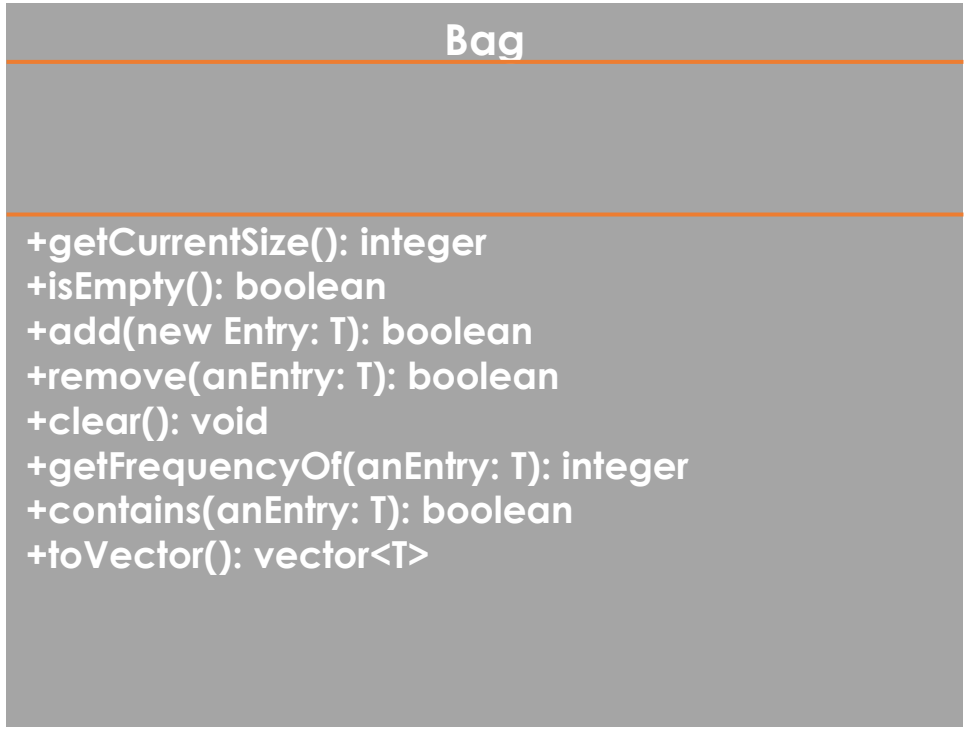
UML Notation




```
/** #file BagInterface.h */
#ifndef _BagInterface_h
#define _BagInterface_h

#include <vector>

template<class ItemType>
class BagInterface
{
public:
     virtual int getCurrentSize() const = 0;
     virtual bool isEmpty() const = 0;
     virtual bool add(const ItemType& newEntry) = 0;
     virtual bool remove(const ItemType& anEntry) = 0;
     virtual void clear() = 0;
     virtual int getFrequencyOf(const ItemType& anEntry) const = 0;
     virtual bool contains(const ItemType& anEntry) const = 0;
     virtual std::vector<ItemType> toVector() const = 0;
     /** Destroys this bag and frees its assigned memory */
     virtual ~BagInterface() { }
};
#endif
```

# Interface of the ADT Bag

UML Notation




```
/** #file BagInterface.h */
#ifndef _BagInterface_h
#define _BagInterface_h

#include <vector>

template<class ItemType>
class BagInterface
{
public:
     virtual int getCurrentSize() const = 0;
     virtual bool isEmpty() const = 0;
     virtual bool add(const ItemType& newEntry) = 0;
     virtual bool remove(const ItemType& anEntry) = 0;
     virtual void clear() = 0;
     virtual int getFrequencyOf(const ItemType& anEntry) const = 0;
     virtual bool contains(const ItemType& anEntry) const = 0;
     virtual std::vector<ItemType> toVector() const = 0;
     virtual ~BagInterface() { }
}; // end BagInterface
#endif
```

# A program for a card guessing game

```
#include <iostream>
#include <string>
#include "Bag.h"

int main()
{
      std::string clubs[] = {"Joker", "Ace", "Two",
      "Three", "Four", "Five", "Six", "Seven", "Eight",
      "Nine", "Ten", "Jack", "Queen", "King" };

      // Create our bag to hold cards
      Bag<std::string> grabBag;

      // Place six cards in the bag
      grabBag.add(clubs[1]);
      grabBag.add(clubs[2]);
      grabBag.add(clubs[4]);
      grabBag.add(clubs[8]);
      grabBag.add(clubs[10]);
      grabBag.add(clubs[12]);

      // Get friend's guess and check it
      int guess = 0;
```

```
      while(!grabBag.isEmpty())
      {
            std::cout << "What is your guess? (1 for Ace
            to 13 for King):";

            // Is card in the bag
            if (grabBag.contains(clubs[guess]))
            {
                  // Good guess - remove card from bag
                  std::cout << "You get the card!\n";
                  grabBag.remove(clubs[guess]);
            }
            else
            {
                  std:: cout <"Card not in bag.\n";
            } // end if
      } //end while
      std::cout << "No more cards in the bag. Game over!\n";
      return 0;
}; // end main
```

# Thank you