std::vector

vector ()
    constructs an empty vector
vector (vector&)
    copy constructor
vector (size, val)
    constructs a vector of 'size' elements, all of default value 'val'
size (): int
    returns the number of elements in this vector
empty (): bool
    returns 'true' if this vector has no elements
begin (): iterator
    returns an iterator pointing to the first element in this vector
end (): iterator
    returns an iterator pointing past the last element in this vector
== (vector&): bool
    returns 'true' if this vector is equal to the vector parameter
!= (vector&): bool
    returns 'true' if this vector is not equal to the vector parameter
= (vector&): vector
    assigns this vector to the value of the vector parameter
insert (iterator, T&)
    inserts parameter element of type T into this vector, at the position pointed to by the iterator parameter
erase (iterator): iterator
    removes from this vector the element pointed to by the iterator parameter
clear ()
    removes all elements from this vector
at (int):  T&
    returns a reference to the element at position 'int' in this vector
push_back (T&)
    inserts a copy of element of type T at the end of this vector
pop_back ()
    removes the last element of this vector
std::set

set ()
  constructs an empty set
set (set&)
  copy constructor
size (): int
  returns the number of elements in this set
empty (): bool
  returns 'true' if this set has no elements
begin (): iterator
  returns an iterator pointing to the first element in this set
end (): iterator
  returns an iterator pointing past the last element in this set
== (set&): bool
  returns 'true' if this set is equal to the set parameter
!=(set&): bool
  returns 'true' if this set is not equal to the set parameter
=(set&): set
  assigns this set to the value of the set parameter
insert (T&): pair<iterator, bool>
  ensures that the parameter element of type T is in this set, and returns a pair containing a boolean
  indicating whether the element was already in the set (value false) or was added by this operation (value
  true), and an iterator pointing to the element in this set.
erase(iterator)
  removes from this set the element pointed to by the parameter iterator
erase(T&)
  removes from this set the parameter element of type T
clear ()
  removes all elements from this set
find (T&): iterator
  returns an iterator that points to the element in this set that equals the parameter element (returns end()
  if the parameter element is not in this set)
std::map

map ()
constructs an empty map
map (map&)
copy constructor
size () : int
returns the number of elements in this map
empty () : bool
returns 'true' if this map has no elements
begin () : iterator
returns an iterator pointing to the first element in this map
end () : iterator
returns an iterator pointing past the last element in this map
== (map&): bool
returns 'true' if this map is equal to the map parameter
!= (map&): bool
returns 'true' if this map is not equal to the map parameter
= (map&): map
assigns this map to the value of the map parameter
insert (K&): pair<iterator, bool>
ensures that a map element whose key value equals the parameter is in this map, and returns a pair
containing a boolean indicating whether the element was already in the map (value false) or was added
by this operation (value true), and an iterator pointing to the element in this map.
erase(iterator)
removes from this map the element pointed to by the parameter iterator
erase(K&)
removes from this map the element whose key value equals the parameter
clear ()
removes all elements from this map
find (K&): iterator
returns an iterator that points to the element in this map whose key value equals the parameter element
(returns end() if there is no such element in the map)
[K&] : T&
finds the map element whose key value equals the parameter and returns a reference to the
 corresponding mapped value
std::multimap

multimap ()
    constructs an empty multimap
multimap (multimap&)
    copy constructor
size (): int
    returns the number of elements in this multimap
empty (): bool
    returns 'true' if this multimap has no elements
begin (): iterator
    returns an iterator pointing to the first element in this multimap
end (): iterator
    returns an iterator pointing past the last element in this multimap
== (multimap&): bool
    returns 'true' if this multimap is equal to the multimap parameter
!= (multimap&): bool
    returns 'true' if this multimap is not equal to the multimap parameter
= (multimap&): multimap
    assigns this multimap to the value of the multimap parameter
insert (K&)
    inserts a new multimap element whose key value equals the parameter is in this map, and returns an
    iterator pointing to the new element.
erase(K&)
    removes from this multimap all elements whose key value equals the parameter
clear ()
    removes all elements from this multimap
find (K&): iterator
    returns an iterator that points to the first element in this multimap whose key value equals the parameter
element (returns end() if there is no such element in the map)
count (K&): int
    returns the number of multimap elements whose key value equals the parameter
UML model used in Question 3

UML model used in Question 7
C++ declarations and definitions used in Question 5.

```cpp
#include <vector>

class IntVector {
private:
    std::vector<int> ivec;

public:
    IntVector(int size);
    void setVal(int index, int value);
    IntVector* Add (IntVector *v2);
};

IntVector::IntVector(int size) : ivec (std::vector<int>(size, 0)) {}

void IntVector::setVal(int index, int value) {
    ivec.at(index) = value;
}

IntVector* IntVector::Add (IntVector *v2) {
    IntVector *result = new IntVector(ivec.size());
    int size = ivec.size();

    for (int i=0; i<size; i++) {
        result->ivec.at(i) = ivec.at(i) + v2->ivec.at(i);
    }
    return result;
}
```

**Standard Library Exception Hierarchy**

```plaintext
#include<exception>
#include<exception>
#include<stdexcept>
#include<new>
#include<stdexcept>
```