CS302 - Data Structures *using C++*

Topic: Other ArrayBag Methods

Kostas Alexis

Test Core Methods

- **Must implement all interface methods**

Test Core Methods

- **Must implement all interface methods**
  - Stub other methods

```
template<class ItemType>
bool ArrayBag<ItemType>::remove(const ItemType& anEntry)
{
          bool canRemoveItem = false;
          return canRemoveItem;
} // end remove
```

Test Core Methods

- **Must implement all interface methods**
    - Stub other methods

```
template<class ItemType>
bool ArrayBag<ItemType>::remove(const ItemType& anEntry)
{
          bool canRemoveItem = false;
          return canRemoveItem;
} // end remove

template<class ItemType>
void ArrayBag<ItemType>::clear()
{

} // end clear
```

Test Core Methods

- **Must implement all interface methods**
  - Stub other methods

```
template<class ItemType>
bool ArrayBag<ItemType>::getFrequencyOf(const ItemType&
anEntry) const
{
          int frequency = -1;
          return frequency;
} // end getFrequencyOf
```

Test Core Methods

- **Must implement all interface methods**
  - Stub other methods

```
template<class ItemType>
bool ArrayBag<ItemType>::getFrequencyOf(const ItemType&
anEntry) const
{
          int frequency = -1;
          return frequency;
} // end getFrequencyOf

template<class ItemType>
bool ArrayBag<ItemType>::contains(const ItemType& anEntry)
const
{
          return false;
} // end contains
```

Test Core Methods

- **Must implement all interface methods**
  - Stub other methods

```
template<class ItemType>
bool ArrayBag<ItemType>::getFrequencyOf(const ItemType&
anEntry) const
{
          int frequency = -1;
          return frequency;
} // end getFrequencyOf

template<class ItemType>
bool ArrayBag<ItemType>::contains(const ItemType& anEntry)
const
{
          return false;
} // end contains

template<class ItemType>
bool ArrayBag<ItemType>::getIndexOf(const ItemType& target)
const
{
          int result = -1;
          return return;
} // end getIndexOf
```

Test Core Methods

- **Must implement all interface methods**
  - Stub other methods
- **Test Constructor (basic)**

Test Core Methods

- **Must implement all interface methods**
  - Stub other methods
- **Test Constructor (basic)**
  - Create an ArrayBag

```
ArrayBag<std::string> bag;
```

Test Core Methods

- **Must implement all interface methods**
  - Stub other methods
- **Test Constructor (basic)**
  - Create an ArrayBag
  - Validate the bag is empty

```
ArrayBag<std::string> bag;
std::cout << “isEmpty: returns “ << bag.isEmpty()
     << “; should be 1 (true)” << std::endl;
std::cout << “The bag contains “ << bag.getCurrentSize()
     << “items:” << std::endl;

std::vector<std::string> bagItems = bag.toVector();
int numberOfEntries = bagItems.size();
for (int i = 0; i < numberOfEntries; i++)
{
     std::cout << bagItems[i] << “ ”;
} // end for
```

Test Core Methods

- **Must implement all interface methods**
  - Stub other methods
- **Test Constructor (basic)**
  - Create an ArrayBag
  - Validate the bag is empty
- **Add items**

```
ArrayBag<std::string> bag;
std::cout << "isEmpty: returns " << bag.isEmpty()
     << "; should be 1 (true)" << std::endl;
std::cout << "The bag contains " << bag.getCurrentSize()
     << "items:" << std::endl;

std::vector<std::string> bagItems = bag.toVector();
int numberOfEntries = bagItems.size();
for (int i = 0; i < numberOfEntries; i++)
{
     std::cout << bagItems[i] << " ";
} // end for


std::string items[] = {"one", "two", "three", "four", "five",
     "one"};

std::cout << "Add 6 items to the bag: " << std::end;
for (int i = 0; i < 6; i++)
{
     bag.add(items[i]);
} // end for
```

Test Core Methods

- **Must implement all interface methods**
  - Stub other methods
- **Test Constructor (basic)**
  - Create an ArrayBag
  - Validate the bag is empty
- **Add items**
  - Validate that the items are in the bag
- **Fill bag**
  - Validate the bag is full
  - Validate additional adds fail

```
ArrayBag<std::string> bag;
std::cout << "isEmpty: returns " << bag.isEmpty()
     << "; should be 1 (true)" << std::endl;
std::cout << "The bag contains " << bag.getCurrentSize()
     << "items:" << std::endl;

std::vector<std::string> bagItems = bag.toVector();
int numberOfEntries = bagItems.size();
for (int i = 0; i < numberOfEntries; i++)
{
     std::cout << bagItems[i] << " ";
} // end for


std::string items[] = {"one", "two", "three", "four", "five",
     "one"};

std::cout << "Add 6 items to the bag: " << std::end;
for (int i = 0; i < 6; i++)
{
     bag.add(items[i]);
} // end for

std::cout << "The bag contains " << bag.getCurrentSize() << "
     items:" << std::endl;

bagItems = bag.toVector(bagItems);
numberOfEntries = bagItems.size();
for (int i = 0; i <numberOfEntries; i+)
{
     std::cout << bagItems[i] << " ";
} // end for
```

Additional Methods

- **Additional status methods**
  - Status of collection
  - Status of an item

Additional Methods

- **Additional status methods**
  - Status of collection
  - Status of an item

```
template<class ItemType>
int ArrayBag<ItemType>::getFrequencyOf(const ItemType& anEntry) const
{
      int frequency = 0;
      int curIndex = 0; // Current array index
      while (curIndex < itemCount)
      {
            if (items[curIndex] == anEntry)
            {
                  frequency++

            } // end if

            curIndex++; // Increment to next entry
      } // end while

      return frequency;
} // end getFrequencyOf
```

Additional Methods

- **Additional status methods**
    - Status of collection
    - Status of an item

```cpp
template<class ItemType>
int ArrayBag<ItemType>::getFrequencyOf(const ItemType& anEntry) const
{
      int frequency = 0;
      int curIndex = 0; // Current array index
      while (curIndex < itemCount)
      {
            if (items[curIndex] == anEntry)
            {
                  frequency++

            } // end if

            curIndex++; // Increment to next entry
      } // end while

      return frequency;
} // end getFrequencyOf


template<class ItemType>
int ArrayBag<ItemType>::contains(const ItemType& anEntry) const
{
      return getIndexOf(anEntry) > -1;
} //end contains
```

Additional Methods

- **Additional status methods**
  - Status of collection
  - Status of an item
    - Method **`contains`** through the private method **`getIndexOf`**

```
template<class ItemType>
int ArrayBag<ItemType>::getFrequencyOf(const ItemType& anEntry) const
{
      int frequency = 0;
      int curIndex = 0; // Current array index
      while (curIndex < itemCount)
      {
            if (items[curIndex] == anEntry)
            {
                  frequency++

            } // end if

            curIndex++; // Increment to next entry
      } // end while

      return frequency;
} // end getFrequencyOf


template<class ItemType>
int ArrayBag<ItemType>::contains(const ItemType& anEntry) const
{
      return getIndexOf(anEntry) > -1;
} //end contains
```

Additional Methods

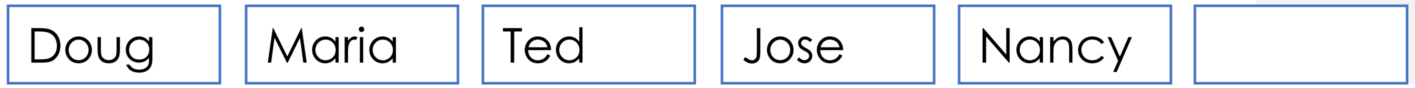
- **Additional status methods**
  - Status of collection
  - Status of an item
    - Method **`contains`** through the private method **`getIndexOf`**

```
// private
template<class ItemType>
int ArrayBag<ItemType>::getIndexOf(const ItemType& target) const
{
      bool found = false;
      int result = -1;
      int searchIndex = 0;
      while (!found && (searchIndex < itemCount))
      {
            if (items[searchIndex] == target)
            {
                  found = true;
                  result = searchIndex;

            }
            else
            {
                  searchIndex++;
            } // end if
      } // end while

      return result;
} // end getIndexOf
```

Additional Methods

- **Additional status methods**
  - Status of collection
  - Status of an item
- **Removing items from the collection**
  - All items
  - A specific item

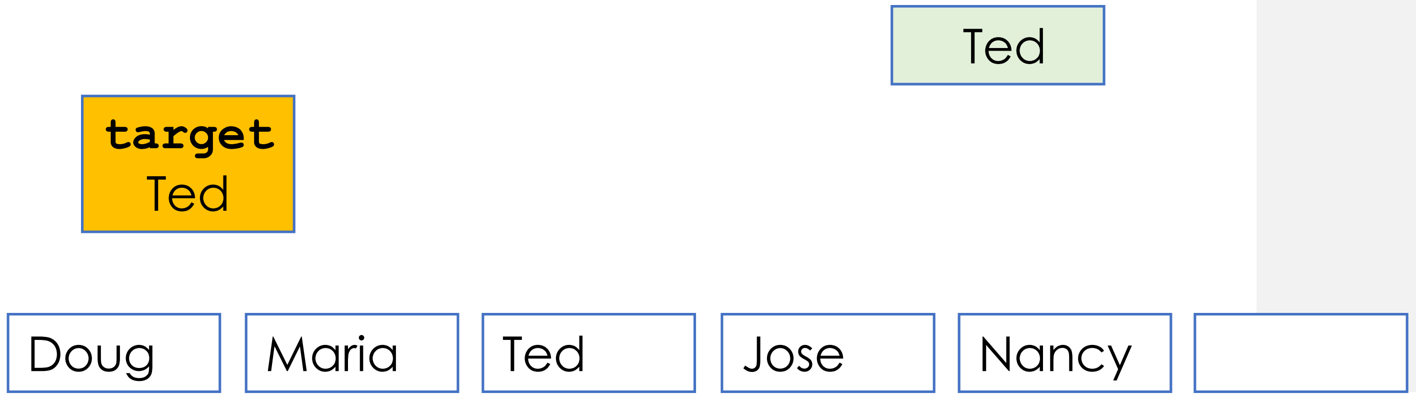
```
// private
template<class ItemType>
bool ArrayBag<ItemType>::remove(const ItemType& target)
{
```



```
// private
template<class ItemType>
bool ArrayBag<ItemType>::remove(const ItemType& target)
{
```

Additional Methods

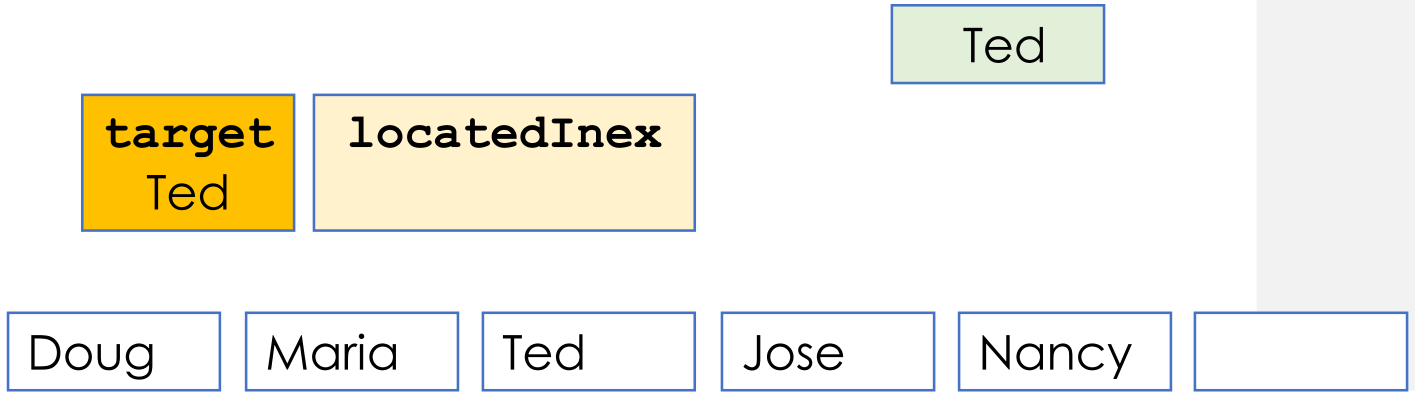
- **Additional status methods**
  - Status of collection
  - Status of an item
- **Removing items from the collection**
  - All items
  - A specific item



```
// private
template<class ItemType>
bool ArrayBag<ItemType>::remove(const ItemType& target)
{
```

Additional Methods

- **Additional status methods**
  - Status of collection
  - Status of an item
- **Removing items from the collection**
  - All items
  - A specific item

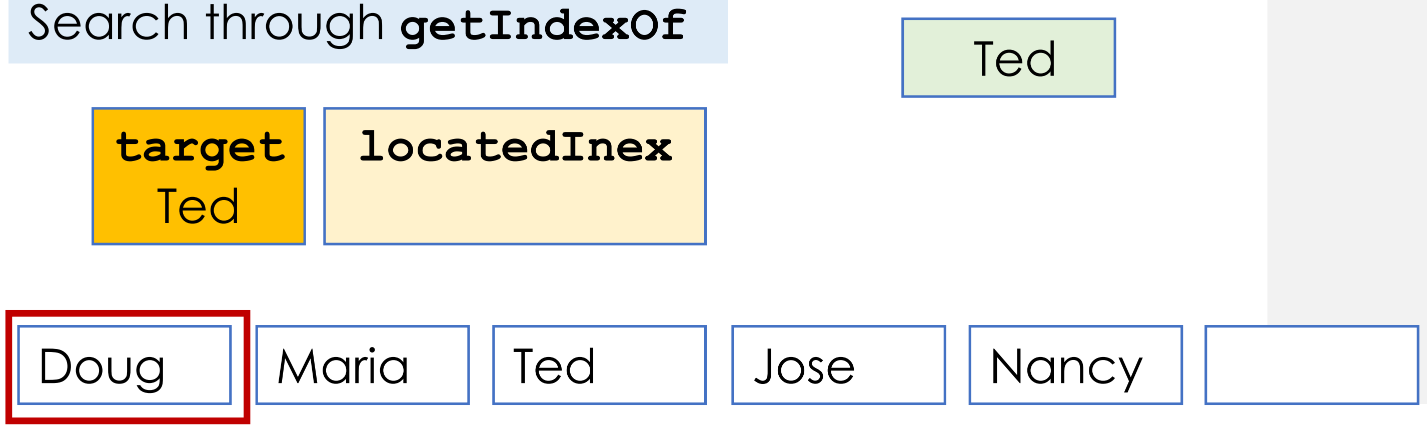


Additional Methods

- **Additional status methods**
  - Status of collection
  - Status of an item
- **Removing items from the collection**
  - All items
  - A specific item

```
// private
template<class ItemType>
bool ArrayBag<ItemType>::remove(const ItemType& target)
{
```

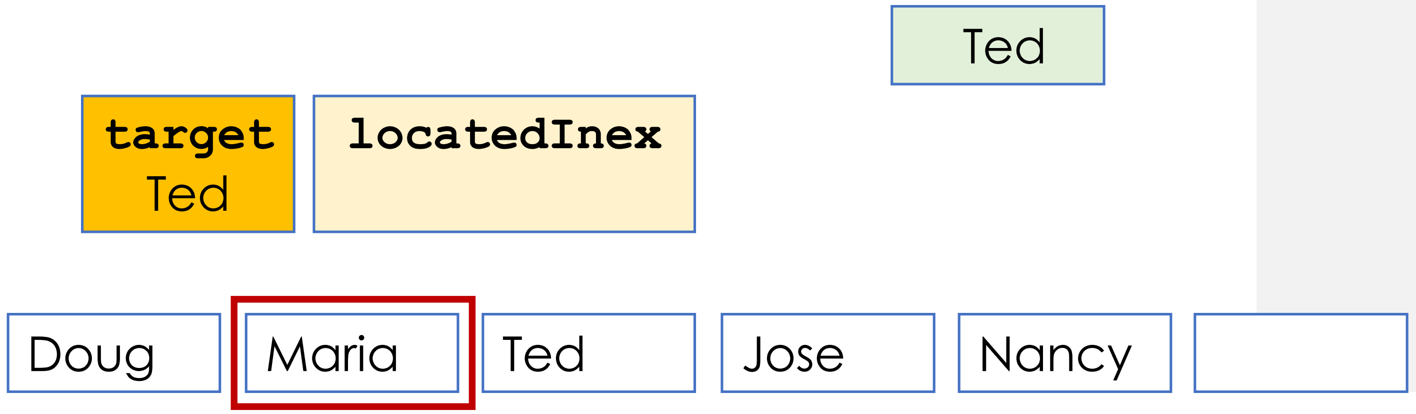
Search through **`getIndexOf`**



Additional Methods

- **Additional status methods**
  - Status of collection
  - Status of an item
- **Removing items from the collection**
  - All items
  - A specific item

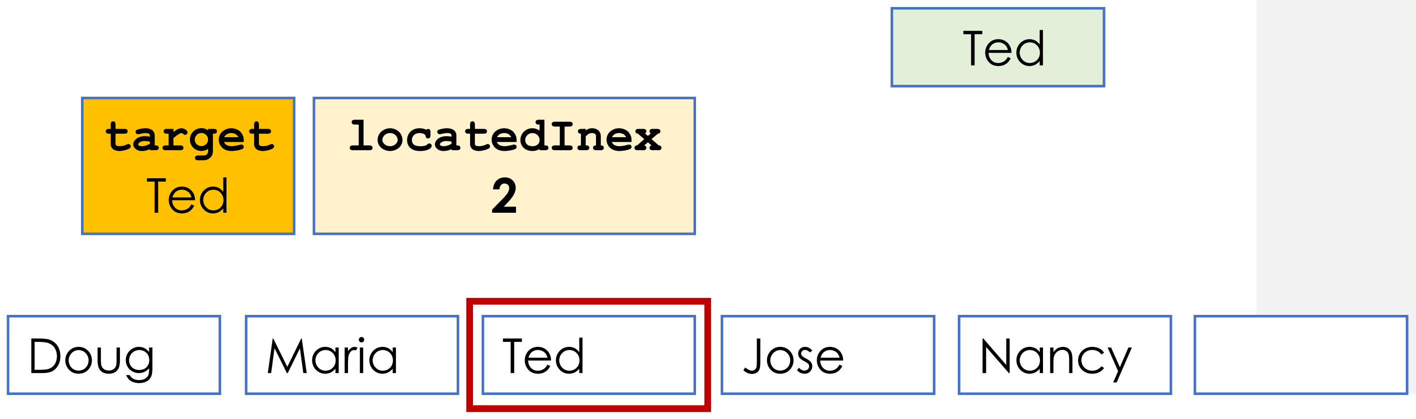
```
// private
template<class ItemType>
bool ArrayBag<ItemType>::remove(const ItemType& target)
{
```



Additional Methods

- **Additional status methods**
  - Status of collection
  - Status of an item
- **Removing items from the collection**
  - All items
  - A specific item

```
// private
template<class ItemType>
bool ArrayBag<ItemType>::remove(const ItemType& target)
{
```

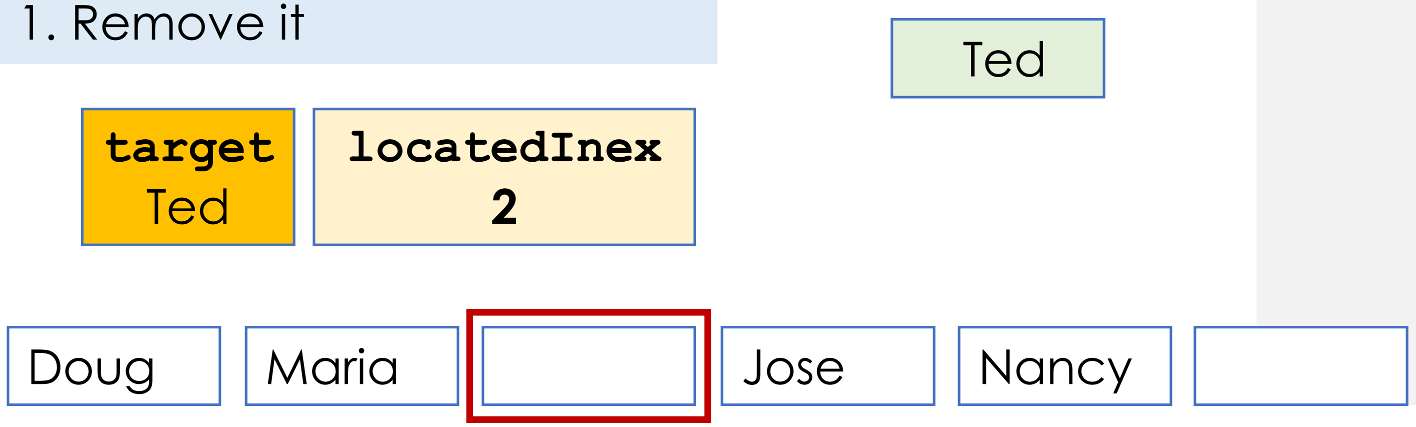


Additional Methods

- **Additional status methods**
  - Status of collection
  - Status of an item
- **Removing items from the collection**
  - All items
  - A specific item

```
// private
template<class ItemType>
bool ArrayBag<ItemType>::remove(const ItemType& target)
{
```

1. Remove it

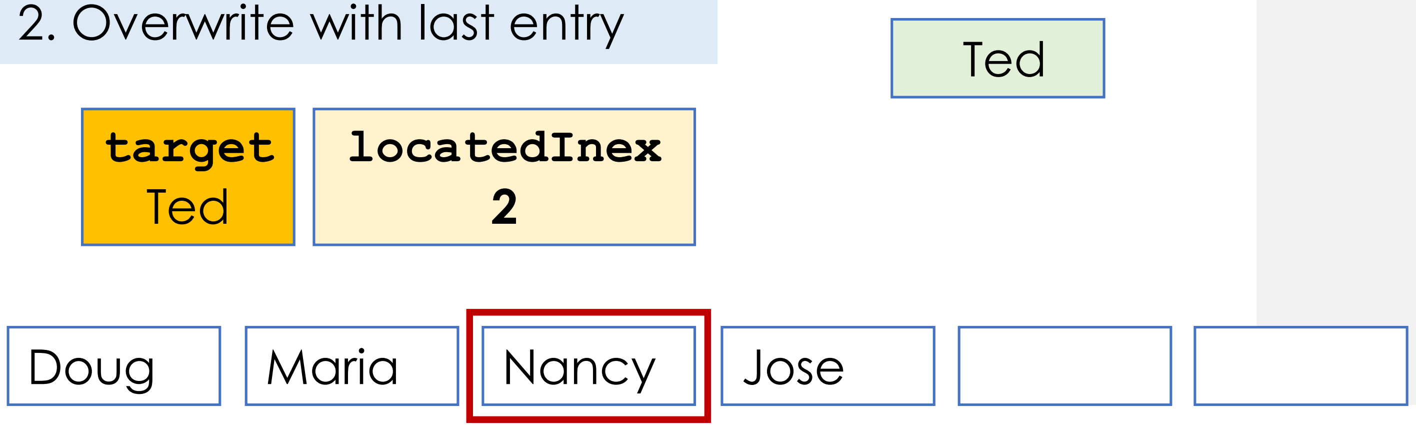


```cpp
// private
template<class ItemType>
bool ArrayBag<ItemType>::remove(const ItemType& target)
{
```

Additional Methods

- **Additional status methods**
  - Status of collection
  - Status of an item
- **Removing items from the collection**
  - All items
  - A specific item

2. Overwrite with last entry

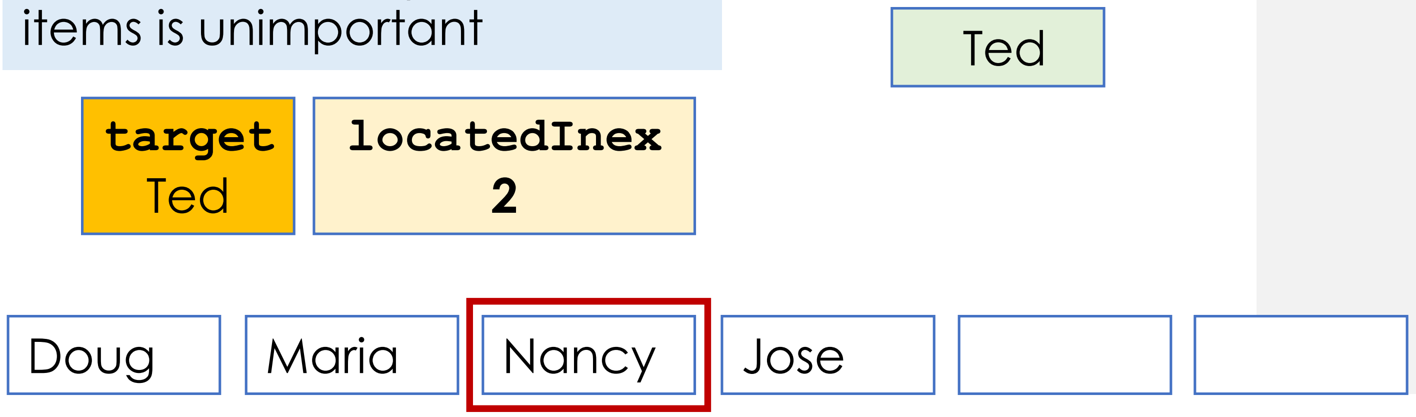


Additional Methods

- **Additional status methods**
  - Status of collection
  - Status of an item
- **Removing items from the collection**
  - All items
  - A specific item

Remember that position of items is unimportant

```
// private
template<class ItemType>
bool ArrayBag<ItemType>::remove(const ItemType& target)
{
```

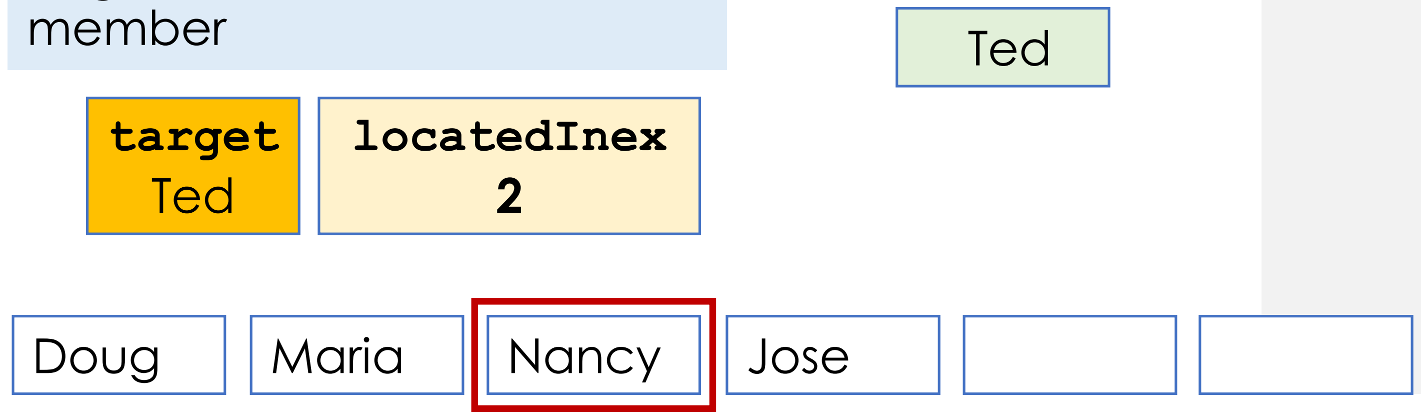


```
// private
template<class ItemType>
bool ArrayBag<ItemType>::remove(const ItemType& target)
{
```

Additional Methods

- **Additional status methods**
  - Status of collection
  - Status of an item
- **Removing items from the collection**
  - All items
  - A specific item

Bag now has one less member

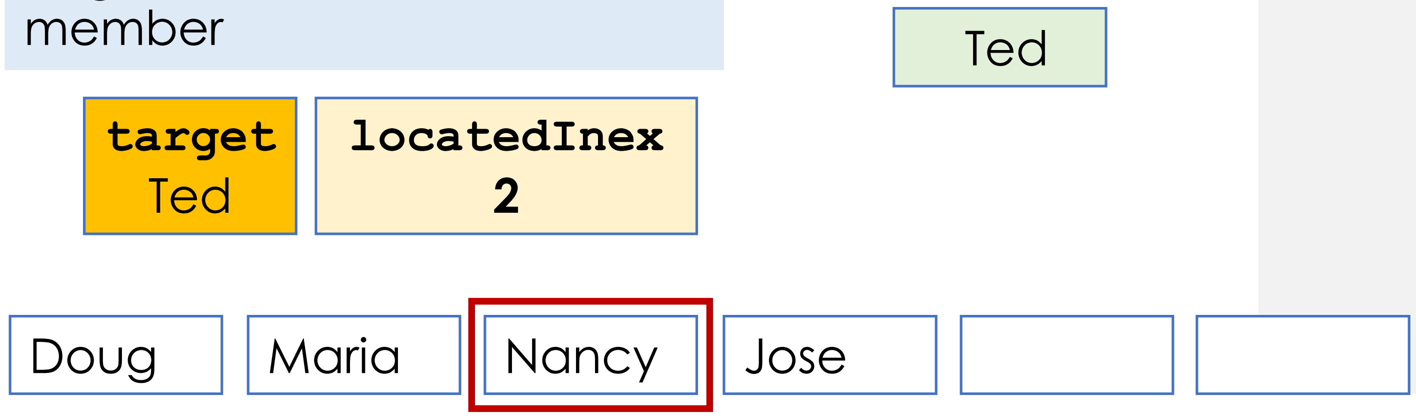


Additional Methods

- **Additional status methods**
  - Status of collection
  - Status of an item
- **Removing items from the collection**
  - All items
  - A specific item

```
// private
template<class ItemType>
bool ArrayBag<ItemType>::remove(const ItemType& target)
{
      int locatedIndex = getIndexOf(target);
      bool canRemoveItem = !isempty() && (locatedIndex > -1);
      if (canRemoveItem)
      {
            items[locatedIndex] = items[itemCount-1];
            itemCount--;
      } // end if

      return canRemoveItem;
} // end remove
```

Bag now has one less member



Thank you