

Templates

Rupesh Nasre.

OOAIA
January 2018

Queue Interface

```
class Queue {  
public:  
    Queue();  
    ~Queue();  
    void insert(int x);  
    int remove();  
  
private:  
    int a[100];  
    int head, tail;  
};
```

Queue Implementation

```
class Queue {  
public:  
    Queue() {  
        head = 0;  
        tail = 0;  
    }  
    ~Queue() { }  
    void insert(int x);  
    int remove();  
  
private:  
    int a[100];  
    int head, tail;  
};
```

Queue Implementation

```
class Queue {  
public:  
    Queue() {  
        head = 0;  
        tail = 0;  
    }  
    ~Queue() {}  
    void insert(int x);  
    int remove();  
  
private:  
    int a[100];  
    int head, tail;  
};  
void Queue::insert(int x) { ← Name resolution  
    // insert code.  
}
```

This allows us to separate interface from its implementation.

Queue Implementation

```
class Queue {  
public:  
    Queue() {  
        head = 0;  
        tail = 0;  
    }  
    ~Queue() {}  
    void insert(int x);  
    int remove();  
  
private:  
    int a[100];  
    int head, tail;  
};  
void Queue::insert(int x) {  
    // insert code.  
}  
int Queue::remove() {  
    // ...  
}
```

Can we do anything about the dependence on `int`?

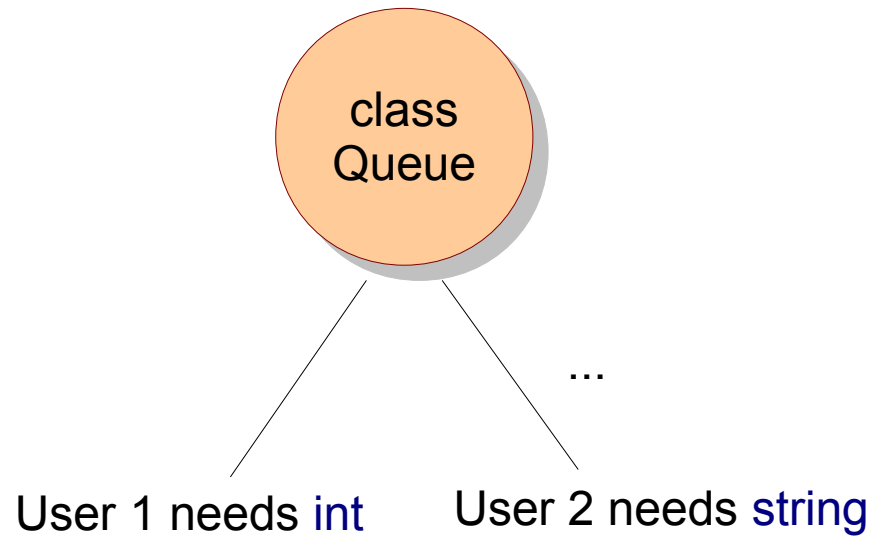
Type Generality

```
#define TYPE int
class Queue {
public:
    Queue() {
        head = 0;
        tail = 0;
    }
    ~Queue() {}
    void insert(TYPE x);
    TYPE remove();

private:
    TYPE a[100];
    int head, tail;
};
void Queue::insert(TYPE x) {
    // insert code.
}
TYPE Queue::remove() {
    // ...
}
```

I need to change the interface for different users.

Type Generality



Type Generality

Tricks / Hacking

```
#define TYPE int  
#include "queue"
```

```
void main() {  
    Queue q;  
    q.insert(10);  
    ...  
}
```

User 1

```
#define TYPE string  
#include "queue"
```

```
void main() {  
    Queue q;  
    q.insert("cs24");  
    ...  
}
```

User 2

User also needs to know which variable to define (TYPE).

Type Generality

```
#include "queue"
```

```
void main() {  
    Queue<int> q;  
    q.insert(10);  
    ...  
}
```

User 1

```
#include "queue"
```

```
void main() {  
    Queue<string> q;  
    q.insert("cs24");  
    ...  
}
```

User 2

Templates

```
template <class TYPE>
```

I need NOT change the interface for different users.

```
class Queue {  
public:  
    Queue() {  
        head = 0;  
        tail = 0;  
    }  
    ~Queue() {}  
    void insert(TYPE x);  
    TYPE remove();  
};
```

```
private:  
    TYPE a[100];  
    int head, tail;
```

```
};  
void Queue::insert(TYPE x) {  
    // insert code.}
```

These don't compile.

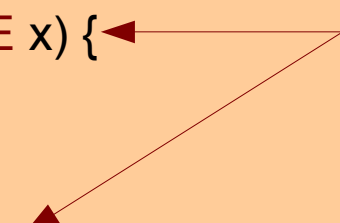
```
TYPE Queue::remove() {  
    // ...  
}
```

Templates

```
template <class TYPE>
class Queue {
public:
    Queue() {
        head = 0;
        tail = 0;
    }
    ~Queue() { }
    void insert(TYPE x);
    TYPE remove();

private:
    TYPE a[100];
    int head, tail;
};
template <class TYPE>
void Queue::insert(TYPE x) {
    // insert code.
}
template <class TYPE>
TYPE Queue::remove() {
    // ...
}
}
```

Still don't compile.



Templates

```
template <class TYPE>
class Queue {
public:
    Queue() {
        head = 0;
        tail = 0;
    }
    ~Queue() { }
    void insert(TYPE x);
    TYPE remove();

private:
    TYPE a[100];
    int head, tail;
};
template <class TYPE>
void Queue<TYPE>::insert(TYPE x) {
    // insert code.
}
template <class TYPE>
TYPE Queue<TYPE>::remove() {
    // ...
}
```

Compiles successfully.

Classwork

- Create a class Group templated with the type of elements to be stored in the group.
- Implement methods: add and find.
- Instantiate integer Group and check add+find.
- Instantiate string Group and check add+find.

```

#include <iostream>
#include <vector>
#include <algorithm>

template<typename T>
class Group {
public:
    void add(T element);
    bool find(T element);
    T findwrapper(T element);

private:
    std::vector<T> group;
};
...

```

```

template<typename T>
void Group<T>::add(T element) {
    group.push_back(element);
}

template<typename T>
bool Group<T>::find(T e) {
    std::find(group.begin(), group.end(), e)
    != group.end();
}

template<typename T>
T Group<T>::findwrapper(T e) {
    std::cout <<
        (find(e) ? "Found " : "Not found ");
    return e;
}

```

```
int main() {  
    Group<int> group;  
  
    group.add(5);  
    group.add(6);  
    group.add(8);  
    group.add(5);  
    std::cout << group.findwrapper(5) << std::endl;  
    std::cout << group.findwrapper(2) << std::endl;  
    std::cout << group.findwrapper(6) << std::endl;  
  
    Group<std::string> groupstr;  
    groupstr.add("one");  
    groupstr.add("two");  
    groupstr.add("three");  
    groupstr.add("five");  
  
    std::cout << groupstr.findwrapper("two") << std::endl;  
    std::cout << groupstr.findwrapper("four") << std::endl;  
    std::cout << groupstr.findwrapper("five") << std::endl;  
  
    return 0;  
}
```