```
/* HEADER FILE - ARRAY IMPLEMENTATION OF A STACK */

#include "underflowoverflow.h"

template <class Object>
class Stack
{
  public:

    // Constructor for a stack of a given capacity

    Stack( int capacity = 10 );

    // Push x on the stack - error if capacity exceeded

    void push( const Object & x ) throw(Overflow);

    // Pop top item from stack - error if empty

    void pop( ) throw(Underflow);

    // Access top item on stack - error if empty

    const Object & top( ) const throw(Underflow);
    // Remove and return top item - error if empty

    Object topAndPop( ) throw(Underflow);

    // Test to see whether stack is empty

    bool empty( ) const;

    // Test to see whether stack is full (at capacity)

    bool full( ) const;

    // Make the stack empty, discarding all items now on it

    void makeEmpty( );

  private:
    Object * theArray;
    int topOfStack;
};
```

```
/* IMPLEMENTATION FILE - ARRAY IMPLEMENTATION OF A STACK */

#include "stacka.h"

template <class Object>
Stack<Object>::Stack( int capacity )
{
    theArray = new Object[capacity];
    topOfStack = -1;
}
```

```
template <class Object>
void Stack<Object>::push( const Object & x ) throw(Overflow)
{
    if( full( ) )
        throw Overflow( );
        // Alternately, could resize theArray vector
    theArray[ ++topOfStack ] = x;
}
```

```
template <class Object>
void Stack<Object>::pop( ) throw(Underflow)
{
    if( empty( ) )
        throw Underflow( );
    topOfStack--;
}
```

```
template <class Object>
const Object & Stack<Object>::top( ) const throw(Underflow)
{
    if( empty( ) )
        throw Underflow( );
    return theArray[ topOfStack ];
}
```

```
template <class Object>
Object Stack<Object>::topAndPop( ) throw(Underflow)
{
    if( empty( ) )
        throw Underflow( );
    return theArray[ topOfStack-- ];
}
```

```
template <class Object>
bool Stack<Object>::empty( ) const
{
    return topOfStack == -1;
}
```

```
template <class Object>
bool Stack<Object>::full( ) const
{
    return topOfStack == theArray.size( ) - 1;
}
```

```
template <class Object>
void Stack<Object>::makeEmpty( )
{
    topOfStack = -1;
}
```