CS 617 Object Oriented Systems
Lecture 4
ADTs, Contracts and The Design by Contract Method
3:30-5:00pm Mon, Jan 14

Rushikesh K Joshi

Department of Computer Science and Engineering
Indian Institute of Technology Bombay

Outline

Outline

From ADTs to classes: Through a Familiar Example

- Identify Constructors
- Moving to Imperative Version (from applicative specification)
  Receiver is modified and not returned

Converting Stack ADT to interface of class Stack

Stack new (void) →constructor *Stack()*
Stack push (E,Stack) → *push (E)*
E top(Stack) → *E(top)*
Stack removetop(Stack) → *removetop()*
Boolean empty (Stack) → *Boolean empty()*

Thus we get:

```
interface Stack {
    Stack();
    push (E);
    E top();
    removetop();
    Boolean Empty();
}
```

Interfaces, Deferred Classes (Abstract Classes) and Concrete Classes

- Interfaces: Only the interface functions, cannot be instantiated
- Deferred Classes: Partial implementation, cannot be instantiated
- Concrete Classes: Fully implemented, can be instantiated

Our ADT Example

Types:

E is the element type and T be Stack type.

Functions:

T new (void)
T push (E,T)
E top(T)
T removetop(T)
Boolean empty (T)

Axioms:

empty(new())
top(push(e,t)) = e
removetop(push(e,t)) = t
not empty(push(e,t))

Preconditions:

.. removetop (T) requires not empty (T)
.. pop (T) requires not empty (T)

An Abstract Class Specification, Extracting Postconditions

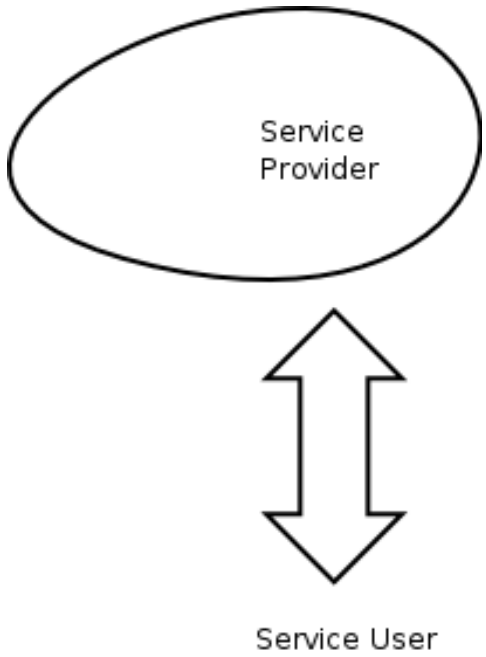
```
class UnboundedStack {
    Stack();
                precondition: none
                postcondition: stack is empty
    push (E e);
                precondition: none
                postcondition: (1) top() is e (2) stack not empty
    E top();
                precondition: stack not empty
                postcondition: no change to stack
    removetop();
                precondition: stack not empty
                postcondition: stack has one element less
    Boolean isEmpty();
}
```

Outline

Contracts

What are contracts?
How to specify them?
How to use contracts in OO Software Development?

Contracts



- involved between collaborating parties
- caller-callee systems: service user and service provider

Outline

The Design by Contract Method of Meyer

	Obligations	Benefits
Service User	fulfil preconditions of partial functions	postconditions are ensured
Service Provider	ensure postconditions	clean business logic assume preconditions are satisfied

Contract for Member Function Push

	Obligations	Benefits
Service User	donot call push on a full stack	get stack updated, stack will not be empty, element on top, do not check whether push failed
Service Provider	update stack such that element is on top, update status	do not check whether stack is full, simply make the updates

Contract for Member Function Pop (tmp=top, removetop,return tmp)

	Obligations	Benefits
Service User	donot call pop on an empty stack	get stack updated, element on top is returned, do not check whether pop failed
Service Provider	update stack such that element is top is returned, update status	do not check whether stack is empty, simply focus on the business logic

Specifications in Eiffel following Design by Contract I

```
class Stack [E]
    ..
    top:E is
            .. top element
    require
            not empty
    do
            ...
    end
    push(e:E) is
            .. add e on top
    require
            not full
```

Specifications in Eiffel following Design by Contract II

```
do
        ...
ensure
        not empty
        top=e
        size=old size+1
end
removetop is
        .. removes top element
require
        not empty
do
        ...
ensure
```

Specifications in Eiffel following Design by Contract III

```
            not full
            size=old size - 1
    end
end
```

Preconditions, Postconditions and Invariants

class invariant: a predicate of which the value is true over the entire lifetime of the object

member function precondition: should be satisfied before the execution of the member function

member function postcondition: should be satisfied after the execution of the member function

No Redundancy in Implementation

The actual function bodies do not check for preconditions.

Also postconditions are not checked by callers.

What happens when a contract is violated

An error can be generated, an exception can be thrown.

Outline

Using Assertions

The assert macro in C, C++, assertion support in Java
Start with assertions

Add implementations later

1. Design first, then implement
2. Protect implementation against bugs and errors
3. Buggy implementations are caught by the contracts, assertions

Preconditions, Postconditions and Inheritance

What happens to them in inheritance?
Can subclasses change them?

Readings

Bertrand Meyer: Applying Design by Contract, IEEE Computer, October 1992, pages 40-51.