# **CMPT 280**

Tutorial: Specifications

CMPT 280 1/11

# Stacks: A Refresher

- A "last-in, first out" (LIFO) container
- Three fundamental operations:
  - Put new item on top of stack
  - Look at item at top of stack
  - Remove the item on the top of the stack.
- Stacks are also a type of *dispenser* (defined in Chapter 10 of the textbook).

CMPT 280 2/11

What does a stack actually need?

### Methods

- newStack
- Push
- Pop
- Top
- isEmpty
- isFull

### Sets

- set of all stacks
- set of items that can be in a stack
- booleans: {true, false}

CMPT 280 3/11

#### Name: Stack< G >

#### Sets:

S: set of all stacks containing elements from G

G: set of items that can be in the stack S B: {true, false}

### **Signatures**

 $\begin{array}{l} \mathsf{newStack} {<} G {>} : \to S \\ S.\mathsf{isEmpty} : \to B \\ S.\mathsf{isFull} : \to B \\ S.\mathsf{push}(g) \colon G \not\to S \end{array}$ 

S.pop:  $\rightarrow$  S S.top:  $\rightarrow$  G **Preconditions:**  $\forall s \in S, g \in G$ 

newStack < G > : nor

s.isFull: none

 $s.\mathsf{push}:s$  is not full  $s.\mathsf{pop}:s$  is not empt

 $s.\mathsf{top}: s \text{ is not empty}$ 

### **Semantics:** $\forall s \in S, g \in G$

 $\operatorname{ewStack} < G >$ : Construct a new stack

that can store elements of G

s.isEmpty: return  ${f true}$  if s is empty,  ${f false}$ 

s.isFull: return **true** if s is full, **false** otherwise

 $s.\mathsf{push}(g)$ :  $\mathsf{push}(g)$  onto top of stack  $s.\mathsf{pop}$  remove top element g from stack  $s.\mathsf{top}$ : fetch top element g from stack

4/11

Name: Stack<G>

#### Sets:

S: set of all stacks containing elements from G

G: set of items that can be in the stack S B: {true, false}

### Signatures:

 $\begin{array}{l} \mathsf{newStack}{<}G{>}{:} \to S \\ S.\mathsf{isEmpty}{:} \to B \\ S.\mathsf{isFull}{:} \to B \\ S.\mathsf{push}(g){:} G \not\to S \end{array}$ 

 $S.\mathsf{pop}$ :  $\not\to S$  $S.\mathsf{top}$ :  $\not\to G$  **Preconditions:**  $\forall s \in S, g \in G$ 

newStack < G > : none

s.isEmpty: noi

 $s.\mathsf{push}: s$  is not full  $s.\mathsf{pop}: s$  is not empt

 $s.\mathsf{top}: s$  is not empty

**Semantics:**  $\forall s \in S, g \in G$ 

 $\operatorname{newStack} < G >$ : Construct a new stack

that can store elements of G

s.isEmpty: return  ${f true}$  if s is empty,  ${f false}$  otherwise

s.isFull: return true if s is full, false otherwise

 $s.\mathsf{push}(g)$ : push g onto top of stack  $s.\mathsf{pop}$  remove top element g from stack  $s.\mathsf{top}$ :

fetch top element q from stack

Name: Stack< G >

#### Sets:

S: set of all stacks containing elements from G

 $G\!\colon$  set of items that can be in the stack S

B: {true, false}

### Signatures:

 $\begin{array}{l} \mathsf{newStack}{<}G{>}: \to S \\ S.\mathsf{isEmpty}: \to B \\ S.\mathsf{isFull}: \to B \\ S.\mathsf{push}(g): G \not\to S \end{array}$ 

S.pop:  $\not\rightarrow S$ S.top:  $\not\rightarrow G$  **Preconditions:**  $\forall s \in S, g \in G$ 

 $\begin{array}{l} \mathsf{newStack} {<} G {>} : \mathsf{none} \\ s.\mathsf{isEmpty} : \mathsf{none} \\ s.\mathsf{isFull} : \mathsf{none} \end{array}$ 

s.push : s is not full s.pop : s is not empty s.top : s is not empty

**Semantics:**  $\forall s \in S, g \in G$ 

 $\mathsf{newStack} < G > : \mathsf{Construct} \ \mathsf{a} \ \mathsf{new} \ \mathsf{stack}$ 

that can store elements of G

s.isEmpty: return  ${f true}$  if s is empty,  ${f false}$  otherwise

s.isFull: return true if s is full, false otherwise

 $s.\mathsf{push}(g)$ :  $\mathsf{push}(g)$  onto top of stack  $s.\mathsf{pop}$ : remove top element g from stack  $s.\mathsf{top}$ :

CMPT 280 4/11

Name: Stack< G >

#### Sets:

S: set of all stacks containing elements from G

G: set of items that can be in the stack S B: {true, false}

### Signatures:

newStack< $G>: <math>\rightarrow S$ S.isEmpty:  $\rightarrow B$ S.isFull:  $\rightarrow B$ S.push $(q): G \not\rightarrow S$ 

 $S.\mathsf{pop}$ :  $\not\to S$  $S.\mathsf{top}$ :  $\not\to G$  Preconditions:  $\forall s \in S, g \in G$ 

newStack < G > : none

s.isEmpty: none s.isFull: none s.push: s is not full

 $s.\mathsf{pop}: s$  is not empty  $s.\mathsf{top}: s$  is not empty

**Semantics:**  $\forall s \in S, g \in G$ 

newStack < G > : Construct a new stack

that can store elements of  ${\cal G}$ 

s.isEmpty: return **true** if s is empty, **false** otherwise

s.isFull: return **true** if s is full, **false** otherwise

 $s.\mathsf{push}(g)$ :  $\mathsf{push}\ g$  onto top of stack  $s.\mathsf{pop}$ : remove top element g from stack  $s.\mathsf{top}$ :

fetch top element g from stack

### **Implementation**

- Specifications done in this way can be translated into any language.
- We only happen to be using Java.

CMPT 280 5/11

# Implementation of Signatures

Signatures translate to method headers.

- newStack<G $> <math>\rightarrow S$
- $S.\mathsf{isEmpty} \to B$
- $S.\mathsf{isFull} \to B$
- $S.\mathsf{push}(g) \ G \not\to S$
- $S.pop \not\rightarrow S$
- $S.\mathsf{top} \not\to G$

```
public class Stack<G> {

public Stack() {}

public boolean isEmpty() {}

public boolean isFull() {}

public void push(G g) {}

public void pop() {}

public d top() {}

public G top() {}
}
```

### Implementation of Signatures

Signatures translate to method headers.

- newStack<G $> <math>\rightarrow S$
- $S.\mathsf{isEmpty} \to B$
- $S.\mathsf{isFull} \to B$
- $S.\mathsf{push}(g) \ G \not\to S$
- $S.pop \not\rightarrow S$
- $S.\mathsf{top} \not\to G$

```
public class Stack<G> {

public Stack() {}

public boolean isEmpty() {}

public boolean isFull() {}

public void push(G g) {}

public void pop() {}

public G top() {}

public G top() {}
```

### Implementation of Preconditions

Preconditions are javadocs, exceptions, and if-statements.

- newStack < G > : none
- s.isEmpty : none
- s.isFull : none

- ullet  $s.\mathsf{push}$  : s is not full
- ullet  $s.\mathsf{pop}$  : s is not empty
- s.top: s is not empty

CMPT 280 7/11

### Implementation of Preconditions

Preconditions are javadocs, exceptions, and if-statements.

- newStack < G > : none
- s.isEmpty : none
- s.isFull : none

- ullet  $s.\mathsf{push}$  : s is not full
- ullet  $s.\mathsf{pop}$  : s is not empty
- s.top: s is not empty

```
/** @precond the stack is not full */
    public void push(G g)
      throws IllegalStateException {
        if (this.isFull()) {
             throw new
             IllegalStateException();
    7
    /** @precond the stack is not empty */
10
    public void pop()
11
      throws IllegalStateException {
12
        if (this.isEmpty()) {
13
             throw new
14
             IllegalStateException();
15
    }
16
17
    /** @precond the stack is not empty */
18
    public G top()
      throws IllegalStateException {
19
        if (this.isEmpty()) {
20
21
             throw new
             IllegalStateException():
23
```

CMPT 280 7/11

Semantics become the javadoc comments (and later, code).

- newStack<G>: Construct a new stack to hold  $g \in G$
- s.isEmpty: return true if ⇒
   s is empty, false
   otherwise
- s.isFull: return **true** if s is full, **false** otherwise

CMPT 280 8/11

Semantics become the javadoc comments (and later, code).

- newStack<G>: Construct a new stack to hold  $q \in G$
- s.isEmpty: return true if ⇒
   s is empty, false
   otherwise
- s.isFull: return **true** if s is full, **false** otherwise

```
public class Stack < G > {
          * Create a new Stack
         public Stack() {}
            Tests whether the stack is empty
10
            Oreturns true if the stack is empty,
               false otherwise
11
12
13
         public boolean isEmpty() {}
14
15
16
          * Tests whether the stack is full
            Oreturns true if the stack is full,
17
18
                 false otherwise
19
20
         public boolean isFull() {}
21
22
       [...]
```

CMPT 280 8/11

- s.push(g): push g onto top of stack
- ullet s.pop: remove top element g from stack

- s.push(g): push g onto top of stack
- ullet s.pop: remove top element g from stack

```
[...]
 3
 4
          * Pushes element g onto
 5
               the top of the stack
 6
          * Oprecond the stack is not full
 7
          */
 8
         public void push(G g)
 9
           throws IllegalStateException {
10
             if (this.isFull()) {
11
                  throw new
12
                  IllegalStateException();
13
         }
14
15
16
          * Removes the top element
17
               from the stack
18
          * Oprecond the stack is not empty
19
20
         public void pop()
21
           throws IllegalStateException {
22
             if (this.isEmpty()) {
23
                  throw new
24
                  IllegalStateException();
25
26
27
       [...]
```

ullet  $s. {
m top:}$  fetch top element g from stack

```
1
2
2
3
4**
4  * Returns the top element
5  * from the stack
6  * @precond the stack is not empty
7  * @return the top element of the stack
8  */
9  public G top()
10  throws IllegalStateException {
11  if (this.isEmpty()) {
12  throw new
13  IllegalStateException();
14  }
15
16
17 }
```

 $\begin{tabular}{ll} \bullet & s. {\tt top:} & {\tt fetch top element} \\ g & {\tt from stack} \end{tabular}$ 

```
[...]
 3
          * Returns the top element
              from the stack
          * Oprecond the stack is not empty
          * Oreturn the top element of the stack
         public G top()
10
           throws IllegalStateException {
11
             if (this.isEmpty()) {
12
                 throw new
13
                 IllegalStateException();
14
15
16
17
```

### Implementation - the rest

- Add the data structures and method implementations
- Don't forget to actually check your preconditions
- ADTs can be implemented in any language, using any implementation

CMPT 280 11/11