An Introduction to Z and Object-Z

Acknowledgement
These slides are mainly based on the book: *The Object-Z Specification Language, by Graeme Smith*, Software Verification Research Centre, University of Queensland.

Source: The Object-Z Specification Language, Graeme Smith, Software Verification Research Centre, University of Queensland
Z Notation
• A mathematical language of logic, sets, and relations
• A schema language of patterns and objects
• A theory of refinement between abstract data types.

Source: Using Z by Jim Davies & Jim Woodcock
Description

• We can use Z to:
  – Describe data structures
  – Model system state
  – Formalize properties

Source: Using Z by Jim Davies & Jim Woodcock
Reasoning

• We can use Z to:
  – Explain design intentions
  – Verify development steps
  – Compare descriptions at different levels of abstraction.

Source: Using Z by Jim Davies & Jim Woodcock
The method

• The Z notation is used to model systems in terms of state: we
  – describe the state of the system, and explain the relationship
  – between this and the state of various components.

Source: Using Z by Jim Davies & Jim Woodcock
Stack of Item

Stack[Item]

items: seq Item
top: Item

top = last(items)

StackInit[Item]

Stack[Item]

items = <>

Push[Item]

Δ Stack[Item]

item?: Item

items' = items ∪ <item?>

Pop[Item]

Δ Stack

item!: Item

items ≠ <>

items = items' ∩ <item!>

Top[Item]

Ξ Stack

top!: Item

top! = top

Item is the formal generic parameter

Source: The Object-Z Specification Language, Graeme Smith, Software Verification Research Centre University of Queensland, with modification
Queue of Item

\[
\text{Queue[Item]} = \triangleleft \\
\text{items: seq Item} \\
\text{front: Item} \\
\text{front} = \text{first}(\text{items})
\]

\[
\text{QueueInit[Item]} = \\
\text{Queue[Item]} \\
\text{items} = \langle \rangle
\]

\[
\text{Join[Item]} = \\
\Delta \text{Queue[Item]} \\
\text{item? : Item} \\
\text{items}' = \text{items} \uparrow \text{item}?
\]

\[
\text{Leave[Item]} = \\
\Delta \text{Queue[Item]} \\
\text{item! : Item} \\
\text{items} \neq \langle \rangle \\
\text{items} = \langle \text{item!} \rangle \uparrow \text{items}'
\]

\[
\text{Front[Item]} = \\
\exists \text{Queue} \\
\text{front! : Item} \\
\text{front!} = \text{front}
\]

Source: The Object-Z Specification Language, Graeme Smith, Software Verification Research Centre University of Queensland, with modification
Stack of Item (constraint on length)

Stack[Item]
items: seq Item
top: Item

#items ≤ 100
top = last(items)

StackInit[Item]
Stack[Item]
items = <>

Push[Item]
Δ Stack
item? : Item

#items < 100
items' = items ~ <item?>

Pop[Item]
Δ Stack
item! : Item

items ≠ <>
items = items' ~ <item!>

Top[Item]
Ξ Stack
top! : Item

top! ≠ top

Source: The Object-Z Specification Language, Graeme Smith, Software Verification Research Centre  University of Queensland, with modification
Queue of Item (counting the elements)

```
Queue[Item] ----
| items: seq Item
| front: Item
| count : N

front = first(items)
```

```
QueueInit[Item] ----
| Queue[Item]
| items = <>
| count = 0
```

Variable `count` records the total number of items which have ever joined the queue.

```
Join[Item] ----
| Δ Queue[Item]
| item ? : Item

items' = items ∪ item?
count' = count + 1
```

```
Leave[Item] ----
| Δ Queue[Item]
| item ! : Item

items ≠ <>
items = <item !> ∩ items'
count' = count - 1
```

```
Front[Item] ----
| Ξ Queue
| front !: Item

front ! = front
```

Source: The Object-Z Specification Language, Graeme Smith, Software Verification Research Centre  University of Queensland, with modification
Object Z
Queue of Items Class

Queue[Item]

items: seq Item
front: Item

front = first(items)

Init
items = <>

Joint

Δ(items)
item : Item
items' = items ~ <item>

Leave

Δ(items)
item : Item
items ≠ <>
items = <item> ~ items'

Front

Δ()
front : Item
front ! = front

Source: The Object-Z Specification Language, Graeme Smith, Software Verification Research Centre University of Queensland, with modification
Instantiation

Stack of Nat

Stack of Nat =

    Stack[N][nat?/item?; nat!/item!]
Queue[Item] with a visibility list

Source: The Object-Z Specification Language, Graeme Smith, Software Verification Research Centre, University of Queensland, with modification
Object-Z classes

- **Generic classes** with formal generic parameters
- **Formal generic parameters** represent types of items
- The scope of the generic parameter is the entire class (unlike in Z, no need to introduce the formal generic parameter to each schema definition)
- The visibility list of a class explicitly lists constants, state variables, initial state schema and operations which are “visible” to the environment of objects of the class. (the absence of a visibility list implies all features are visible)
- The state schema (and the state schema in primed form) is implicitly included in any operation
- The **Δ-list** is a list of state variables which may be changed by the operation (all state variables not in the Δ-list remain unchanged)
Inheritance

- **BoundedQueue[Item]** inherits from **Queue** *(maintaining the same interface)*
- **BoundedQueue[Item]** is a specialization or extension of **Queue**
- **BoundedQueue[Item]** is the inheriting class
  - The inheriting class (also called **subclass**), implicitly includes all of the features of the inherited class, and may also modify and add to these features
- The inherited class **Queue** is also called a superclass
Bounded Queue With Visibility list (specified without using inheritance)

```
BoundedQueue[Item]  
\((\text{count}, \text{INIT, Join, Leave})\)

max: \(N\)

items: seq Item  
count: \(N\)

\#items \leq \text{max}

\text{Init}
items = <>  
count = 0

\text{Join}
\Delta(\text{items, count })
item ? : Item

\#items < max  
items' = items \concat <\text{item}?>  
count' = count + 1

\text{Leave}
\Delta(\text{items })
item ! : Item

items \neq <>  
items = <\text{item}!> \concat \text{items'}  
count' = count - 1
```

Source: The Object-Z Specification Language, Graeme Smith, Software Verification Research Centre, University of Queensland, with modification