Abstract Data Types Documentation

Documentation

- Users are only interested in the properties of the ADT
- Programmers and designers require all the information which a user needs AND all information pertaining to the design and implementation
- Useful to think of the documentation as being an annotated definition of an abstract data type

Documentation Table of Contents

- Cover page, table of contents and abstract
- Document introduction
  - » **Informal overview of the facilities provided. Help readers determine if this is what they need**
- Data type objects
  - » **Description of all the objects – include diagrams**
  - » **Split into**
    - > **Imported – which predefined objects are used**
    - > **Exported – for others to use**
    - > **Hidden – used in the implementation**

Operations TOC – 2

- Operations
  - **Give**
    - **Signature**
    - **Informal description**
    - **pre- and post- conditions**
  - **Use natural language, mathematics, diagrams – whatever best gets the meaning across.**
  - **Be simple, complete, clear, precise, concise as possible**

Operations TOC – 3

- Example – partial axiomatic description of bank accounts
  - » **The operation signatures only – no pre- post- given**

`new        : []  →  account`

– Create an account with a zero balance

`withdraw : account X amount  →  account`

– Remove amount from account

`deposit  : account X amount  →  account`

– Add amount to account

`balance  : account  →  amount`

– What is the amount in the account?

Operations TOC – 4

- Operation interaction
  - **Previous section describes operations in isolation**
  - **Provide better understanding by showing properties when operations are used in combination**
  - **Common descriptive method in use is axiomatic**
    - **List of axioms or statements which must be true if the ADT is implemented and used correctly**

Operations TOC – 5

- Axioms about the data type
  - **Axiom 1: New account has a balance of zero dollars**

    balance(new) = 0

  - **Axiom 2: Cannot withdraw from a new account**

    withdraw(new, amt) = error

  - **Axiom 3: Deposit amt and then withdraw amt with no intervening operations the balance does not change**

    balance(withdraw(deposit(acct, amt) , amt)
    = balance(acct)

  - **Axiom 4: Only withdraw if the balance is ≥ the amount to withdraw. The amount is deducted from the balance**

    balance(acct) < amt → withdraw(acct, amt) = error

    balance(acct) ≥ amt →
    balance(withdraw(acct, amt)) = balance(acct) – amt

TOC – 6

- How to use the ADT
  - » **Tutorial guide on use. Dwell on nuances. Describe various examples**
- Dictionary
  - » **Define new terminology or domain specific jargon that implementers or users may not know**
- Undesired Event Dictionary
  - » **Description of possible errors which can occur**
  - » **Contains warnings**
  - » **How to recognize error situations**
  - » **How to recover from error situations**
  - » **What to do if recovery is impossible**

TOC – 7

- ADT generation parameters
  - » Describe how instances and variations can be implemented from this generic data type
    - > How to change base types
    - > How to change amount of storage for a customer name
  - » Describe changes that can be made that will not violate assumptions and specifications. Design for a class of similar data types
  - » State what programming tools can be used to modify the implementation

TOC – 8

- Design issues
  - » **What were the design choices and why were the actual choices chosen. Help guide future changes to keep in the spirit of the original**
    - > **Why was fixed memory allocation used instead of dynamic?**
    - > **Why were size limits imposed?**
    - > **Why was a particular data structure chosen?**

TOC – 9

- Implementation notes
  - » **Designer may have information of use to the implementer. Know properties that can improve implementation**
- List of assumptions – those assumptions that
  - » **Cannot be violated**
  - » **Not implicit in the context**
  - » **Global**
  - » **Note: cannot state all assumptions so state those that**
    - > **Are most important**
    - > **Most likely to cause problems if violated**
    - > **Are not easily detected as causing problems until a long time later**

TOC – 10

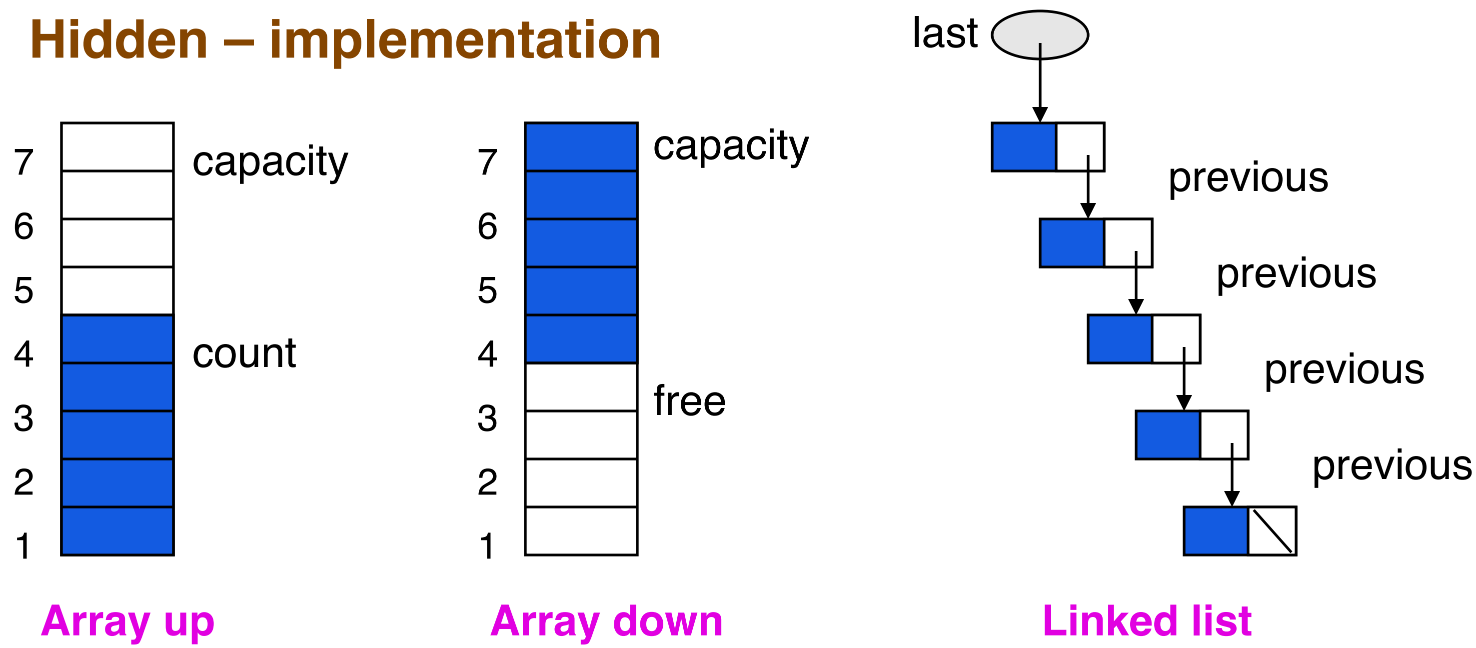
- Normal use assumptions
  - » **Information available from the ADT**
  - » **Information that must be supplied to the ADT**
  - » **Events reported by the ADT**
  - » **Tasks that can be performed by the ADT**
  - » **Operating states of the ADT and how they affect the Information obtained from and supplied to the ADT**
  - » **Failure states of the ADT and how they affect the information obtained from and supplied to the ADT**

TOC – 11

- Incorrect use assumptions
  - » **Associated with run time undesired events**
  - » **What may or may not happen if the production version has undesired event handling code removed to speed up the system**
- Program source text
  - » **If the source test is small may be included with the description of the operations**
- Facilities index
  - » **A quick look up reference of all programs, modules, operations, objects and terms defined**

Minimal Documentation

- Objects
  - » Types — Diagrams where possible
- Example – stack
  - » Imported – none
  - » Exported – STACK [G]
  - » Hidden – implementation



Minimal Documentation – 2

- Operations – example for a stack
  - » **Signatures, pre & post conditions**
    - > **push : STACK [G] x G $\rightarrow$ STACK [G]**
      - – **require** true
        ensure result = x ^ s & count = old count + 1
    - > **pop : STACK [G] $\rightarrow$ STACK [G]**
      - – **require** not empty (s)
        ensure result = s' & count = old count - 1
    - > **top : STACK [G] $\rightarrow$ G**
      - – **require** not empty (s)
        ensure result = s_1

Minimal Documentation – 3

- Operations – example for a stack cont'd

  > **empty : STACK [G] $\rightarrow$ BOOLEAN**

  – **require** **true**
  ensure **result = (count = 0)**

  > **new : [ ] $\rightarrow$ STACK [G]**

  – **require** **true**
  ensure **result = STACK [G] $\wedge$ count = 0**

  » **Note: often "require true" is not written but is assumed**

  » **It is better to write it as then one can wonder if it was left out by accident**

  > **"nothing" is often represented with a special symbol. e.g. nil , λ, ε, △**

Minimal Documentation – 4

- Operations – example for a stack cont'd
  - » **Axioms**
    - > $\forall$**x : G, s : STACK [G]** •
      top (push (s, x)) = x
      $\wedge$ **pop (push (s, x)) = s**
      $\wedge$ **empty (new)**
      $\wedge$ **~ empty (push (s, x))**

• **is read as "it is the case that"**

  - » **Alternately can use natural language**
    - > **forall x : G, s : STACK [G } ::**
      top (push (s, x)) = x
      and pop (push (s, x)) = s
      and empty (new)
      and not empty (push (s, x))

ADT Invariants

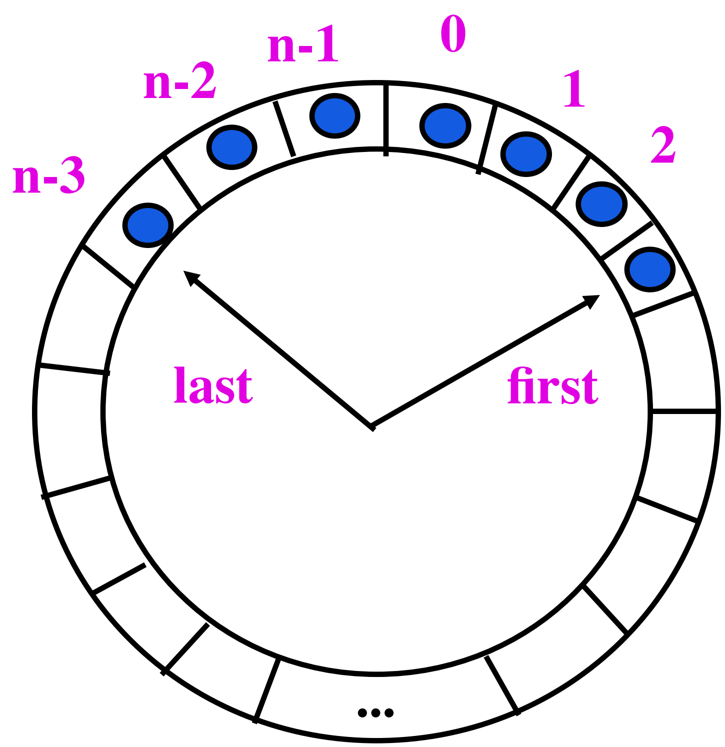
- Conditions that must be true after the execution of any method in the the class

- The conditions that hold, at all times, among the objects in an instance of the ADT
  - » **More on this when we discuss design by contract**

Example Circular Queue

isEmpty → length = 0 ∧ (last-1) mod Size = first
isFull → length = Size - 1
not isFull → length = (Size + first - last + 1) mod Size

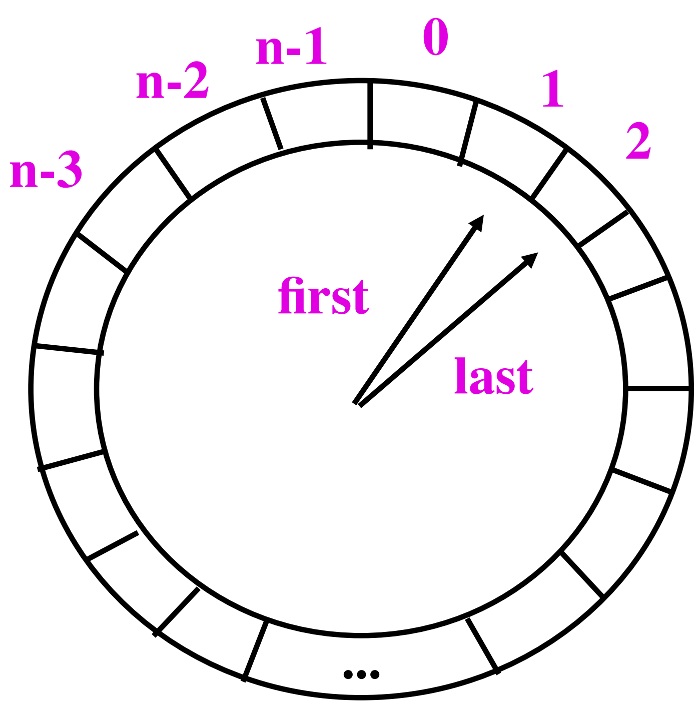
last is the last
Item to remove



first is the first
Item to remove

Empty Circular Queue

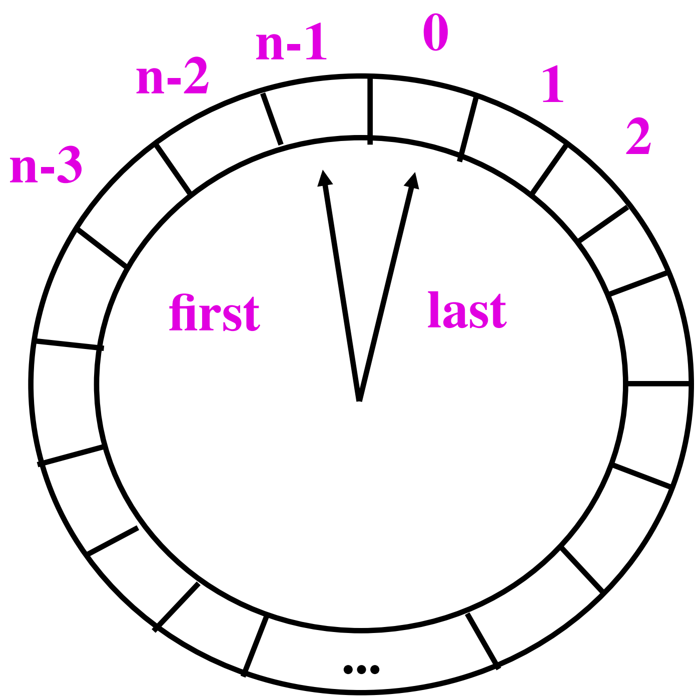
isEmpty $\rightarrow$ length = 0 $\wedge$ (last-1) mod Size = first
isFull $\rightarrow$ length = Size - 1
not isFull $\rightarrow$ length = (Size + first - last + 1) mod Size



length = (n + 1 - 2 + 1) mod n
= (n + 0) mod n
= 0

Empty Circular Queue – 2

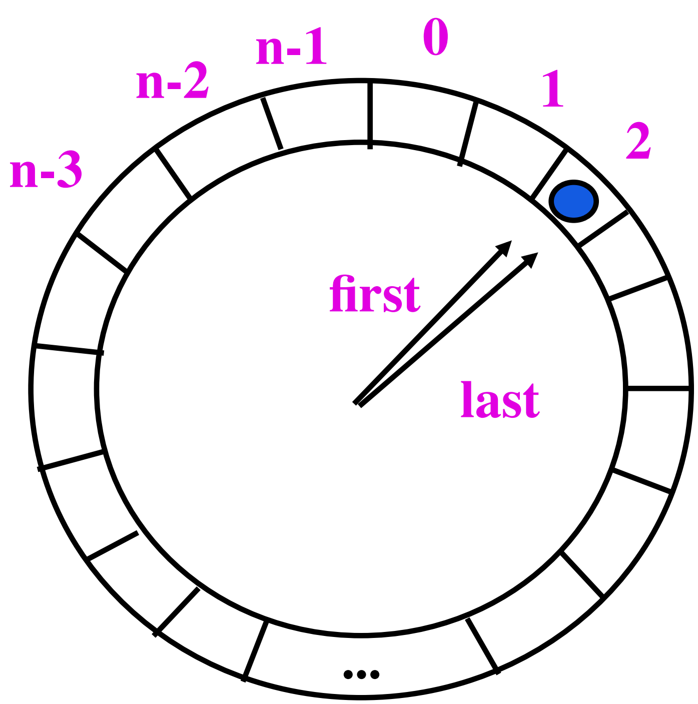
isEmpty → length = 0 ∧ (last-1) mod Size = first
isFull → length = Size - 1
not isFull → length = (Size + first - last + 1) mod Size



length = (n + (n-1) - 0 + 1) mod n
= (2n + 0) mod n
= 0

Length 1 Circular Queue

isEmpty $\rightarrow$ length = 0 $\wedge$ (last-1) mod Size = first
isFull $\rightarrow$ length = Size - 1
not isFull $\rightarrow$ length = (Size + first - last + 1) mod Size



length = (n + 2 - 2 + 1) mod n
= (n + 1) mod n
= 1

Longer length Circular Queue

length = (Size + first - last + 1) mod Size

length = (n + 3 - (n-3) + 1) mod n
= (7) mod n
= 7