

**SE 2AA4, CS 2ME3 (Introduction to Software  
Development)**

**Winter 2017**

# **10 Abstract Data Types (Ghezzi Ch. 4)**

Dr. Spencer Smith

Faculty of Engineering, McMaster University

January 25, 2017



# Abstract Data Types

- Administrative details
- File transfer on mills
- Implementation of a sequence abstract object
- Specification of abstract data types
- Example (similar to A2)
  - ▶ PointADT
  - ▶ LineADT
  - ▶ CircleADT
  - ▶ Deque

# Administrative Details

- Assignment 1
  - ▶ Files due by 11:59 pm January 28
  - ▶ E-mail partner files by 11:59 pm January 29
  - ▶ Lab report due by 11:59 pm February 2
  - ▶ Using Python 2.7, doxygen, make, LaTeX, git
  - ▶ Make sure everything runs on mills
  - ▶ Use the folder structure and filenames given!

# File Transfer To the CAS Servers

- sftp
- Samba: <smb://nts5.cas.mcmaster.ca/macid>
- Mount a drive over sftp using SSHFS  
<https://igikorn.com/sshfs-windows-10/>

# Implementation of An Abstract Object in Python

See the sample files in the repo.

# Specification of ADTs

- Similar template to abstract objects
- “Template Module” as opposed to “Module”
- “Exported Types” that are abstract use a ?
  - ▶ `pointT = ?`
  - ▶ `pointMassT = ?`
- Access routines know which abstract object called them
- Use “self” to refer to the current abstract object
- Use a dot “.” to reference methods of an abstract object
  - ▶ `p.xcoord()`
  - ▶ `self.pt.dist(p.point())`
- Similar notation to Python or Java

# Syntax Point ADT Module

## Template Module

pointADT

## Uses

N/A

## Exported Types

pointT = ?

## Syntax Point ADT Module Continued

<b>Routine name</b>	<b>In</b>	<b>Out</b>	<b>Exceptions</b>
new pointT	real, real	pointT	
xcoord		real	
ycoord		real	
dist	pointT	real	
rotate	real		



# Semantics Point ADT Module

## State Variables

$xc$ : real

$yc$ : real

## State Invariant

None

## Assumptions

None

# Access Routine Semantics Point ADT Module

new pointT ( $x, y$ ):

- transition:  $xc, yc := x, y$
- output:  $out := self$
- exception: none

xcoord:

- output:  $out := xc$
- exception: none

ycoord:

- output:  $out := yc$
- exception: none

# Semantics Point ADT Module Continued

$\text{dist}(p)$ :

- output:  $out := \sqrt{(xc - p.xcoord)^2 + (yc - p.ycoord)^2}$
- exception: none

$\text{rotate}(\varphi)$ :

- $\varphi$  is in radians
- transition:

$$\begin{bmatrix} xc \\ yc \end{bmatrix} := \begin{bmatrix} \cos \varphi & -\sin \varphi \\ \sin \varphi & \cos \varphi \end{bmatrix} \begin{bmatrix} xc \\ yc \end{bmatrix}$$

- exception: none

# Syntax Line ADT Module

## Template Module

lineADT

## Uses

pointADT

## Exported Types

lineT = ?

# Syntax Line ADT Module Continued

<b>Routine name</b>	<b>In</b>	<b>Out</b>	<b>Exceptions</b>
new lineT	pointT, pointT	lineT	
start		pointT	
end		pointT	
length		real	
midpoint		pointT	
rotate	real		

# Semantics Line ADT Module

## State Variables

s: pointT

e: pointT

## State Invariant

None

## Assumptions

None

# Access Routine Semantics Line ADT Module

new lineT ( $p_1, p_2$ ):

- transition:  $s, e := p_1, p_2$
- output:  $out := self$
- exception: none

start:

- output:  $out := s$
- exception: none

end:

- output:  $out := e$
- exception: none

# Access Routine Semantics Continued

length:

- output:  $out := s.dist(e)$
- exception: none

midpoint:

- output:  $out :=$

$new\ pointT(avg(s.xcoord, e.xcoord), avg(s.ycoord, e.ycoord))$

- exception: none

rotate ( $\varphi$ ):

$\varphi$  is in radians

- transition:  $s.rotate(\varphi), e.rotate(\varphi)$
- exception: none



# Line ADT Local Functions

## Local Functions

avg:  $\text{real} \times \text{real} \rightarrow \text{real}$

$$\text{avg}(x_1, x_2) \equiv \frac{x_1 + x_2}{2}$$

# Syntax Circle ADT Module

## Template Module

circleADT

## Uses

pointADT, lineADT

## Exported Types

circleT = ?

## Syntax Circle ADT Module Continued

<b>Routine name</b>	<b>In</b>	<b>Out</b>	<b>Exceptions</b>
new circleT	pointT, real	circleT	
centre		pointT	
radius		real	
intersect	circleT	boolean	
connection	circleT	lineT	

# Semantics Circle ADT Module

## State Variables

*c*: pointT

*r*: real

## State Invariant

None

## Assumptions

None

# Access Routine Semantics Circle ADT Module

new circleT (*cinput*, *rinput*):

- transition:  $c, r := cinput, rinput$
- output:  $out := self$
- exception: none

centre:

- output:  $out := c$
- exception: none

radius:

- output:  $out := r$
- exception: none

# Access Routine Semantics Continued

`intersect(ci):`

- output:

$\exists(p : \text{pointT} \mid \text{insideCircle}(p, ci) : \text{insideCircle}(p, self))$

- exception: none

`connection(ci):`

- output: *out* := new lineT(*c*, *ci*.centre)

- exception: none

# Circle ADT Local Functions

## Local Functions

insideCircle:  $\text{pointT} \times \text{circleT} \rightarrow \text{boolean}$

$\text{insideCircle}(p, c) \equiv p.\text{dist}(c.\text{centre}) \leq c.\text{radius}$

# Syntax Deque Of Circles Module

## **Module**

DequeCircleModule

## **Uses**

circleADT

## **Exported Constants**

max\_size = 20



# Syntax Deque Of Circles Module Continued

<b>Routine name</b>	<b>In</b>	<b>Out</b>	<b>Exceptions</b>
init			
pushBack	circleT		FULL
pushFront	circleT		FULL
popBack			EMPTY
popFront			EMPTY
back		circleT	EMPTY
front		circleT	EMPTY
size		integer	
disjoint		boolean	EMPTY
totalArea		real	EMPTY
averageRadius		real	EMPTY

# Semantics Deque Of Circles Module

## State Variables

$s$ : sequence of circleT

## State Invariant

$$|s| \leq \text{max\_size}$$

## Assumptions

$\text{init}()$  is called before any other access program.

# Access Routine Semantics Deque Of Circles Module

init():

- transition:  $s := \langle \rangle$
- exception: none

pushBack( $c$ ):

- transition:  $s := s || \langle c \rangle$
- exception:  $exc := (|s| = \text{max\_size} \Rightarrow \text{FULL})$

pushFront( $c$ ):

- transition:  $s := \langle c \rangle || s$
- exception:  $exc := (|s| = \text{max\_size} \Rightarrow \text{FULL})$

# Access Routine Semantics Continued

popBack():

- transition:  $s := s[0..|s| - 2]$
- exception:  $exc := (|s| = 0 \Rightarrow \text{EMPTY})$

popFront():

- transition:  $s := s[1..|s| - 1]$
- exception:  $exc := (|s| = 0 \Rightarrow \text{EMPTY})$

back():

- output:  $out := s[|s| - 1]$
- exception:  $exc := (|s| = 0 \Rightarrow \text{EMPTY})$

# Access Routine Semantics Continued

front():

- output:  $out := s[0]$
- exception:  $exc := (|s| = 0 \Rightarrow \text{EMPTY})$

size():

- output:  $out := |s|$
- exception: none

disjoint():

- output  
 $out := \forall(i, j : \mathbb{N} | i \in [0..|s| - 1] \wedge j \in [0..|s| - 1] \wedge i \neq j : \neg s[i].\text{intersect}(s[j]))$
- exception:  $exc := (|s| = 0 \Rightarrow \text{EMPTY})$

# Access Routine Semantics Continued

totalArea():

- output

*out* := ?

- exception:  $\text{exc} := (|s| = 0 \Rightarrow \text{EMPTY})$

averageRadius():

- output

*out* := ?

- exception:  $\text{exc} := (|s| = 0 \Rightarrow \text{EMPTY})$