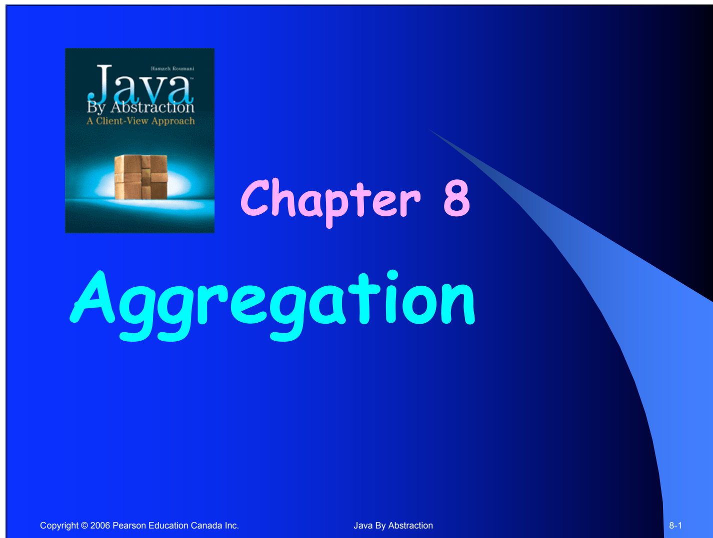


Outline

8.1 What is Aggregation?
8.1.1 Definition and Terminology
8.1.2 The Aggregate's Constructor
8.1.3 Accessors and Mutators
8.1.4 The Client's Perspective
8.1.5 Case Study: I/O Streams
8.1.6 Case Study: Graphics

8.2 Working with Collections
8.2.1 Creating the Collection
8.2.2 Adding/Removing Elements
8.2.3 Indexed Traversals
8.2.4 Chained Traversals
8.2.5 Searching
8.2.6 Search Complexity

Copyright © 2006 Pearson Education Canada Inc. Java By Abstraction 8-2

8.1 What is Aggregation?

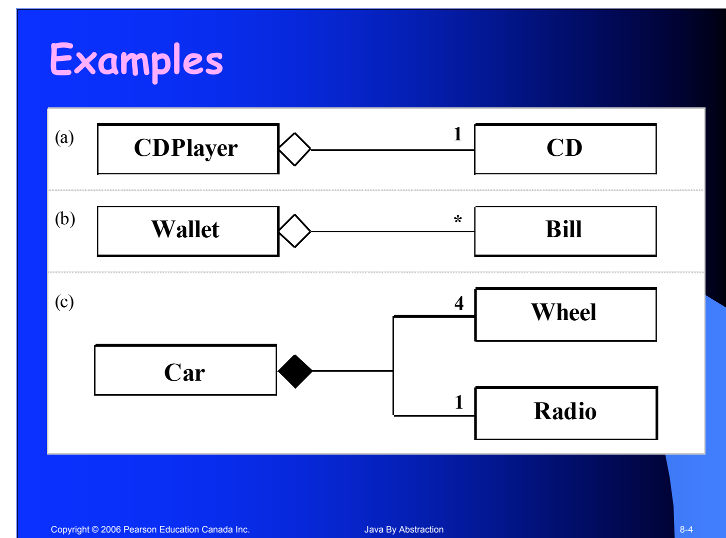
If one of the attributes of a class *C* is an object reference of type T*, then *C* is an aggregate and T is the aggregated part.

Every instance of *C* must have an instance of T (or else the attribute would be null).

Aggregation = has-a

* T != String

Copyright © 2006 Pearson Education Canada Inc. Java By Abstraction 8-3



Copyright © 2006 Pearson Education Canada Inc.

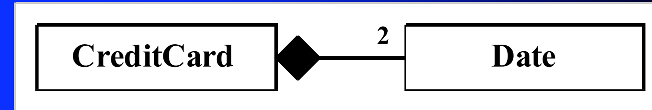
8.1.1 Definition and Terminology

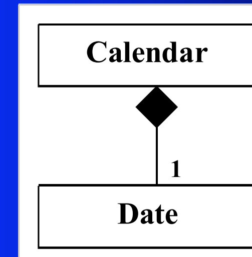
- Multiplicity
- Variable Multiplicity
- Collections (part=element)
- Composition (shared lifetime)

The Camera - Film Relation

Copyright © 2006 Pearson Education Canada Inc. Java By Abstraction 8-5

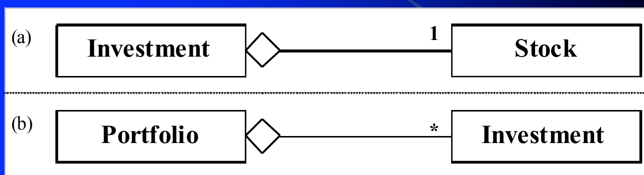
Examples





Copyright © 2006 Pearson Education Canada Inc. Java By Abstraction 8-6

Examples



Copyright © 2006 Pearson Education Canada Inc. Java By Abstraction 8-7

8.1.2 The Aggregate's Constructor

- When a client instantiates C, who instantiates T?
- Create an Investment
- Create a CreditCard
- What signature (for the Investment constructor) makes Investment a composition?

Copyright © 2006 Pearson Education Canada Inc. Java By Abstraction 8-8

Copyright © 2006 Pearson Education Canada Inc.

8.1.3 Accessors and Mutators

- Aggregates must provide an accessor through which the part can be accessed
- In a composition, the accessor returns a clone of the part
- An aggregate may provide a mutator so the client can mutate the part
- In a non-composition, such a mutator is not needed (why?)

Copyright © 2006 Pearson Education Canada Inc. Java By Abstraction 8-9

8.1.4 The Client's Perspective

- Aggregation = Layered Abstraction
- Sounds like an implementer's concern
- Why don't implementers hide it? If they did:
  - Investment would have to handle symbol, name, and price
  - CreditCard would have to accept day, month, and year.

Copyright © 2006 Pearson Education Canada Inc. Java By Abstraction 8-10

Example-1: Copying an Aggregate

Given a reference x to an aggregate, make a copy of it and call it y.

Copyright © 2006 Pearson Education Canada Inc. Java By Abstraction 8-11

Example-1: Copying an Aggregate

Given a reference x to an aggregate, make a copy of it and call it y.

Three different copies:

- An Alias
- A Shallow Copy
- A Deep Copy

Copyright © 2006 Pearson Education Canada Inc. Java By Abstraction 8-12

8.1.5 Case Study: I/O Streams

InputStream	InputStreamReader	BufferedReader
read(): int *reads one byte*	read(): int *reads one character*	readLine(): String *reads one line*

Input Stream Reader

Buffered Reader

```
BufferedReader buffer =
  new BufferedReader(
  new InputStreamReader(System.in));
```


File Input:

InputStream	FileInputStream	InputStreamReader	BufferedReader
read(): int *reads one byte*	read(): int *reads one byte*	read(): int *reads one c haracter*	readLine(): String *reads one line*

```
BufferedReader filer =
  new BufferedReader(
  new InputStreamReader(
  new FileInputStream(filename));
```


8.1.6 Case Study: Graphics

JFrame

Container

UniPanel

Graphics2D

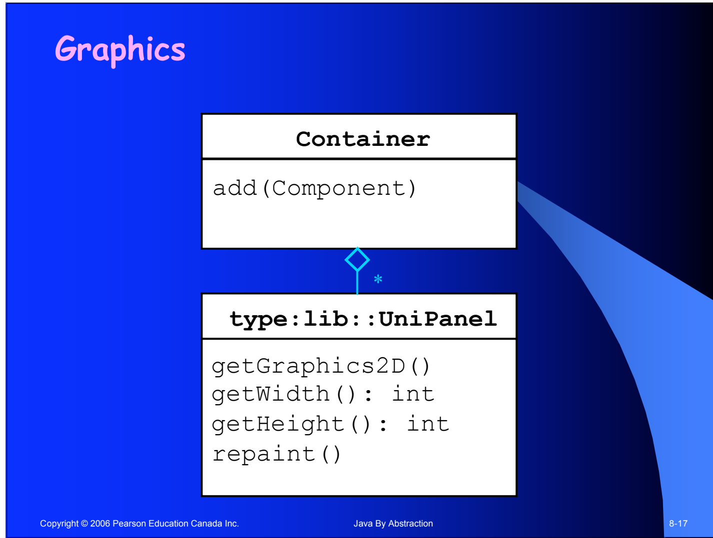
Graphics

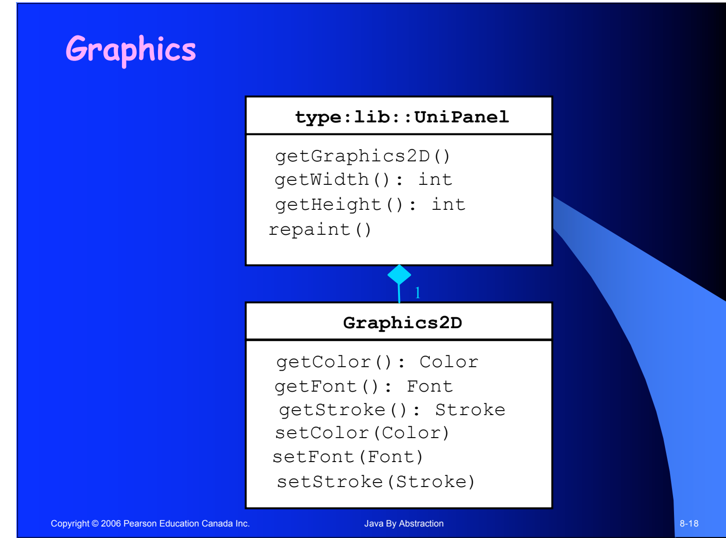
JFrame
getContentPane(): Container setContentPane(Container)

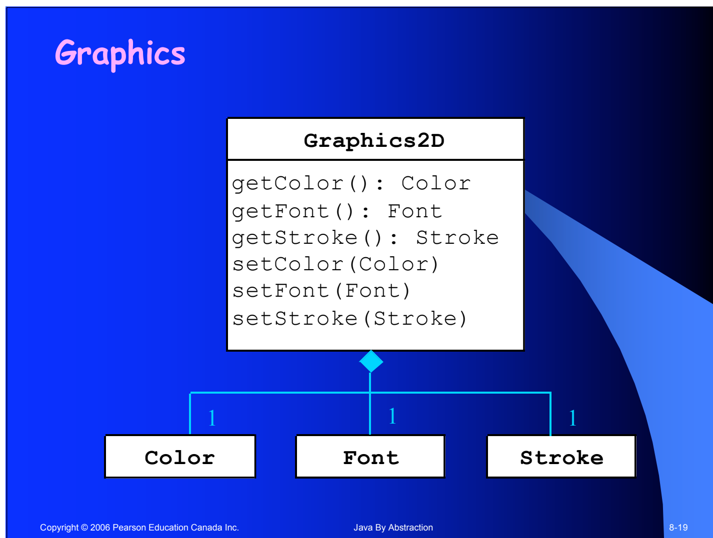
1

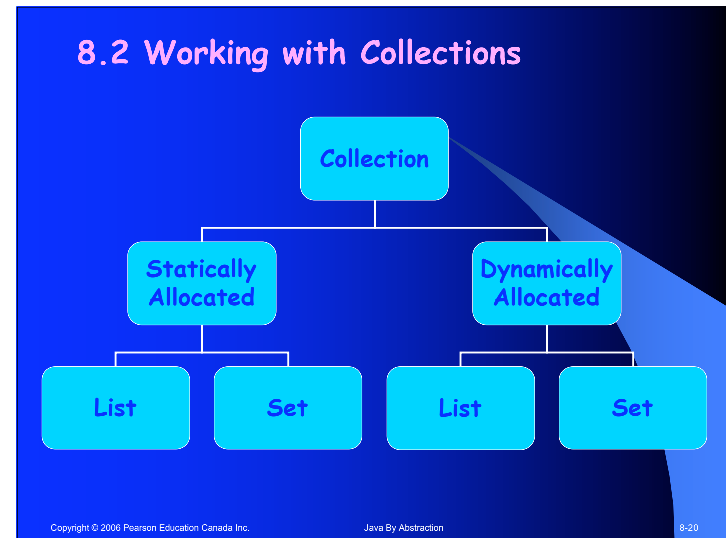
Container
add(Component)

Copyright © 2006 Pearson Education Canada Inc.


Graphics
Container
add(Component)
*
type:lib::UniPanel
getGraphics2D()
getWidth(): int
getHeight(): int
repaint()
Copyright © 2006 Pearson Education Canada Inc.
Java By Abstraction
8-17


Graphics
type:lib::UniPanel
getGraphics2D()
getWidth(): int
getHeight(): int
repaint()
1
Graphics2D
getColor(): Color
getFont(): Font
getStroke(): Stroke
setColor(Color)
setFont(Font)
setStroke(Stroke)
Copyright © 2006 Pearson Education Canada Inc.
Java By Abstraction
8-18


Graphics
Graphics2D
getColor(): Color
getFont(): Font
getStroke(): Stroke
setColor(Color)
setFont(Font)
setStroke(Stroke)
1
1
1
Color
Font
Stroke
Copyright © 2006 Pearson Education Canada Inc.
Java By Abstraction
8-19


8.2 Working with Collections
Collection
Statically Allocated
Dynamically Allocated
List
Set
List
Set
Copyright © 2006 Pearson Education Canada Inc.
Java By Abstraction
8-20

8.2.1 Creating the Collection

- Cannot specify elements as parameters
- Create an empty one then populate

Constructor Summary - Portfolio

`Portfolio(java.lang.String title, int capacity)`
Construct an empty portfolio having the passed name and capable of holding the specified number of investments.

Constructor Summary - GlobalCredit

`GlobalCredit()`
Construct a GC processing centre having the name "NoName".

Copyright © 2006 Pearson Education Canada Inc.

8.2.2 Adding / Removing Elements

- All collections provide a void or a boolean add to enable clients to populate.
- These methods are boolean for diff reasons:

Method Summary - Portfolio

boolean	`add(Investment inv)` Attempt to add the passed investment to `this` portfolio.

Method Summary - GlobalCredit

boolean	`add(CreditCard card)` Attempt to add the passed credit card to `this` GCC.

Copyright © 2006 Pearson Education Canada Inc.

8.2.3 Indexed Traversals

- Traversal in lieu of accessors
- Traverse = Visit each element once. Don't miss and don't over-visit.
- Indexed = Pretend the elements are numbered (0 offset).
- Two methods: get(int) and size()

Copyright © 2006 Pearson Education Canada Inc.

Example of an indexed traversal

Given a reference x to a Portfolio, list all its investments in a tabular fashion:

Inv.	Market	Book	Net
001	3450.00	2870.00	580.00
002	450.00	500.00	-50.00
.	.	.	.
.	.	.	.

Total

Copyright © 2006 Pearson Education Canada Inc.

Copyright © 2006 Pearson Education Canada Inc.

8.2.4 Chained Traversals

- The chain metaphor
- Often used in big and/or distributed databases
- Two methods: getFirst() and getNext()
- Both can return null but for different reasons
- Must invoke getFirst before getNext
- Could it be done with just one method?

Copyright © 2006 Pearson Education Canada Inc. Java By Abstraction 8-25

Example of a chained traversal

Given a reference x to a GlobalCredit, list all its credit cards in a tabular fashion:

```
Card No      Balance      Exp 36m?
907321-5       76.85
671282-1       81.64
464184-0      134.49          <
755917-2      232.43
   .             .            .
   .             .            .
---------------------------------
```

The last column indicates if the card will expire within 36 months

Copyright © 2006 Pearson Education Canada Inc. Java By Abstraction 8-26

Pitfall: On-the-fly Invocation

Given a reference x to a collection that, as a precondition, has at least one element and at most two. List its elements using chained traversal without writing a loop

Copyright © 2006 Pearson Education Canada Inc. Java By Abstraction 8-27

Pitfall: On-the-fly Invocation

Given a reference x to a collection that, as a precondition, has at least one element and at most two. List its elements using chained traversal without writing a loop

```
output.println(x.getFirst());
if (x.getNext() != null)
{
   output.println(x.getNext());
}
```

Copyright © 2006 Pearson Education Canada Inc. Java By Abstraction 8-28

8.2.5 Searching

Searching can be done via a traversal:

- Set up a traversal loop
- In each iteration, compare the element we are searching for with an element of the collection. Set a boolean flag accordingly
- The result (found or not found) must be somehow remembered after the loop is exited.

Copyright © 2006 Pearson Education Canada Inc. Java By Abstraction 8-29

A search example:

Given a reference gc to a random GlobalCredit, determine whether a given card c is in it.

Attempt #1 (incorrect):

```
boolean found = false;
for (Card card=gc.getFirst(); card != null; card=gc.getNext())
{
    found = card.equals(c);
}
```

Copyright © 2006 Pearson Education Canada Inc. Java By Abstraction 8-30

A search example, cont.

Correct it by adding the loop invariant:

```
c is equal to one of the elements seen so far
```

Attempt #2 (correct):

```
boolean found = false;
for (Card card=gc.getFirst(); card != null; card=gc.getNext())
{
    found = found || card.equals(c);
}
```

Copyright © 2006 Pearson Education Canada Inc. Java By Abstraction 8-31

A search example, cont.

Speed it up by exiting once c is found:

Attempt #3 (correct and efficient):

```
boolean found = false;
for (Card card = gc.getFirst(); card != null && !found;
                                card=gc.getNext())
{
    found = found || card.equals(c);
}
```

Copyright © 2006 Pearson Education Canada Inc. Java By Abstraction 8-32

Copyright © 2006 Pearson Education Canada Inc.

8.2.6 Search Complexity

- Traversal-based search is Exhaustive
- N comparisons in the worst case. It is thus a linear search

A bag contains N numbered balls and you can pick one ball one at a time. Can you determine if ball number 55 is in the bag by picking less than N times? In the worst case?

Copyright © 2006 Pearson Education Canada Inc. Java By Abstraction 8-33

Search Complexity

- Traversal-Based search: $O(N)$.
- Complexity of an algorithm can be: $O(1)$, $O(\lg N)$, $O(N)$, $O(N^2)$ … $O(2^N)$, $O(N!)$
- Can break the $O(N)$ barrier by pre-arranging the elements in some manner
- Sorting, Hashing, Tree structures can lead to sub-linear search complexity.
- GlobalCredit offers a non-exhaustive search. It is sub-linear

Copyright © 2006 Pearson Education Canada Inc. Java By Abstraction 8-34

Copyright © 2006 Pearson Education Canada Inc.