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 is String
8.1.1 Definition and Terminology

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

The Camera - Film Relation

Examples

(a) CDPlayer \( \rightarrow \) CD
(b) Wallet \( \rightarrow \) Bill
(c) Car \( \rightarrow \) Wheel, Radio

Examples

CreditCard \( \rightarrow \) Date

Calendar \( \rightarrow \) Date
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?

8.1.3 Accessors and Mutators

- Aggregates must provide an accessor thru 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?)
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.

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
8.1.5 Case Study: I/O Streams

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

8.1.6 Case Study: Graphics

JFrame
8.2 Working with Collections

- Collection
  - Statically Allocated
  - Dynamically Allocated
    - List
    - Set

8.2.1 Creating the Collection

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

**Constructor Summary - Portfolio**

```java
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**

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

<table>
<thead>
<tr>
<th>Method Summary - Portfolio</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean add(Investment inv)</td>
</tr>
<tr>
<td>Attempt to add the passed investment to this portfolio.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method Summary - GlobalCredit</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean add(CreditCard card)</td>
</tr>
<tr>
<td>Attempt to add the passed credit card to this GCC.</td>
</tr>
</tbody>
</table>

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()

Example of an indexed traversal

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

<table>
<thead>
<tr>
<th>Inv.</th>
<th>Market</th>
<th>Book</th>
<th>Net</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>3450.00</td>
<td>2870.00</td>
<td>580.00</td>
</tr>
<tr>
<td>002</td>
<td>450.00</td>
<td>500.00</td>
<td>-50.00</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
8.2.4 Iterator-Based Traversals

- More abstract than indexed
- Relies on the enhanced for loop
- Works if the collection implements Iterable

```java
for (E e : bag)
    // visit element e
```

Example of a chained traversal

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

<table>
<thead>
<tr>
<th>Card No</th>
<th>Balance</th>
<th>Exp 36m?</th>
</tr>
</thead>
<tbody>
<tr>
<td>907321-5</td>
<td>76.85</td>
<td></td>
</tr>
<tr>
<td>671282-1</td>
<td>81.64</td>
<td></td>
</tr>
<tr>
<td>464184-0</td>
<td>134.49</td>
<td>&lt;</td>
</tr>
<tr>
<td>755917-2</td>
<td>232.43</td>
<td></td>
</tr>
</tbody>
</table>

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

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

Attempt #1 (incorrect):

```java
boolean found = false;
for (CreditCard card : gc) {
    found = card.equals(c);
}
```

Attempt #2 (correct):

```java
boolean found = false;
for (CreditCard card : gc) {
    found = found || card.equals(c);
}
```

A search example, cont.
Correct it by adding the loop invariant:

```
The value of found is the same as the sentence:
c is equal to one of the elements seen so far
```

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?
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.