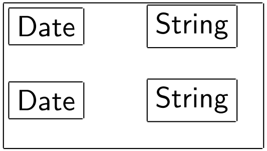
Review
Introduction to Computer Science I
CSE 1020

moodle.yorku.ca

Aggregation

Combine simple data into more complex data.



Aggregation

Definition

Aggregation is a binary relation on classes. The pair (A, P) of classes is in the aggregation relation if class A (aggregate) has an attribute of type P (part).

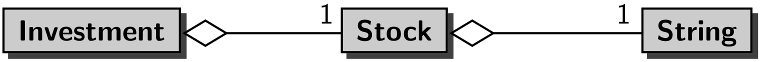
The aggregation relation is also known as the has-a relation. Instead of saying that (A, P) is in the aggregation relation, we often simply say that A has-a P.

Example

`Stock` has-a `String`.

`Investment` has-a `Stock`.

UML Diagrams



How to Copy an Object?

We will show three ways to copy an object:

- create an alias,
- create a shallow copy, and
- create a deep copy.

The created copies are fundamentally different.

Alias

```
Investment investment = Investment.getRandom();
Investment alias = investment;
```

Alias

100	main invocation
investment	400
alias	400
200	String object
	"HR.Z"
300	Stock object
symbol	200
400	Investment object
stock	300
quantity	8
bookValue	25.50

Shallow Copy

```
Investment investment = Investment.getRandom();
Stock stock = investment.getStock();
int quantity = investment.getQty();
double bookValue = investment.getBookValue();
Investment shallowCopy =
  new Investment(stock, quantity, bookValue);
```

Shallow Copy

100	main invocation
investment	400
stock	300
quantity	8
bookValue	25.50
shallowCopy	500
200	String object
	"HR.Z"
300	Stock object
symbol	200
400	Investment object
stock	300
quantity	8
bookValue	25.50
500	Investment object
stock	300
quantity	8
bookValue	25.50

Deep Copy

```
Investment investment = Investment.getRandom();
Stock stock = investment.getStock();
String symbol = stock.getSymbol();
int quantity = investment.getQty();
double bookValue = investment.getBookValue();
Stock stockCopy = new Stock(symbol);
Investment deepCopy =
  new Investment(stockCopy, quantity, bookValue);
```

Deep Copy

100	main invocation
investment	400
deepCopy	500
500	Investment object
stock	600
quantity	8
bookValue	25.50
600	Stock object
symbol	200

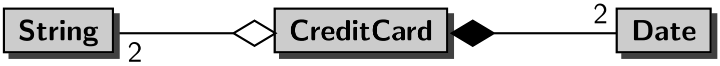
Composition

Composition is a special type of aggregation. The aggregate A and its part P form a composition if "A owns P", that is, each object of type A has exclusive access to its attribute of type P.

The designer and the implementer of a class determine whether an aggregation is a composition.

Java does not provide any special language constructs for implementing compositions. The constructors, accessors and mutators are implemented in a particular way (the details will be covered in CSE1030).

UML Diagrams



Accessor

```
CreditCard card = new CreditCard(123456, "Jane Doe");
Date expiryDate = card.getExpiryDate();
```

Accessor

100	main invocation
card	200
expiryDate	700
200	CreditCard object
number	300
name	400
issueDate	500
expiryDate	600
300	String object
	"123456"
400	String object
	"Jane Doe"
500	Date object
	now
600	Date object
	two year from now
700	Date object
	two years from now

Mutator

```
CreditCard card = new CreditCard(123456, "Jane Doe");
Date expiryDate = card.getExpiryDate();
final int YEAR = 113;
expiryDate.setYear(YEAR); // set year to 1900 + YEAR
```

Mutator

100	main invocation
card	200
expiryDate	700
YEAR	113
200	CreditCard object
number	300
name	400
issueDate	500
expiryDate	600
300	String object
	"123456"
400	String object
	"Jane Doe"
500	Date object
	now
600	Date object
	two years from now
700	Date object
	one year ago

Collection

We distinguish between

- static allocation: the maximum number of elements (capacity) is fixed when the collection is created
- dynamic allocation: the number of elements is unbounded

and

- list: duplicates are allowed and the elements are ordered
- set: duplicates are disallowed and the elements are *not* ordered

Traversals

```
for each element of the collection
    ...
```

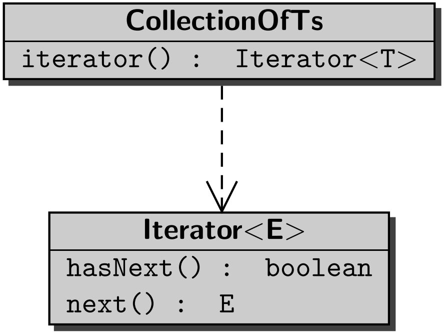
We distinguish two types of traversals:

- indexed traversals
- Iterator-based traversals

Indexed Traversals

```
... collection = ...
...
for (int i = 0; i < collection.size(); i++) {
   ... element = collection.get(i);
   ...
}
```

Iterator-Based Traversals



Iterator-Based Traversals

```
... collection = ...
...
Iterator<...> iterator = collection.iterator();
while (iterator.hasNext()) {
   ... element = iterator.next();
   ...
}
```

Iterator-Based Traversals

```
... collection = ...
...
Iterator<...> iterator = collection.iterator();
while (iterator.hasNext()) {
    ... element = iterator.next();
    ...
}
```

The above can be abbreviated using the advanced for loop:

```
... collection = ...
...
for (... element : collection) {
    ...
}
```