Composition and Aggregation

Java Developer

StayAhead Training; May, 2022

Class Associations*

Composition and Aggregation

- Class A might be associated with class B in a variety of ways
- One of A's methods might have a param. of type B - A **uses a** B
- Instances of A might be composed of one/more instances of B - A **has a** B
- B might inherit its state and behaviour from A - B **is a** A
- Composition and aggregation are forms of **has-a** type associations
- NB: many devs use only the term composition regardless of whether the association is a composite or an aggregate

Class Associations*

Composition and Aggregation

- Consider the following:

```
class TrainingCourse {

  ArrayList<Delegate> delegates;

}
```

- From this we can say that a TrainingCourse **has** Delegates, but we can't say whether the association is a composite or an aggregate

Composition

Composition and Aggregation

- Composition describes a has-a type association where the part **cannot** exist without the whole
- For example, a tennis match is composed of a score - the score has no meaning outside of the match
- For example, a movie is composed of reviews - the reviews are of no value when disconnected from the movie
- Can the part exist on its own without the whole? If no, it's composition

Aggregation

Composition and Aggregation

- Aggregation describes a has-a type association where the part **can** exist without the whole
- For example, a tennis match is composed of tennis players - the players do not depend on the match for their existence
- For example, a movie review is composed of a reviewer - the reviewer is likely to have reviewed other movies
- Can the part exist on its own without the whole? If yes, it's aggregation

Composite Associations

Composition and Aggregation

- In a composition type association the part is created inside the whole, e.g.

```
public class TennisMatch {
  private Score score;
  public TennisMatch() {
    score = new Score();
  }
}
```

Aggregate Associations

Composition and Aggregation

- In an aggregation type association the part is passed into the whole, e.g.

```
public class TennisMatch {
  private Player[] players;
  public TennisMatch(Player player1, Player player2) {
    players = new Player[] {player1, player2};
  }
}
```

Aggregate Associations

Composition and Aggregation

- In an aggregation type association the part is passed into the whole
- The part may be passed in via a constructor or setter method or both
- If the part is passed in via the one and only constructor then we could say that the whole **must** include the part
- A setter method for the part means the part may be exchanged for another

Composition vs. Aggregation*

Composition and Aggregation

- The key question is can the part exist on its own without the whole?
- But this is not the only consideration
- Composition type associations make the code less flexible and much harder to test
- Your editor will generate aggregation type associations automatically

The Problem with Getter Methods

Composition and Aggregation

- What's wrong with this code?

```
class TrainingCourse {
  ...
  public void registerDelegate(Delegate delegate) {
    // TODO add delegate if space available
  }
  public ArrayList<Delegate> getDelegates() {
    return delegates;
  }
}
```

The Problem with Getter Methods

Composition and Aggregation

- A getter method that returns a copy of a reference to a collection inadvertently enables users of the class to bypass business logic/rules
- In the previous example, another developer might do, e.g.

```
trainingCourse.getDelegates().add(delegate);
```

- The business logic that is implemented in the registerDelegate method (check that there is space available) is bypassed
- One solution to this problem is to have the getDelegates method return a copy of the ArrayList, e.g. `new ArrayList<>(delegates);`