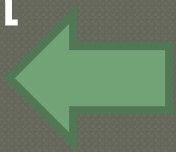
Design Patterns

- The Strategy Pattern
- The Factory Method
- Generics
- The Abstract Factory Pattern
- The State Pattern
- The Observer Pattern
- The Adapter Pattern
- The Composite Pattern
- **The Iterator Pattern**
- The Builder Pattern
- Fallen Patterns
  - The Singleton Pattern
  - The Visitor Pattern

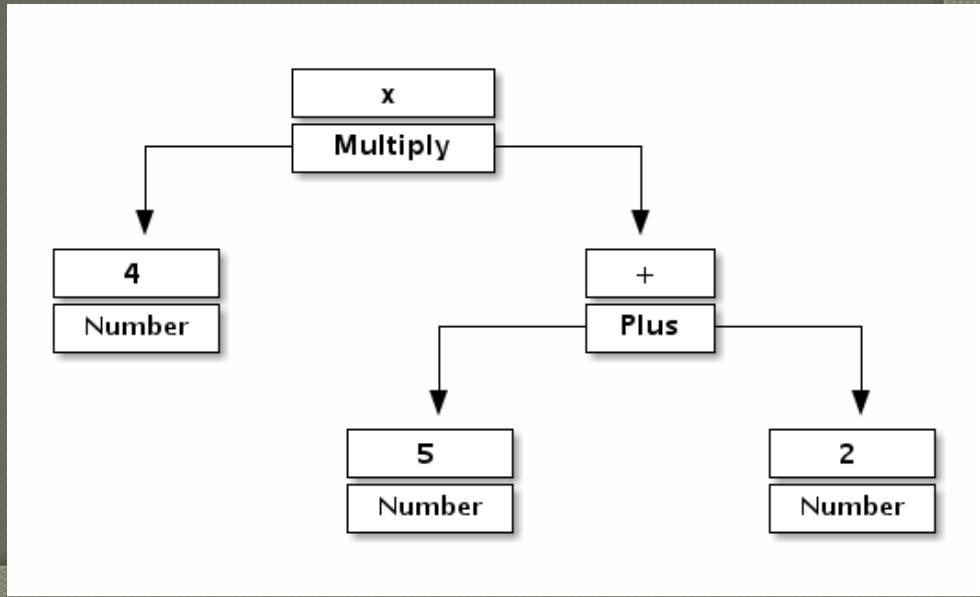


The Iterator

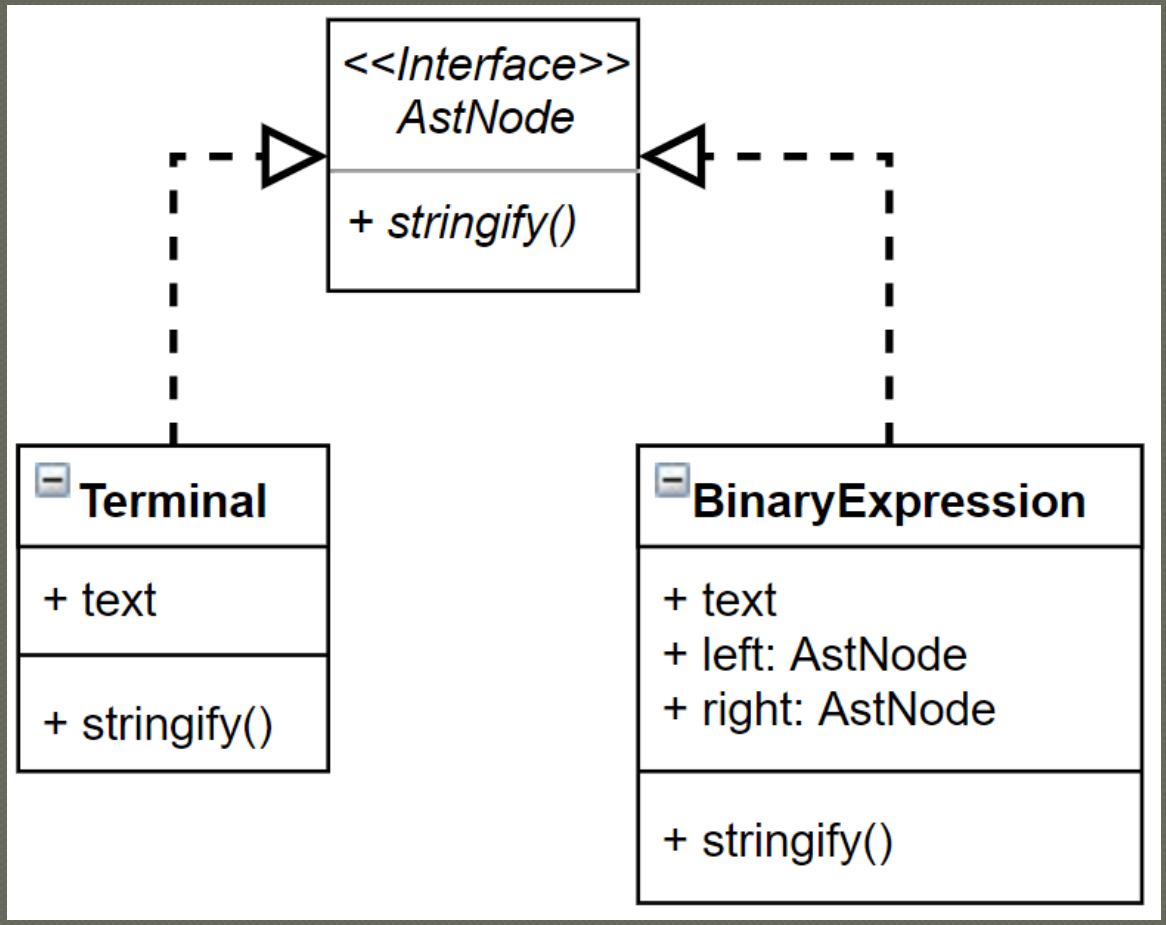
Let's go back to our composite example.

The abstract syntax tree node (AstNode).

How did it work again?



AstNode


<<Interface>>
AstNode
+ stringify()
Terminal
+ text
+ stringify()
BinaryExpression
+ text
+ left: AstNode
+ right: AstNode
+ stringify()

Usage

- We can create an abstract syntax tree easily:

```
new BinaryExpression( “+”,
   new Terminal( “7” ),
   new BinaryExpression( “*”,
         new Terminal( “4” ),
         new Terminal( “3” )
   )
)
```

- Represents 7 + (4 * 3)

- When we call `stringify()`, the `AstNode` is converted to `String`
- But, depending on implementation, you may not have control over the order of this operation

Solution: The Iterator

An iterator is an object that gives you a way of iterating over a data structure.

We can use iterators to specify what order we want.

An iterator "points" to a specific object inside the structure.

We can then move the iterator forward (or sometimes back)

How Iterators Work

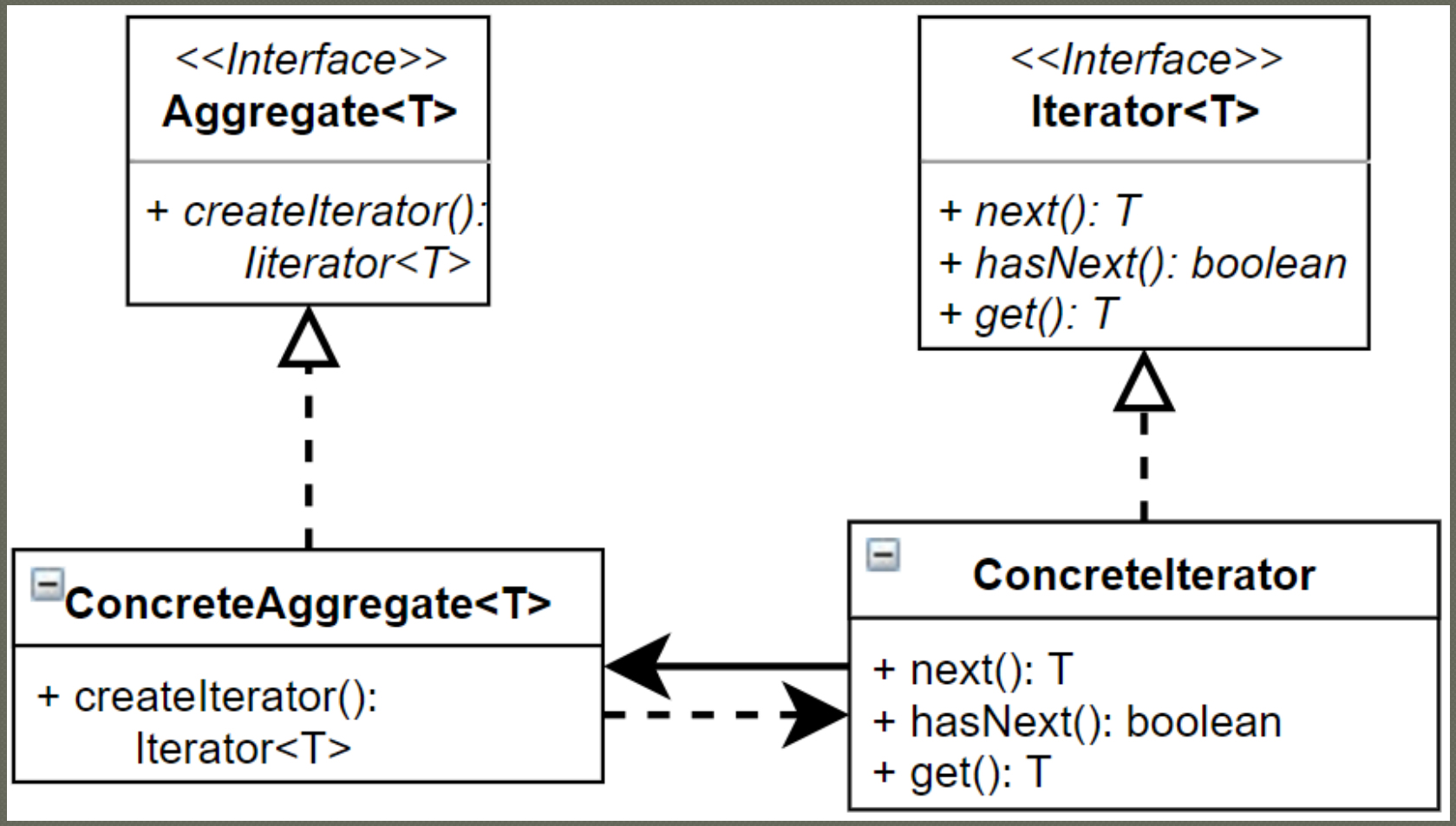
- Have a `next()` method to advance the iterator
- Have a `hasNext()` method to see if we're done
- https://docs.oracle.com/javase/8/docs/api/java/util/Iterator.html

Important Notes

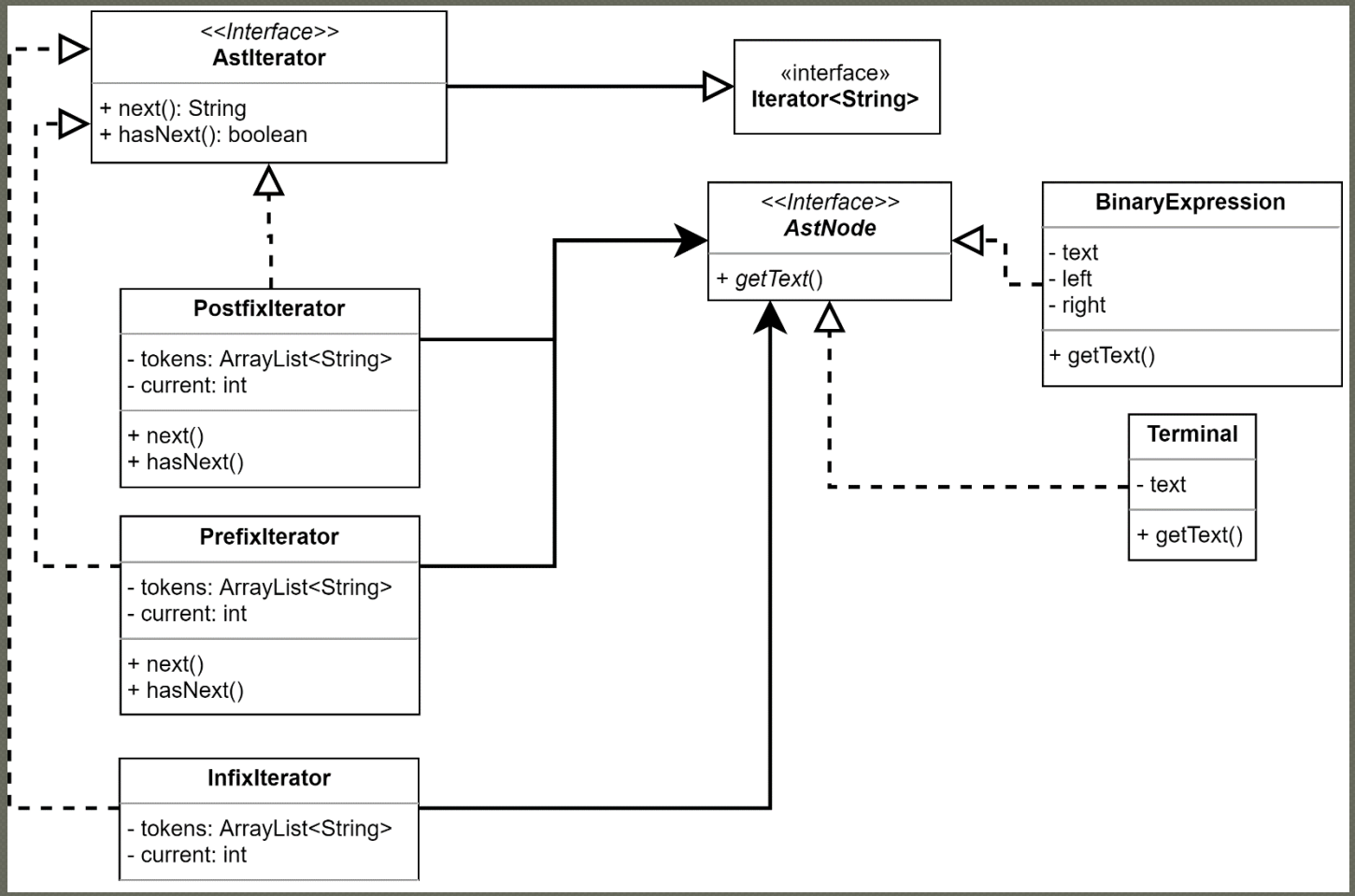
- Iterators are how the enhanced `for` loop works in Java
  - `for ( int elem : listOfNumbers )` …
  - called "for each" in every other language
- The list `listOfNumbers` has the ability to return an iterator (implementing the `Iterable<T>` interface)
- Java hides the iterator from you. This is what's really happening:

```
ListIterator<int> iter = listOfNumbers.listIterator()
   while( iter.hasNext() ) {
      int elem = iter.next();
       //…
   }
```

Iterator


<<Interface>>
Aggregate<T>
+ createIterator(): Iterator<T>
<<Interface>>
Iterator<T>
+ next(): T
+ hasNext(): boolean
+ get(): T
ConcreteAggregate<T>
+ createIterator(): Iterator<T>
ConcreteIterator
+ next(): T
+ hasNext(): boolean
+ get(): T

AstNode with Iterator



What Did We Gain?

We have removed "iteration" as a responsibility from Ast

We have improved flexibility

We have enhanced encapsulation

What Did We Lose?

- It's more complicated now

Applying The Iterator Pattern to Your Project

- What data structure will you use and how will you iterate over it?