The Iterator Pattern

It provides a way to access the elements of an aggregate object sequentially, without exposing its underlying representation

Iterable

Iterable (*adjective*): something that can be iterated

In software, an iterable object is such that it is possible to generate a sequence of values from it:

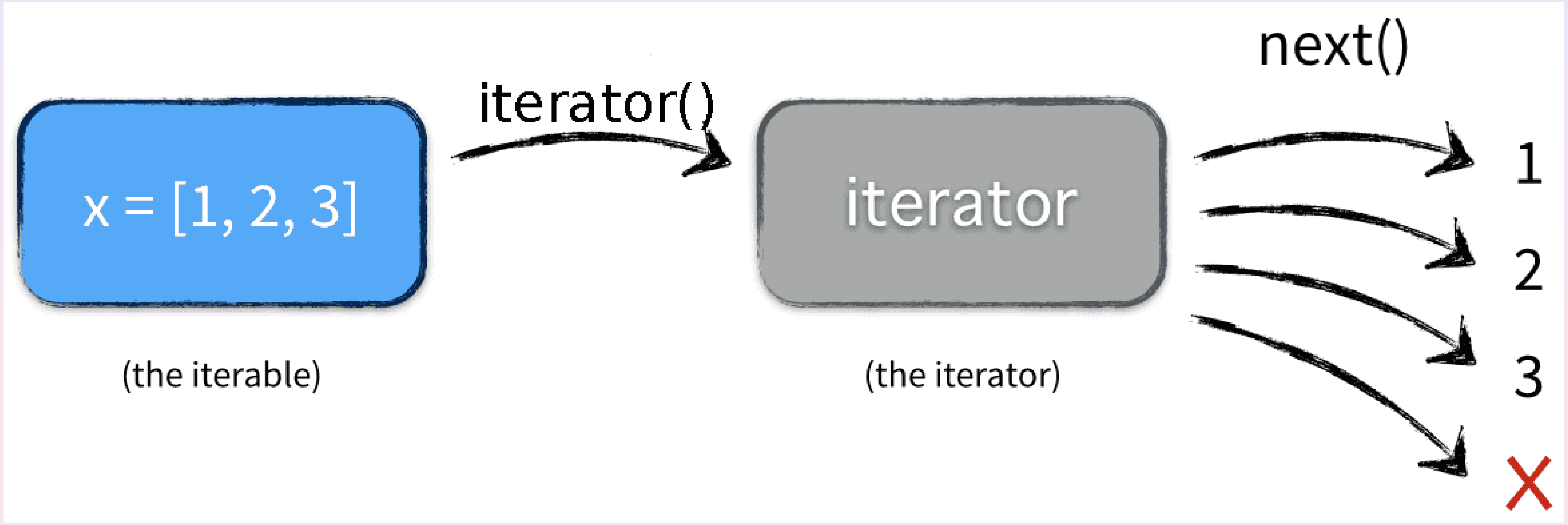
- month ⇒ its dates, in any order
- year ⇒ its months, in any order
- year ⇒ its dates, in any order
- the set of prime numbers ⇒ the prime numbers, in any order
- a set ⇒ its elements, in any order
- a tree ⇒ its leaves, in any order

Iterator

Iterator (*noun*): something that provides each value of a sequence of values, iteratively

- the dates of a month
- the months of a year
- the dates of a year
- the prime numbers
- the elements of a set
- the leaves of a tree

Interaction between Iterable and Iterator

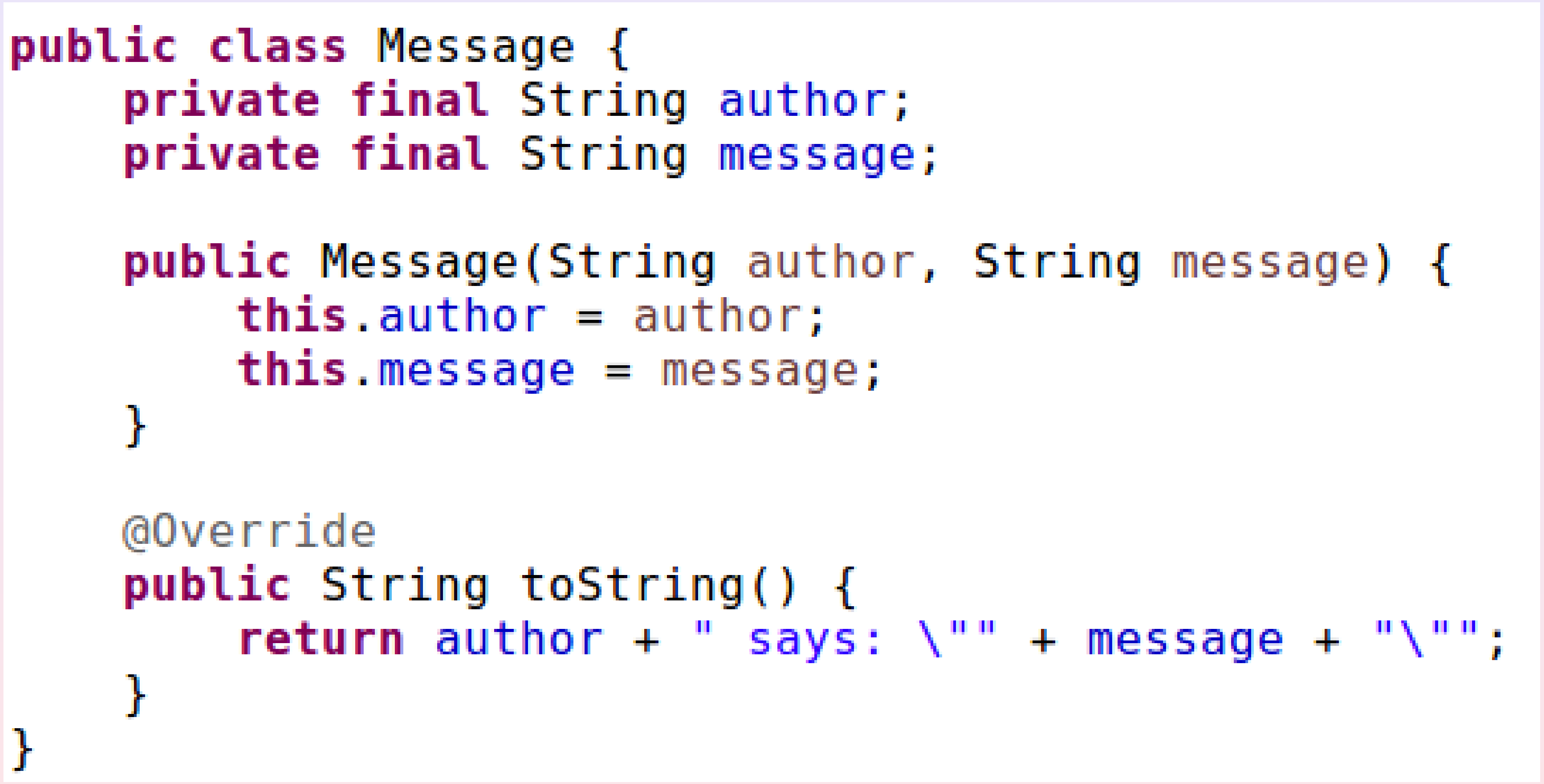


Iterable and Iterator in Java

Part of the standard Java library

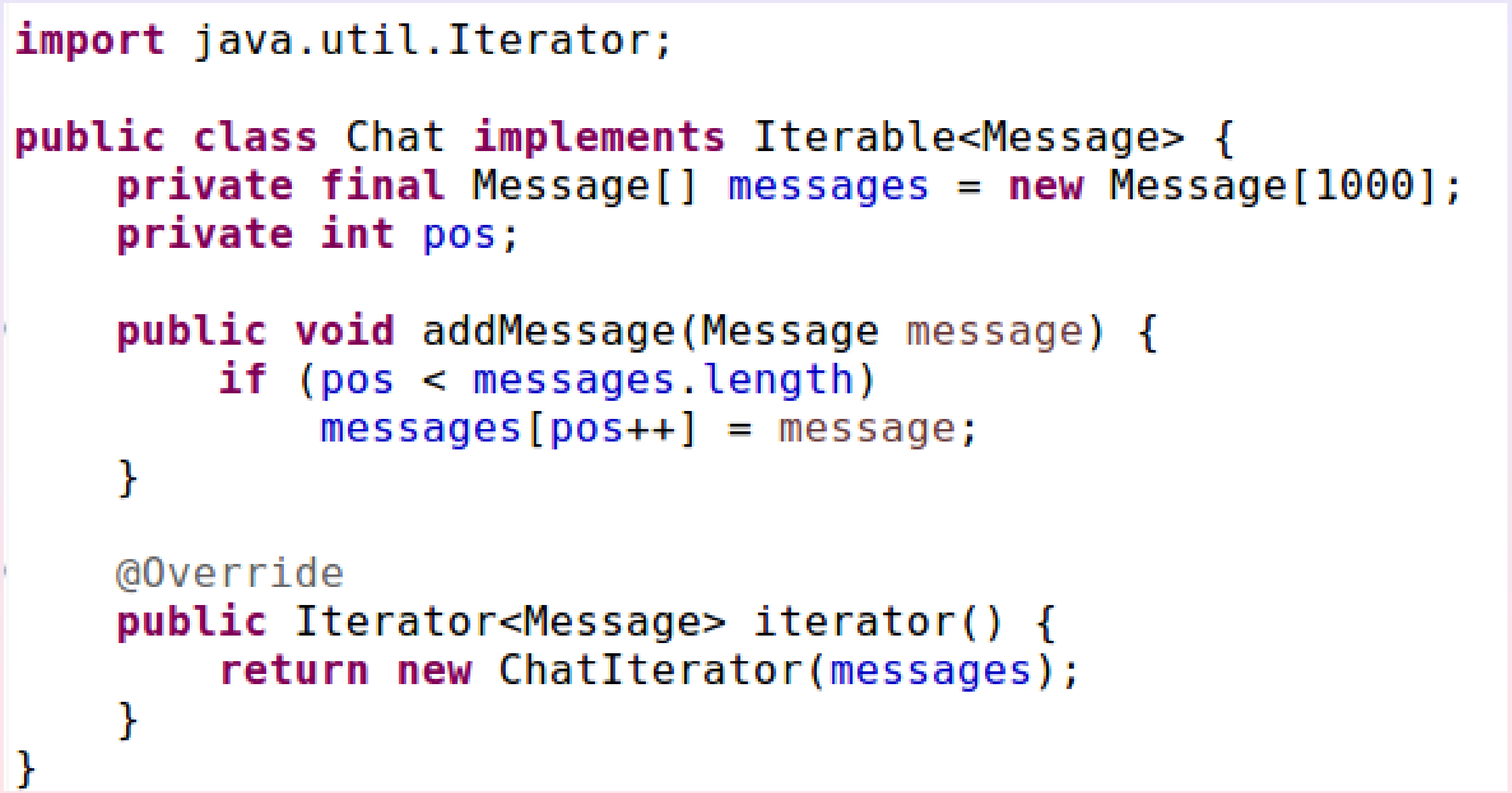
```
public interface java.lang.Iterable<T> {
    Iterator<T> iterator();
}

public interface java.util.Iterator<E> {
    boolean hasNext();
    E next();
}
```

A Simple Chat Message



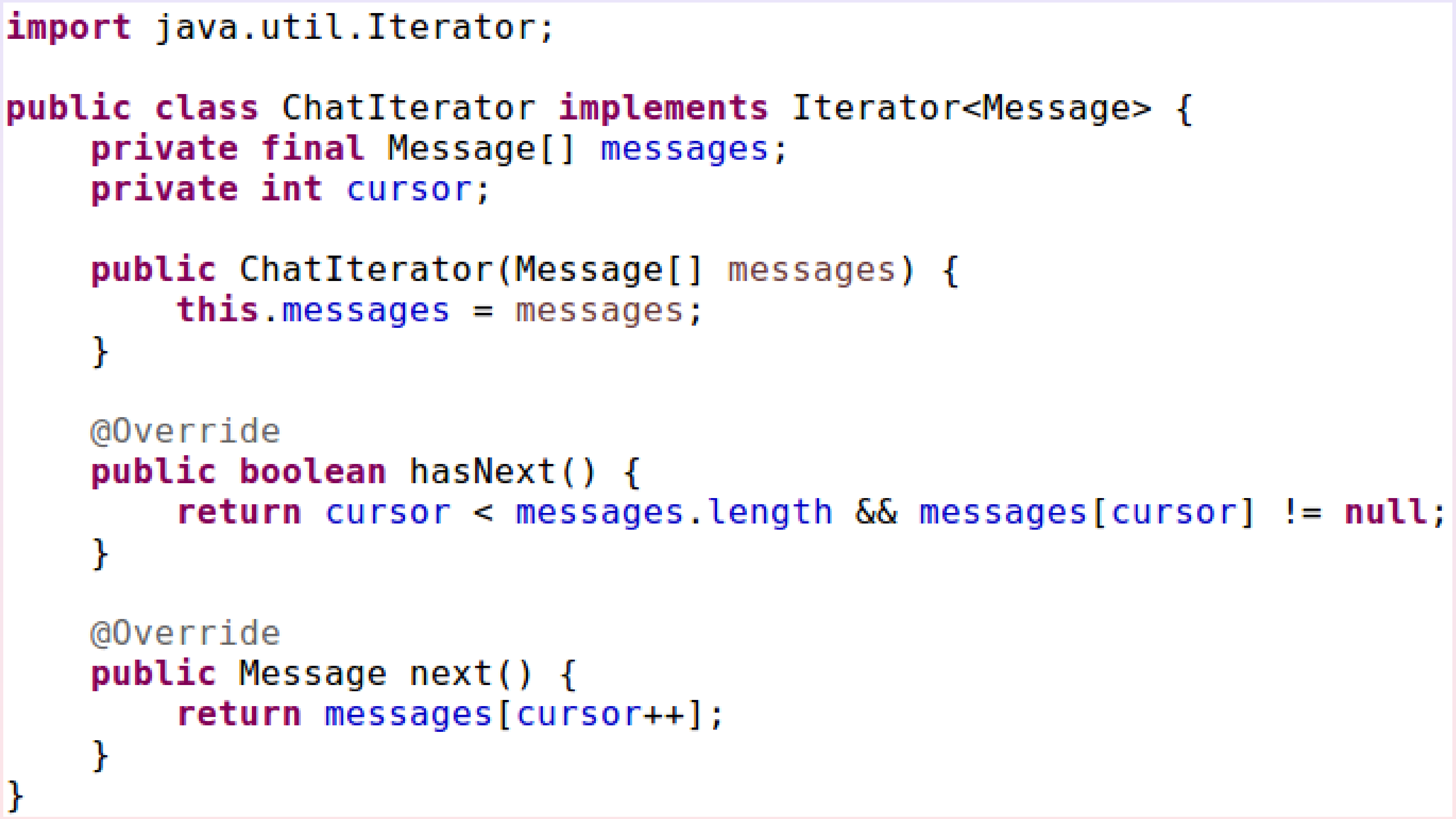
```java
public class Message {
    private final String author;
    private final String message;

    public Message(String author, String message) {
        this.author = author;
        this.message = message;
    }

    @Override
    public String toString() {
        return author + " says: \"" + message + "\"";
    }
}
```

A Chat Contains Many Messages



```java
import java.util.Iterator;

public class Chat implements Iterable<Message> {
    private final Message[] messages = new Message[1000];
    private int pos;

    public void addMessage(Message message) {
        if (pos < messages.length)
            messages[pos++] = message;
    }

    @Override
    public Iterator<Message> iterator() {
        return new ChatIterator(messages);
    }
}
```

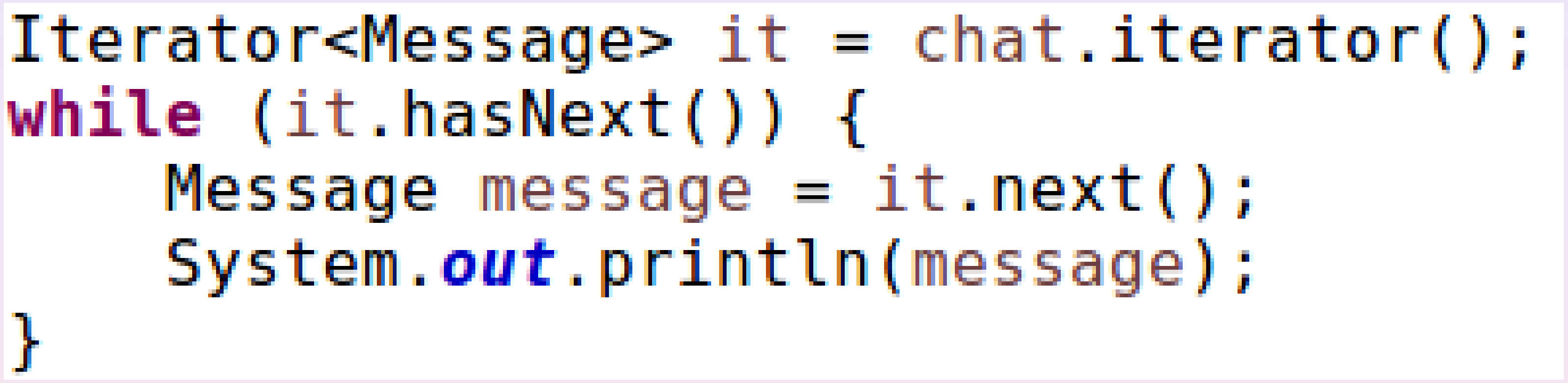
A Chat Iterator Scans the Messages



```java
import java.util.Iterator;

public class ChatIterator implements Iterator<Message> {
    private final Message[] messages;
    private int cursor;

    public ChatIterator(Message[] messages) {
        this.messages = messages;
    }

    @Override
    public boolean hasNext() {
        return cursor < messages.length && messages[cursor] != null;
    }

    @Override
    public Message next() {
        return messages[cursor++];
    }
}
```

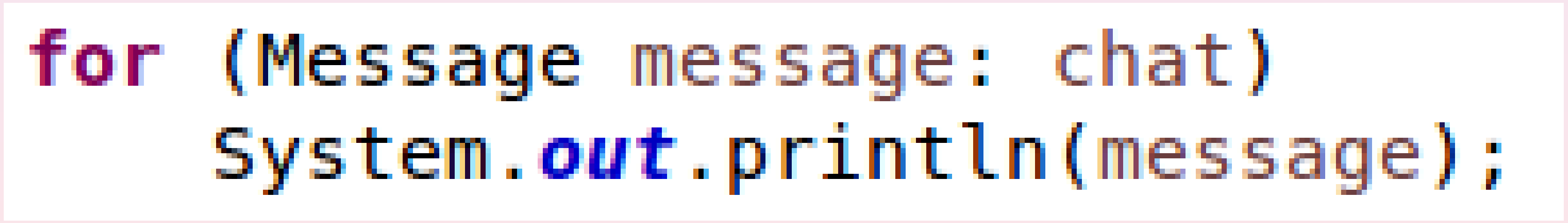
Use it

If variable `chat` has type `Chat` then we can scan its messages with:



```java
Iterator<Message> it = chat.iterator();
while (it.hasNext()) {
    Message message = it.next();
    System.out.println(message);
}
```

or just through the **for each** loop on iterables:

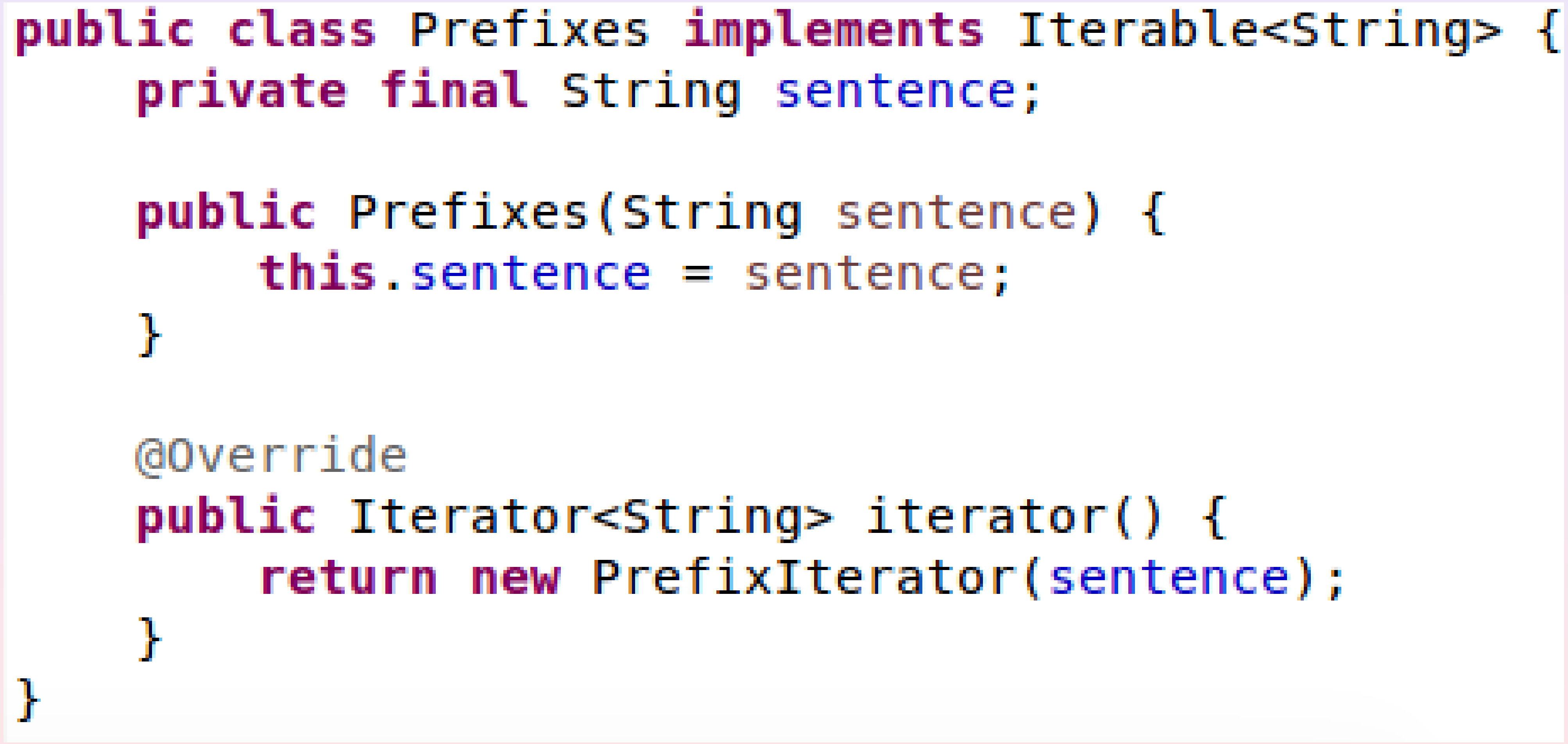


```java
for (Message message: chat)
    System.out.println(message);
```

without exposing the underlying array of a `Messages`!

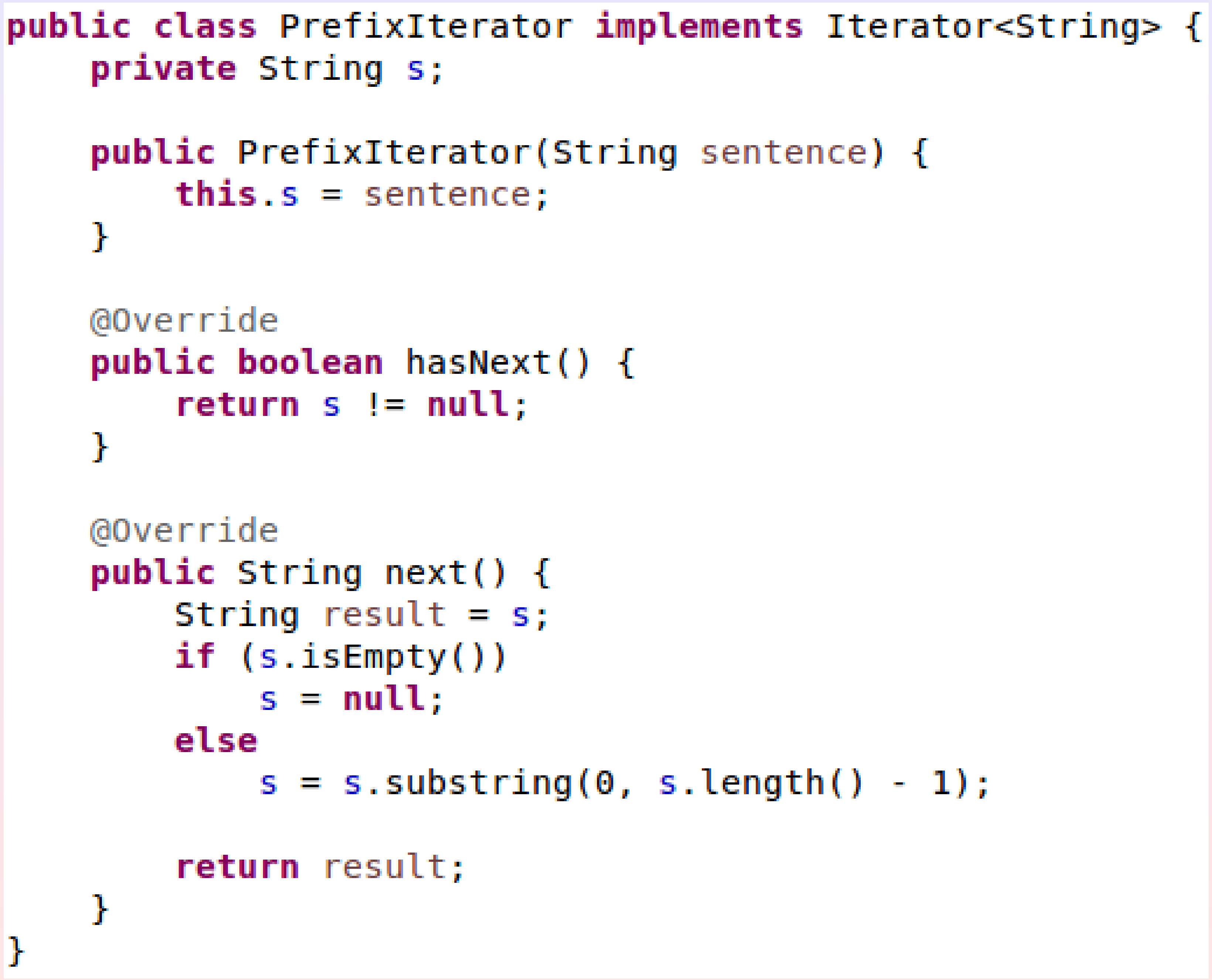
An Iterator can create the iterated elements

The iterable set of prefixes of a string



```
public class Prefixes implements Iterable<String> {
    private final String sentence;

    public Prefixes(String sentence) {
        this.sentence = sentence;
    }

    @Override
    public Iterator<String> iterator() {
        return new PrefixIterator(sentence);
    }
}
```

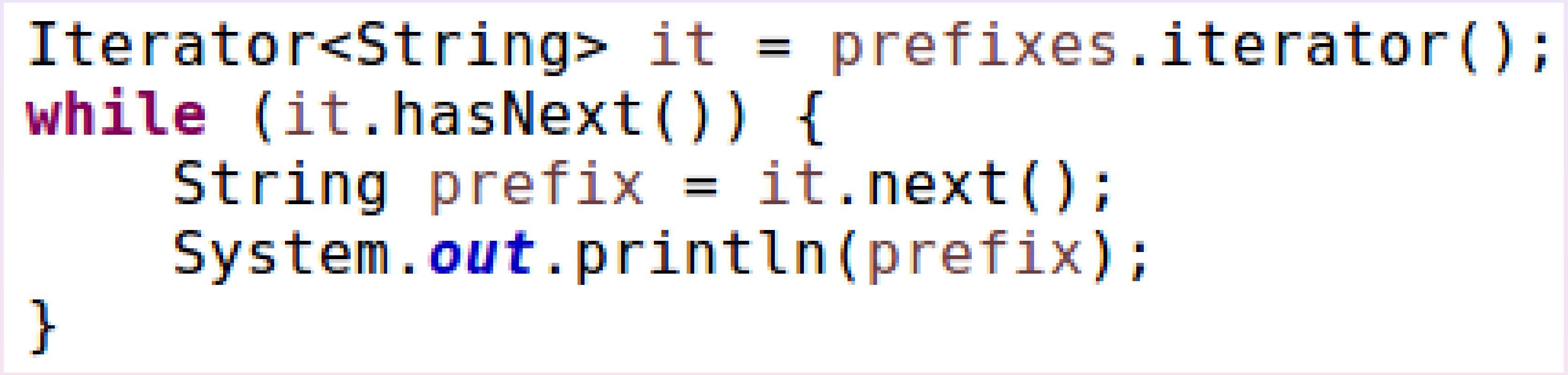
An Iterator can create the iterated elements



```java
public class PrefixIterator implements Iterator<String> {
    private String s;

    public PrefixIterator(String sentence) {
        this.s = sentence;
    }

    @Override
    public boolean hasNext() {
        return s != null;
    }

    @Override
    public String next() {
        String result = s;
        if (s.isEmpty())
            s = null;
        else
            s = s.substring(0, s.length() - 1);

        return result;
    }
}
```

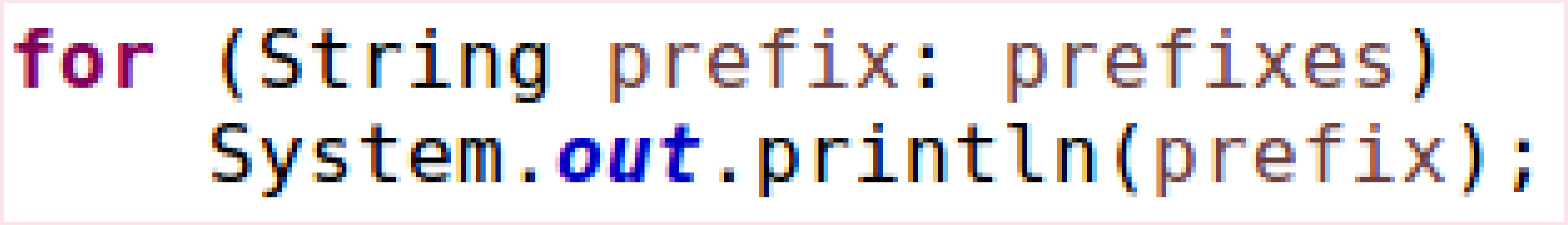
Use it

If variable `prefixes` has type `Prefixes` then we can scan the prefixes with:



```java
Iterator<String> it = prefixes.iterator();
while (it.hasNext()) {
    String prefix = it.next();
    System.out.println(prefix);
}
```

or just through the **for each** loop on iterables:



```java
for (String prefix: prefixes)
    System.out.println(prefix);
```