{JG} JavaGuru

INTRODUCTION TO JAVA

Java 1.0

LISTS

Lesson # 10

COLLECTIONS OVERVIEW

REASONING FOR COLLECTIONS

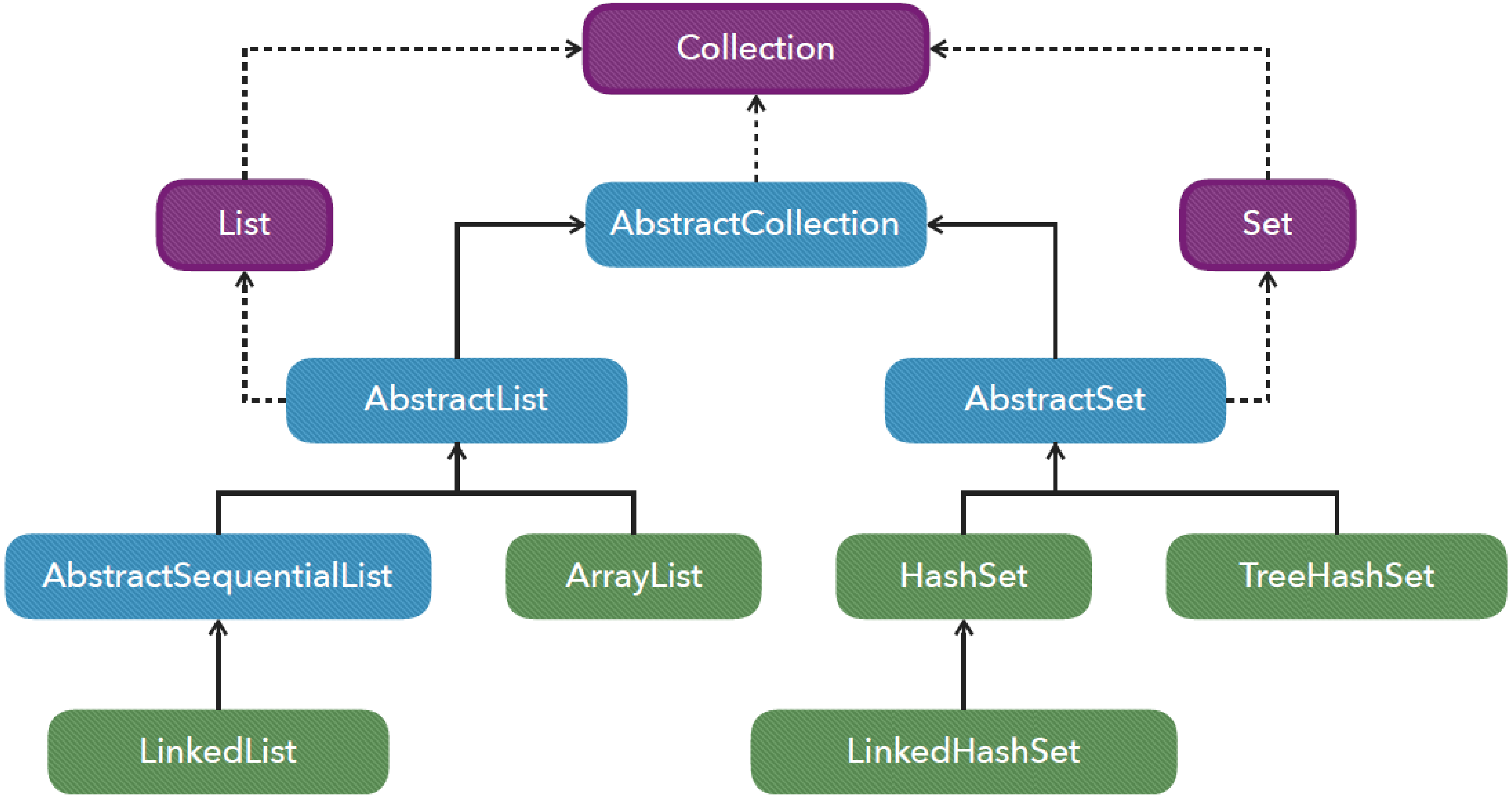
- Plain data structures (e.g., arrays) are simple and fast but cumbersome to work with

- Initially, Java provided some tools to store and manipulate groups of objects, but they lacked a unifying theme

REASONING FOR COLLECTIONS

- Language developers wanted to design such a framework that would meet several goals
  - High performance
  - Support a high degree of interoperability and abstraction
  - Extend and adapt collections easily

COLLECTION API HIERARCHY



COLLECTION CHARACTERISTICS

- Ordered
  - Whether it is possible to iterate over the elements of an ordered collection in a predictable order
- Uniqueness of elements
  - Some collections do not allow duplicate elements
- Thread safety
  - Whether it is safe to work with collection in a multithreaded environment

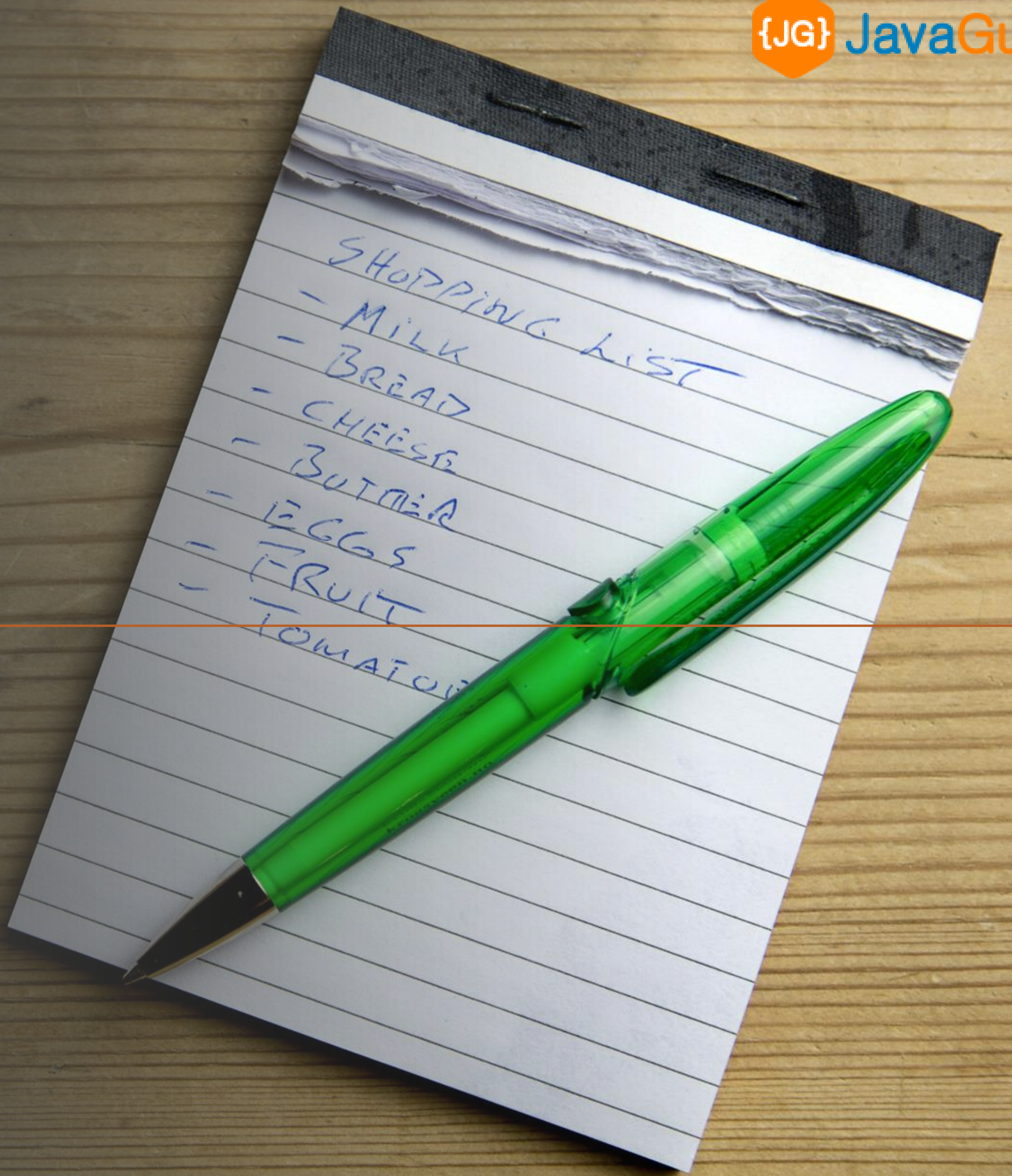
COLLECTION CHARACTERISTICS

- Underlying storage structure
    - Array-based storage
        - Fast to access but slow to remove or insert
    - Linked-list-based storage
        - Efficient at removing or inserting but slower for access

COLLECTION CHARACTERISTICS

- Underlying storage structure
  - Hash-based storage
    - Reasonably efficient access
  - Tree-based storage
    - Efficient for searching

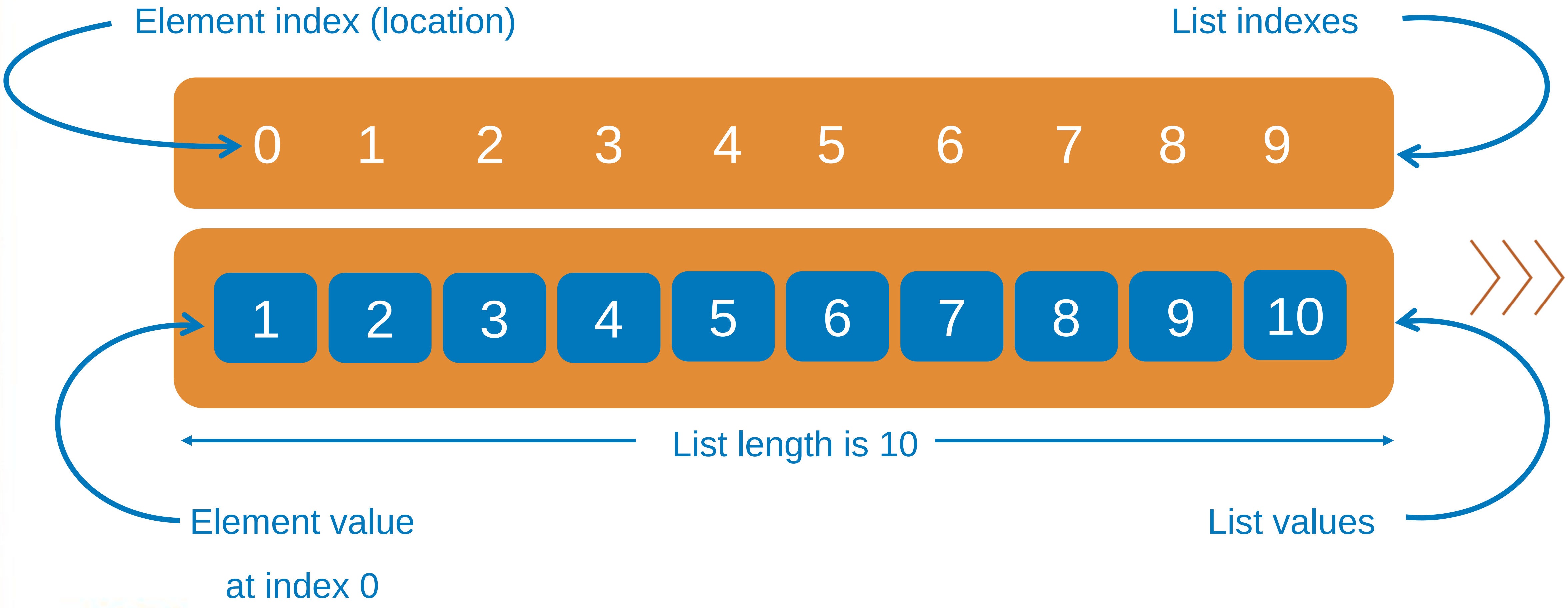
LIST INTERFACE



LIST OVERVIEW

- The List is probably the most useful and widely used type of Collection
- A list collection stores elements by insertion order, just like an array
- The list is a general interface, and ArrayList and LinkedList are implementing classes
- A list can store objects of any type.
- Primitive types are automatically converted to the corresponding wrapper type

LIST VISUALISATION



Element index (location)

List indexes

0	1	2	3	4	5	6	7	8	9
1	2	3	4	5	6	7	8	9	10

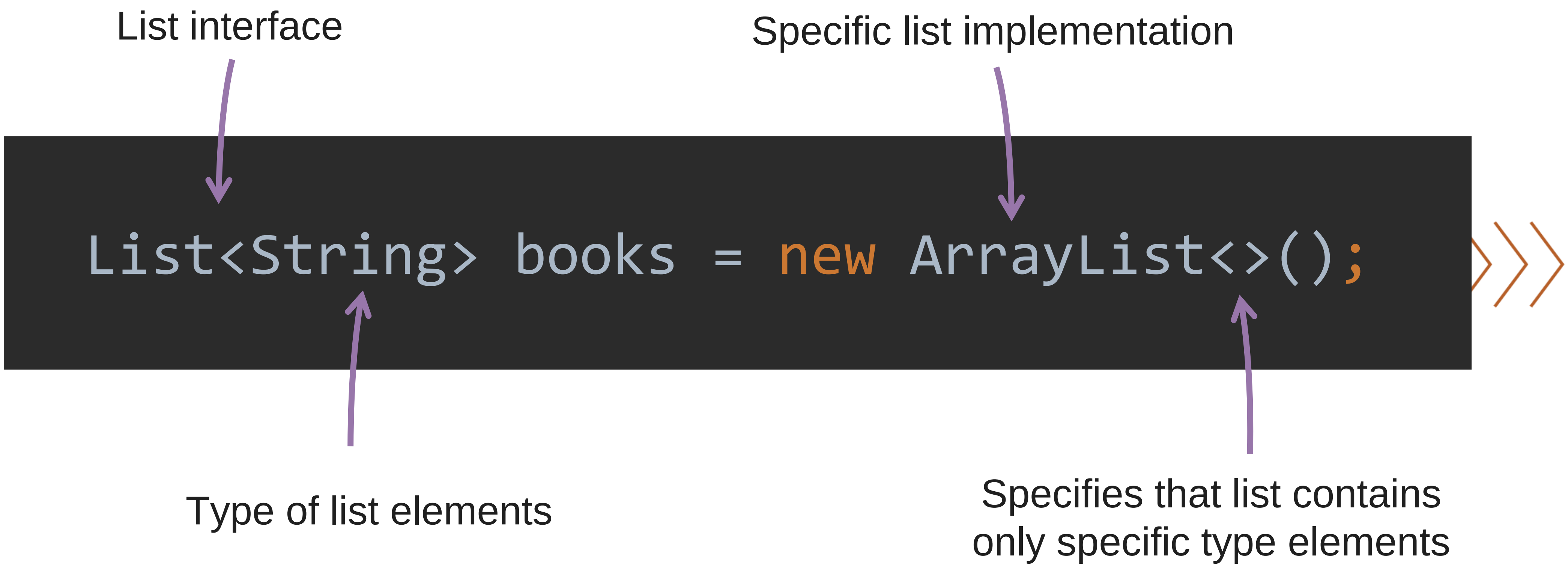
List length is 10

Element value

at index 0

List values

LIST INITIALIZATION



List interface

Specific list implementation

```
List<String> books = new ArrayList<>();
```

Type of list elements

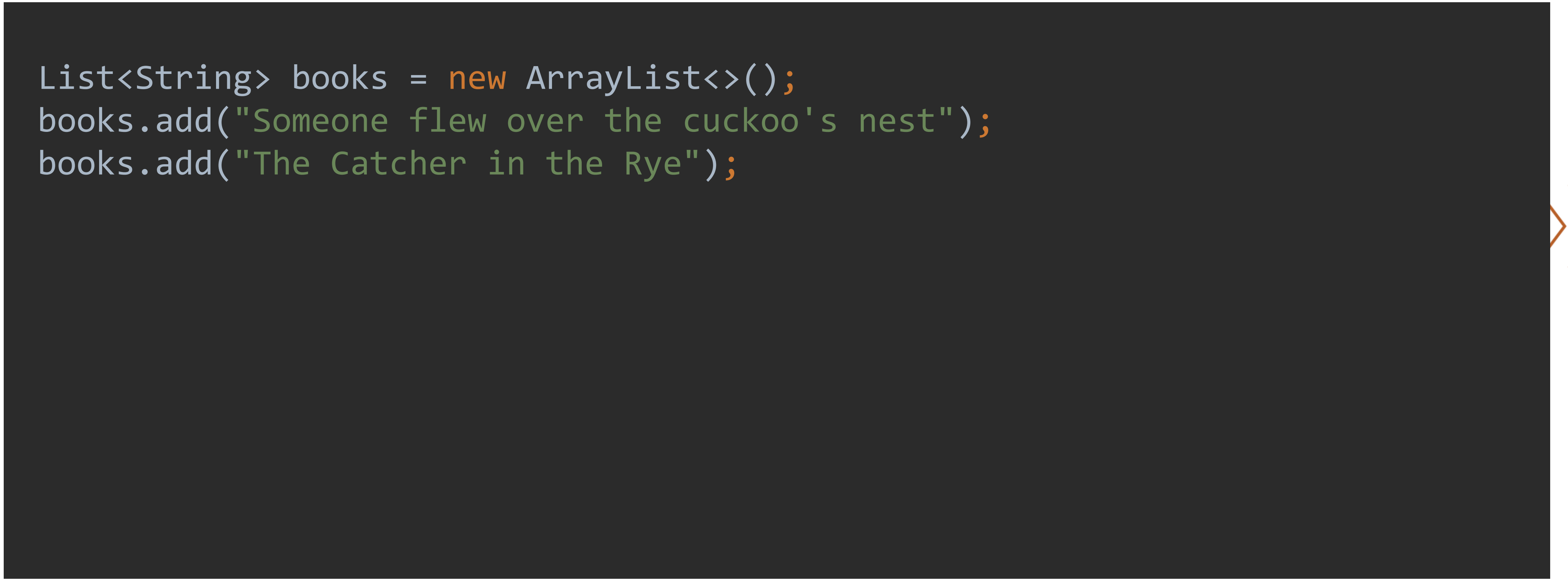
Specifies that list contains only specific type elements

BASIC LIST OPERATIONS

Method	Purpose
add(Object obj)	Adds a new element at the end of the list
add(Object obj, int index)	Adds a new element into the list at the given index
get(int index)	Returns the element at the given index
remove(Object obj)	Removes the first occurrence of the specified element from this list
remove(int index)	Removes the element at the given index

ADDING OBJECT TO LIST

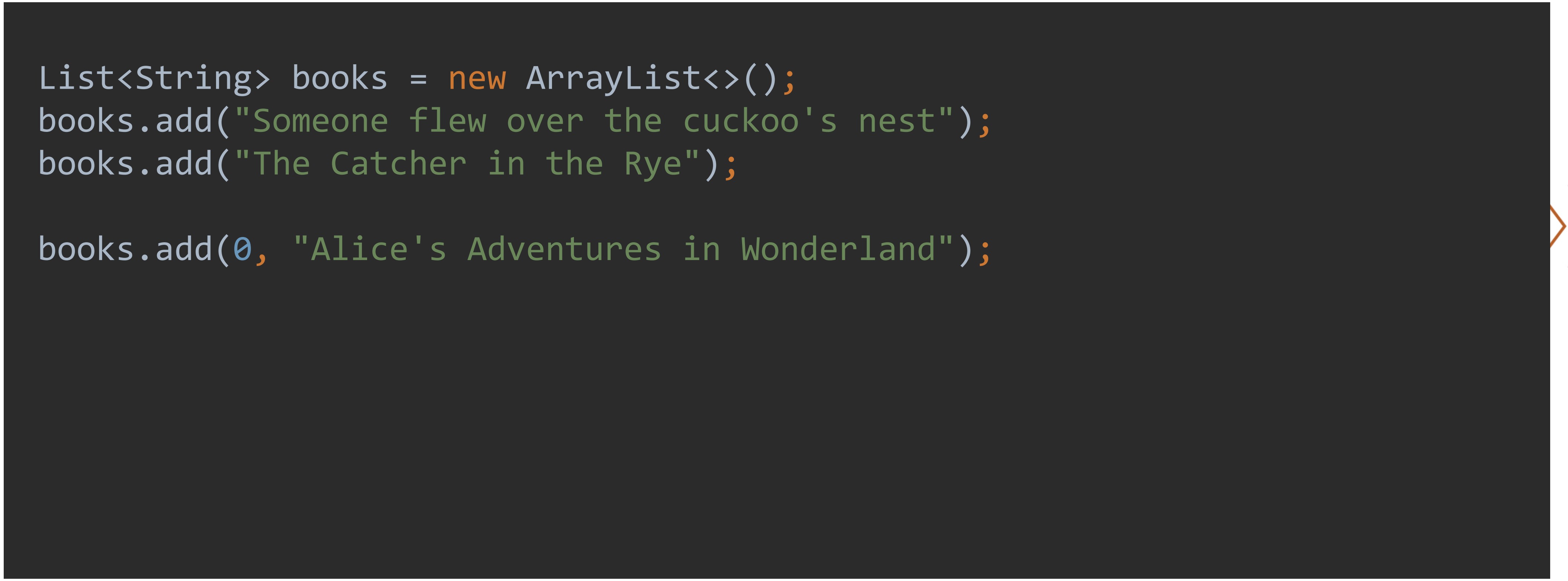
Code



```
List<String> books = new ArrayList<>();
books.add("Someone flew over the cuckoo's nest");
books.add("The Catcher in the Rye");
```

INSERTING OBJECT IN LIST

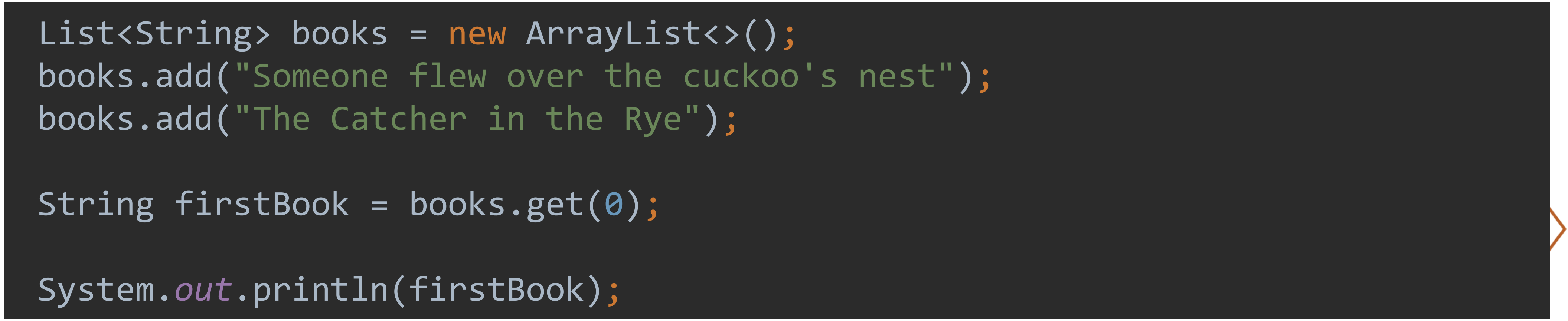
Code



```
List<String> books = new ArrayList<>();
books.add("Someone flew over the cuckoo's nest");
books.add("The Catcher in the Rye");

books.add(0, "Alice's Adventures in Wonderland");
```

RETRIEVING OBJECT FROM LIST

Code



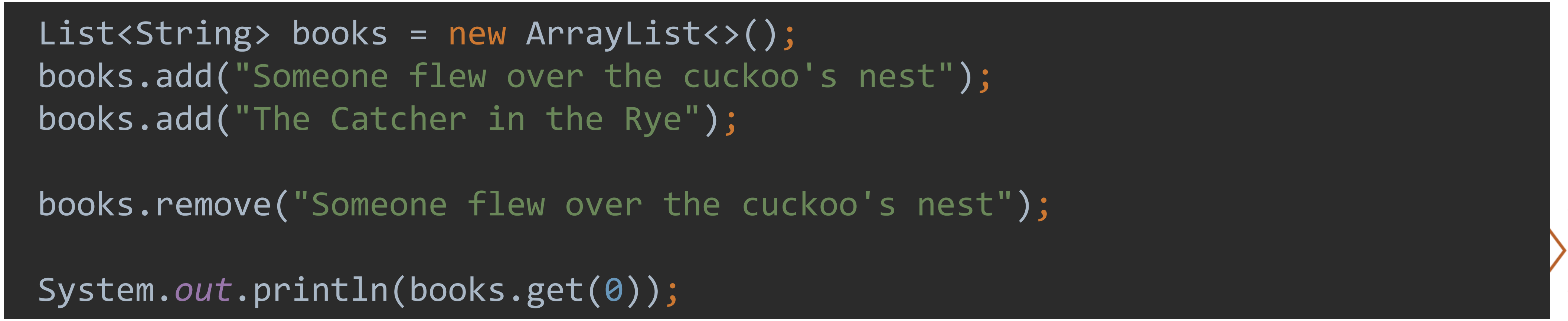
```
List<String> books = new ArrayList<>();
books.add("Someone flew over the cuckoo's nest");
books.add("The Catcher in the Rye");

String firstBook = books.get(0);

System.out.println(firstBook);
```

Console output

```
Someone Flew Over the Cuckoo’s Nest
```

REMOVING OBJECT FROM LIST

Code



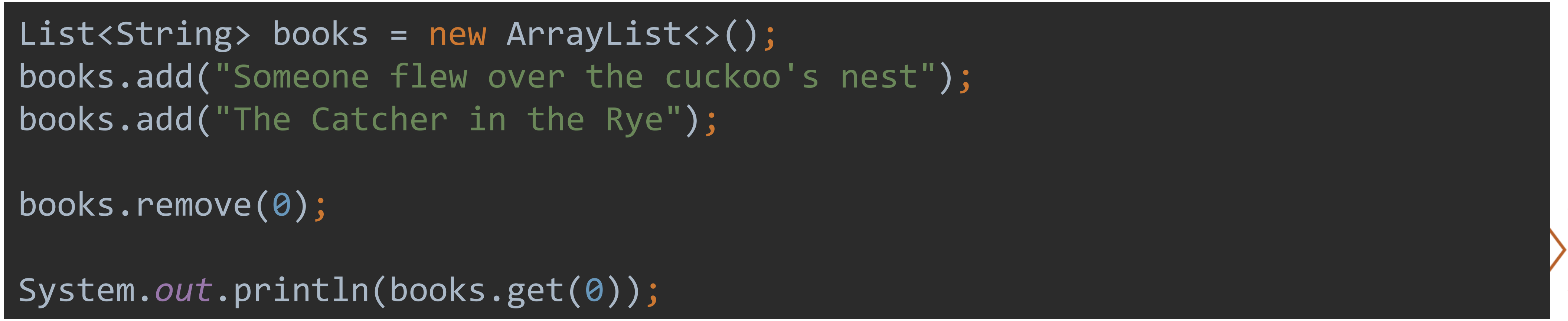
```java
List<String> books = new ArrayList<>();
books.add("Someone flew over the cuckoo's nest");
books.add("The Catcher in the Rye");

books.remove("Someone flew over the cuckoo's nest");

System.out.println(books.get(0));
```

Console output

```
The Catcher in the Rye
```

REMOVING OBJECT AT GIVEN INDEX

Code



```
List<String> books = new ArrayList<>();
books.add("Someone flew over the cuckoo's nest");
books.add("The Catcher in the Rye");

books.remove(0);

System.out.println(books.get(0));
```

Console output

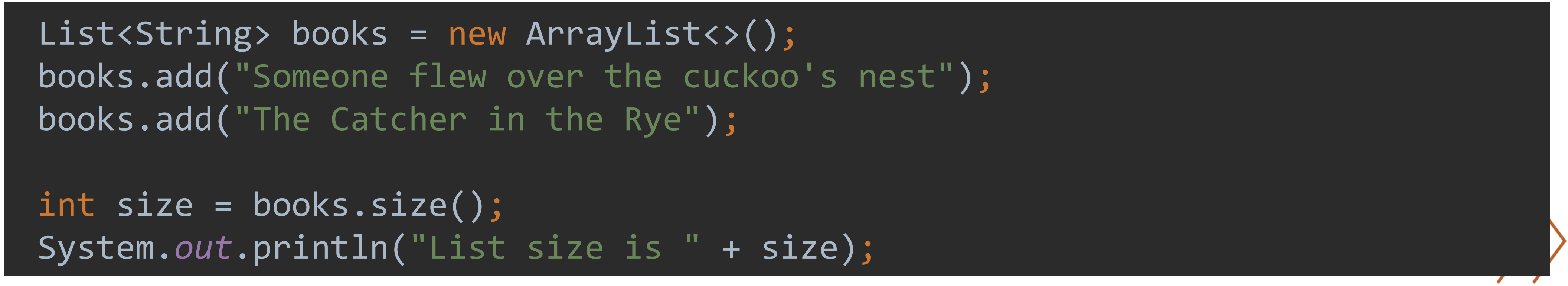
```
The Catcher in the Rye
```

LIST BASIC UTILITY METHODS

Method	Purpose
int size()	Number of elements in the list
boolean isEmpty()	True if the list is empty
contains(Object target)	True if the list contains the given target element
void clear()	Removes all the elements in the list
int indexOf(Object target)	Returns the int index of the first appearance of target in the list

RETRIEVING LIST SIZE

Code



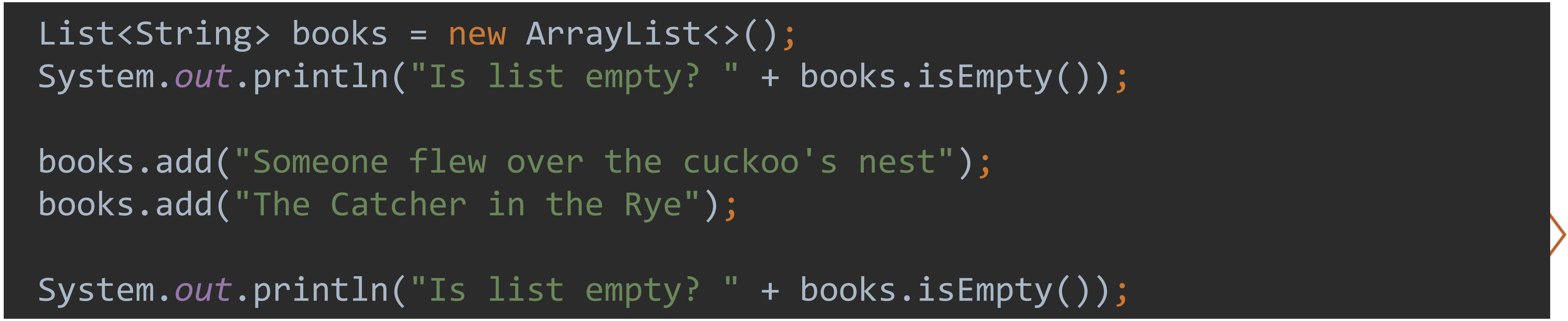
```java
List<String> books = new ArrayList<>();
books.add("Someone flew over the cuckoo's nest");
books.add("The Catcher in the Rye");

int size = books.size();
System.out.println("List size is " + size);
```

Console output

```
List size is 2
```

CHECKING IF LIST IS EMPTY

Code



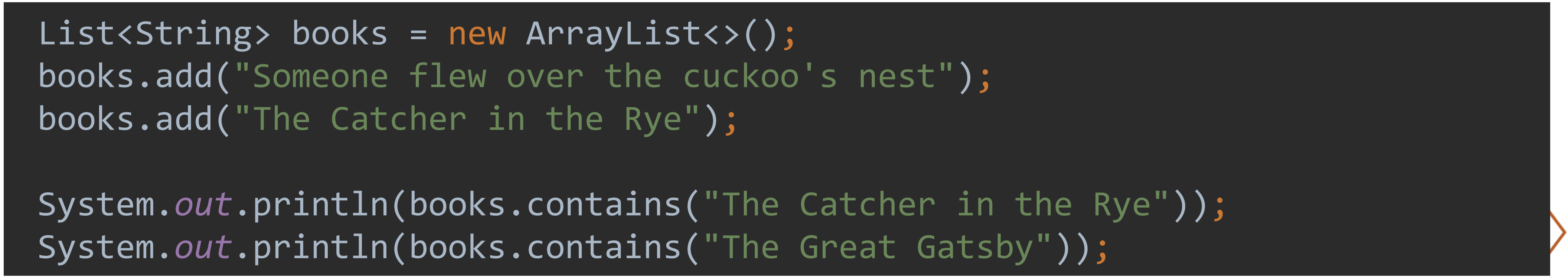
```java
List<String> books = new ArrayList<>();
System.out.println("Is list empty? " + books.isEmpty());

books.add("Someone flew over the cuckoo's nest");
books.add("The Catcher in the Rye");

System.out.println("Is list empty? " + books.isEmpty());
```

Console output

```
Is list empty? true
Is list empty? false
```

CHECKING IF LIST CONTAINS ELEMENT

Code



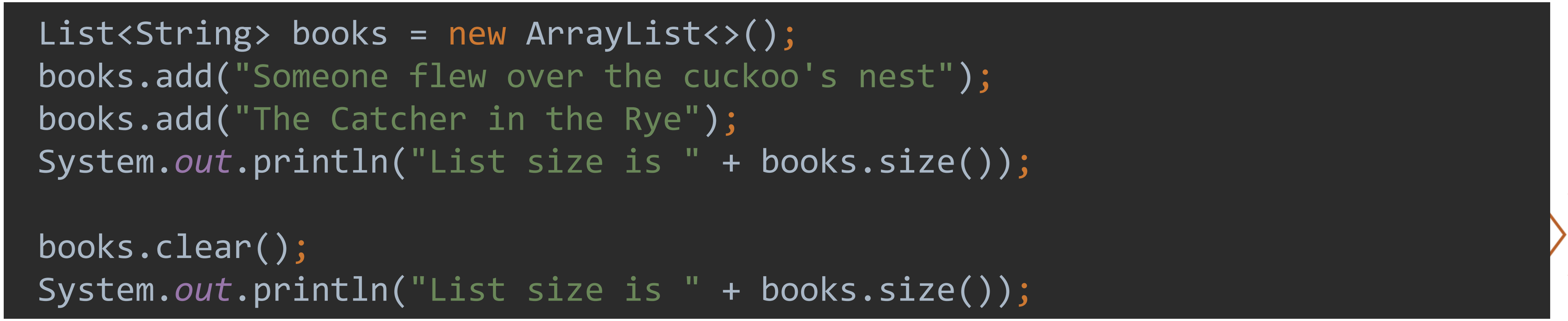
```java
List<String> books = new ArrayList<>();
books.add("Someone flew over the cuckoo's nest");
books.add("The Catcher in the Rye");

System.out.println(books.contains("The Catcher in the Rye"));
System.out.println(books.contains("The Great Gatsby"));
```

Console output

```
true
false
```

CLEARING LIST

Code



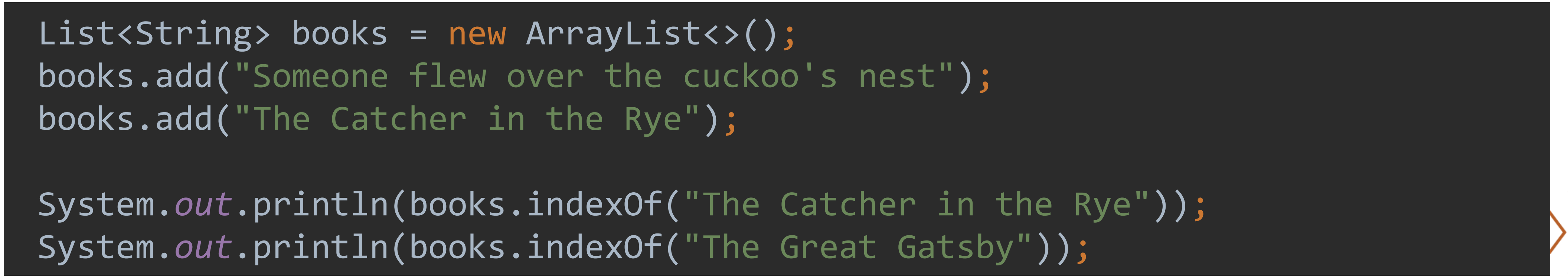
```java
List<String> books = new ArrayList<>();
books.add("Someone flew over the cuckoo's nest");
books.add("The Catcher in the Rye");
System.out.println("List size is " + books.size());

books.clear();
System.out.println("List size is " + books.size());
```

Console output

```
List size is 2
List size is 0
```

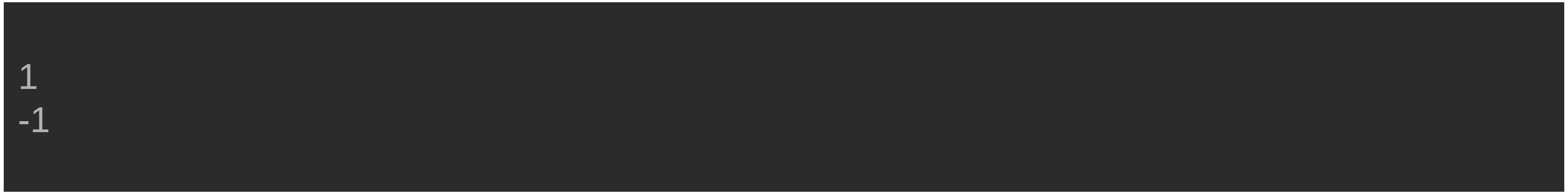
GETTING INDEX OF ELEMENT

Code



```java
List<String> books = new ArrayList<>();
books.add("Someone flew over the cuckoo's nest");
books.add("The Catcher in the Rye");

System.out.println(books.indexOf("The Catcher in the Rye"));
System.out.println(books.indexOf("The Great Gatsby"));
```

Console output



```
1
-1
```

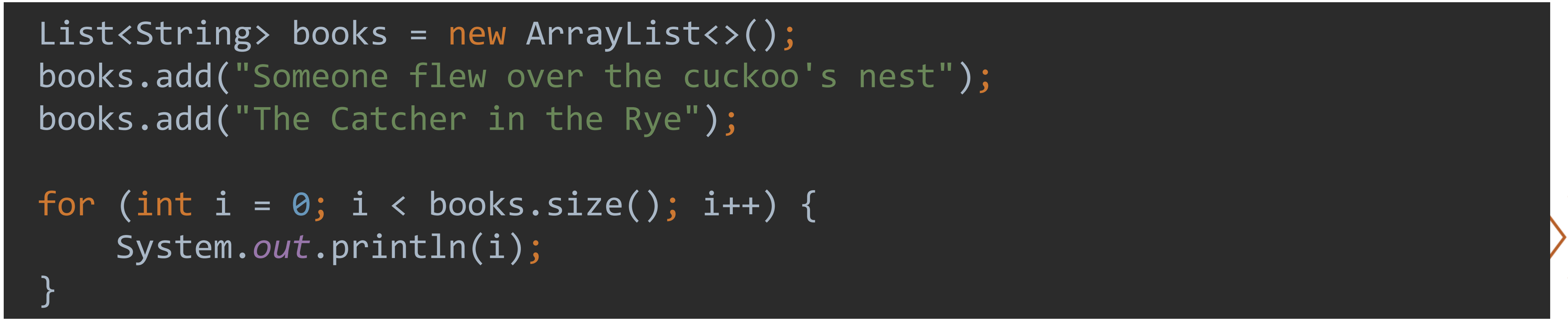
LOOPING THROUGH LIST ITEMS

- Java Collection interface and, therefore, List interface support iterative processing of its items

- For loop and For-Each loop are the most common iterative techniques applied to lists in Java.

LIST FOR LOOP EXAMPLE

Code



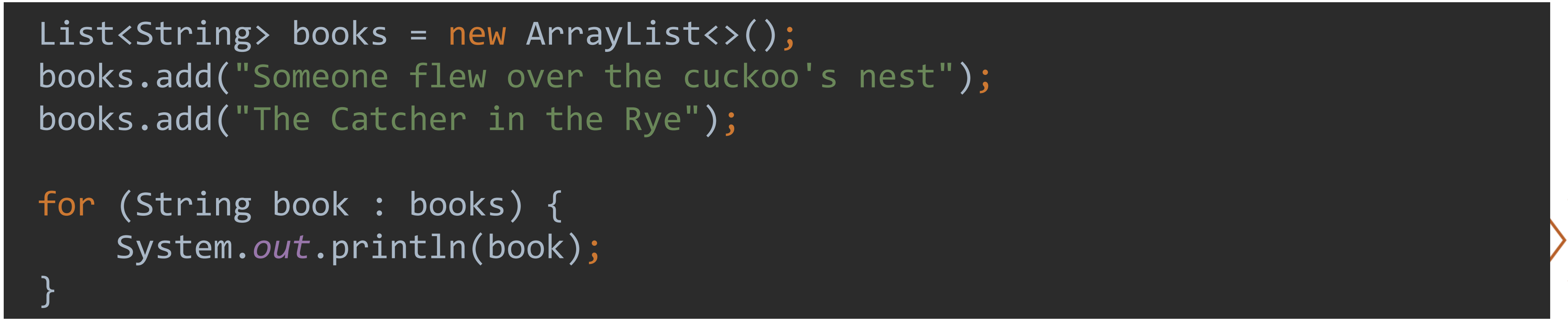
```java
List<String> books = new ArrayList<>();
books.add("Someone flew over the cuckoo's nest");
books.add("The Catcher in the Rye");

for (int i = 0; i < books.size(); i++) {
    System.out.println(i);
}
```

Console output

```
Someone Flew Over the Cuckoo’s Nest
The Catcher in the Rye
```

LIST FOR-EACH LOOP EXAMPLE

Code

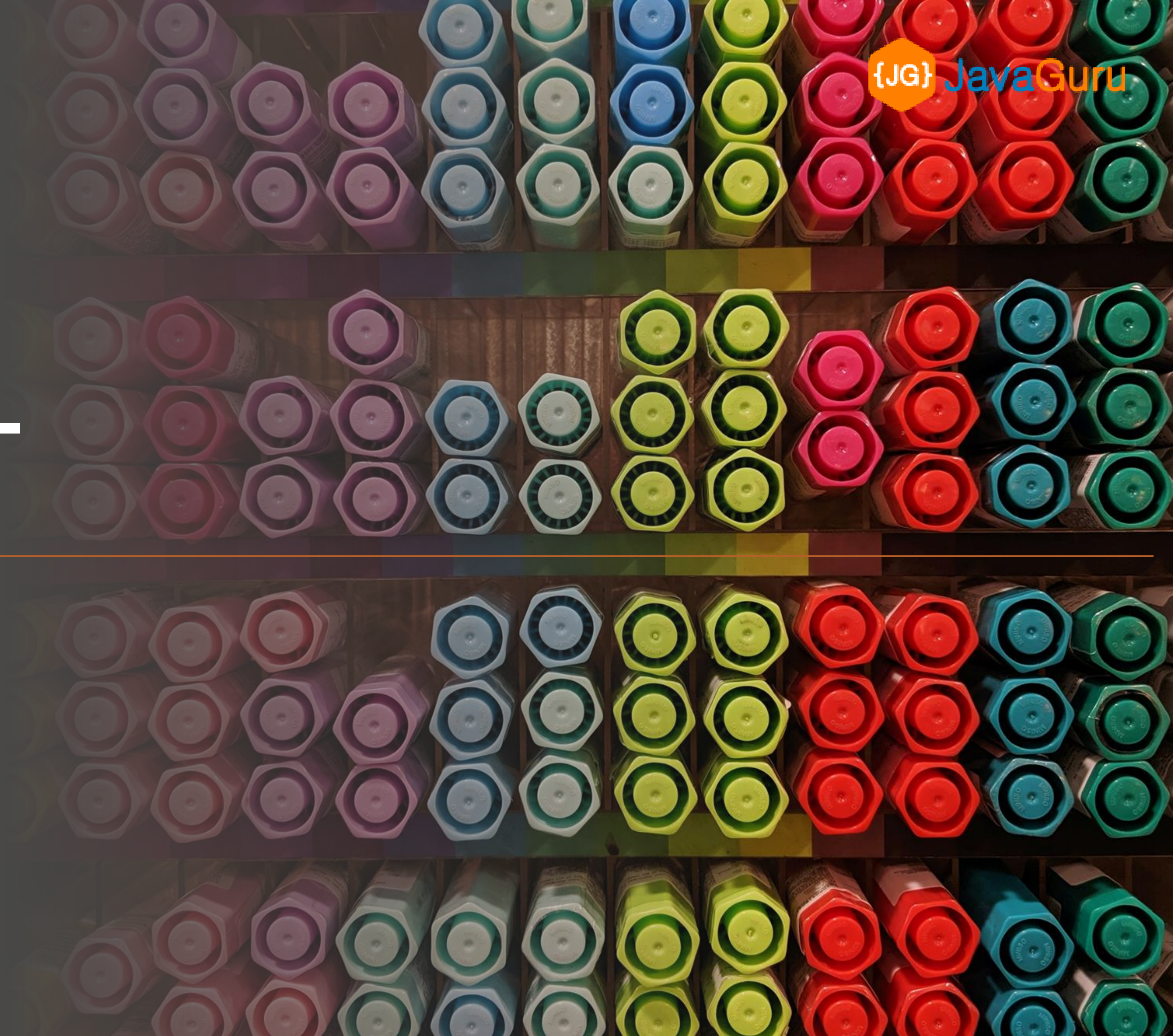


```java
List<String> books = new ArrayList<>();
books.add("Someone flew over the cuckoo's nest");
books.add("The Catcher in the Rye");

for (String book : books) {
    System.out.println(book);
}
```

Console output

```
Someone Flew Over the Cuckoo’s Nest
The Catcher in the Rye
```

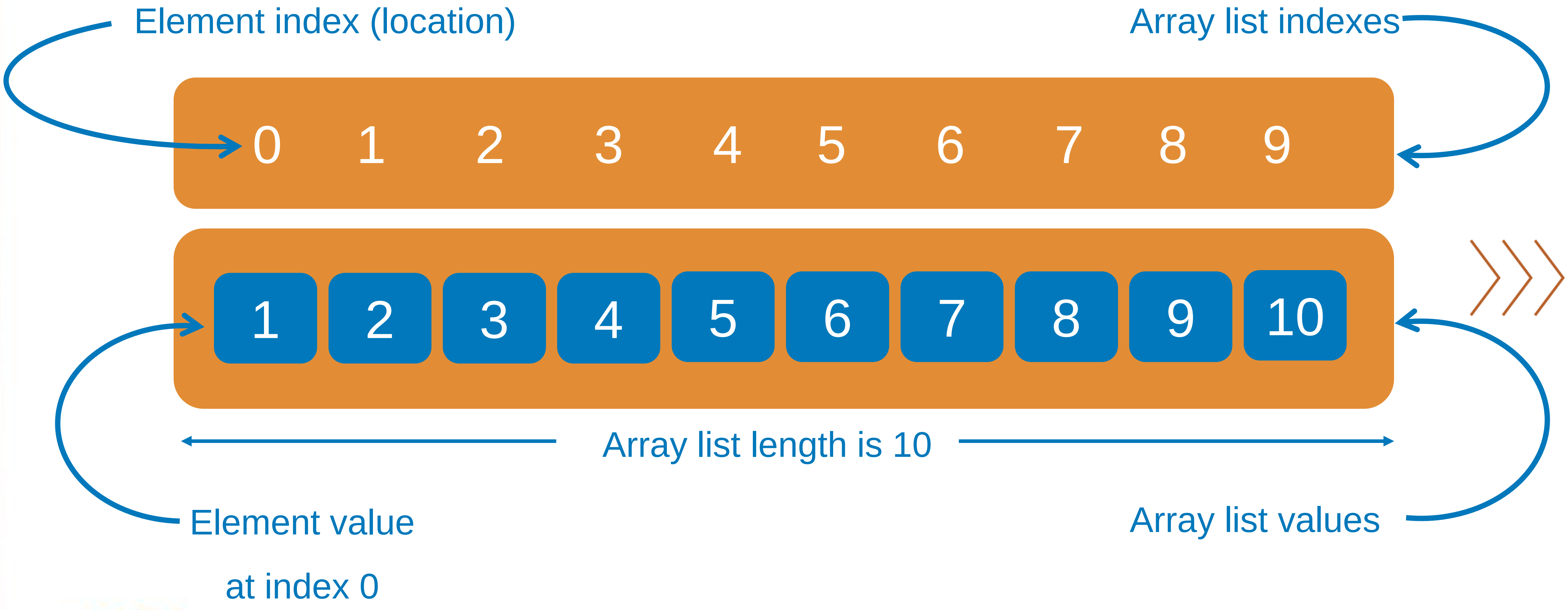
ARRAY LIST



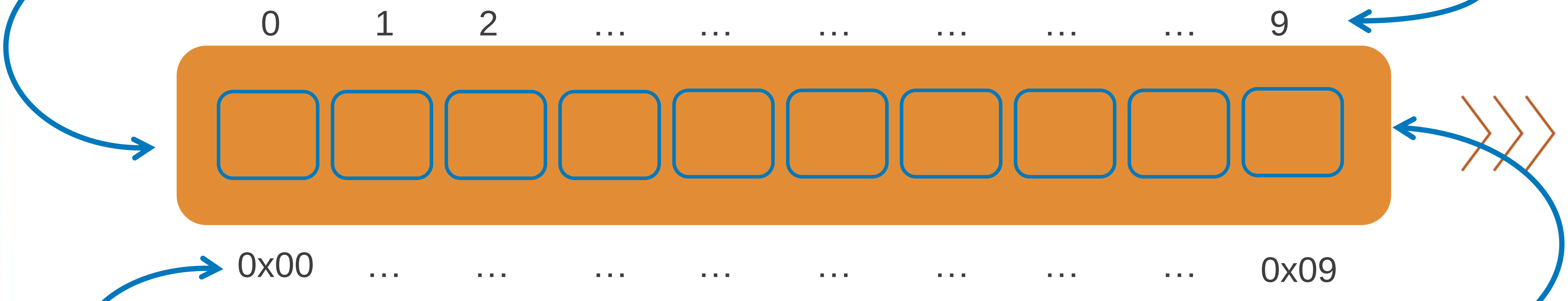
ARRAY LIST CHARACTERISTICS

- It is a resizable array, also called a dynamic array
- It internally uses an array to store the elements
- It allows duplicate values
- It is an ordered collection

ARRAY LIST VISUALISATION



ARRAY LIST INSERTION



Initial array size by default is 10

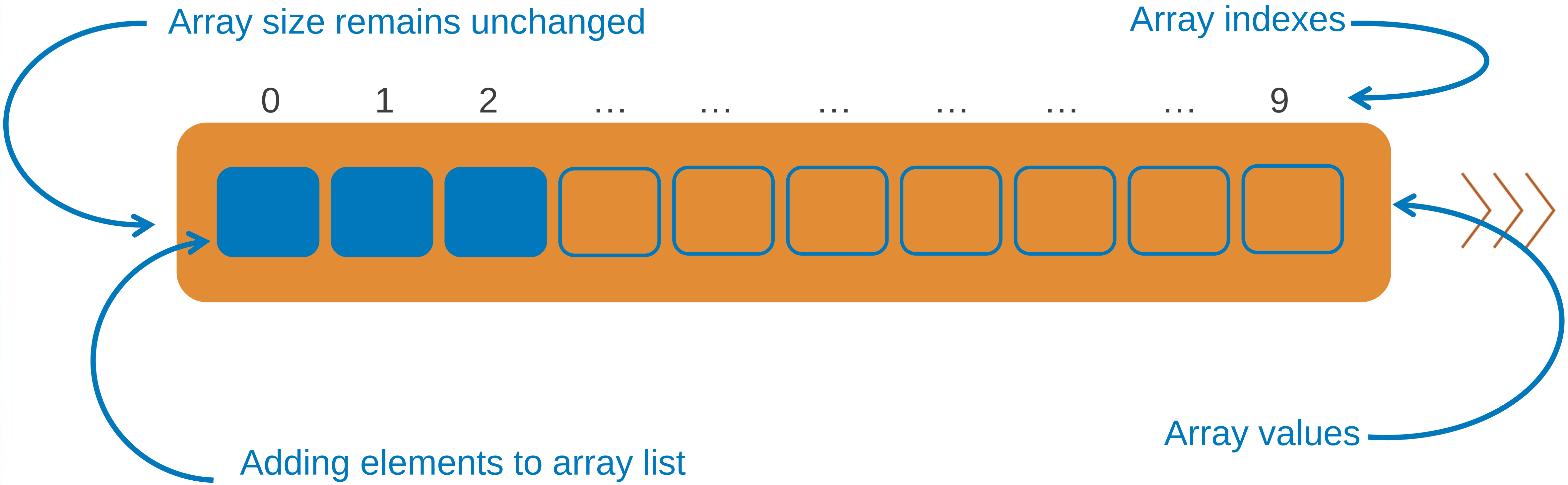
Array indexes

Array values

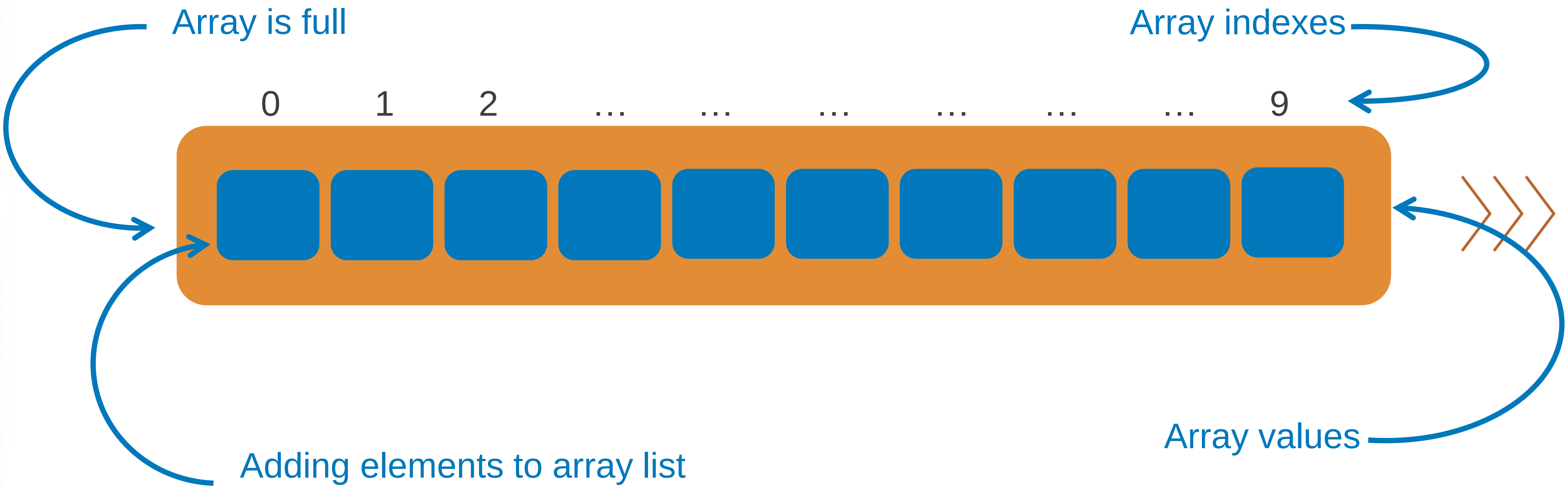
Memory allocated sequentially

for the whole array

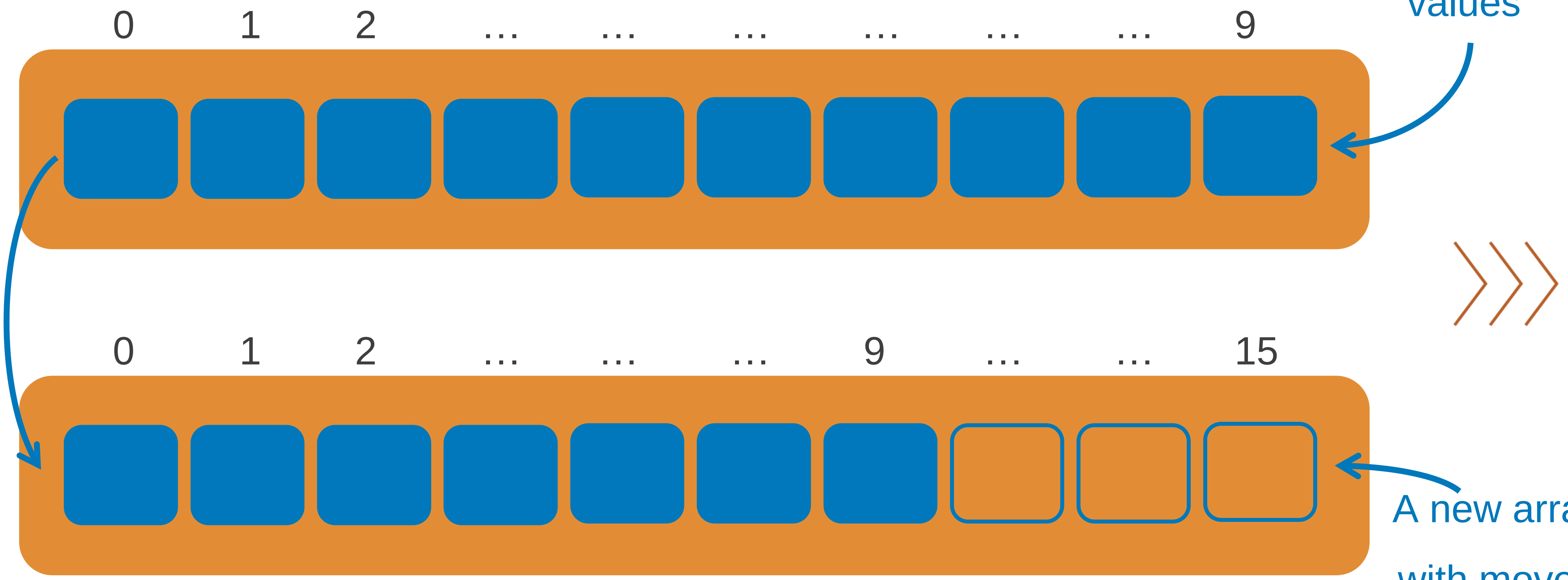
ARRAY LIST INSERTION



ARRAY LIST INSERTION



ARRAY LIST INSERTION



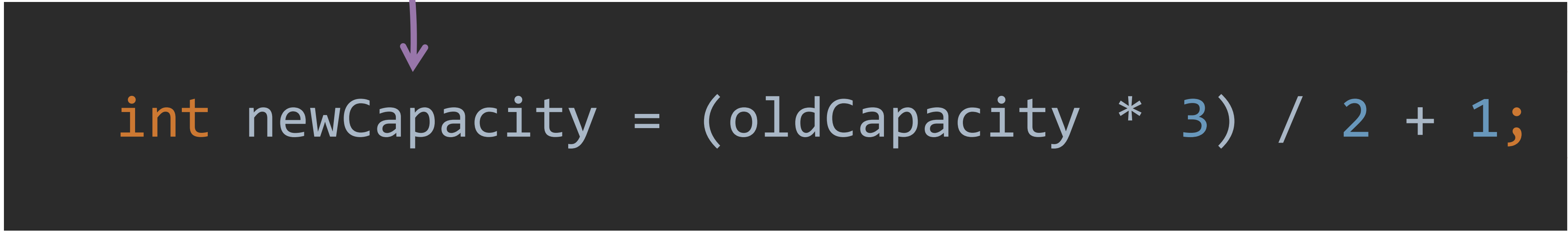
Old array values

Move elements to a new array

A new array with moved values

ARRAY LIST CAPACITY INCREMENT

Capacity is increased by roughly 50%



```
int newCapacity = (oldCapacity * 3) / 2 + 1;
```

ARRAY LIST WITH DEFAULT CAPACITY

Code

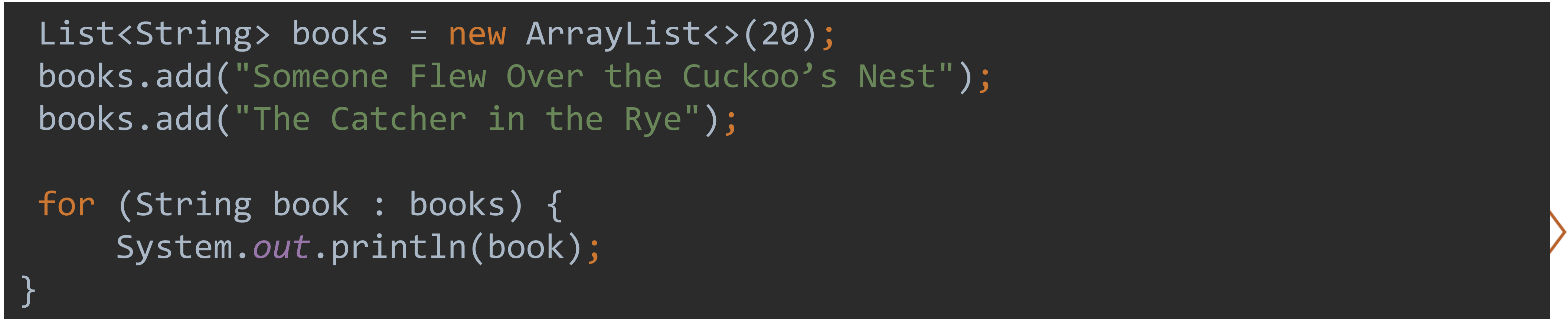
```
 List<String> books = new ArrayList<>();
 books.add("Someone Flew Over the Cuckoo’s Nest");
 books.add("The Catcher in the Rye");

 for (String book : books) {
     System.out.println(book);
}
```

Console output

```
Someone Flew Over the Cuckoo’s Nest
The Catcher in the Rye
```

ARRAY LIST WITH SPECIFIED CAPACITY

Code



```
 List<String> books = new ArrayList<>(20);
 books.add("Someone Flew Over the Cuckoo’s Nest");
 books.add("The Catcher in the Rye");

 for (String book : books) {
     System.out.println(book);
}
```

Console output

```
Someone Flew Over the Cuckoo’s Nest
The Catcher in the Rye
```

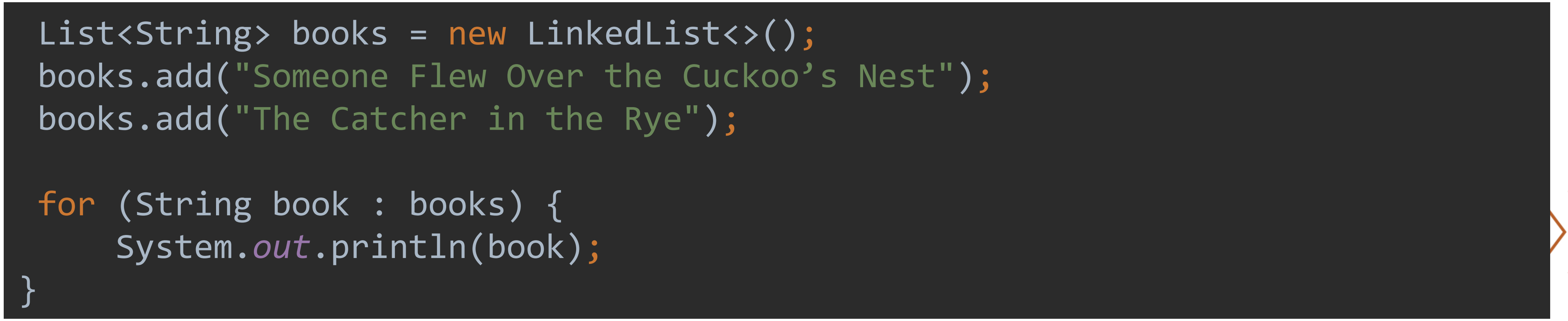
LINKED LIST

LINKED LIST

- Internally uses distinct objects which are referencing each other
- It allows duplicate and null values
- It is an ordered collection

LINKED LIST EXAMPLE

Code



```
 List<String> books = new LinkedList<>();
 books.add("Someone Flew Over the Cuckoo’s Nest");
 books.add("The Catcher in the Rye");

 for (String book : books) {
     System.out.println(book);
}
```

Console output

```
Someone Flew Over the Cuckoo’s Nest
The Catcher in the Rye
```

LINKED LIST VS ARRAY LIST

ARRAY LIST AND LINKED LIST COMPARISON

- Memory consumption:
  - LinkedList consumes more memory than an ArrayList because it also stores the next and previous references along with the data

ARRAY LIST AND LINKED LIST COMPARISON

- Accessing data:
  - An element can be accessed in an ArrayList in O(1) time (directly by index)
  - It takes O(n) time to access an element in a LinkedList (traverse to the desired element through references)

ARRAY LIST AND LINKED LIST COMPARISON

- Addition or removal:
  - ArrayList is usually slower because the elements in the ArrayList need to be shifted if an element is added or removed in the middle (capacity changes matter as well)
  - LinkedList is faster because only references must be changed

REFERENCES

REFERENCES

- https://www.callicoder.com/java-arraylist/
- https://www.callicoder.com/java-linkedlist/
- https://www.netjstech.com/2015/08/how-arraylist-works-internally-in-java.html
- https://www.netjstech.com/2015/08/how-linked-list-class-works-internally-java.html

QUESTIONS?

THANK YOU!