Java Basic for Tester

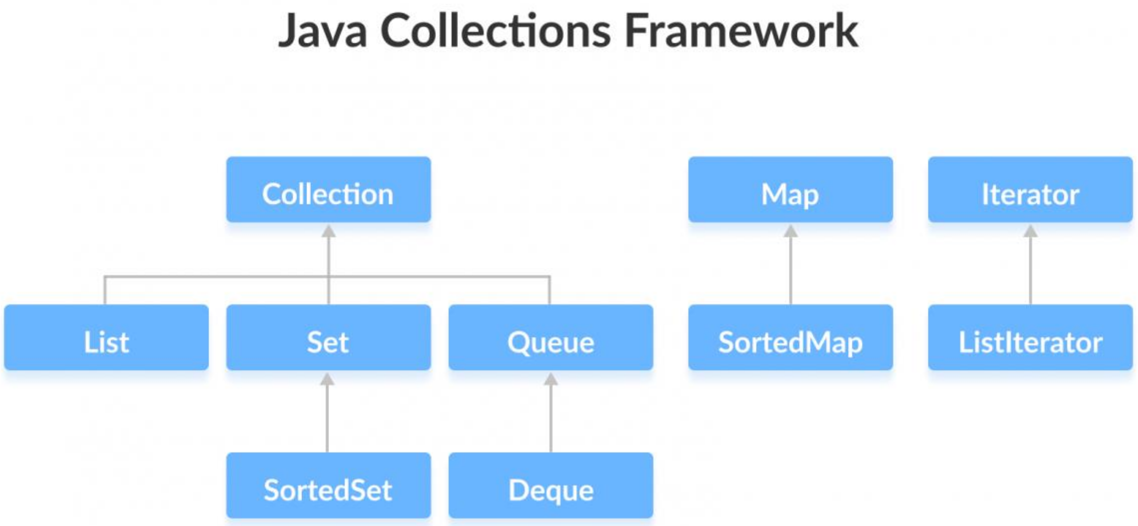
Java List

Agenda

- *Java Collections Framework*
- *Java Collection Interface*
- *Java List Interface*
- *Java ArrayList*
- *Java Vector*
- *Java Stack*

Java Collections Framework

The Java collections framework provides a set of interfaces and classes to implement various data structures and algorithms.

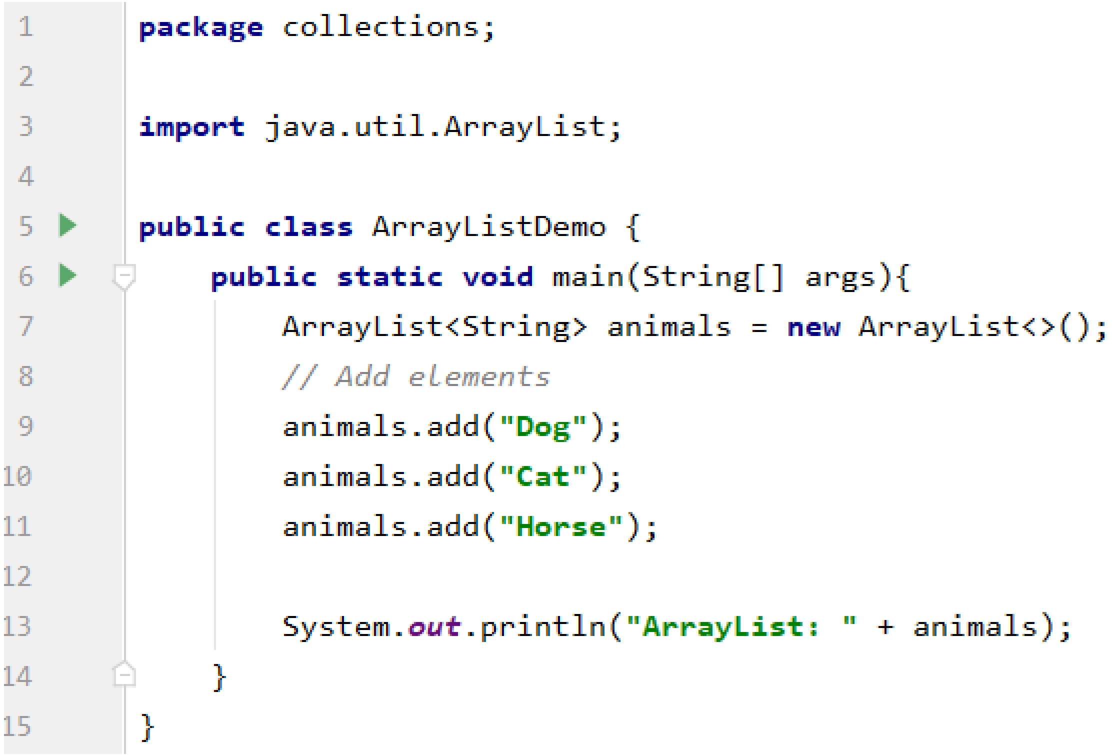


Why the Collections Framework?

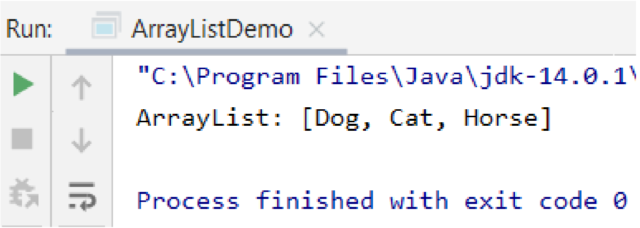
1. We do not have to write code to implement these data structures and algorithms manually.

2. Our code will be much more efficient as the collections framework is highly optimized.

Java Collections Framework



```java
package collections;

import java.util.ArrayList;

public class ArrayListDemo {
    public static void main(String[] args){
        ArrayList<String> animals = new ArrayList<>();
        // Add elements
        animals.add("Dog");
        animals.add("Cat");
        animals.add("Horse");

        System.out.println("ArrayList: " + animals);
    }
}
```

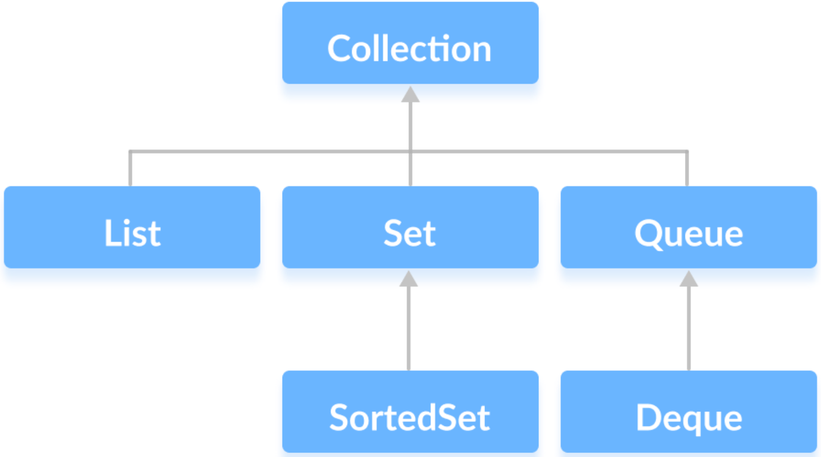


```
"C:\Program Files\Java\jdk-14.0.1\
ArrayList: [Dog, Cat, Horse]

Process finished with exit code 0
```

Java Collection Interface

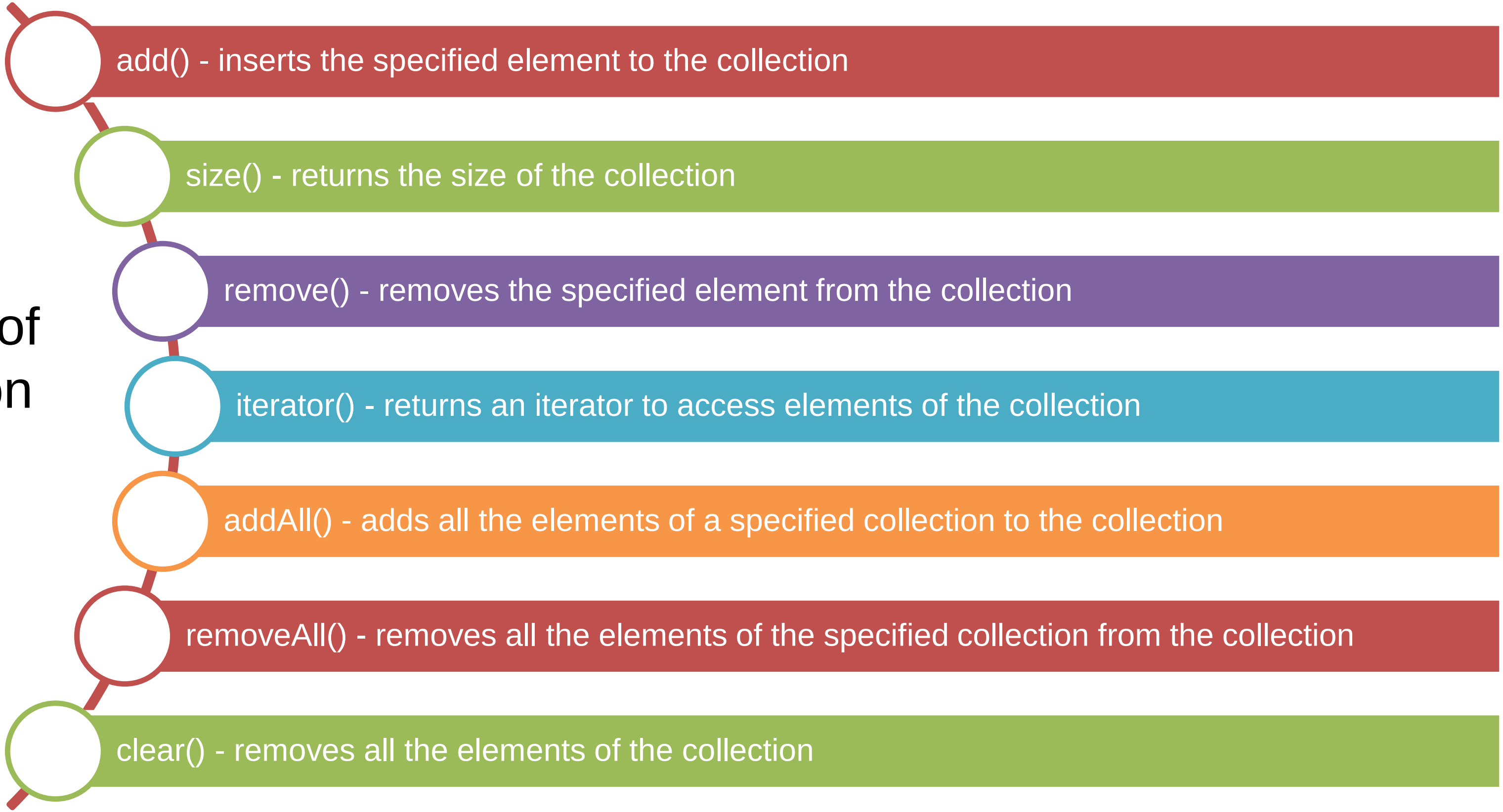
The Collection interface is the root interface of the Java collections framework.

There is no direct implementation of this interface. However, it is implemented through its subinterfaces like List, Set, and Queue.



Java Collection Interface

Methods of Collection



add() - inserts the specified element to the collection

size() - returns the size of the collection

remove() - removes the specified element from the collection

iterator() - returns an iterator to access elements of the collection

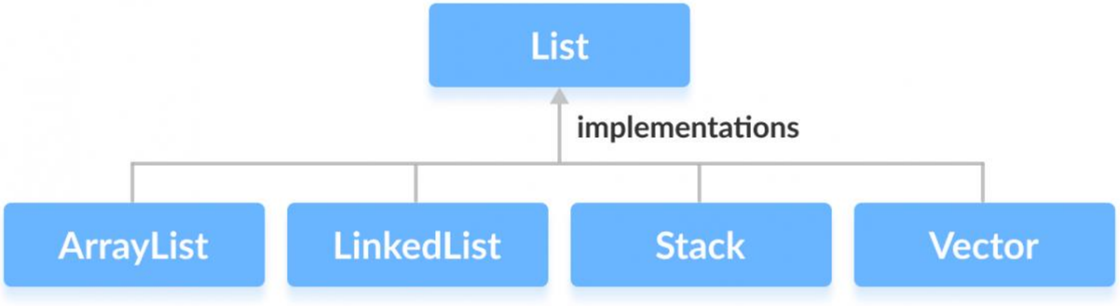
addAll() - adds all the elements of a specified collection to the collection

removeAll() - removes all the elements of the specified collection from the collection

clear() - removes all the elements of the collection

Java List

In Java, the List interface is an ordered collection that allows us to store and access elements sequentially. It extends the Collection interface.



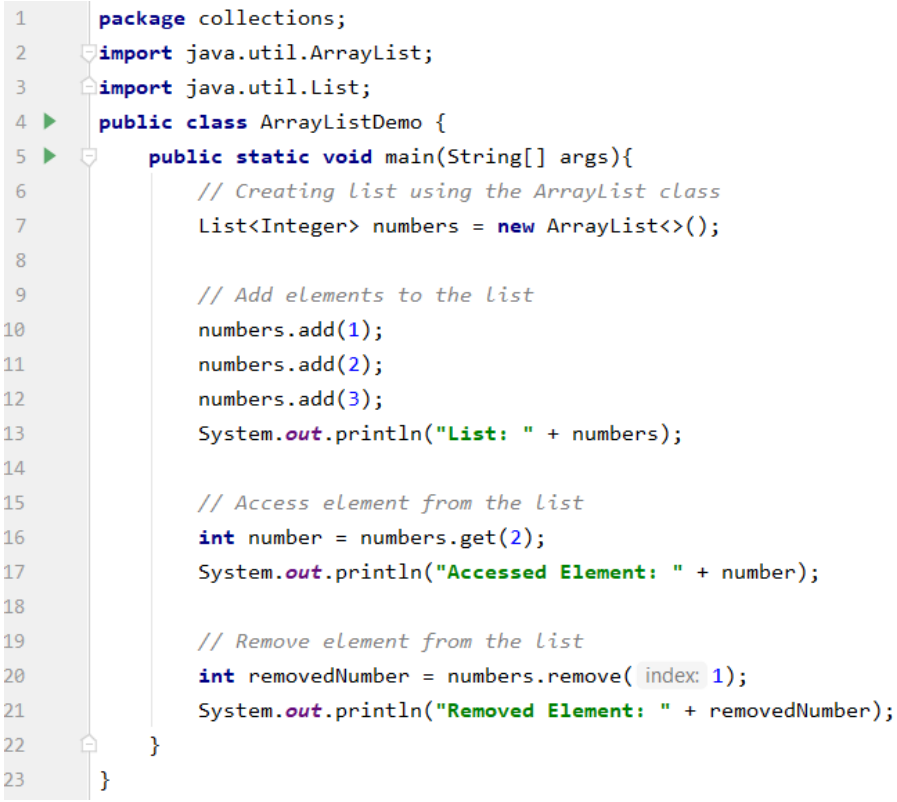
```
// ArrayList implementation of List
List<String> list1 = new ArrayList<>();

// LinkedList implementation of List
List<String> list2 = new LinkedList<>();
```

Java Collection Interface

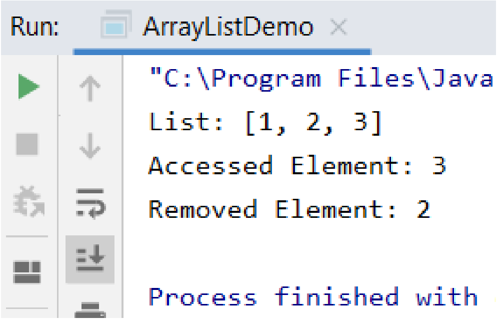
Methods of List

add() - adds an element to a list
addAll() - adds all elements of one list to another
get() - helps to randomly access elements from lists
iterator() - returns iterator object that can be used to sequentially access elements of lists
set() - changes elements of lists
remove() - removes an element from the list
removeAll() - removes all the elements from the list
clear() - removes all the elements from the list (more efficient than removeAll())
size() - returns the length of lists
toArray() - converts a list into an array
contains() - returns true if a list contains specified element

Java Collection Interface

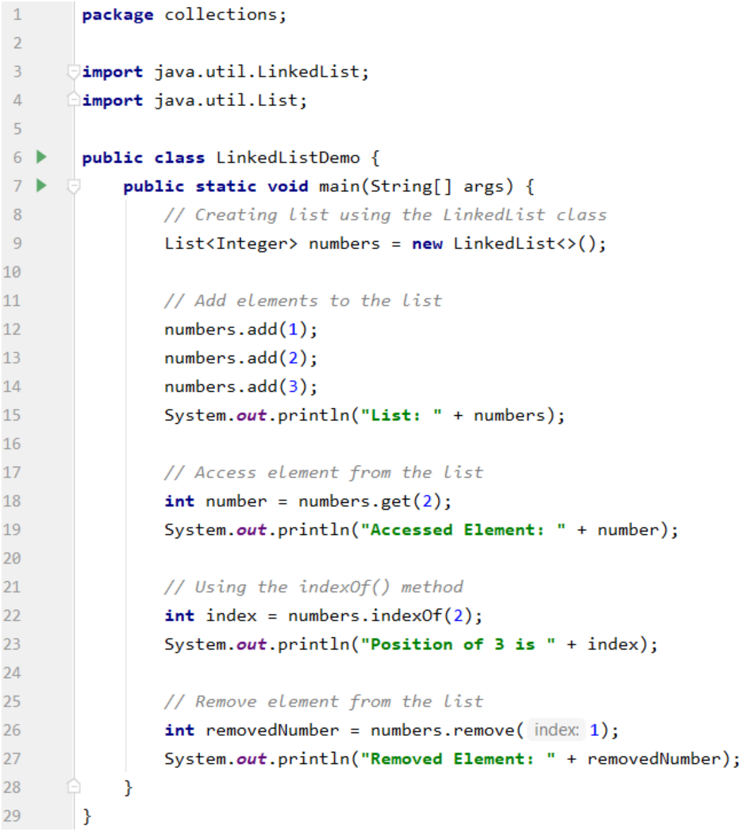


```java
package collections;
import java.util.ArrayList;
import java.util.List;
public class ArrayListDemo {
    public static void main(String[] args){
        // Creating list using the ArrayList class
        List<Integer> numbers = new ArrayList<>();

        // Add elements to the list
        numbers.add(1);
        numbers.add(2);
        numbers.add(3);
        System.out.println("List: " + numbers);

        // Access element from the list
        int number = numbers.get(2);
        System.out.println("Accessed Element: " + number);

        // Remove element from the list
        int removedNumber = numbers.remove( index: 1);
        System.out.println("Removed Element: " + removedNumber);
    }
}
```

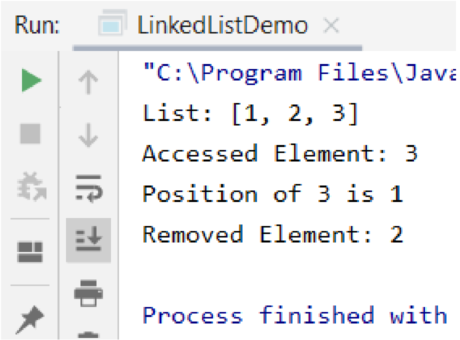


```
"C:\Program Files\Java
List: [1, 2, 3]
Accessed Element: 3
Removed Element: 2

Process finished with
```

Java Collection Interface



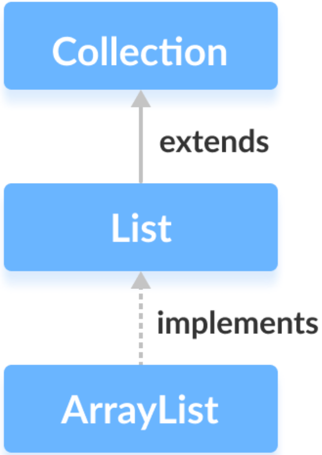
```java
package collections;

import java.util.LinkedList;
import java.util.List;

public class LinkedListDemo {
    public static void main(String[] args) {
        // Creating list using the LinkedList class
        List<Integer> numbers = new LinkedList<>();

        // Add elements to the list
        numbers.add(1);
        numbers.add(2);
        numbers.add(3);
        System.out.println("List: " + numbers);

        // Access element from the list
        int number = numbers.get(2);
        System.out.println("Accessed Element: " + number);

        // Using the indexOf() method
        int index = numbers.indexOf(2);
        System.out.println("Position of 3 is " + index);

        // Remove element from the list
        int removedNumber = numbers.remove( index: 1);
        System.out.println("Removed Element: " + removedNumber);
    }
}
```



```
"C:\Program Files\Java
List: [1, 2, 3]
Accessed Element: 3
Position of 3 is 1
Removed Element: 2

Process finished with
```

Java ArrayList Class

The ArrayList class is an implementation of the List interface that allows us to create resizable-arrays.



Java ArrayList Class

Java Array Vs ArrayList

In Java, we need to declare the size of an array before we can use it. Once the size of an array is declared, it's hard to change it.

To handle this issue, we can use the ArrayList class. The ArrayList class present in the java.util package allows us to create resizable arrays.

Unlike arrays, array lists (objects of the ArrayList class) can automatically adjust its capacity when we add or remove elements from it. Hence, array lists are also known as dynamic arrays.

Java ArrayList Class

Creating an ArrayList

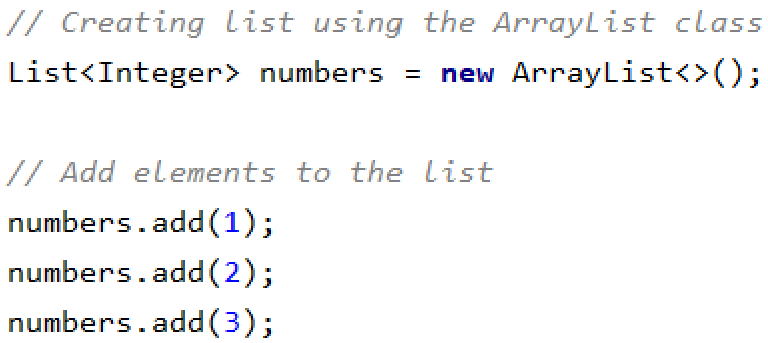
```
// create Integer type arraylist
ArrayList<Integer> arrayList = new ArrayList<>();

// create String type arraylist
ArrayList<String> arrayList = new ArrayList<>();
```

Note: We can not create array lists of primitive data types like int, float, char, etc. Instead, we have to use their corresponding wrapper class.

Java ArrayList Class

Add Elements to an ArrayList

1. Using the add() method

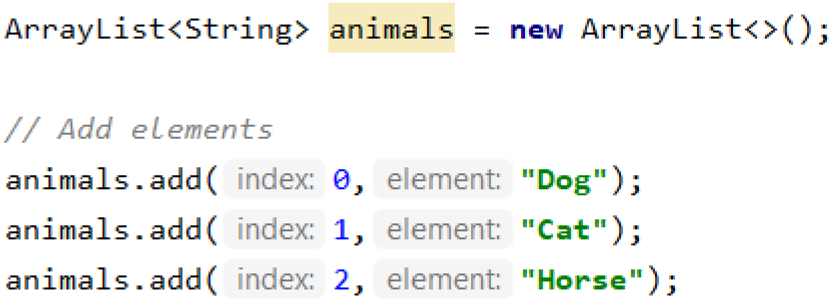


```java
// Creating list using the ArrayList class
List<Integer> numbers = new ArrayList<>();

// Add elements to the list
numbers.add(1);
numbers.add(2);
numbers.add(3);
```

Java ArrayList Class

Add Elements to an ArrayList

2. Using index number

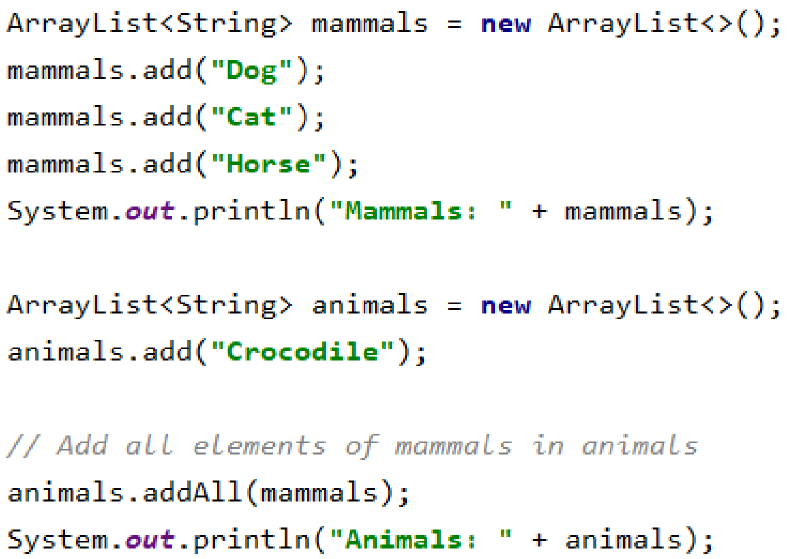


```
ArrayList<String> animals = new ArrayList<>();

// Add elements
animals.add( index: 0,  element: "Dog");
animals.add( index: 1,  element: "Cat");
animals.add( index: 2,  element: "Horse");
```

Java ArrayList Class

Add Elements to an ArrayList

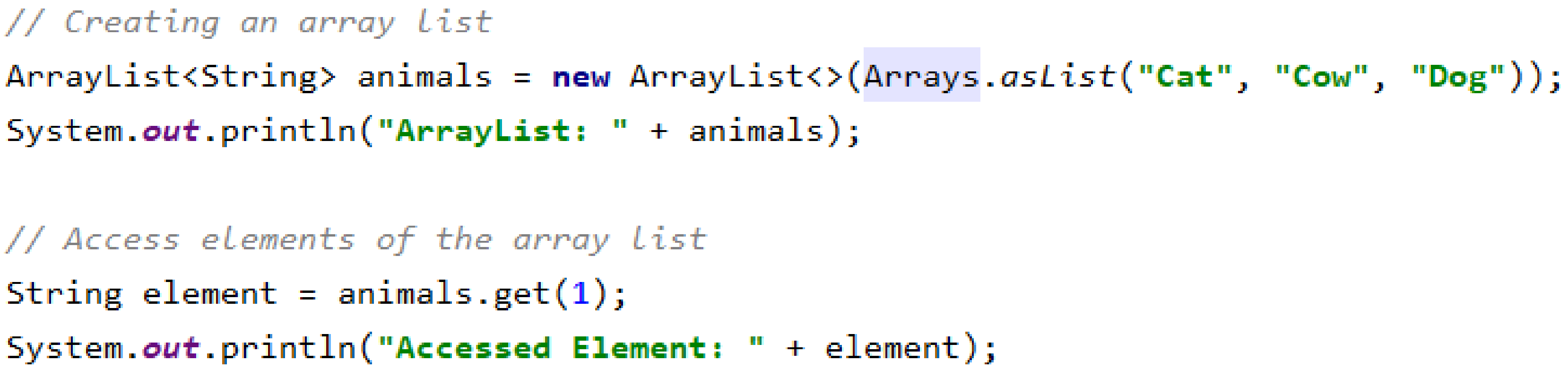
3. Add elements of an array list to another array list



```
ArrayList<String> mammals = new ArrayList<>();
mammals.add("Dog");
mammals.add("Cat");
mammals.add("Horse");
System.out.println("Mammals: " + mammals);

ArrayList<String> animals = new ArrayList<>();
animals.add("Crocodile");

// Add all elements of mammals in animals
animals.addAll(mammals);
System.out.println("Animals: " + animals);
```

Java ArrayList Class

Initialize an ArrayList Using asList()

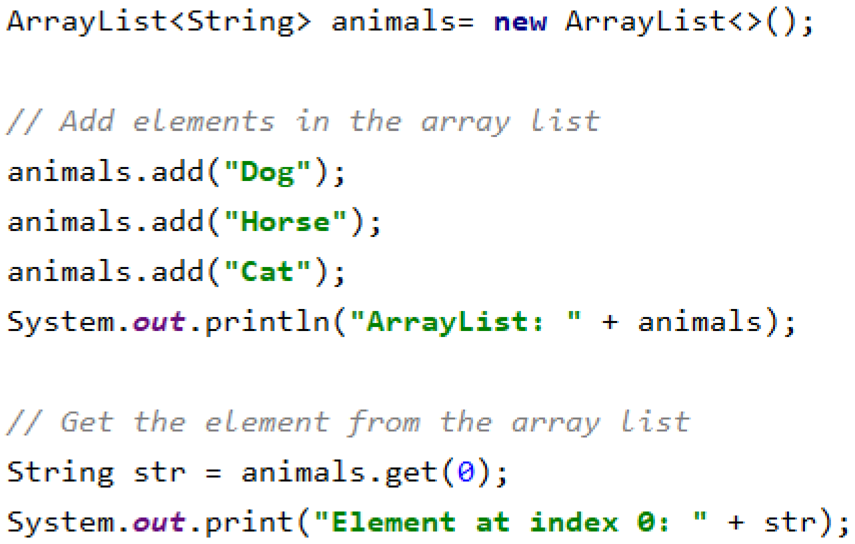


```
// Creating an array list
ArrayList<String> animals = new ArrayList<>(Arrays.asList("Cat", "Cow", "Dog"));
System.out.println("ArrayList: " + animals);

// Access elements of the array list
String element = animals.get(1);
System.out.println("Accessed Element: " + element);
```

Java ArrayList Class

Access ArrayList Elements

1. Using get() Method

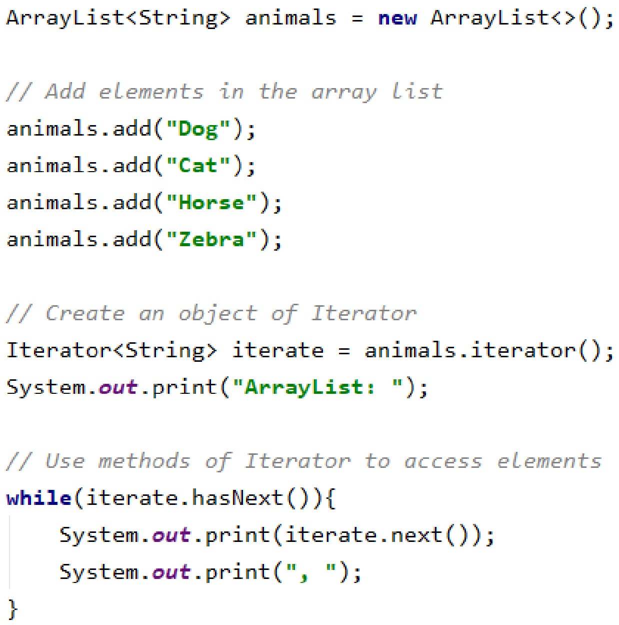


```
ArrayList<String> animals= new ArrayList<>();

// Add elements in the array list
animals.add("Dog");
animals.add("Horse");
animals.add("Cat");
System.out.println("ArrayList: " + animals);

// Get the element from the array list
String str = animals.get(0);
System.out.print("Element at index 0: " + str);
```

Java ArrayList Class

Access ArrayList Elements

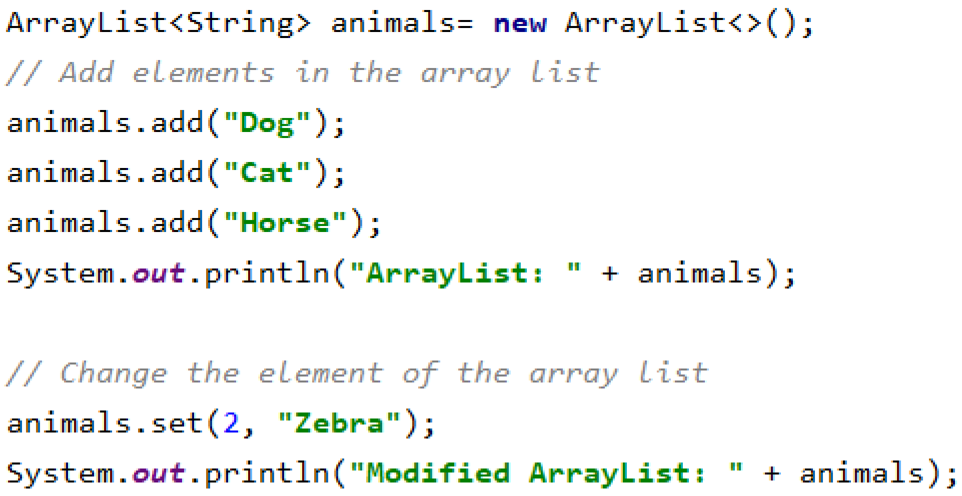
2. Using iterator() Method



```
ArrayList<String> animals = new ArrayList<>();

// Add elements in the array list
animals.add("Dog");
animals.add("Cat");
animals.add("Horse");
animals.add("Zebra");

// Create an object of Iterator
Iterator<String> iterate = animals.iterator();
System.out.print("ArrayList: ");

// Use methods of Iterator to access elements
while(iterate.hasNext()){
    System.out.print(iterate.next());
    System.out.print(", ");
}
```

Java ArrayList Class

Change ArrayList Elements

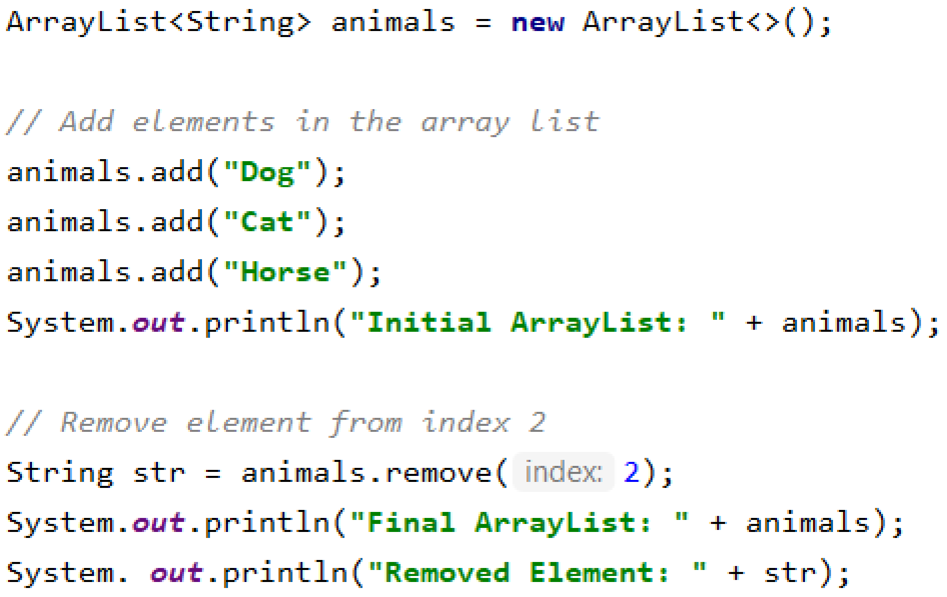


```
ArrayList<String> animals= new ArrayList<>();
// Add elements in the array list
animals.add("Dog");
animals.add("Cat");
animals.add("Horse");
System.out.println("ArrayList: " + animals);

// Change the element of the array list
animals.set(2, "Zebra");
System.out.println("Modified ArrayList: " + animals);
```

Java ArrayList Class

Remove ArrayList Elements

1. Using remove() Method

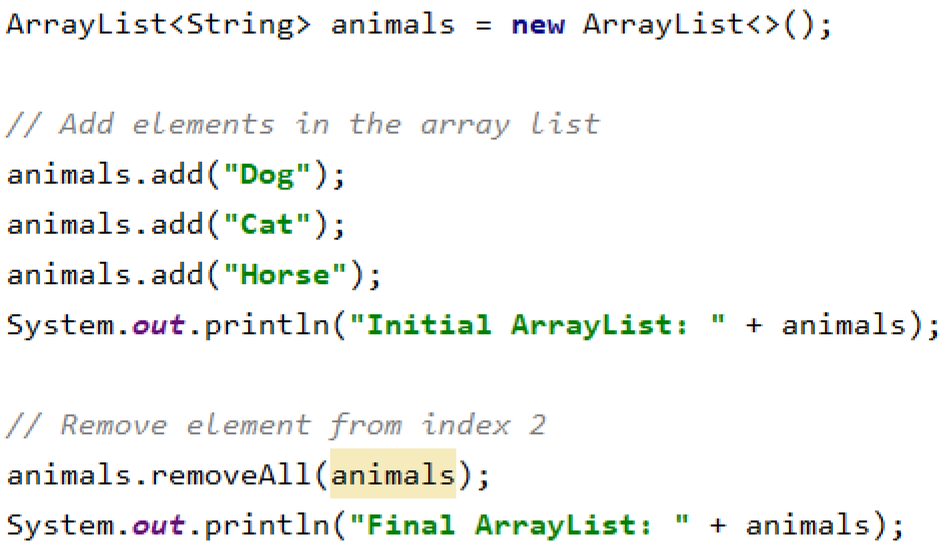


```
ArrayList<String> animals = new ArrayList<>();

// Add elements in the array list
animals.add("Dog");
animals.add("Cat");
animals.add("Horse");
System.out.println("Initial ArrayList: " + animals);

// Remove element from index 2
String str = animals.remove( index: 2);
System.out.println("Final ArrayList: " + animals);
System. out.println("Removed Element: " + str);
```

Java ArrayList Class

Remove ArrayList Elements

2. Using removeAll() method

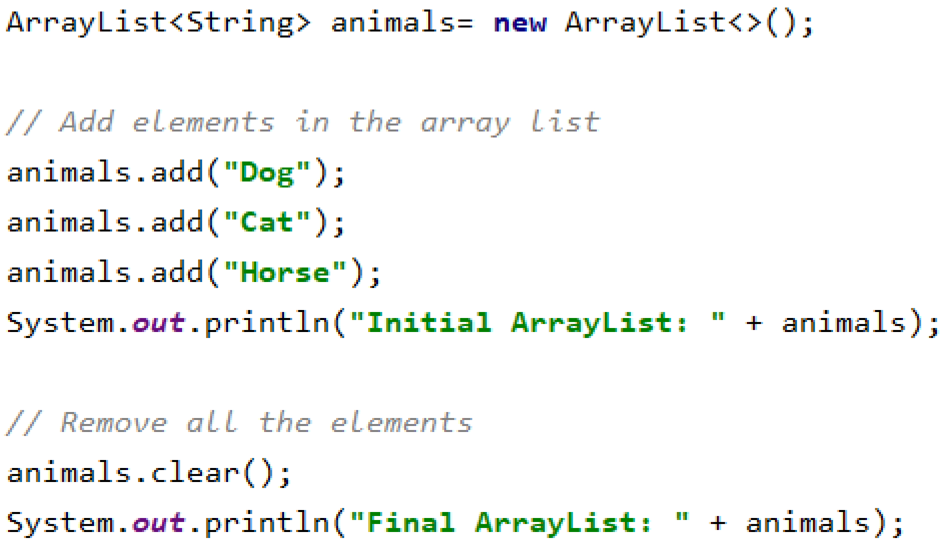


```
ArrayList<String> animals = new ArrayList<>();

// Add elements in the array list
animals.add("Dog");
animals.add("Cat");
animals.add("Horse");
System.out.println("Initial ArrayList: " + animals);

// Remove element from index 2
animals.removeAll(animals);
System.out.println("Final ArrayList: " + animals);
```

Java ArrayList Class

Remove ArrayList Elements

3. Using clear() Method



```
ArrayList<String> animals= new ArrayList<>();

// Add elements in the array list
animals.add("Dog");
animals.add("Cat");
animals.add("Horse");
System.out.println("Initial ArrayList: " + animals);

// Remove all the elements
animals.clear();
System.out.println("Final ArrayList: " + animals);
```

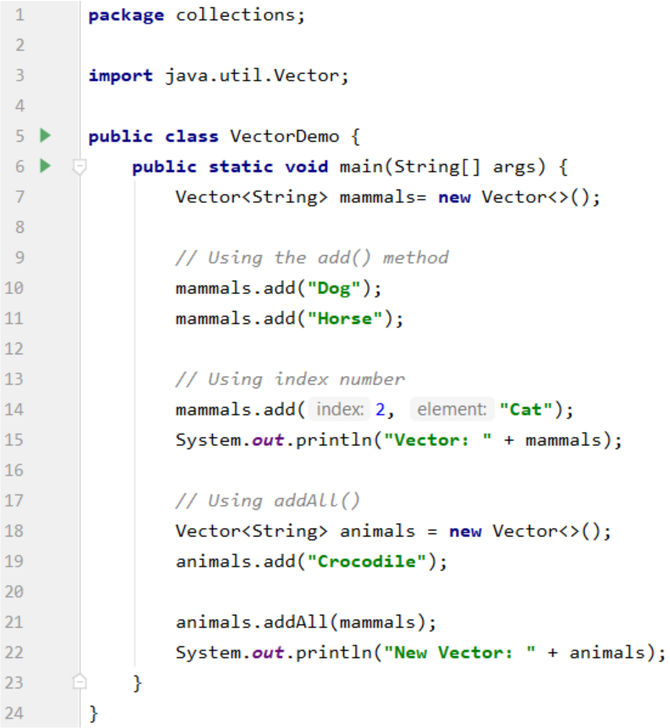
Java Vector

The **Vector** class is an implementation of the List interface that allows us to create resizable-arrays similar to the ArrayList class.

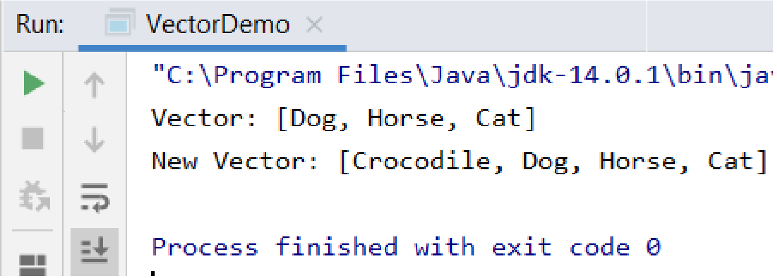
The **Vector** class synchronizes each individual operation. This means whenever we want to perform some operation on vectors, the Vector class automatically applies a lock to that operation.

```
// create Integer type linked list
Vector<Integer> vector= new Vector<>();

// create String type linked list
Vector<String> vector= new Vector<>();
```

Add Elements to Vector



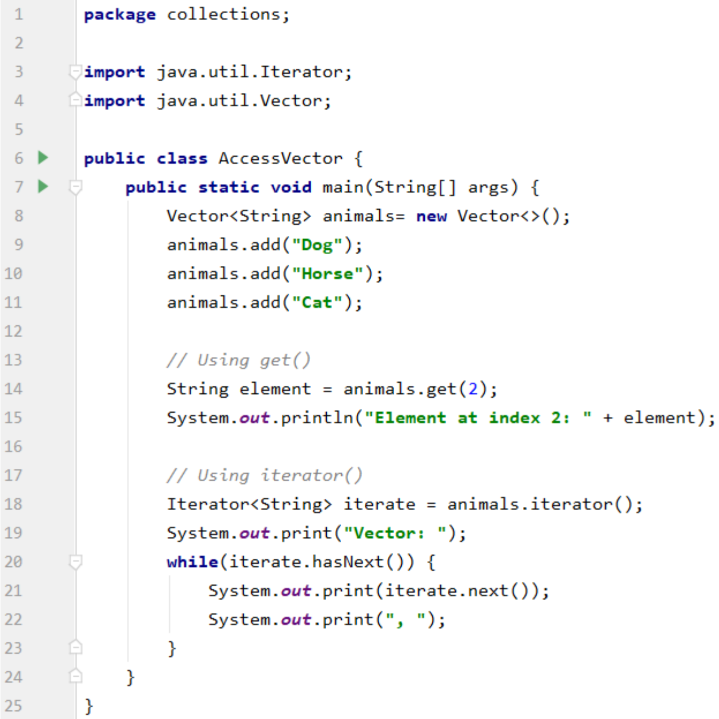
```
package collections;

import java.util.Vector;

public class VectorDemo {
    public static void main(String[] args) {
        Vector<String> mammals= new Vector<>();

        // Using the add() method
        mammals.add("Dog");
        mammals.add("Horse");

        // Using index number
        mammals.add( index: 2,  element: "Cat");
        System.out.println("Vector: " + mammals);

        // Using addAll()
        Vector<String> animals = new Vector<>();
        animals.add("Crocodile");

        animals.addAll(mammals);
        System.out.println("New Vector: " + animals);
    }
}
```



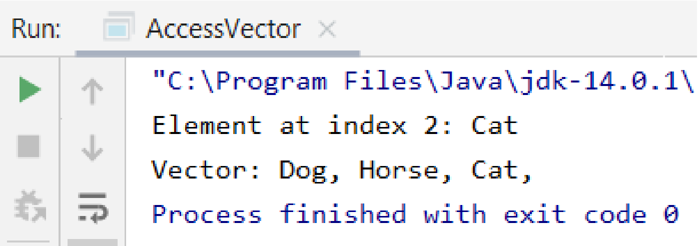
```
"C:\Program Files\Java\jdk-14.0.1\bin\ja
Vector: [Dog, Horse, Cat]
New Vector: [Crocodile, Dog, Horse, Cat]

Process finished with exit code 0
```

Java Vector

Access Vector Elements

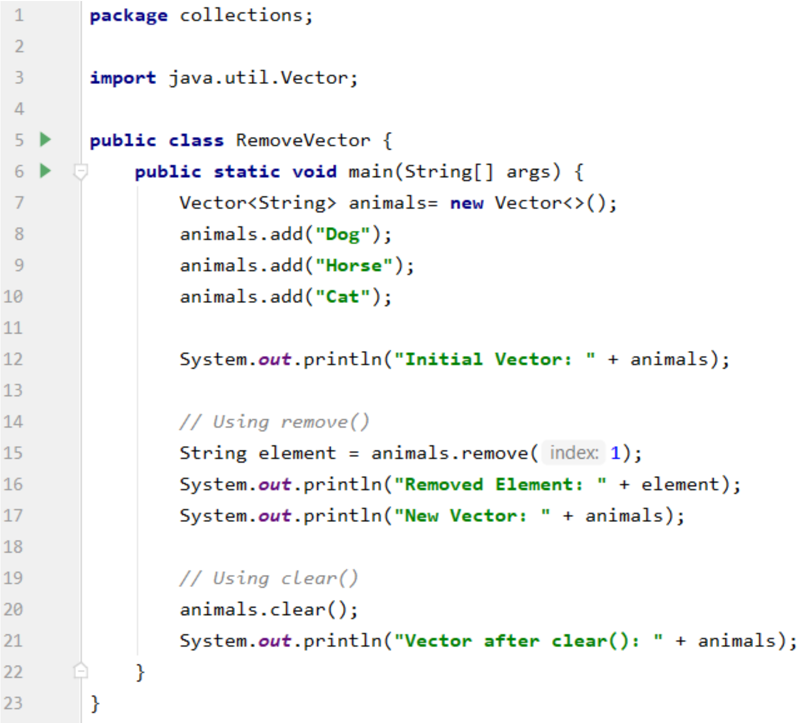


```java
package collections;

import java.util.Iterator;
import java.util.Vector;

public class AccessVector {
    public static void main(String[] args) {
        Vector<String> animals= new Vector<>();
        animals.add("Dog");
        animals.add("Horse");
        animals.add("Cat");

        // Using get()
        String element = animals.get(2);
        System.out.println("Element at index 2: " + element);

        // Using iterator()
        Iterator<String> iterate = animals.iterator();
        System.out.print("Vector: ");
        while(iterate.hasNext()) {
            System.out.print(iterate.next());
            System.out.print(", ");
        }
    }
}
```

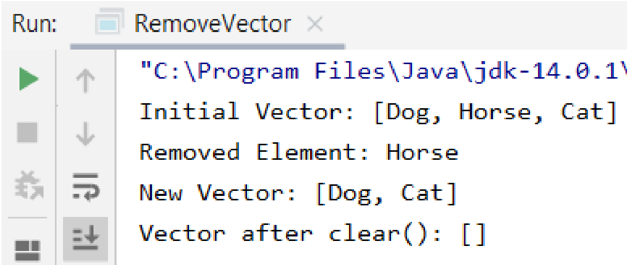


```
"C:\Program Files\Java\jdk-14.0.1\
Element at index 2: Cat
Vector: Dog, Horse, Cat,
Process finished with exit code 0
```

Remove Vector Elements



```java
package collections;

import java.util.Vector;

public class RemoveVector {
    public static void main(String[] args) {
        Vector<String> animals= new Vector<>();
        animals.add("Dog");
        animals.add("Horse");
        animals.add("Cat");

        System.out.println("Initial Vector: " + animals);

        // Using remove()
        String element = animals.remove( index: 1);
        System.out.println("Removed Element: " + element);
        System.out.println("New Vector: " + animals);

        // Using clear()
        animals.clear();
        System.out.println("Vector after clear(): " + animals);
    }
}
```



```
"C:\Program Files\Java\jdk-14.0.1\
Initial Vector: [Dog, Horse, Cat]
Removed Element: Horse
New Vector: [Dog, Cat]
Vector after clear(): []
```

Java Vector

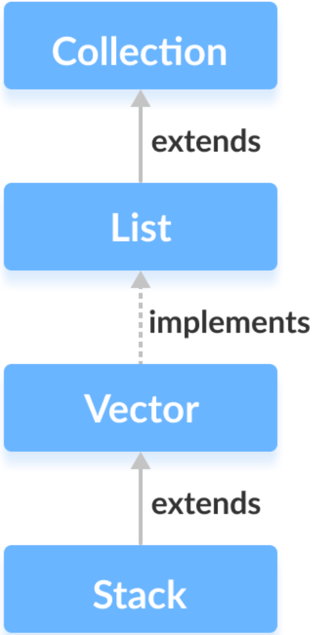
Others Vector Methods

Methods	Descriptions
set()	changes an element of the vector
size()	returns the size of the vector
toArray()	converts the vector into an array
toString()	converts the vector into a String
contains()	searches the vector for specified element and returns a boolean result

Java Stack Class

The Java collections framework has a class named Stack that provides the functionality of the stack data structure.

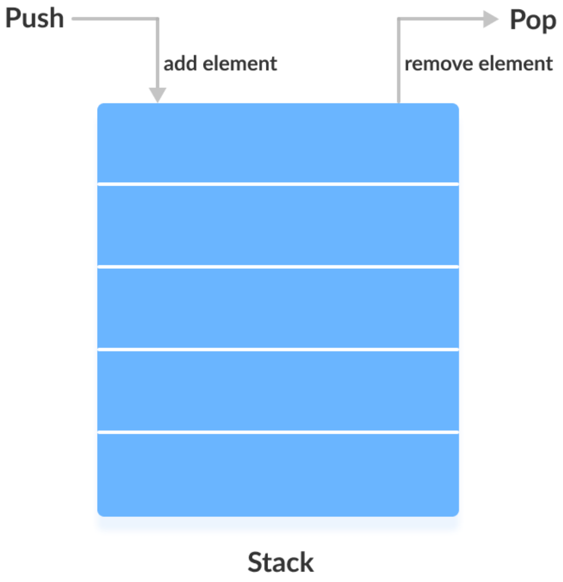
The **Stack** class extends the **Vector** class.



```
// Create Integer type stack
Stack<Integer> stacks = new Stack<>();

// Create String type stack
Stack<String> stacks = new Stack<>();
```

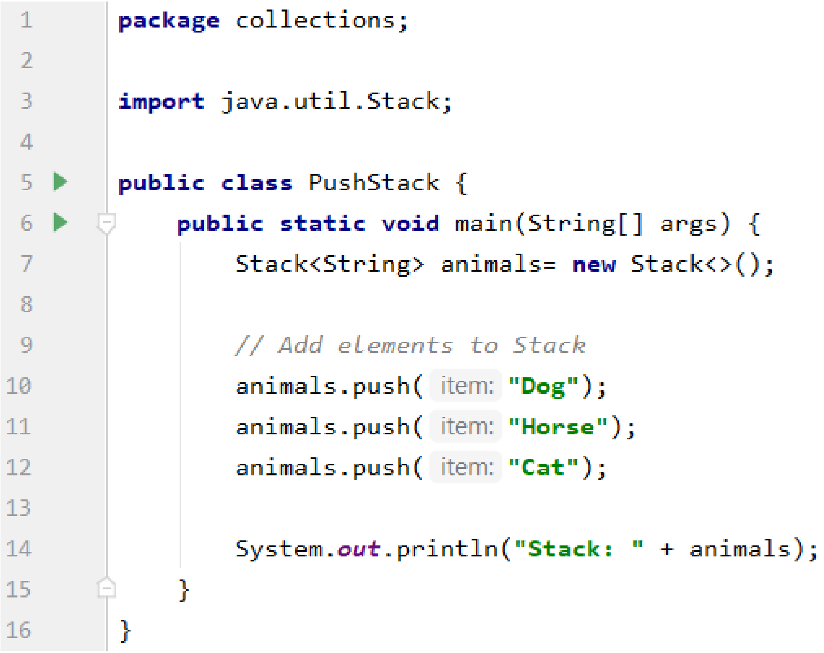
Java Stack Class

In stack, elements are stored and accessed in Last In First Out manner. That is, elements are added to the top of the stack and removed from the top of the stack.

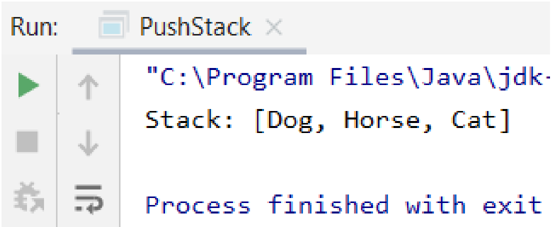


Java Stack Class

push() Method: To add an element to the top of the stack



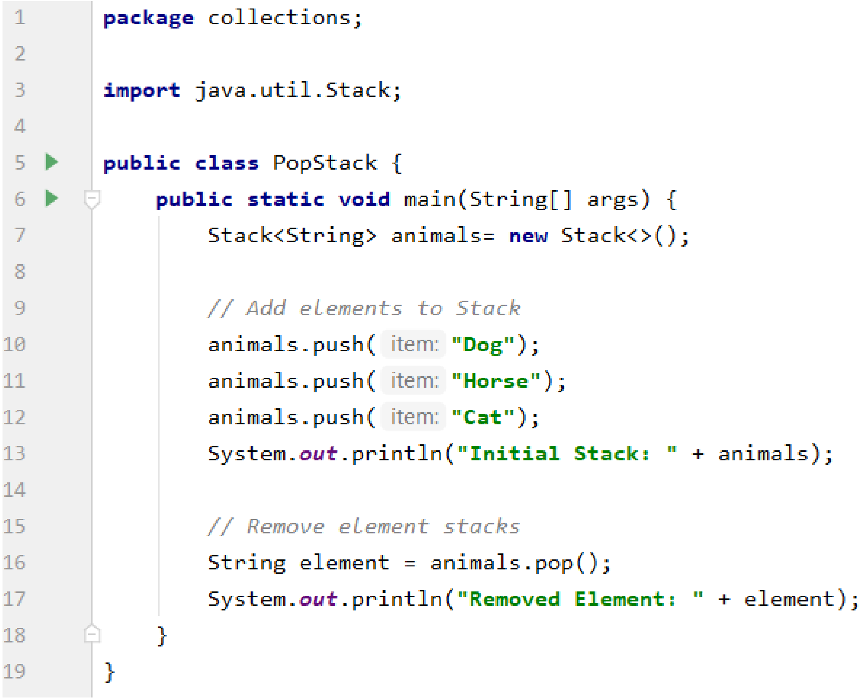
```
package collections;

import java.util.Stack;

public class PushStack {
    public static void main(String[] args) {
        Stack<String> animals= new Stack<>();

        // Add elements to Stack
        animals.push( item: "Dog");
        animals.push( item: "Horse");
        animals.push( item: "Cat");

        System.out.println("Stack: " + animals);
    }
}
```



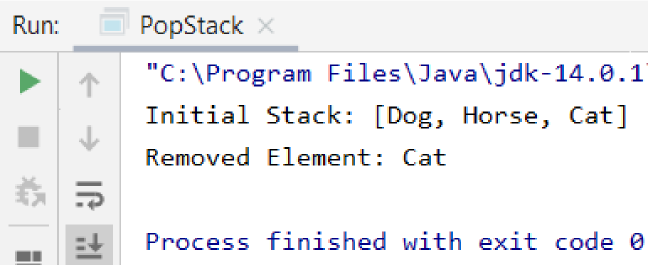
```
"C:\Program Files\Java\jdk-
Stack: [Dog, Horse, Cat]

Process finished with exit
```

Java Stack Class

pop() Method: To remove an element from the top of the stack



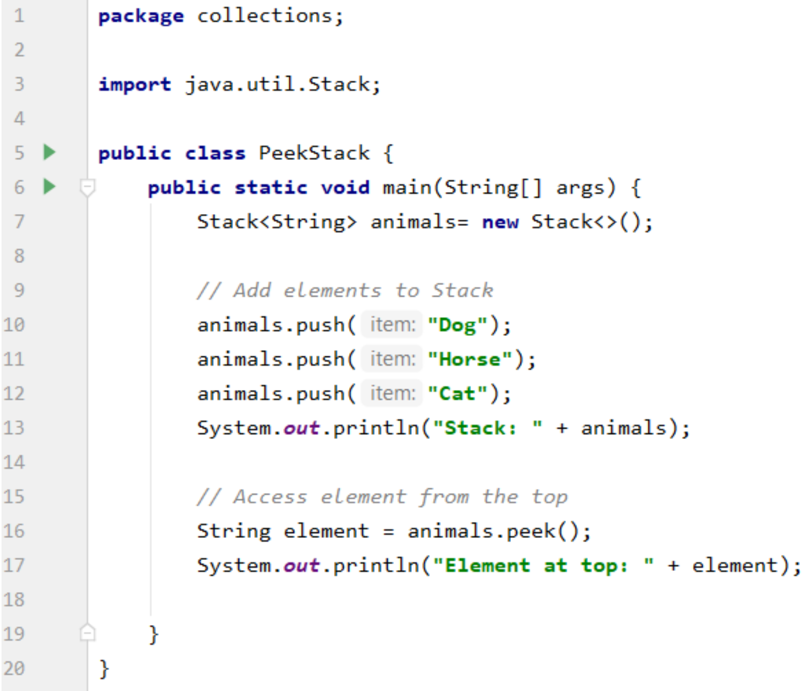
```
package collections;

import java.util.Stack;

public class PopStack {
    public static void main(String[] args) {
        Stack<String> animals= new Stack<>();

        // Add elements to Stack
        animals.push( item: "Dog");
        animals.push( item: "Horse");
        animals.push( item: "Cat");
        System.out.println("Initial Stack: " + animals);

        // Remove element stacks
        String element = animals.pop();
        System.out.println("Removed Element: " + element);
    }
}
```



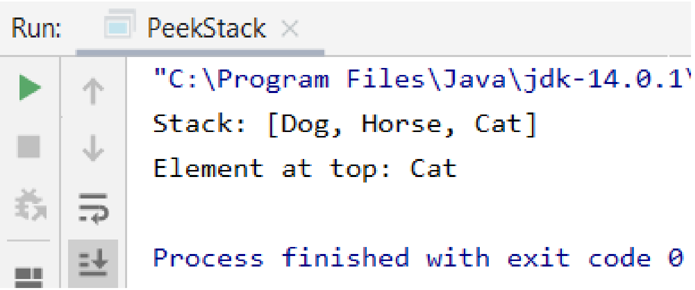
```
"C:\Program Files\Java\jdk-14.0.1
Initial Stack: [Dog, Horse, Cat]
Removed Element: Cat


Process finished with exit code 0
```

Java Stack Class

peek() Method: returns an object from the top of the stack



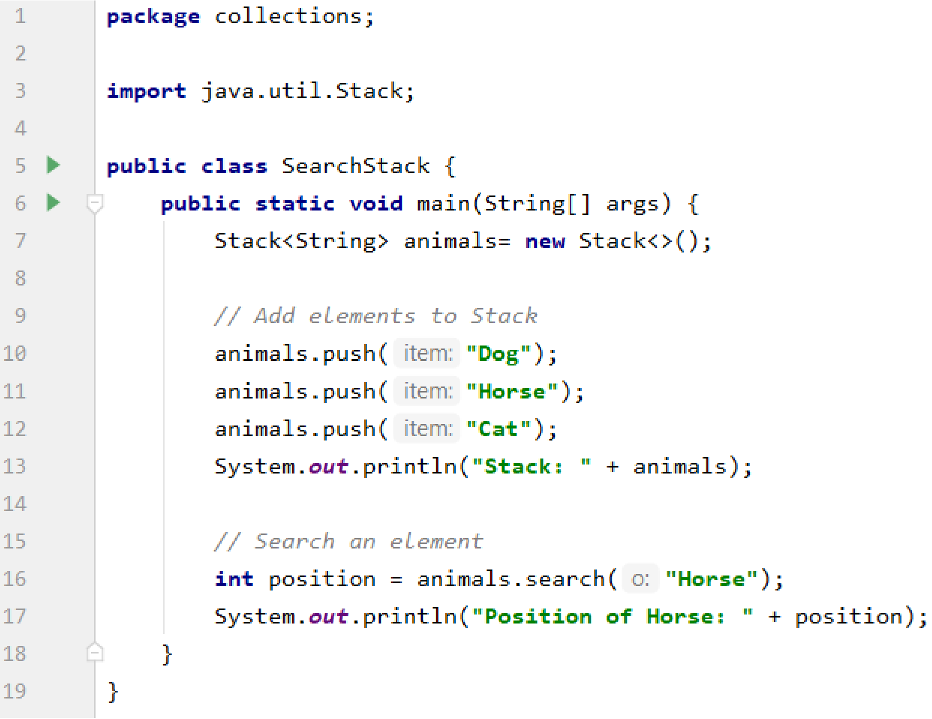
```
package collections;

import java.util.Stack;

public class PeekStack {
    public static void main(String[] args) {
        Stack<String> animals= new Stack<>();

        // Add elements to Stack
        animals.push( item: "Dog");
        animals.push( item: "Horse");
        animals.push( item: "Cat");
        System.out.println("Stack: " + animals);

        // Access element from the top
        String element = animals.peek();
        System.out.println("Element at top: " + element);

    }
}
```



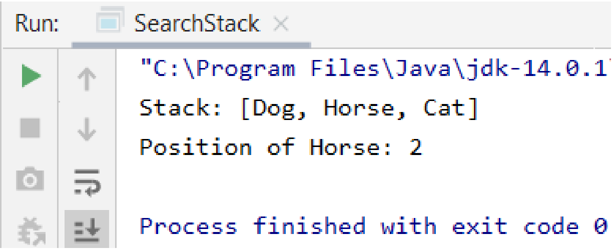
```
"C:\Program Files\Java\jdk-14.0.1
Stack: [Dog, Horse, Cat]
Element at top: Cat


Process finished with exit code 0
```

Java Stack Class

search() Method: To search an element in the stack



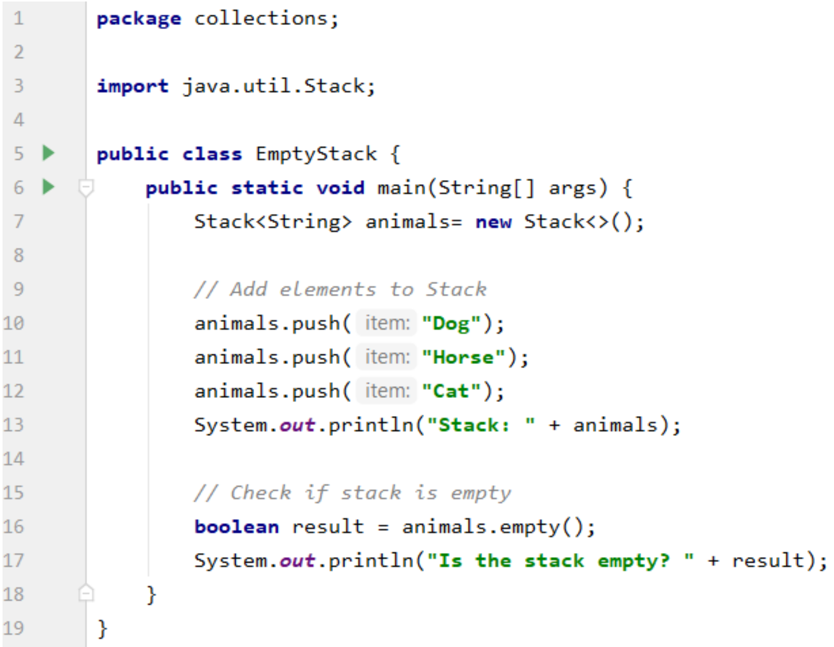
```java
package collections;

import java.util.Stack;

public class SearchStack {
    public static void main(String[] args) {
        Stack<String> animals= new Stack<>();

        // Add elements to Stack
        animals.push( item: "Dog");
        animals.push( item: "Horse");
        animals.push( item: "Cat");
        System.out.println("Stack: " + animals);

        // Search an element
        int position = animals.search( o: "Horse");
        System.out.println("Position of Horse: " + position);
    }
}
```



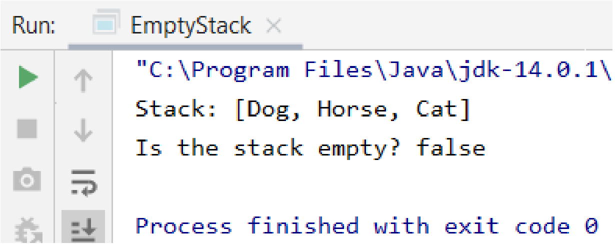
```
"C:\Program Files\Java\jdk-14.0.1
Stack: [Dog, Horse, Cat]
Position of Horse: 2

Process finished with exit code 0
```

Java Stack Class

empty() Method: To check whether a stack is empty or not



```java
package collections;

import java.util.Stack;

public class EmptyStack {
    public static void main(String[] args) {
        Stack<String> animals= new Stack<>();

        // Add elements to Stack
        animals.push( item: "Dog");
        animals.push( item: "Horse");
        animals.push( item: "Cat");
        System.out.println("Stack: " + animals);

        // Check if stack is empty
        boolean result = animals.empty();
        System.out.println("Is the stack empty? " + result);
    }
}
```



```
"C:\Program Files\Java\jdk-14.0.1\
Stack: [Dog, Horse, Cat]
Is the stack empty? false

Process finished with exit code 0
```

Thank you