# Chapter 9

## Arrays, ArrayLists, Sorting, and Searching

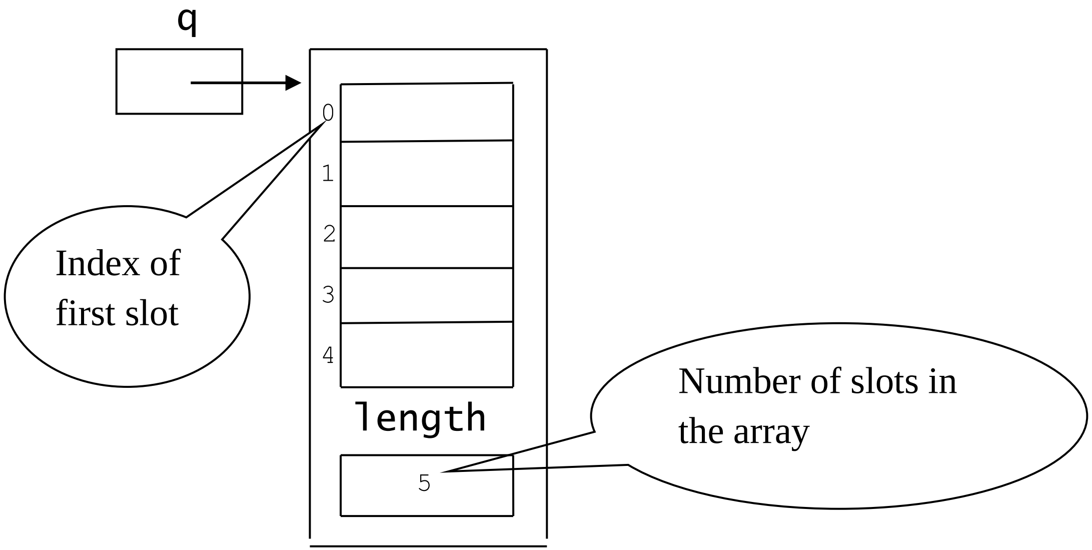
# Creating an array

```
double[] q;
q = new double[5];
```

or

```
double[] q = new double[5];
```

# Structure of the `q` array



Index of first slot

Number of slots in the array

# Working with arrays

```
int i = 2;
q[0] = 2.5;
q[i] = 5.5;
q[i + 2] = 7.7;
System.out.println(q.length);
```

Compare:

| | |
|---|---|
| `s.length()` | length of string |
| `q.length` | length of array |

```
int[] p = {10, 3, 7};
```

# Why we need arrays

Display three `int` variables:

```
System.out.println(x1);
System.out.println(x2);
System.out.println(x3);
```

But what if we wanted to display 1,000,000 `int` variables?

# Do it easily with arrays

```
int[] w = new int[1000000];
//
// init array
//
int i = 0;
while (i < w.length)
{
   System.out.println(w[i]);
   i++;
}
```

# Don't hardcode length

Use

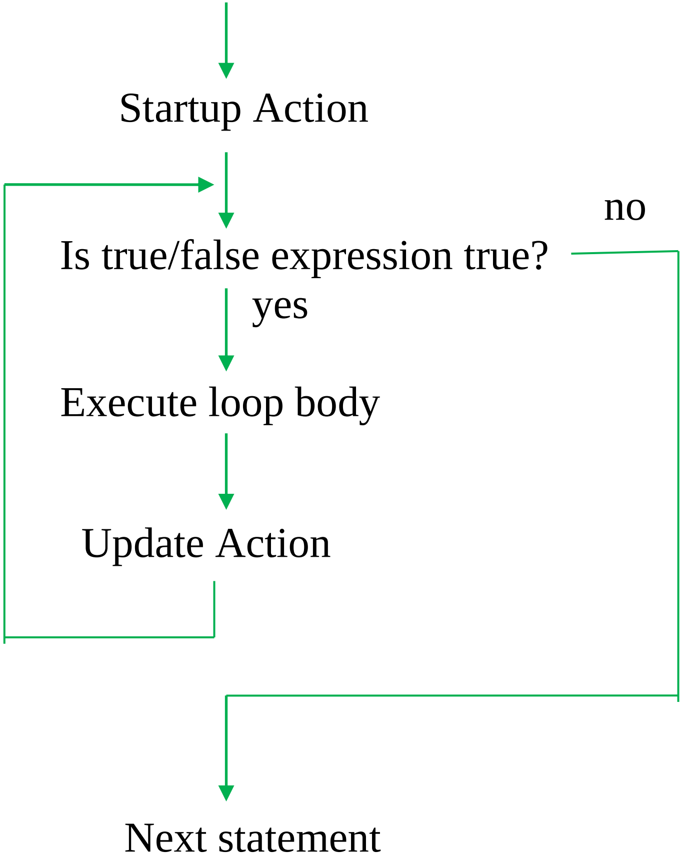
```
i < w.length
```

instead of

```
i < 1000000
```

# `for` Loops

`for (`*startup action*`;` *true/false expression*`;` *update action*`)`<br>
    *statement*



Startup Action

Is true/false expression true?

no

yes

Execute loop body

Update Action

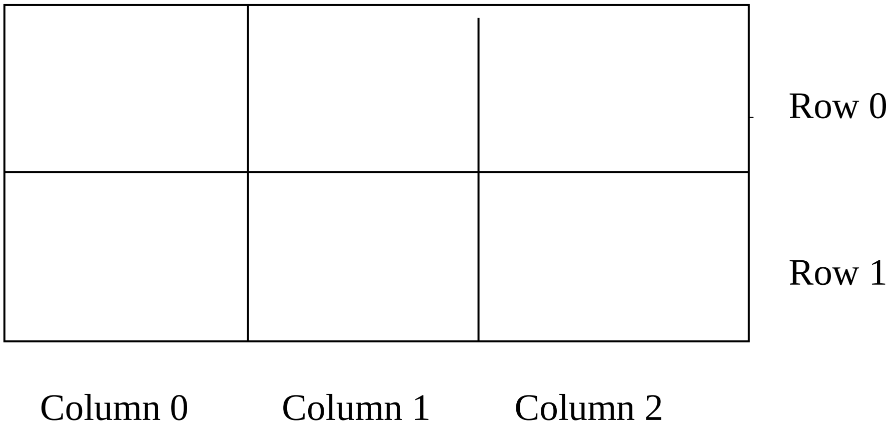
Next statement

# Some `for` loops

```
for (i = 1; i <= 10; i++)
   System.out.println("hello");
```

```
for (int i = 1; i <= 10; i++)
   System.out.println("hello");
```

# Two-dimensional Arrays



Row 0

Row 1

Column 0

Column 1

Column 2

# Working with a 2-D array

```
int m[][];
m = new int[2][3];
```

or

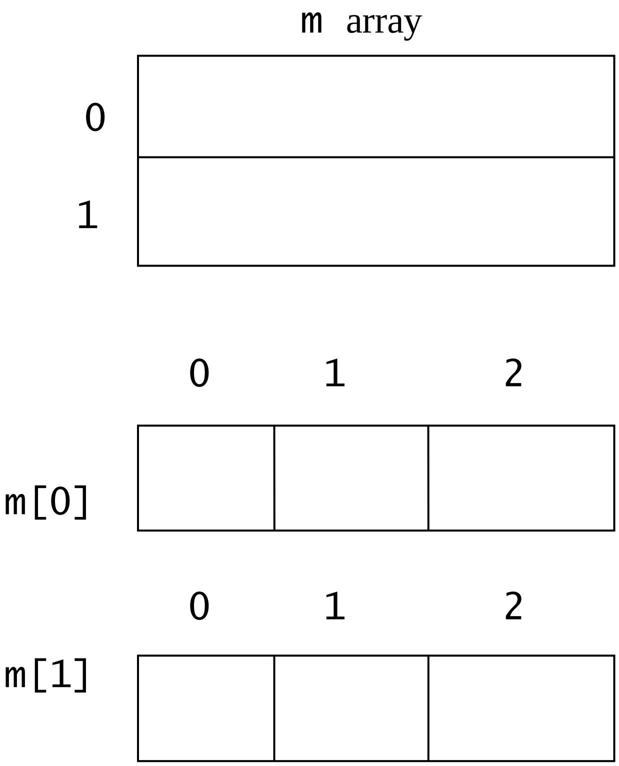
```
int m[][] = new int[2][3];
```

```
m[1][2] = -7;
```

# Use nested for loops to process 2-D array

```
for (int row = 0; row < 2; row++)
   for (int col = 0; col < 3; col++)
      m[row][col] = 20;
```

# 2-D array is 1-D array in which each row is an array



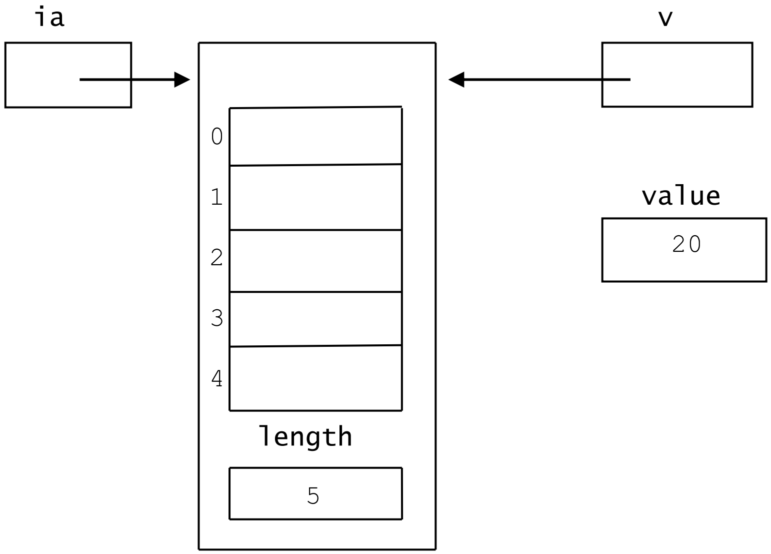
# Use `m.length` and `m[row].length`

```
for (int row = 0; row < m.length; row++)
   for (int col = 0; col < m[row].length; col++)
      m[row][col] = 20;
```

# Passing arrays

```java
class ArrayExample
{
   public static void main(String[] args)
   {
      int[] ia = new int[5];
      initArray(ia, 20);
   }
   //-----------------------------------------
   public static void initArray(int[] v, int value)
   {
      for (int i = 0; i < v.length; i++)
         v[i] = value;
   }
}
```

# ia and v point to same structure

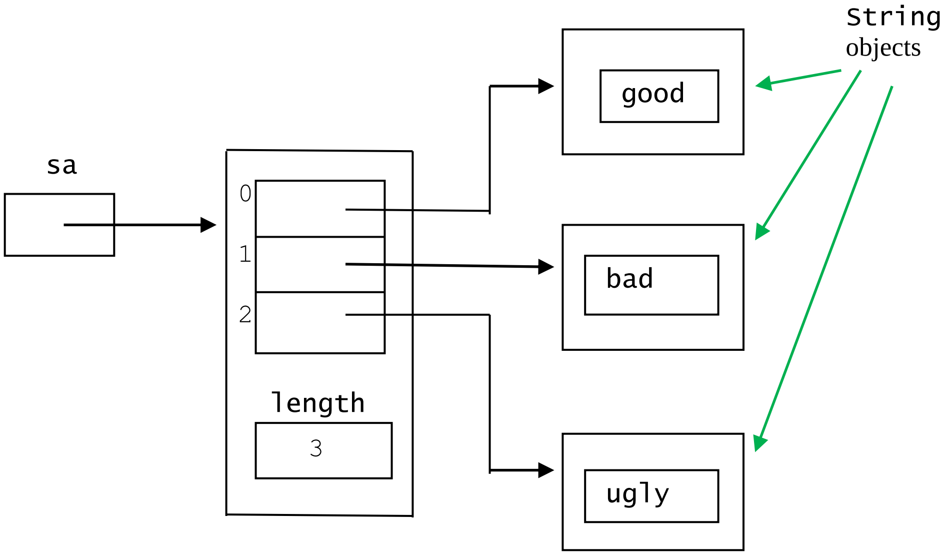


# Returning arrays

```
public int[] returnIntArray()
{
   int[] q = new int[100];
   for (int i = 0; i < q.length; i++)
      q[i] = 7;
   return q;
}
```

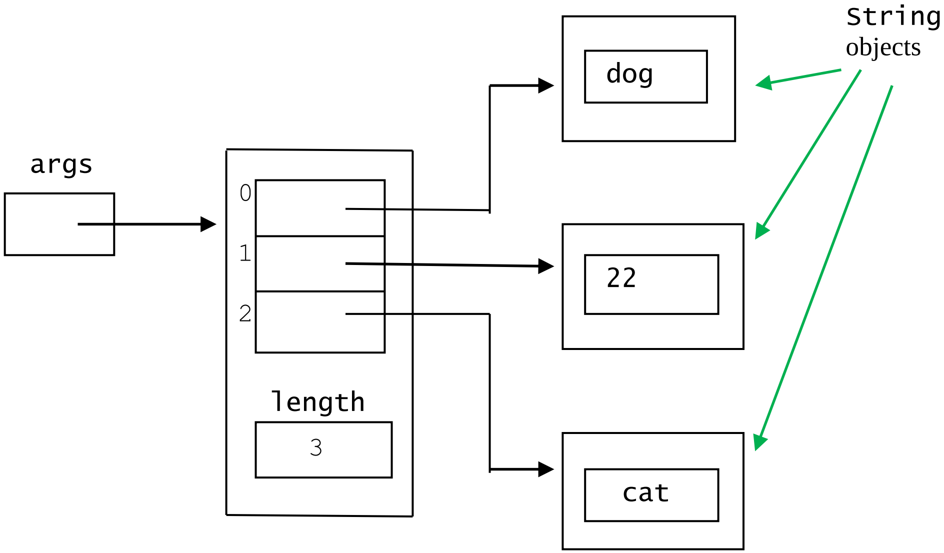
# Arrays of objects

```
String[] sa = new String[3];

sa[0] = "good";
sa[1] = "bad";
sa[2] = "ugly";

for (int i = 0; i < sa.length; i++)
   System.out.println(sa[i]);
```


String
objects
sa
0
1
2
length
3
good
bad
ugly

# Accessing command line arguments

```
java ArgsExample dog 22 cat
```


String
objects
args
0
1
2
length
3
dog
22
cat

# Program to display command line arguments

```
class ArgsExample
{
   public static void main(String[] args)
   {
      for (int i = 0; i < args.length; i++)
         System.out.println(args[i]);
   }
}
```

# Must convert command-line arguments

```
int x;
x = Integer.parseInt(arg[1]);
```

# Sorting

Makes accessing data easier.

What if telephone book entries were not in alphabetical order?

# Selection sort

7 3 2 5

select smallest and exchange with 7

2 3 7 5

select smallest and exchange with 3

2 3 7 5

select smallest and exchange with 7

2 3 5 7

# Program that uses selection sort

```java
import java.util.Random;
class TestSort
{
   public static void main(String[] args)
   {
      int[] z = new int[10];
      Random r = new Random();

      for (int i = 0; i < z.length; i++)
         z[i] = r.nextInt();

      selectionSort(z);

      for (int i = 0; i < z.length; i++)
         System.out.println(z[i]);
   }
   public static void selectionSort(int[] a)
   {
    for (int startIndex=0;startIndex<a.length-1;startIndex++)
      {
         int min = a[startIndex]; // set min to start element
         int indexOfMin = startIndex; // remember its index

         for (int j = startIndex + 1; j < a.length; j++)
            if (a[j] < min)       // found smaller element?
            {
               min = a[j];            // rememeber its value
               indexOfMin = j;        // remember its index
            }

         a[indexOfMin] = a[startIndex];// switch first/min
         a[startIndex] = min;
      }
   }
}
```

# Time and space complexity of selection sort

Number of probes:

$$n + (n-1) + (n-2) + ... + 2$$

which equals

$$(n\times(n+1)/2) - 1$$

which equals

$$n^2/2 + n/2 - 1$$

# Common complexity measures

| | |
|---|---|
| $O(n)$ | Linear |
| $O(n \log n)$ | |
| $O(n^2)$ | Polynomial |
| $O(n^3)$ | |
| $O(c^n)$ | Exponential |
| $O(n!)$ | Factorial--Really bad! |

# Binary Search

1) If `i` is greater than `j` , return `-1` (a return value of `-1` indicates `x` is not in the array)

2) Compute `mid = (i + j)/2`. `mid` is the index of the number in the middle of that portion of the `q` array corresponding to indices `i` to `j`.

3) If `x` is equal to `q[mid]`, return `mid`.

4) If `x` is less than `q[mid]`, set `j` to `mid - 1` else set `i` to `mid + 1`. Go to step 1.

# ArrayList

1) Its base type is not fixed.

2) Its base type must be a class. It cannot be a primitive type.

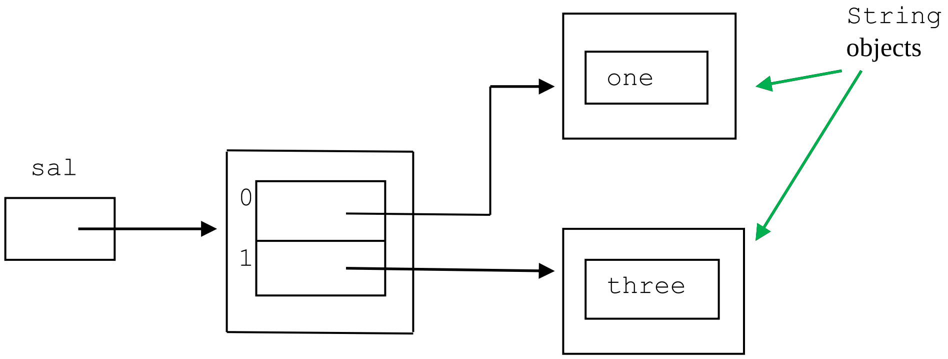
3) An `ArrayList` object can grow and shrink during the execution of a program. In contrast, the size of an array is fixed. Once an array is created, its size cannot change.

4) Elements of an `ArrayList` object are accessed via method calls—not with the square-bracket notation used by arrays.

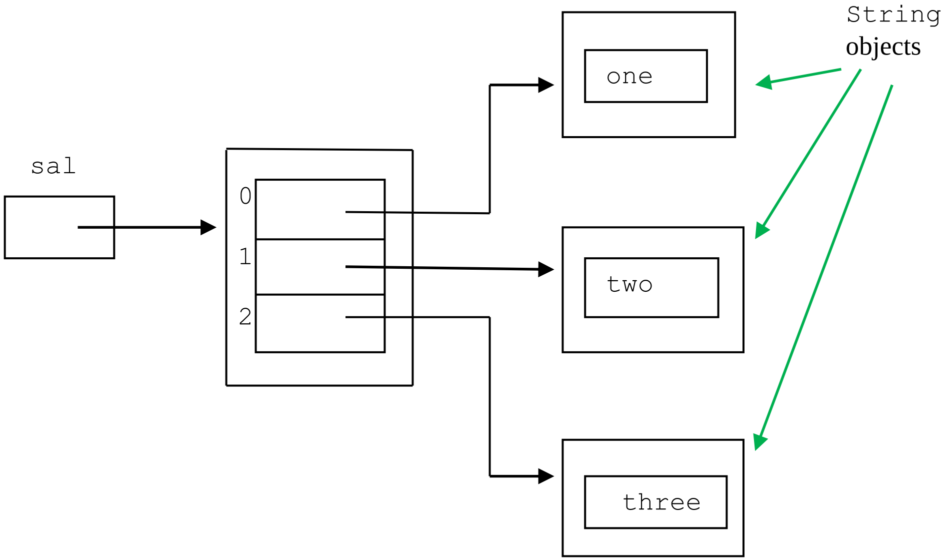
5) `ArrayList` is a class in the `java.util` package. Thus, to use `ArrayList`, you should include an `import` statement at the beginning of your program that imports `java.util.ArrayList`.

# Working with an `ArrayList`

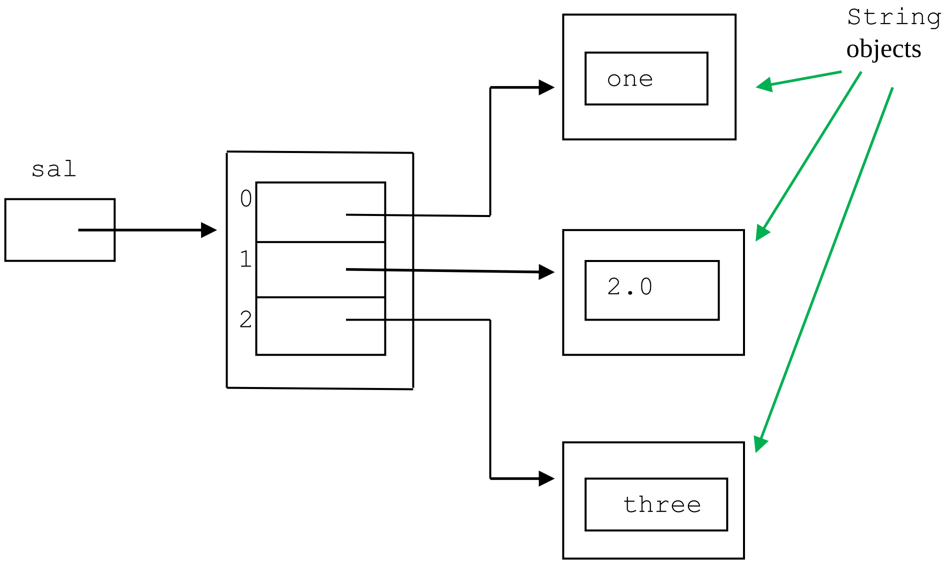
```
ArrayList<String> sal = new ArrayList<String>();
sal.add("one");        // add "one" to end of sal
sal.add("three");      // add "three" to end of sal
```



```
sal.add(1, "two");    // insert "two" at index 1
```



```
sal.set(1, "2.0");    // overlay slot 1 with "2.0"
```



# remove

```
System.out.println("slot 1 contains " + sal.get(1));
System.out.println("Now remove " + sal.remove(1));
```

# indexOf

```
System.out.println("idx of three is " + sal.indexOf("three"));
```

# contains

```
if (sal.contains("three"))
   System.out.println("sal object contains three");
else
   System.out.println("sal object does not contain three");
```

# size, isEmpty, and clear

```
System.out.println("size of sal object is " + sal.size());

if (sal.isEmpty())
   System.out.println("sal object is empty");
else
   System.out.println("sal object is not empty");

sal.clear();              // reset sal to zero size
```

# Milestone

You can now understand

```
class Program1
{
   public static void main(String[] args)
   {
      System.out.println(20 + 3);
      System.out.println("20 + 3");
   }
}
```