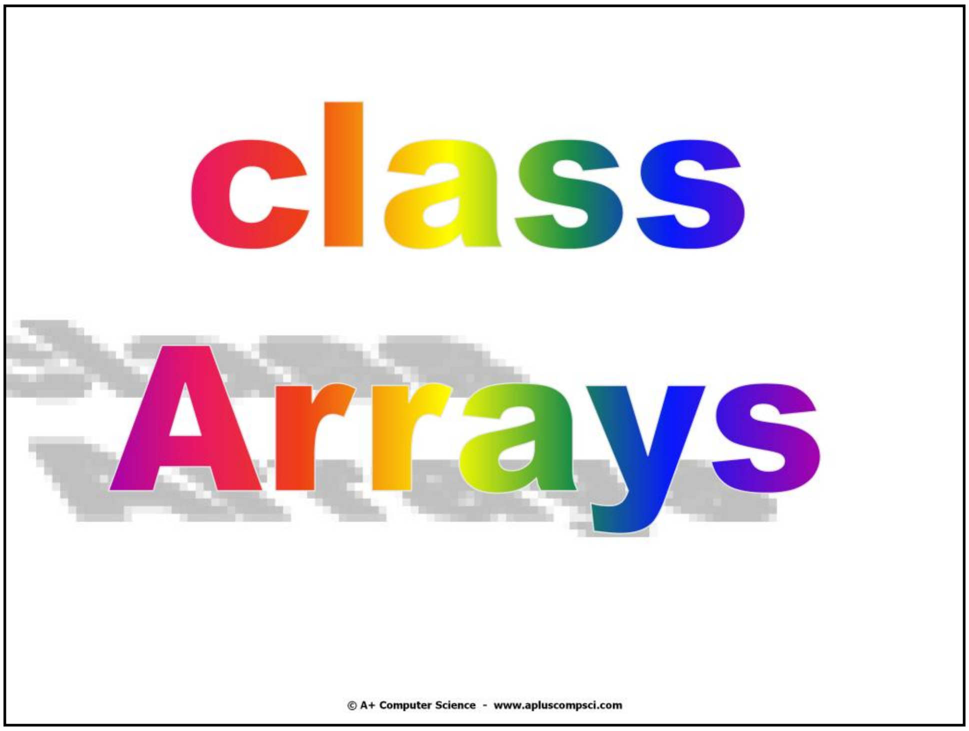
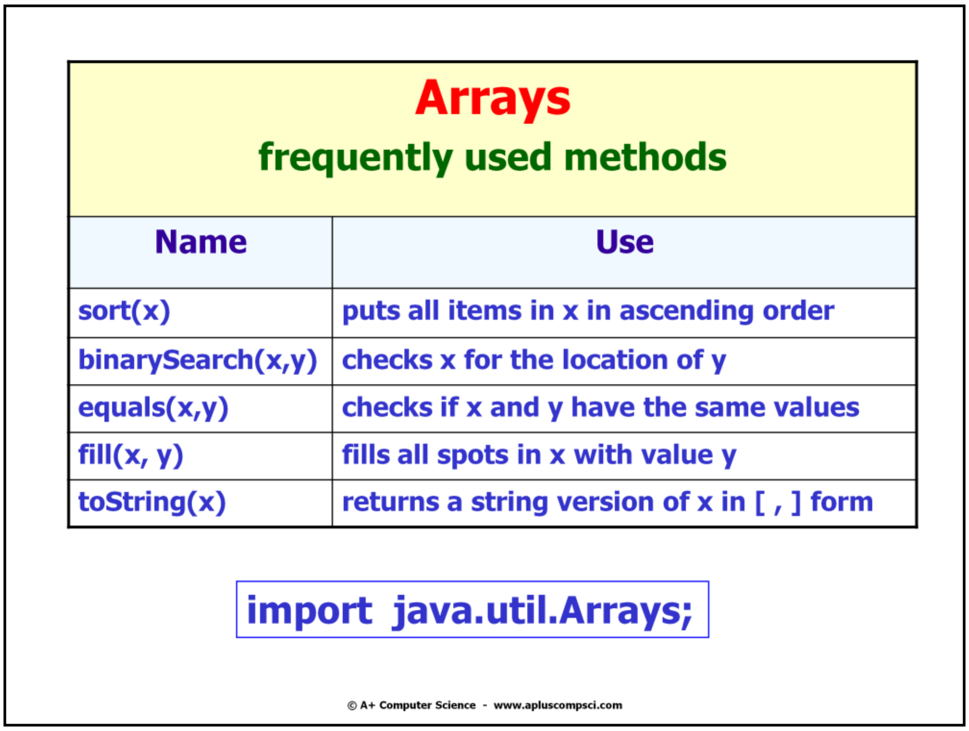
Sorting / Searching String Arrays

© A+ Computer Science - www.apluscompsci.com



class Arrays

© A+ Computer Science - www.apluscompsci.com



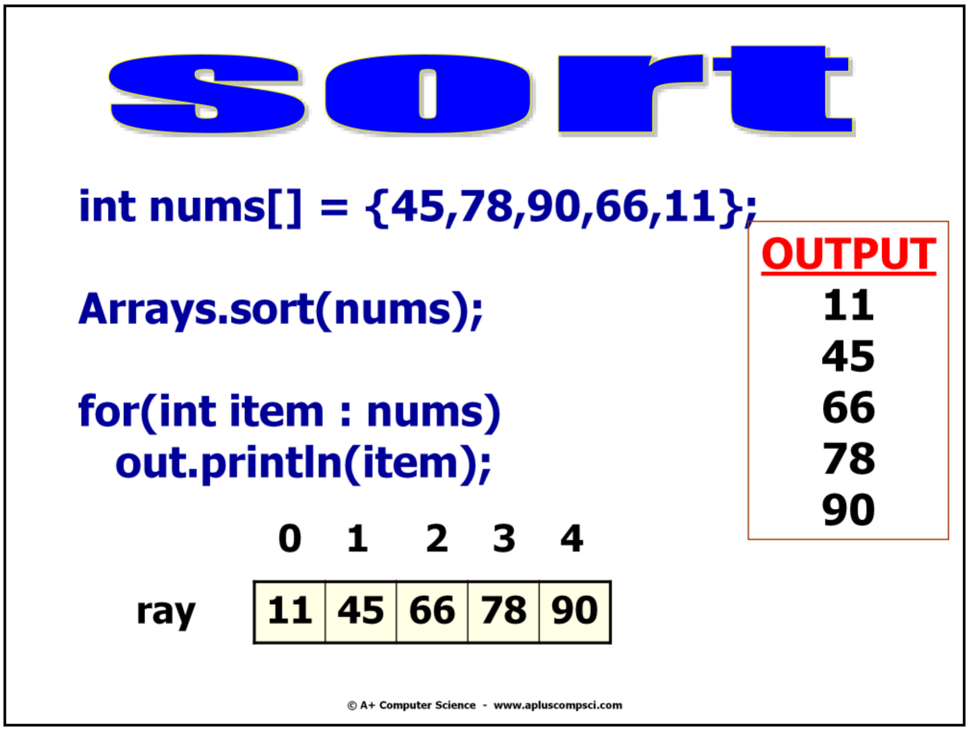
Arrays
frequently used methods

Name	Use
sort(x)	puts all items in x in ascending order
binarySearch(x,y)	checks x for the location of y
equals(x,y)	checks if x and y have the same values
fill(x, y)	fills all spots in x with value y
toString(x)	returns a string version of x in [,] form

```
import  java.util.Arrays;
```

© A+ Computer Science - www.apluscompsci.com

Class Arrays contains many static methods that are very useful when manipulating and analyzing arrays.



sort

```
int nums[] = {45,78,90,66,11};

Arrays.sort(nums);

for(int item : nums)
   out.println(item);
```

OUTPUT

```
11
45
66
78
90
```

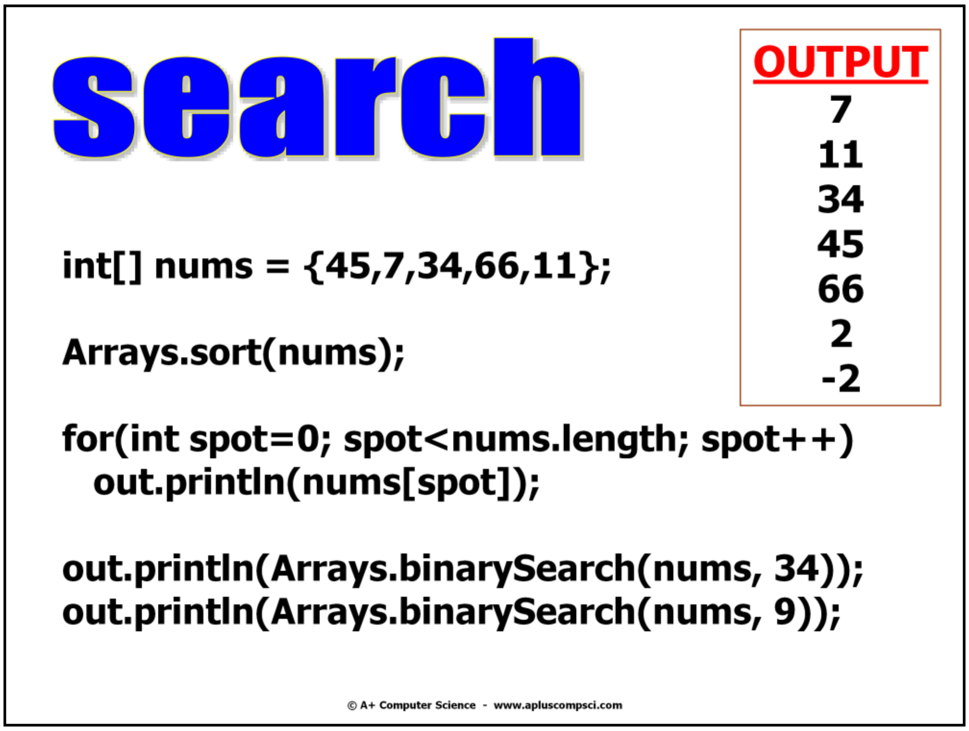
	0	1	2	3	4
ray	11	45	66	78	90

© A+ Computer Science - www.apluscompsci.com

Arrays.sort() will put all items in an array in natural order.

Arrays.sort() uses a quick sort algorithm when sorting primitive values.

Arrays.sort() uses a merge sort algorithm when sorting references.



search

```
int[] nums = {45,7,34,66,11};

Arrays.sort(nums);

for(int spot=0; spot<nums.length; spot++)
   out.println(nums[spot]);

out.println(Arrays.binarySearch(nums, 34));
out.println(Arrays.binarySearch(nums, 9));
```

OUTPUT

```
7
11
34
45
66
2
-2
```

© A+ Computer Science - www.apluscompsci.com

`Arrays.binarySearch()` will search an array for a specified value.

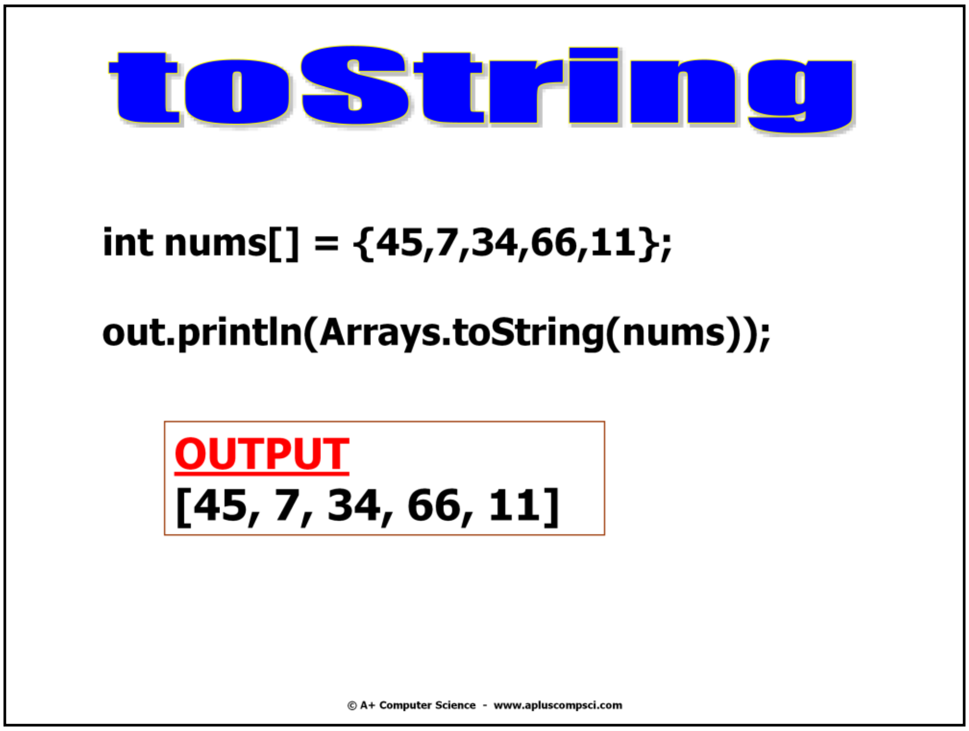
`Arrays.binarySearch()` works best when used on a sorted array.

`Arrays.binarySearch()` will return the spot at which the specified value is found if the value is present.

`Arrays.binarySearch()` will return -1 + -(where it should/would be if it was present).

Given the array `{3,6,8,11,20,25}`, a `binarySearch()` call for 8 would return 2 as 8 is in spot 2.

A `binarySearch()` call for 23 would return -6 as 23 would/should be in spot 5 if it were present. As it is not preset, -1 + -5(-6) is returned.



toString

```
int nums[] = {45,7,34,66,11};

out.println(Arrays.toString(nums));
```

OUTPUT

```
[45, 7, 34, 66, 11]
```

© A+ Computer Science - www.apluscompsci.com

`Arrays.toString()` is useful for printing out an array.

`Arrays.toString()` returns a String will all values in the array separated by commas and bounded by brackets.

open sort.java

© A+ Computer Science - www.apluscompsci.com

open search.java

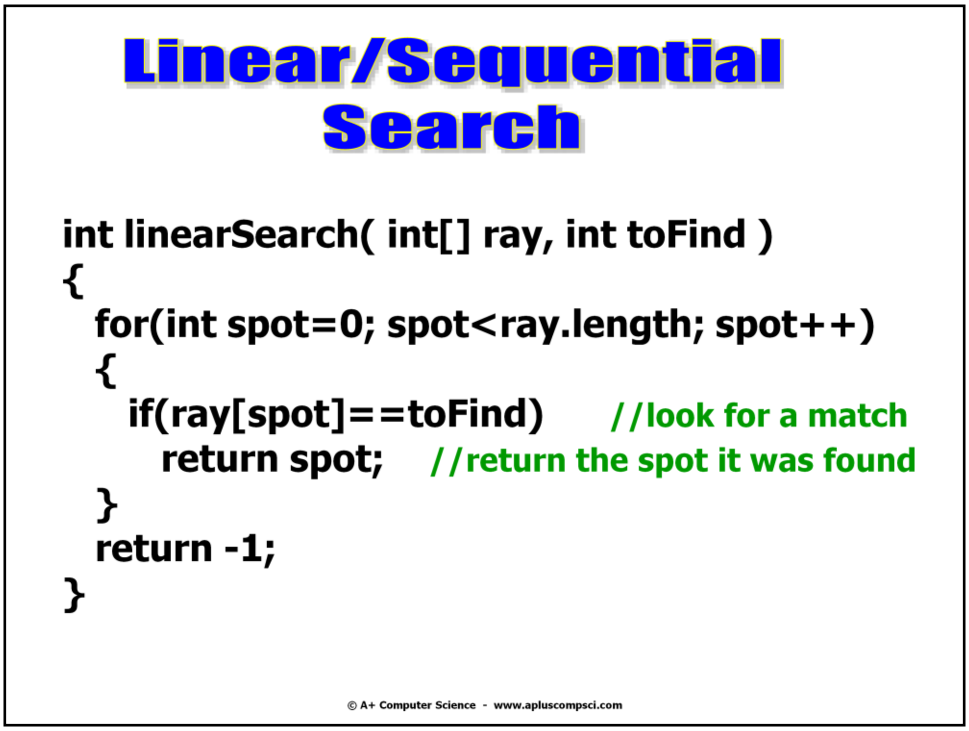
© A+ Computer Science - www.apluscompsci.com

open tostring.java

© A+ Computer Science - www.apluscompsci.com

Searching Algos

© A+ Computer Science - www.apluscompsci.com



Linear/Sequential Search

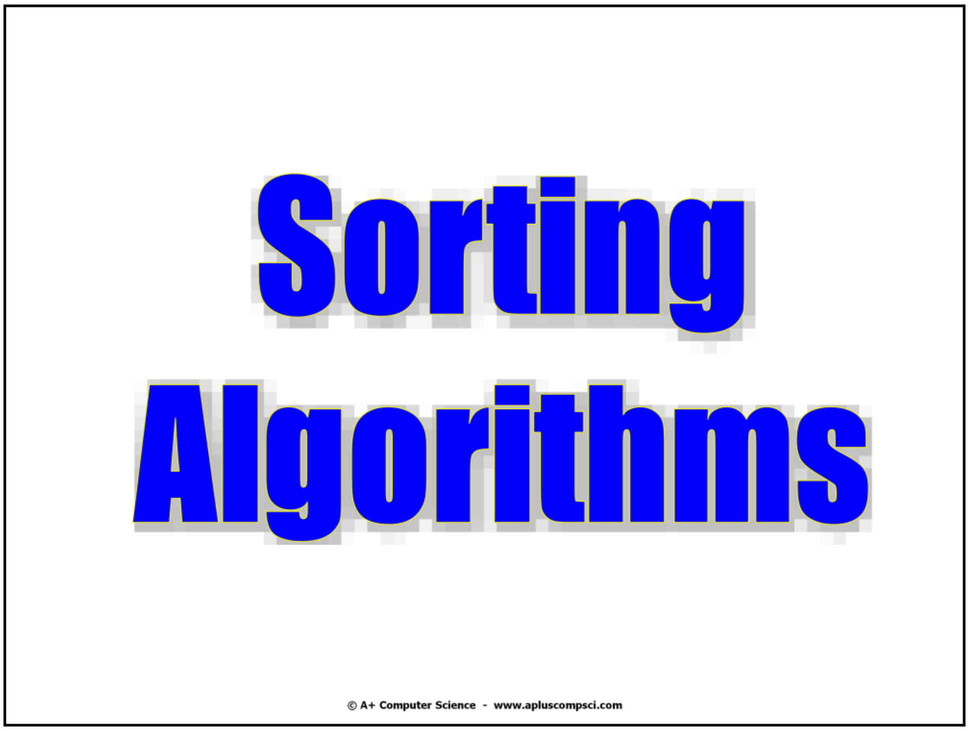
```
int linearSearch( int[] ray, int toFind )
{
  for(int spot=0; spot<ray.length; spot++)
  {
    if(ray[spot]==toFind)      //look for a match
      return spot;    //return the spot it was found
  }
  return -1;
}
```

© A+ Computer Science - www.apluscompsci.com

Linear / sequential search is a very basic search algorithm.

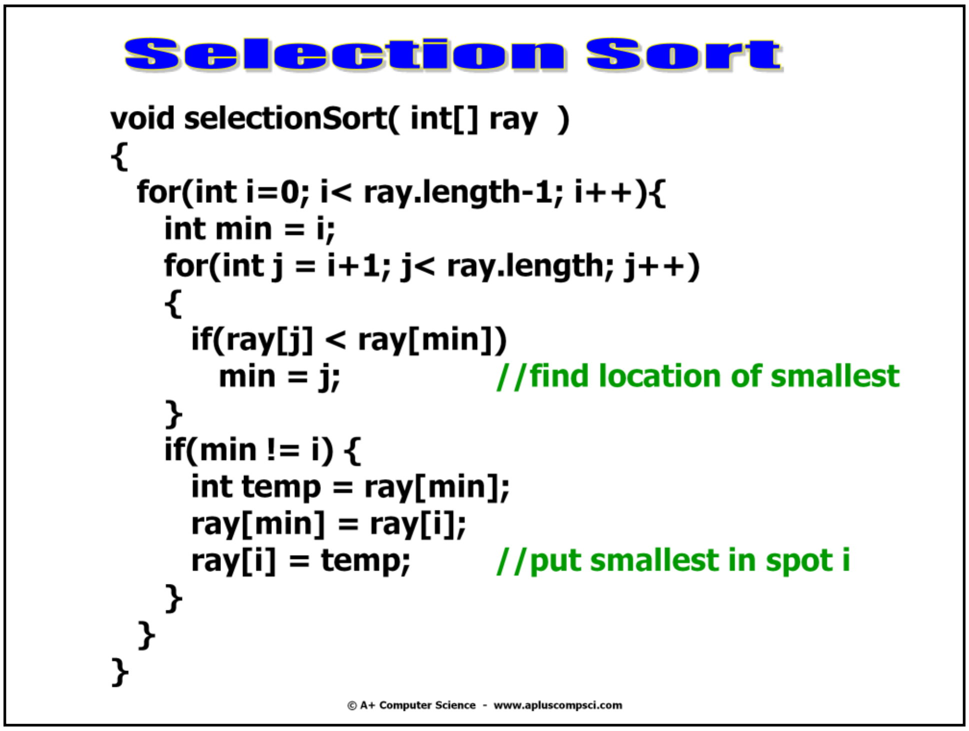
Linear / sequential search accesses each spot in the array and checks each item in each spot to see if that item is the specified search value. If a match occurs, the spot where the match was found is returned.

Linear / sequential search is most commonly written using a for loop and an if statement.



Sorting Algorithms

© A+ Computer Science - www.apluscompsci.com



Selection Sort

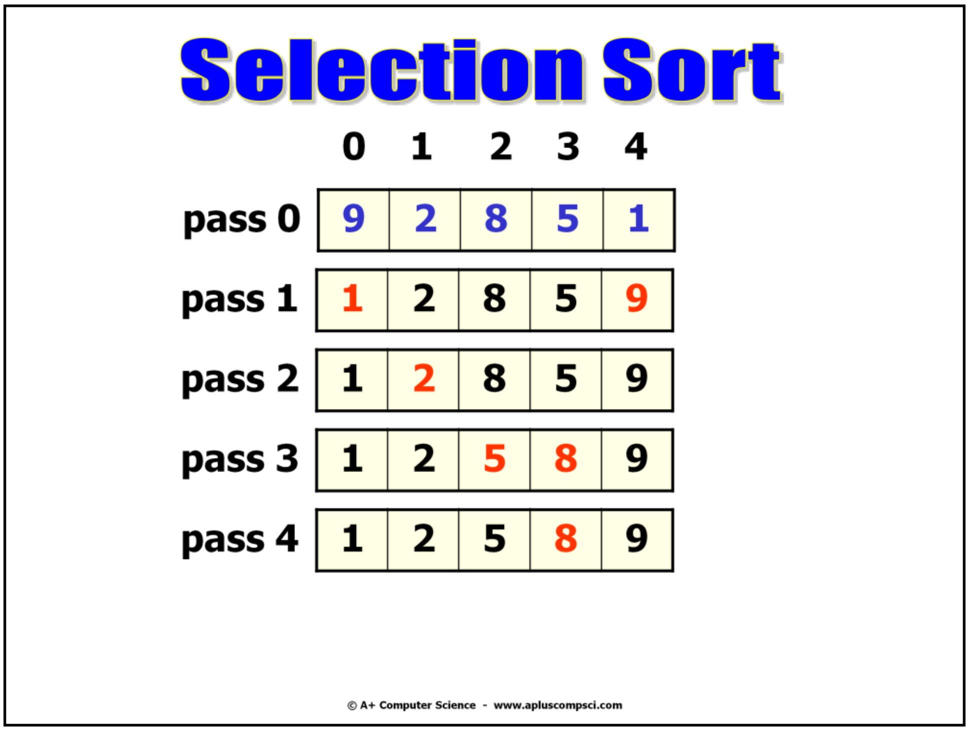
```
void selectionSort( int[] ray  )
{
  for(int i=0; i< ray.length-1; i++){
    int min = i;
    for(int j = i+1; j< ray.length; j++)
    {
      if(ray[j] < ray[min])
        min = j;            //find location of smallest
    }
    if(min != i) {
      int temp = ray[min];
      ray[min] = ray[i];
      ray[i] = temp;        //put smallest in spot i
    }
  }
}
```

© A+ Computer Science - www.apluscompsci.com

Selection sort puts all items in an array in descending or ascending order.

Selection consists of two loops. The outer loop will run length-1 times. The inner loop will run to completion each time the outer loop runs.

The inner loop's job is to find the current smallest or largest item. The spot where the item was found is saved. After the inner loop completes, the saved spot is examined and a swap occurs if needed. As items are places in the correct spot, those spots are no longer accessed. The inner loop start value is i+1 to account for items that have been ordered.



Selection Sort

	0	1	2	3	4
pass 0	9	2	8	5	1
pass 1	1	2	8	5	9
pass 2	1	2	8	5	9
pass 3	1	2	5	8	9
pass 4	1	2	5	8	9

© A+ Computer Science - www.apluscompsci.com

An item is placed in the correct spot after each pass. A pass is one iteration of the outer loop.

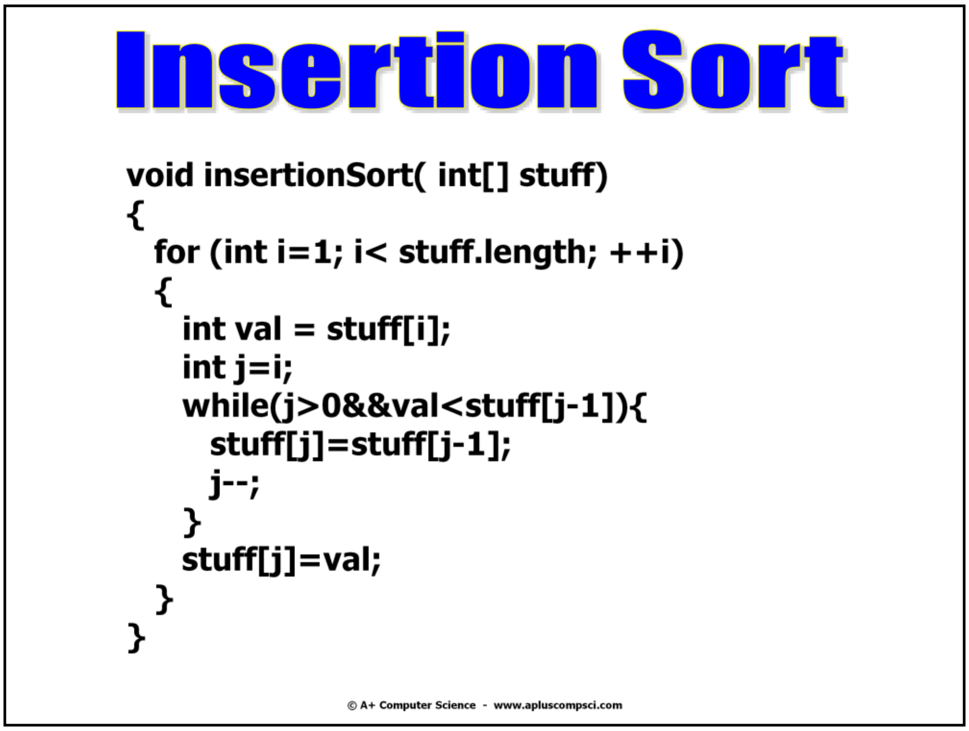
Pass 1 – 1 is placed in spot 0.

Pass 2 – 2 is not moved as it was in spot 1 already.

Pass 3 – 5 is placed in spot 2.

Pass 4 – 8 is not moved as it was in spot 3 already.

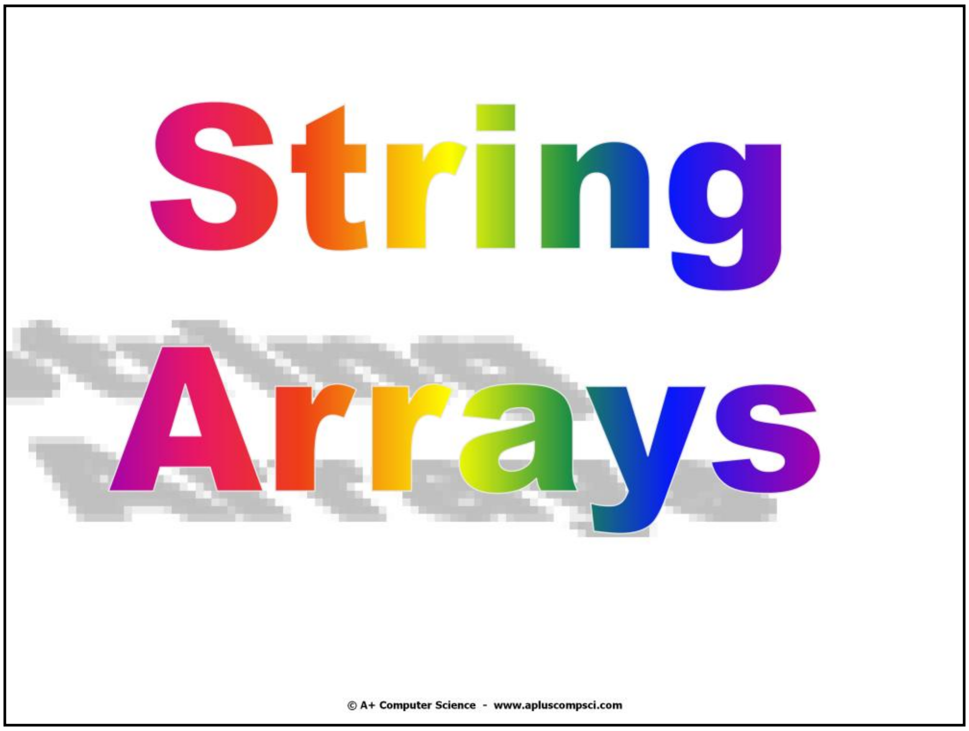
Pass 5 – 9 is not moved as it was in spot 4 already.



Insertion Sort

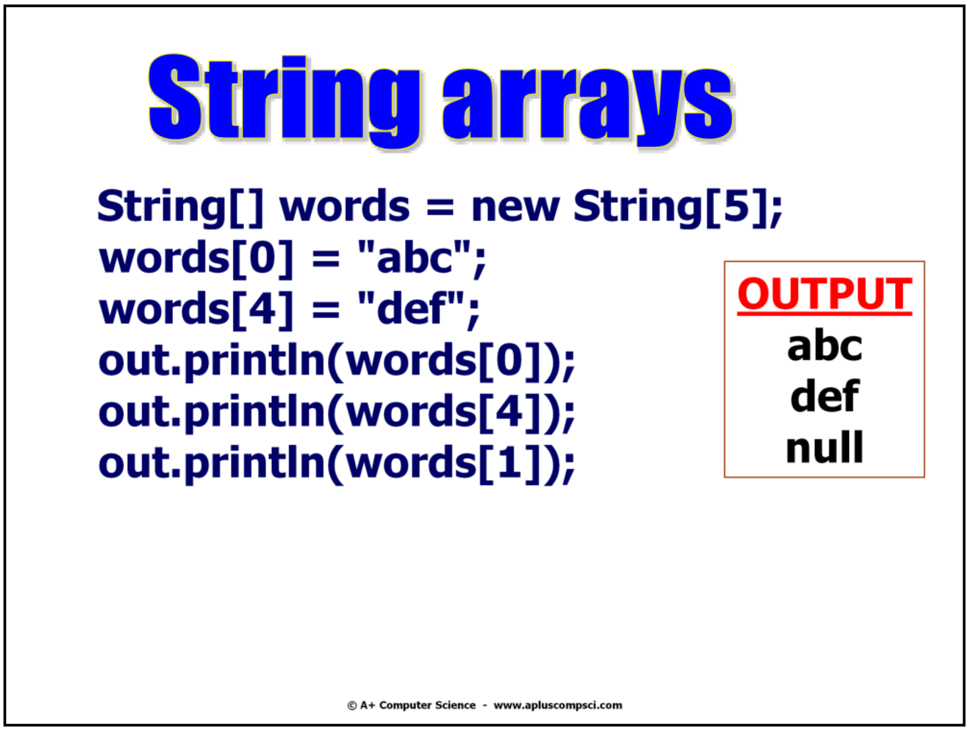
```
void insertionSort( int[] stuff)
{
  for (int i=1; i< stuff.length; ++i)
  {
    int val = stuff[i];
    int j=i;
    while(j>0&&val<stuff[j-1]){
      stuff[j]=stuff[j-1];
      j--;
    }
    stuff[j]=val;
  }
}
```

© A+ Computer Science - www.apluscompsci.com



String Arrays

© A+ Computer Science - www.apluscompsci.com



String arrays

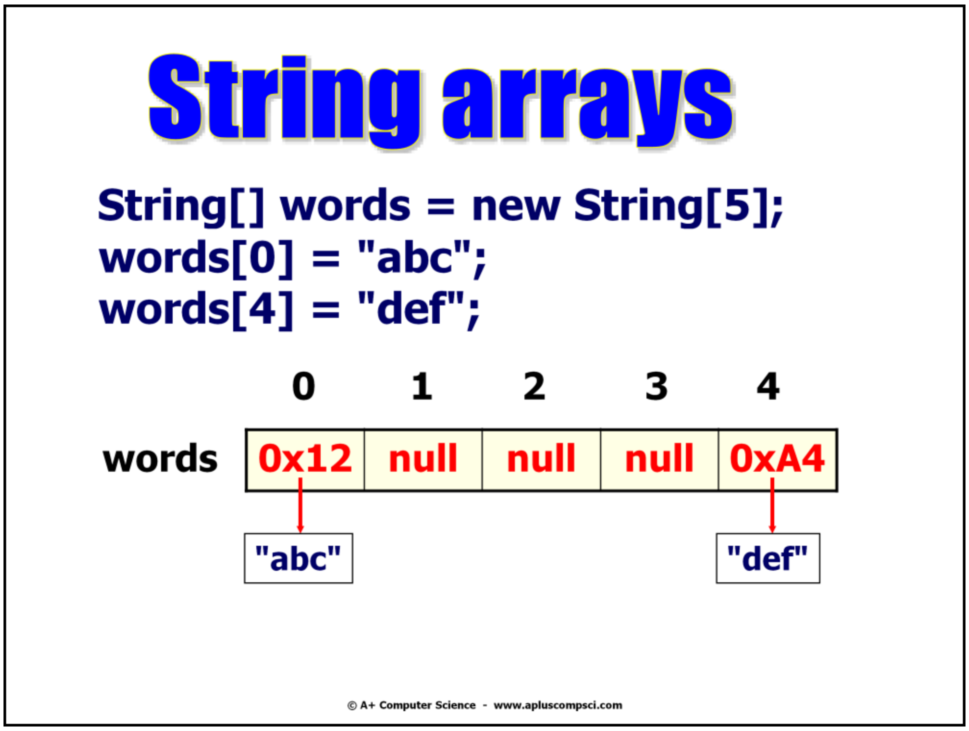
```
String[] words = new String[5];
words[0] = "abc";
words[4] = "def";
out.println(words[0]);
out.println(words[4]);
out.println(words[1]);
```

OUTPUT

```
abc
def
null
```

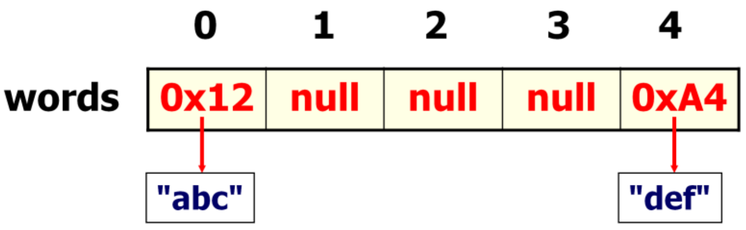
© A+ Computer Science - www.apluscompsci.com

String arrays are arrays of String references. Each spot in the array stores the location/memory address of a String Object. All spots in the array are initialized to null.



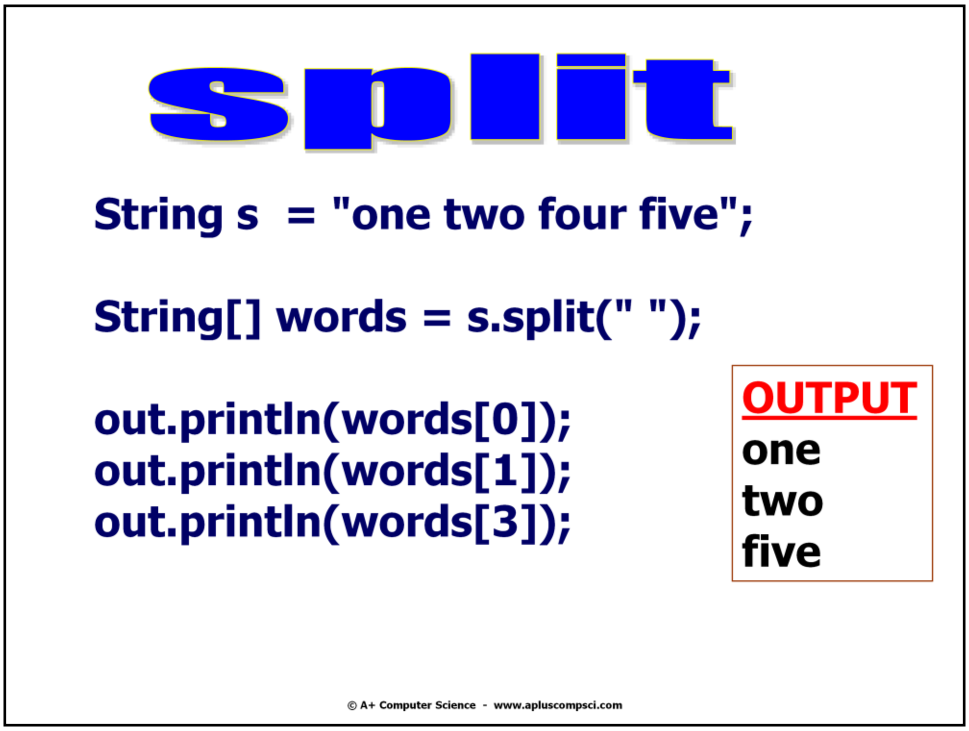
String arrays

```
String[] words = new String[5];
words[0] = "abc";
words[4] = "def";
```



© A+ Computer Science - www.apluscompsci.com

String arrays are arrays of String references. Each spot in the array stores the location/memory address of a String Object. All spots in the array are initialized to null.



split

```
String s  = "one two four five";

String[] words = s.split(" ");

out.println(words[0]);
out.println(words[1]);
out.println(words[3]);
```

OUTPUT

```
one
two
five
```

© A+ Computer Science - www.apluscompsci.com

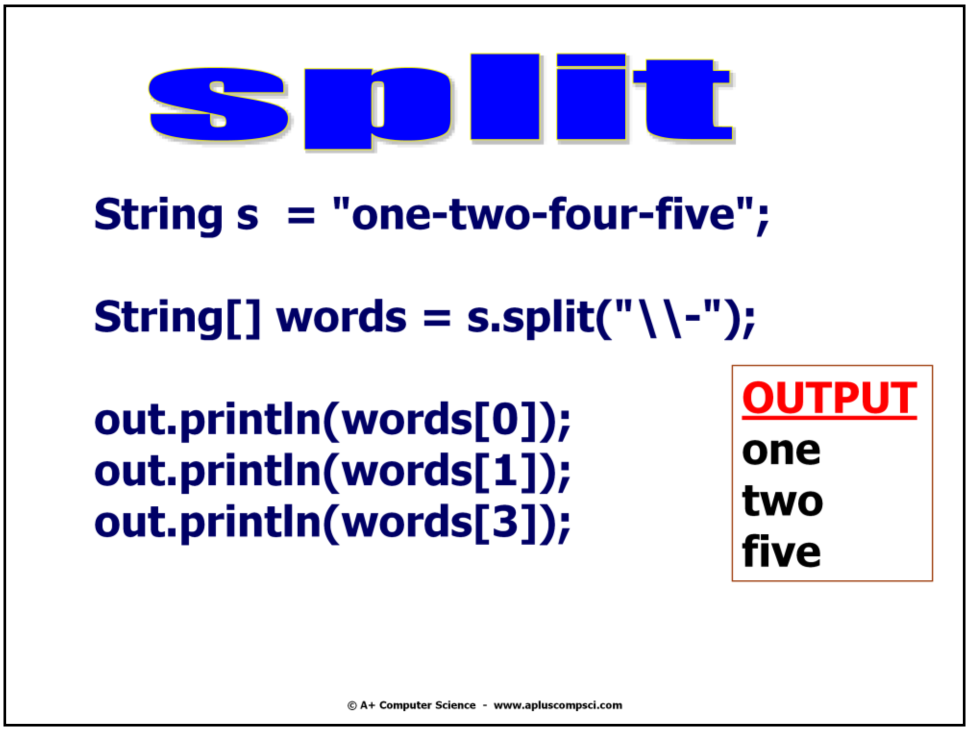
`split()` is a String method that returns an array of String references.

`split()` is very useful and functions like using a Scanner to chop up a multi-word line.

`split()` requires that a split value be provided. The split value tells `split()` what to split around.

stringray.java
splitone.java

© A+ Computer Science - www.apluscompsci.com



split

```
String s  = "one-two-four-five";

String[] words = s.split("\\-");

out.println(words[0]);
out.println(words[1]);
out.println(words[3]);
```

OUTPUT

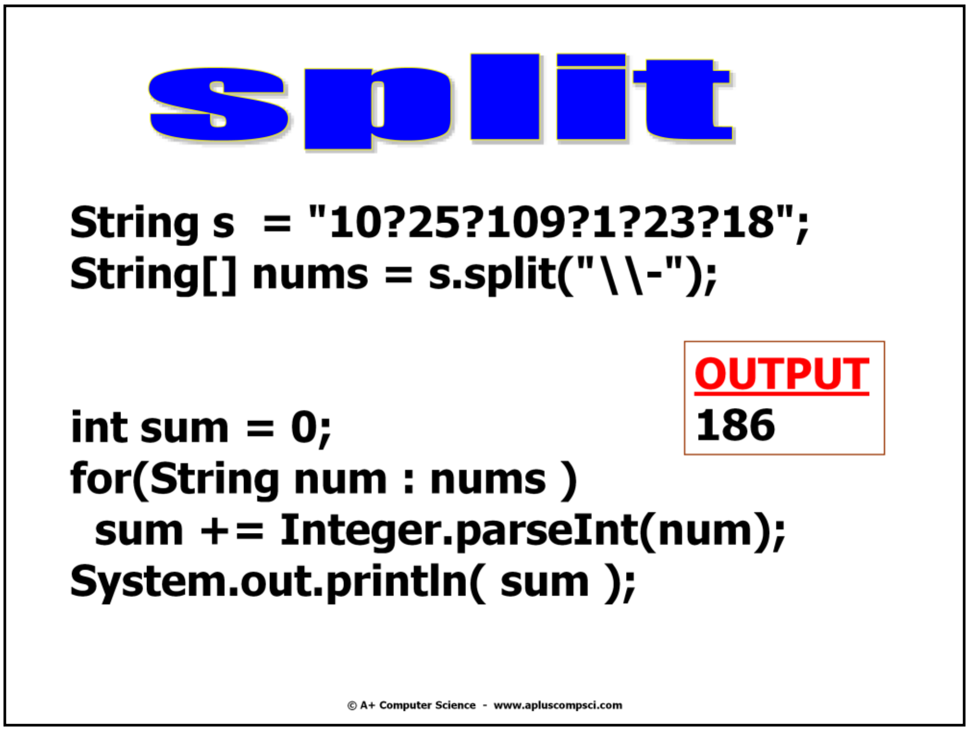
```
one
two
five
```

© A+ Computer Science - www.apluscompsci.com

`split()` is a String method that returns an array of String references.

`split()` is very useful and functions like using a Scanner to chop up a multi-word line.

`split()` requires that a split value be provided. – is being used as the split value in the example above. – is a regular expression symbol; as a result, `\\-` must be used if the – is to be treated literally and not as a reg ex symbol.



split

```
String s  = "10?25?109?1?23?18";
String[] nums = s.split("\\-");

int sum = 0;
for(String num : nums )
  sum += Integer.parseInt(num);
System.out.println( sum );
```

OUTPUT

```
186
```

© A+ Computer Science - www.apluscompsci.com

```
String s  = "10?25?109?1?23?18";
String[] nums = s.split("\\D+");
//split around non-digit characters
int sum = 0;
for(String num : nums )
  sum += Integer.parseInt(num);
//converts the String value to an
integer
System.out.println( sum );
```

This code will also output 186, but uses a different regex notation to accomplish the splitting.

The Pattern class has extensive documentation on regex notation for Java.

splittwo.java
splitthree.java
splitfour.java

© A+ Computer Science - www.apluscompsci.com

Start work on the labs

© A+ Computer Science - www.apluscompsci.com