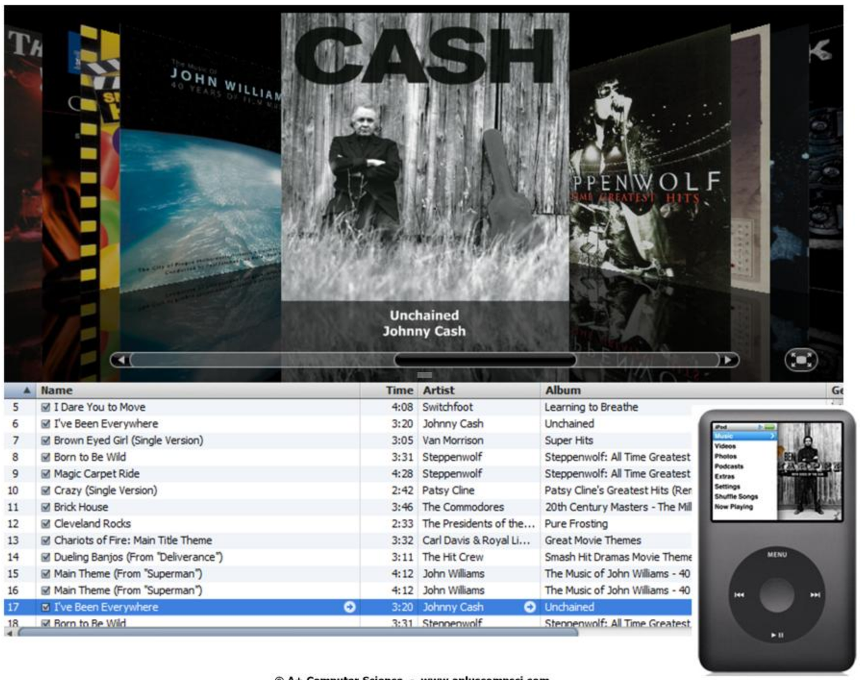
ArrayList and Lists

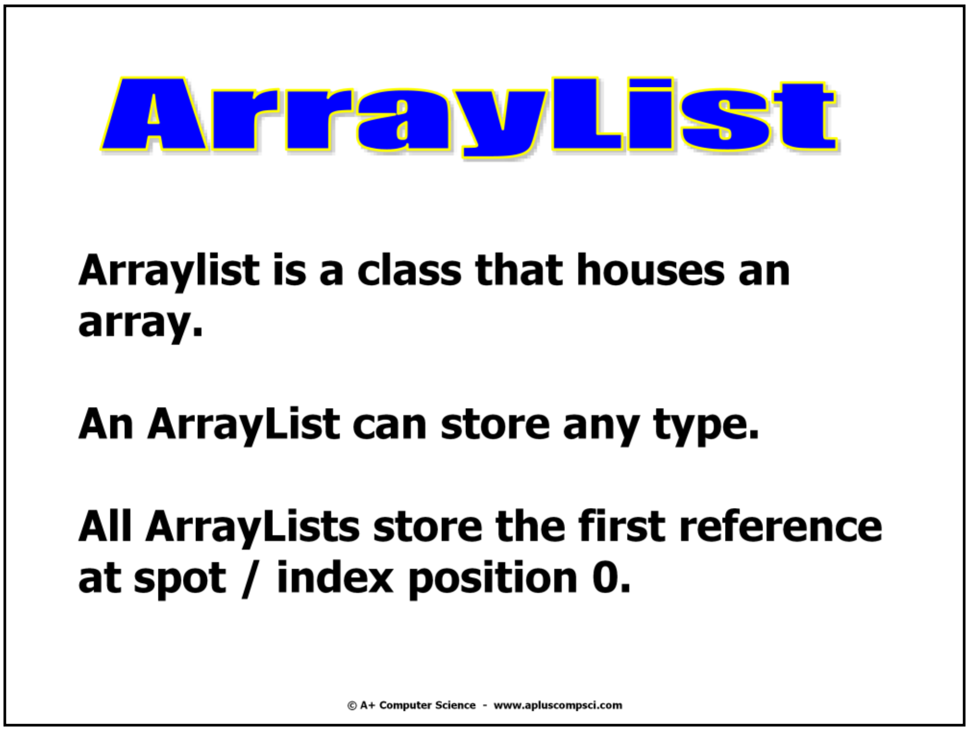
What is a list?



▲	Name	Time	Artist	Album
5	☑ I Dare You to Move	4:08	Switchfoot	Learning to Breathe
6	☑ I've Been Everywhere	3:20	Johnny Cash	Unchained
7	☑ Brown Eyed Girl (Single Version)	3:05	Van Morrison	Super Hits
8	☑ Born to Be Wild	3:31	Steppenwolf	Steppenwolf: All Time Greatest
9	☑ Magic Carpet Ride	4:28	Steppenwolf	Steppenwolf: All Time Greatest
10	☑ Crazy (Single Version)	2:42	Patsy Cline	Patsy Cline's Greatest Hits (Rer
11	☑ Brick House	3:46	The Commodores	20th Century Masters - The Mill
12	☑ Cleveland Rocks	2:33	The Presidents of the…	Pure Frosting
13	☑ Chariots of Fire: Main Title Theme	3:32	Carl Davis & Royal Li…	Great Movie Themes
14	☑ Dueling Banjos (From "Deliverance")	3:11	The Hit Crew	Smash Hit Dramas Movie Theme
15	☑ Main Theme (From "Superman")	4:12	John Williams	The Music of John Williams - 40
16	☑ Main Theme (From "Superman")	4:12	John Williams	The Music of John Williams - 40
17	☑ I've Been Everywhere	3:20	Johnny Cash	Unchained
18	☑ Born to Be Wild	3:31	Steppenwolf	Steppenwolf: All Time Greatest

What is an ArrayList?

© A+ Computer Science - www.apluscompsci.com



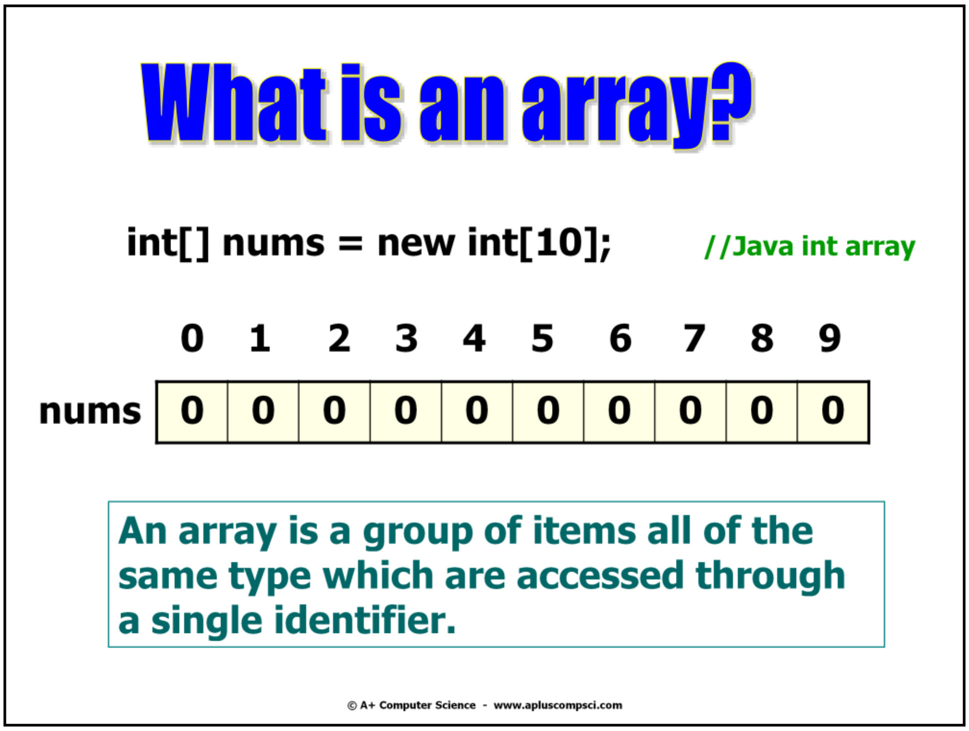
ArrayList

Arraylist is a class that houses an array.

An ArrayList can store any type.

All ArrayLists store the first reference at spot / index position 0.

ArrayList can store a reference to any type of Object.
ArrayList was built using an array[] of object references.

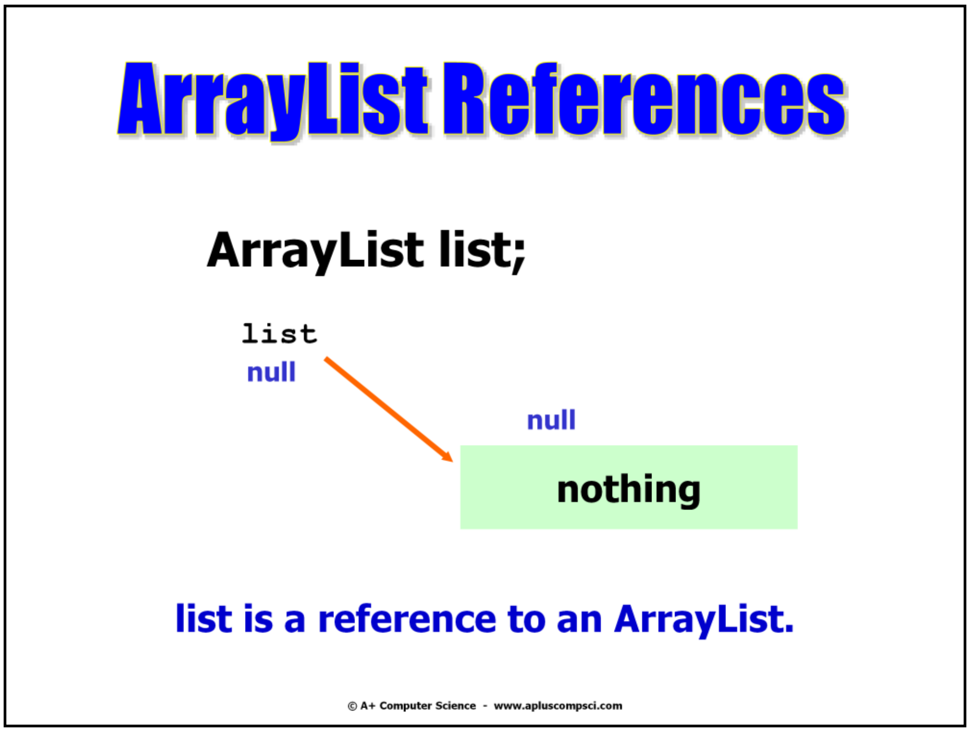


What is an array?

```
int[] nums = new int[10];        //Java int array
```

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

An array is a group of items all of the same type which are accessed through a single identifier.



ArrayList References

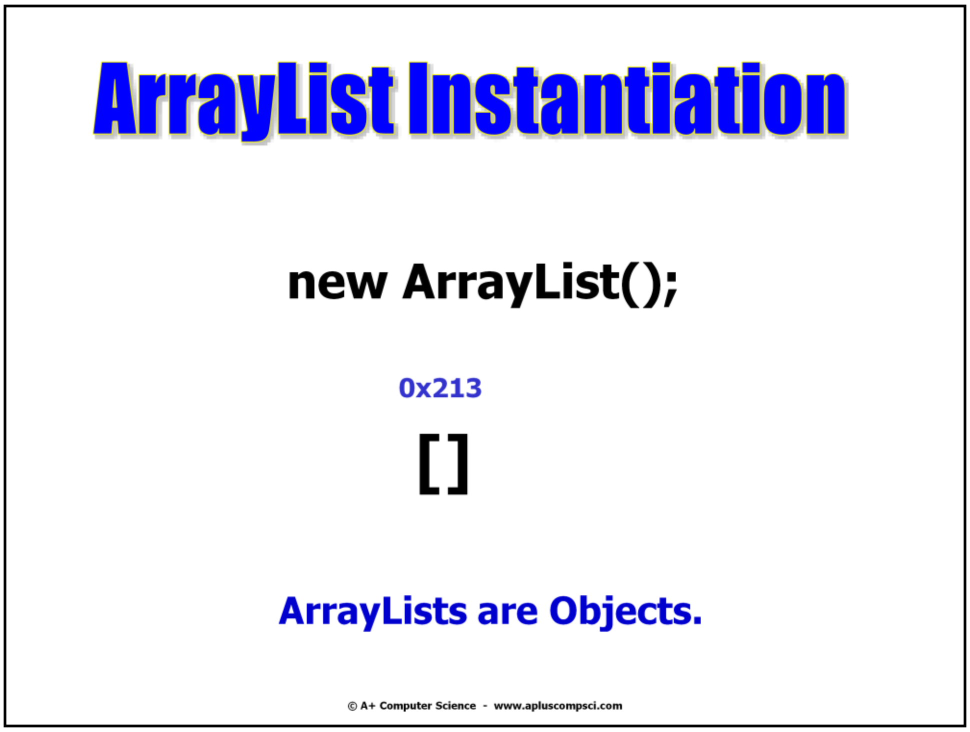
```
ArrayList list;
```

list
null
null
nothing

list is a reference to an ArrayList.

A reference variable is used to store the location of an Object. In most situations, a reference stores the actual memory address of an Object.

`list` stores the location / memory address of an ArrayList.



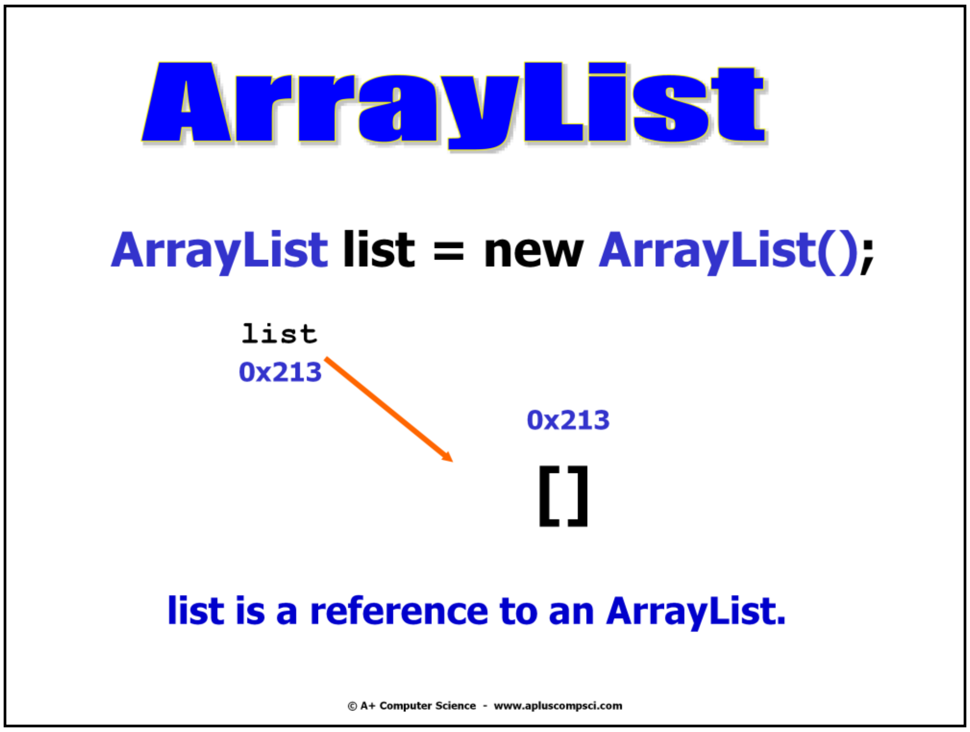
ArrayList Instantiation

```
new ArrayList();
```

0x213

[]

ArrayLists are Objects.



ArrayList

ArrayList list = new ArrayList();

list
0x213
0x213
[]

list is a reference to an ArrayList.

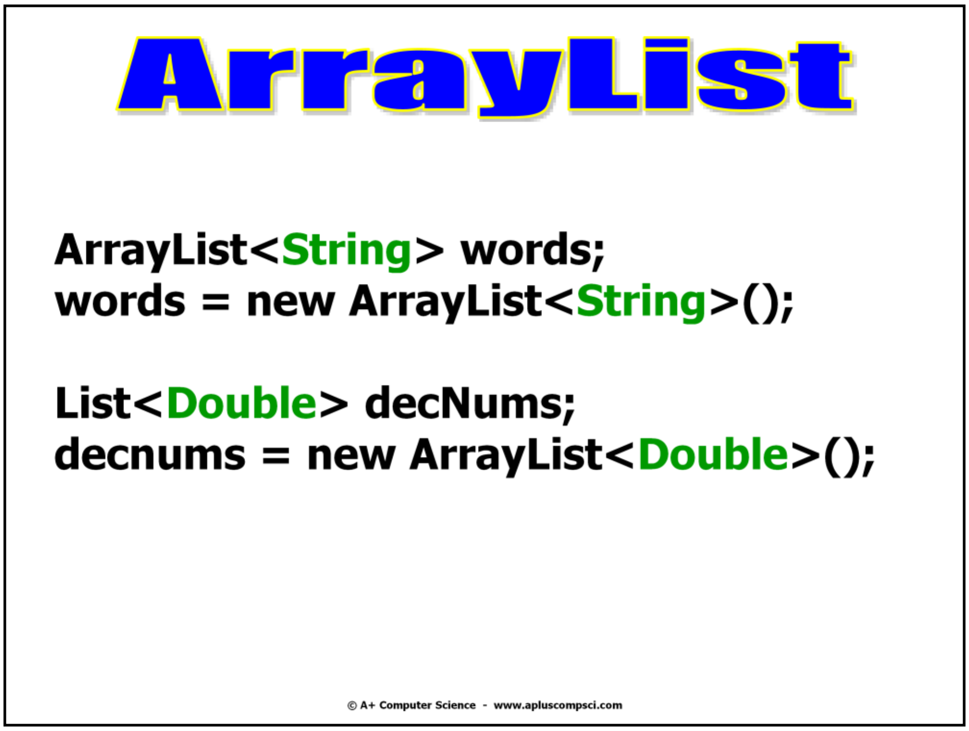
A reference variable is used to store the location of an Object. In most situations, a reference stores the actual memory address of an Object.

`list` stores the location / memory address of an ArrayList.

Open objects.java

© A+ Computer Science - www.apluscompsci.com

Generic ArrayLists



ArrayList

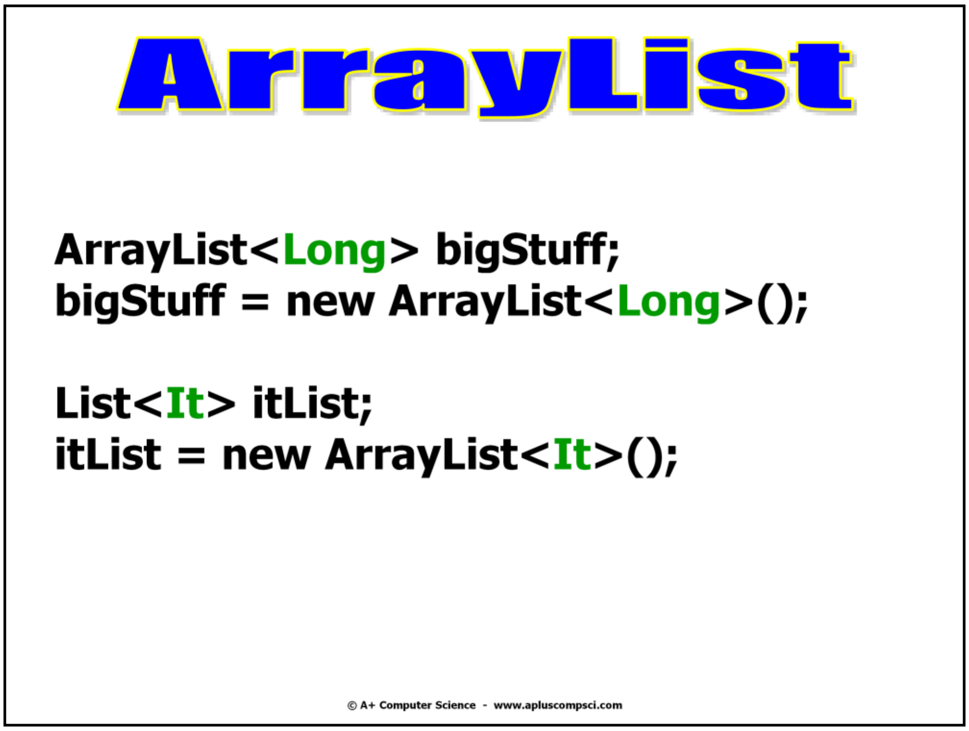
```
ArrayList<String> words;
words = new ArrayList<String>();

List<Double> decNums;
decnums = new ArrayList<Double>();
```

© A+ Computer Science - www.apluscompsci.com

In the example above, words can only store String references.   decNums can only store Double references.

Java knows the exact type of reference in both ArrayLists; thus, there is no need for casting when accessing class specific methods.

```
words.add("Hello");
out.println(words.get(0).charAt(0));
```



ArrayList

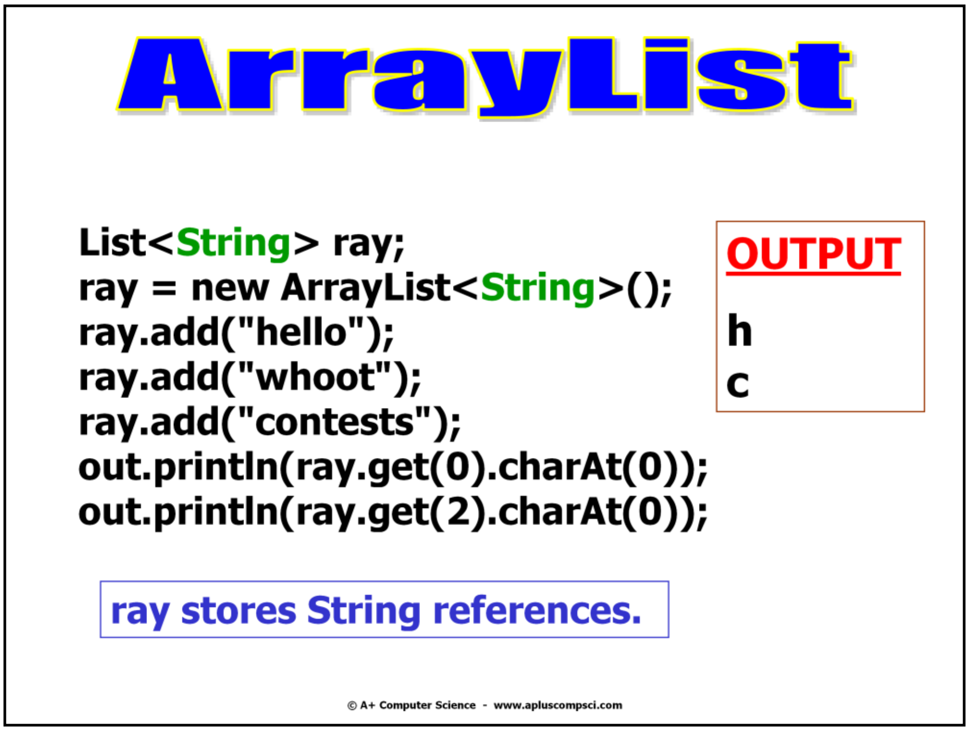
```
ArrayList<Long> bigStuff;
bigStuff = new ArrayList<Long>();

List<It> itList;
itList = new ArrayList<It>();
```

© A+ Computer Science - www.apluscompsci.com

In the example above, words can only store String references. decNums can only store Double references.

Java knows the exact type of reference in both ArrayLists; thus, there is no need for casting when accessing class specific methods.

```
itList.add(new It(34.21));
out.println(itList.get(0).getIt());
```



ArrayList

```
List<String> ray;
ray = new ArrayList<String>();
ray.add("hello");
ray.add("whoot");
ray.add("contests");
out.println(ray.get(0).charAt(0));
out.println(ray.get(2).charAt(0));
```

OUTPUT

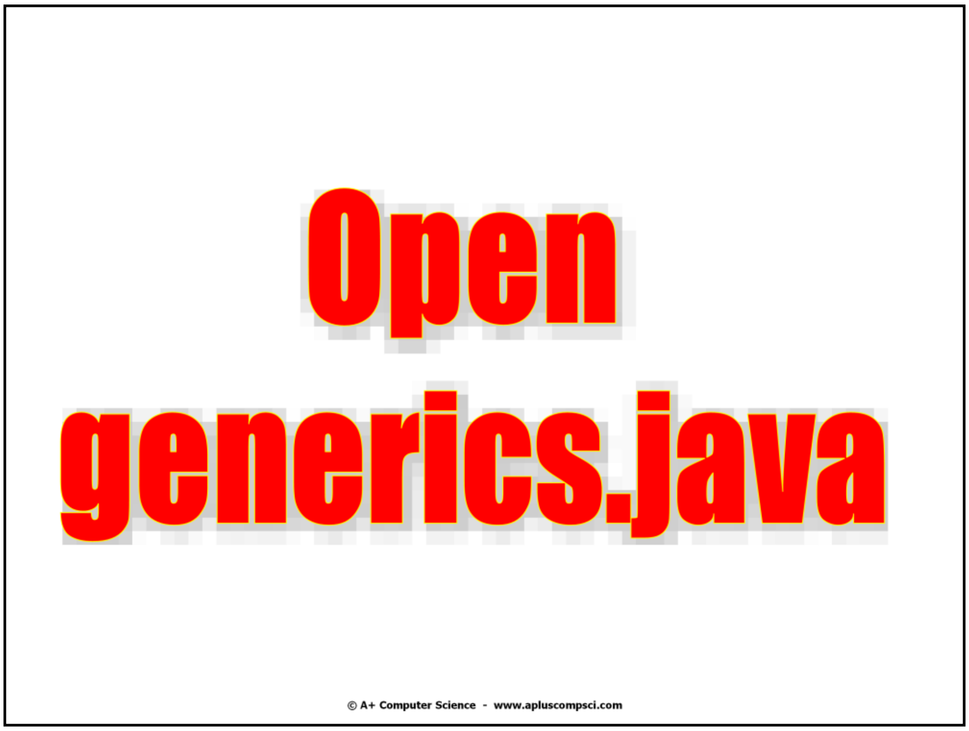
```
h
c
```

ray stores String references.

© A+ Computer Science - www.apluscompsci.com

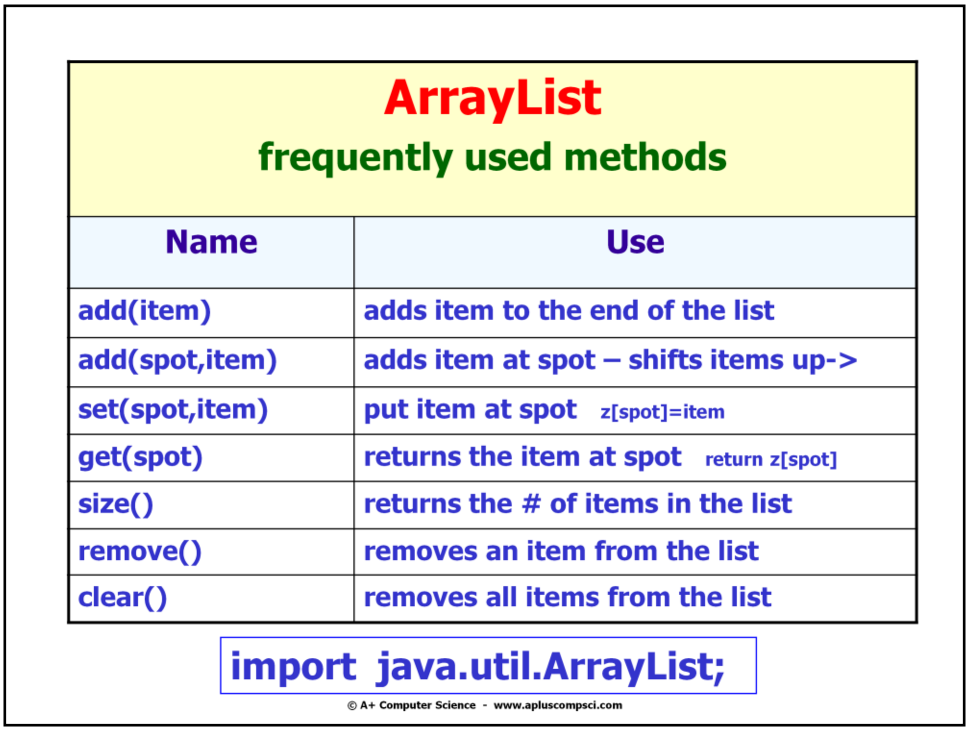
In the example above, ray is an ArrayList that stores String references. Casting would not be required to call non-Object methods on ray.

```
ray.add(0,"hello");
ray.add(1,"chicken");

out.println(ray.get(0).charAt(0));
out.println(ray.get(1).charAt(5));
```



Open generics.java

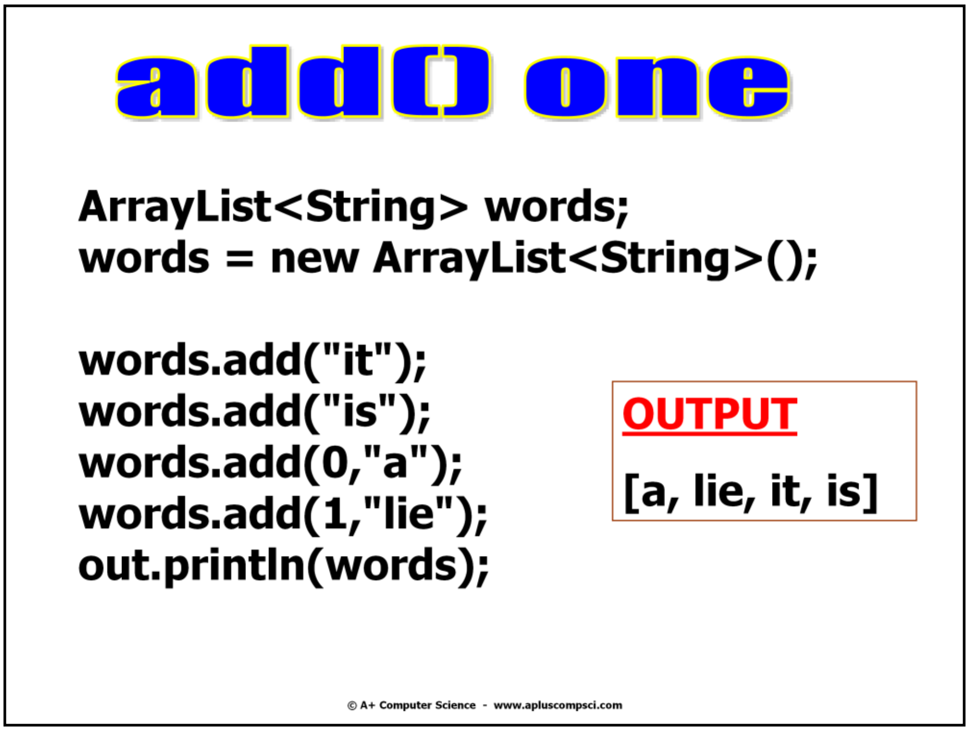
ArrayList Methods



ArrayList
frequently used methods

Name	Use
add(item)	adds item to the end of the list
add(spot,item)	adds item at spot – shifts items up->
set(spot,item)	put item at spot z[spot]=item
get(spot)	returns the item at spot return z[spot]
size()	returns the # of items in the list
remove()	removes an item from the list
clear()	removes all items from the list

```
import  java.util.ArrayList;
```



add() one

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

words.add("it");
words.add("is");
words.add(0,"a");
words.add(1,"lie");
out.println(words);
```

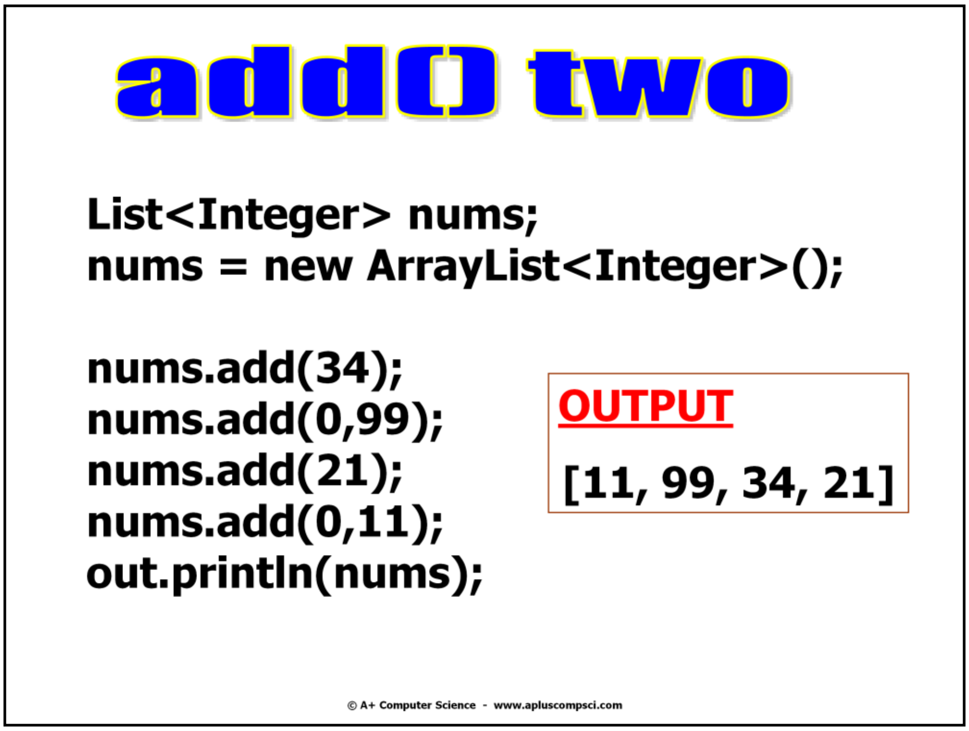
OUTPUT

```
[a, lie, it, is]
```

© A+ Computer Science - www.apluscompsci.com

The `add(item)` method adds the new item to the end of the ArrayList.

The `add(spot, item)` method adds the new item at the spot specified.
All other existing items are shifted toward the end of the ArrayList.
The add method does not override existing values.



add() two

```
List<Integer> nums;
nums = new ArrayList<Integer>();

nums.add(34);
nums.add(0,99);
nums.add(21);
nums.add(0,11);
out.println(nums);
```

OUTPUT

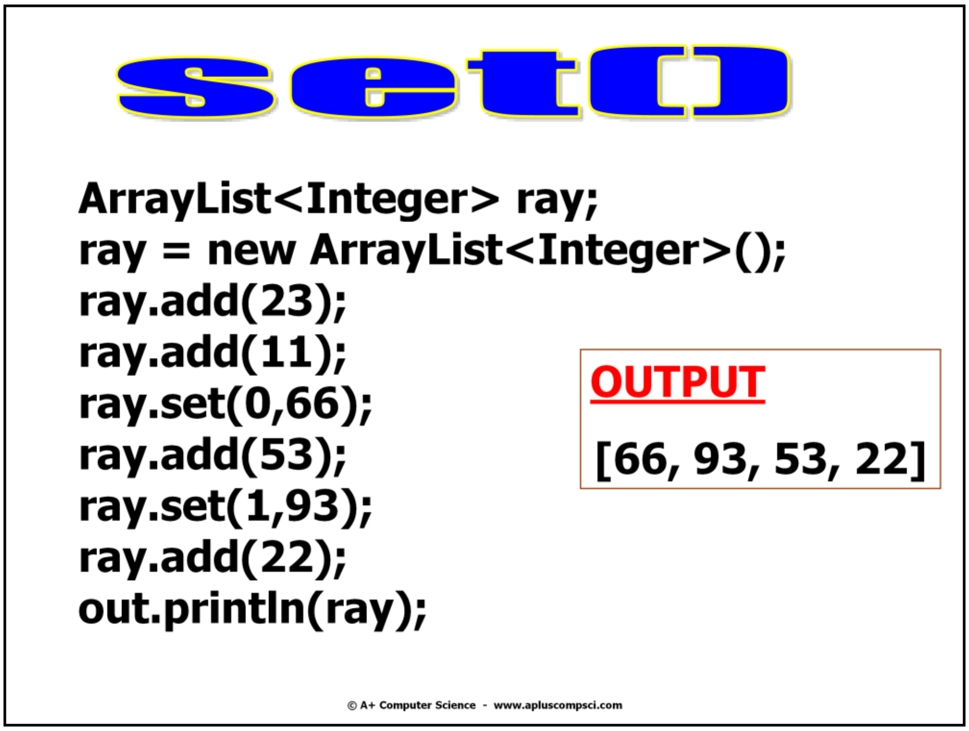
```
[11, 99, 34, 21]
```

© A+ Computer Science - www.apluscompsci.com

The `add(item)` method adds the new item to the end of the ArrayList.

The `add(spot, item)` method adds the new item at the spot specified.
All other existing items are shifted toward the end of the ArrayList.
The add method does not override existing values.

Open addone.java addtwo.java



set()

```
ArrayList<Integer> ray;
ray = new ArrayList<Integer>();
ray.add(23);
ray.add(11);
ray.set(0,66);
ray.add(53);
ray.set(1,93);
ray.add(22);
out.println(ray);
```

OUTPUT

```
[66, 93, 53, 22]
```

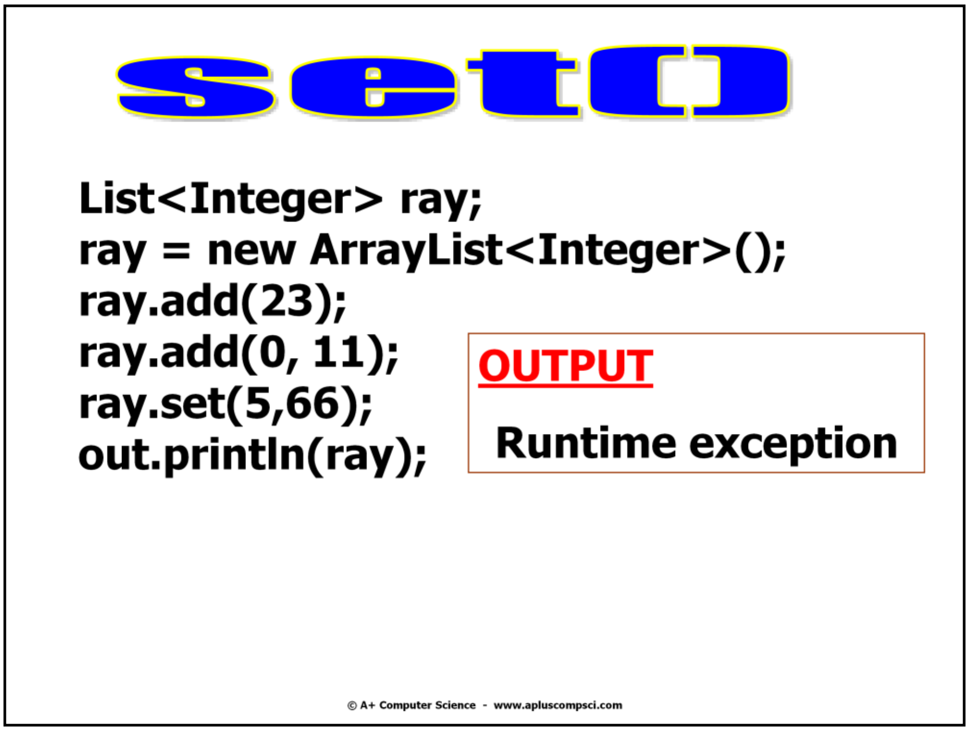
The `add(item)` method adds the new item to the end of the ArrayList.

The `set(spot, item)` method replaces the reference at spot with the new item.

The location / address of item is placed in spot.

You cannot set a location to a value if the location does not already exist.

This will result in an index out of bounds exception.



set()

```
List<Integer> ray;
ray = new ArrayList<Integer>();
ray.add(23);
ray.add(0, 11);
ray.set(5,66);
out.println(ray);
```

OUTPUT

```
Runtime exception
```

© A+ Computer Science - www.apluscompsci.com

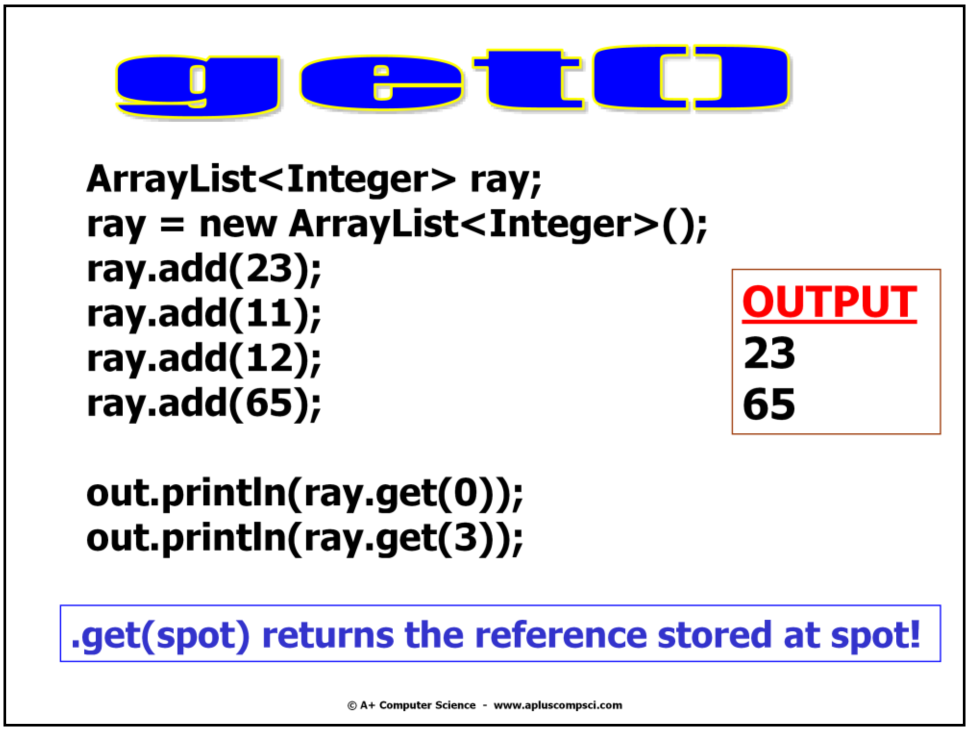
The `add(item)` method adds the new item to the end of the ArrayList.

The `set(spot, item)` method replaces the reference at spot with the new item.

The location / address of item is placed in spot.

You cannot set a location to a value if the location does not already exist.

This will result in an index out of bounds exception.



get()

```
ArrayList<Integer> ray;
ray = new ArrayList<Integer>();
ray.add(23);
ray.add(11);
ray.add(12);
ray.add(65);

out.println(ray.get(0));
out.println(ray.get(3));
```

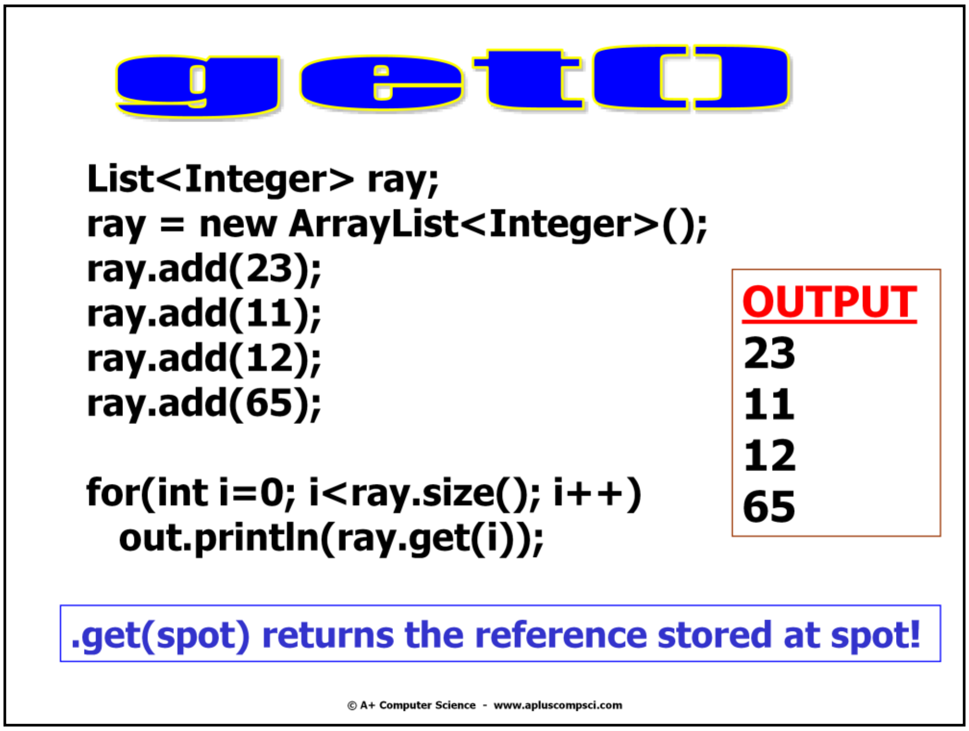
OUTPUT

```
23
65
```

.get(spot) returns the reference stored at spot!

© A+ Computer Science - www.apluscompsci.com

The get(spot) method returns the reference stored at spot.



get()

```
List<Integer> ray;
ray = new ArrayList<Integer>();
ray.add(23);
ray.add(11);
ray.add(12);
ray.add(65);

for(int i=0; i<ray.size(); i++)
   out.println(ray.get(i));
```

OUTPUT

```
23
11
12
65
```

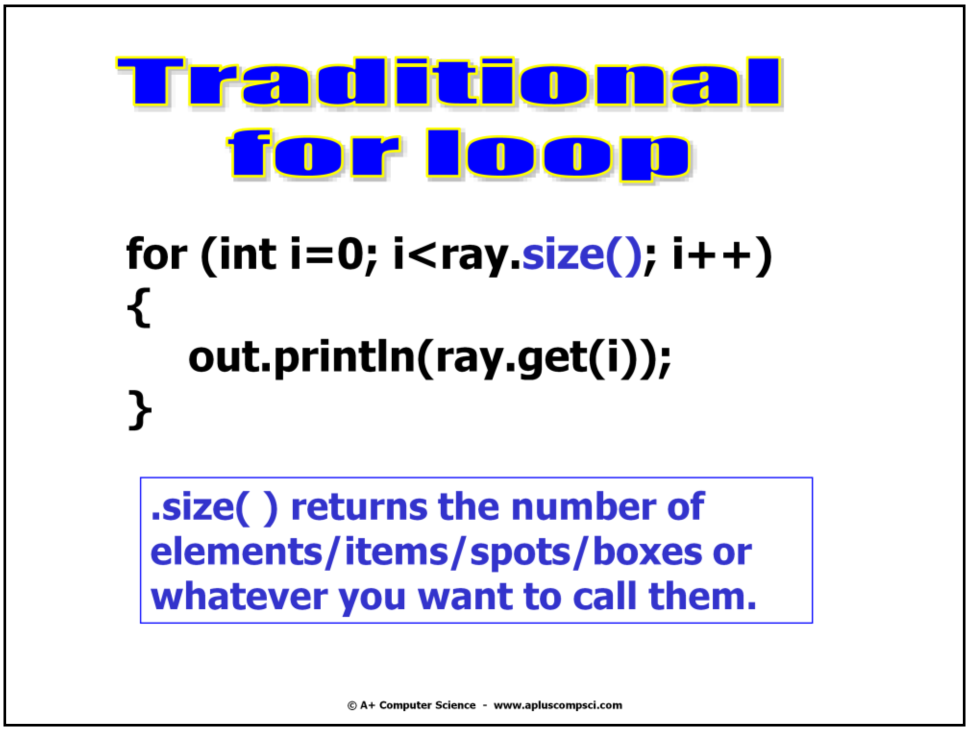
.get(spot) returns the reference stored at spot!

© A+ Computer Science - www.apluscompsci.com

The get(spot) method returns the reference stored at spot.

Open
set.java
get.java

Processing a list using loops



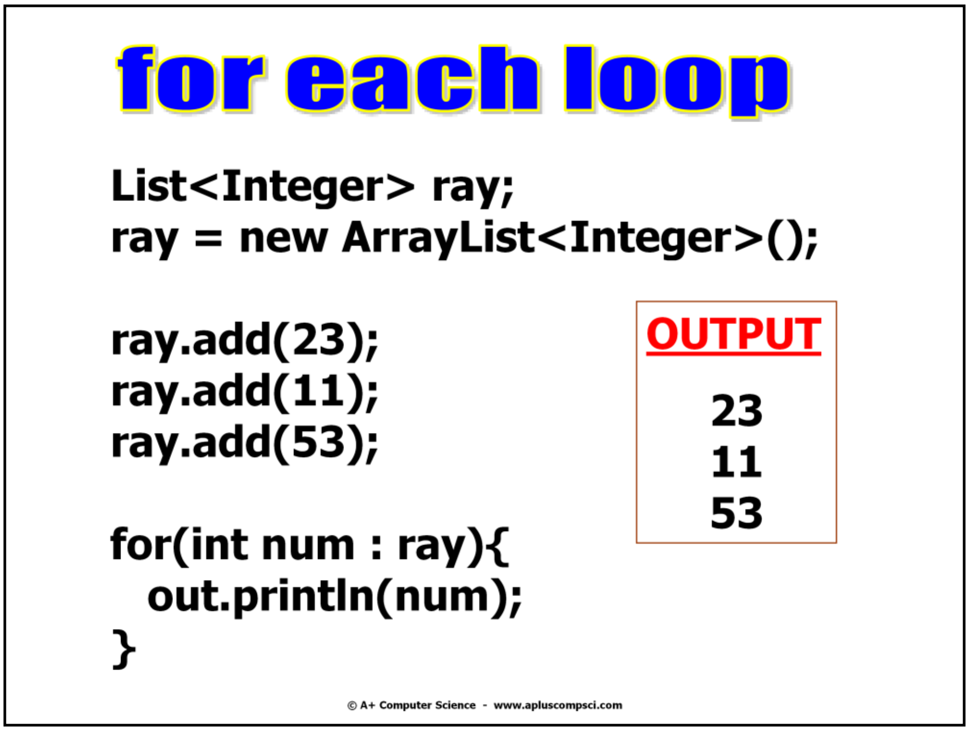
Traditional for loop

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

.size( ) returns the number of elements/items/spots/boxes or whatever you want to call them.

© A+ Computer Science - www.apluscompsci.com

The `size()` method returns the number of items in the ArrayList. If the ArrayList is storing seven references, `size()` would return a 7.



for each loop

```
List<Integer> ray;
ray = new ArrayList<Integer>();

ray.add(23);
ray.add(11);
ray.add(53);

for(int num : ray){
  out.println(num);
}
```

OUTPUT

```
23
11
53
```

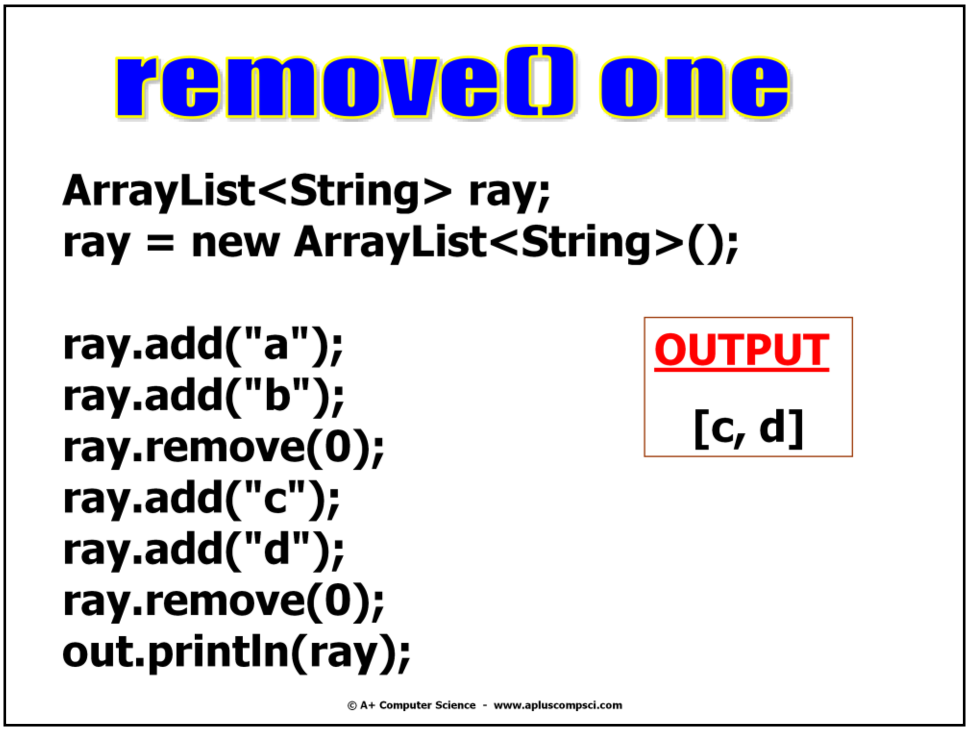
© A+ Computer Science - www.apluscompsci.com

The new for loop is great to print out Arrays and Collections. The new for loop extracts an item from ray each time it iterates. The new for loop is an iterator based loop. Once the loop reaches the end of ray, it stops iterating.

Open foreachloopone.java

© A+ Computer Science - www.apluscompsci.com

remove methods



remove() one

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

ray.add("a");
ray.add("b");
ray.remove(0);
ray.add("c");
ray.add("d");
ray.remove(0);
out.println(ray);
```

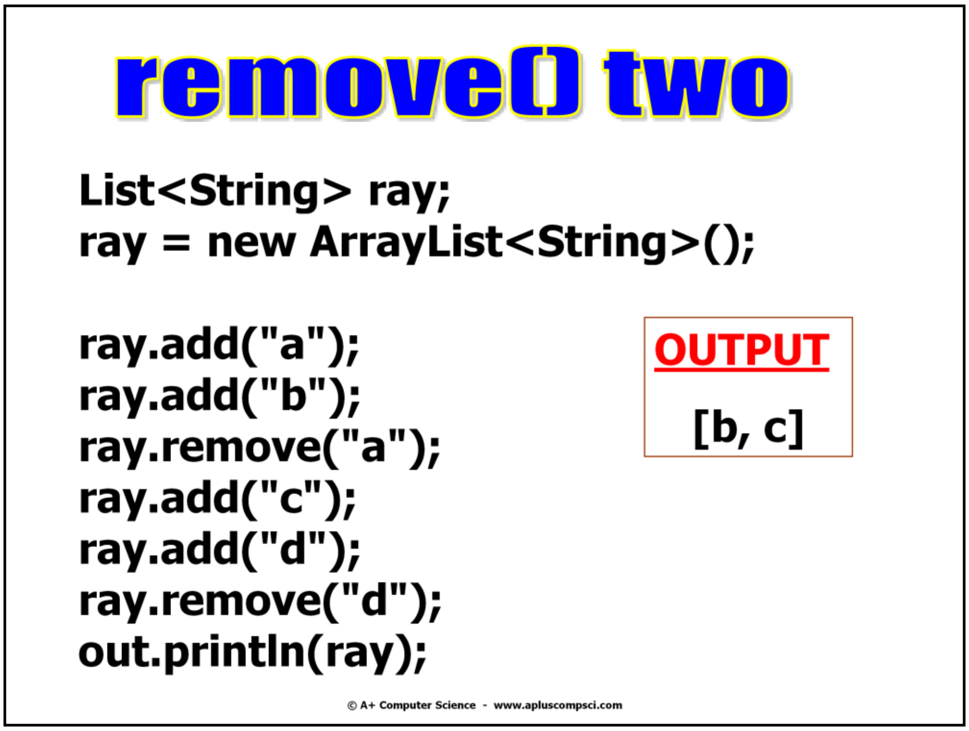
OUTPUT

```
[c, d]
```

The remove method will remove the item at the specified spot / location or the specified value. When an item is removed, all items above the removed item are shifted down toward the front of the ArrayList. All items are shifted to the left.

`[a, b]` becomes `[b]`

`[b, c, d]` becomes `[c, d]`



remove() two

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

ray.add("a");
ray.add("b");
ray.remove("a");
ray.add("c");
ray.add("d");
ray.remove("d");
out.println(ray);
```

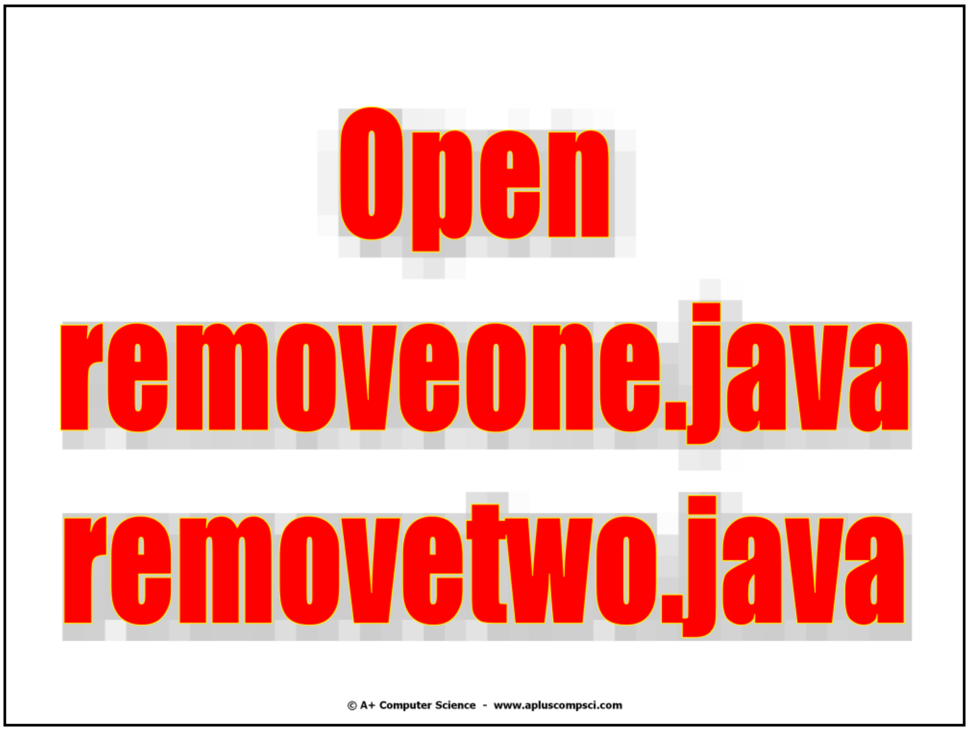
OUTPUT

```
[b, c]
```

The remove method will remove the item at the specified spot / location or the specified value. When an item is removed, all items above the removed item are shifted down toward the front of the ArrayList. All items are shifted to the left.

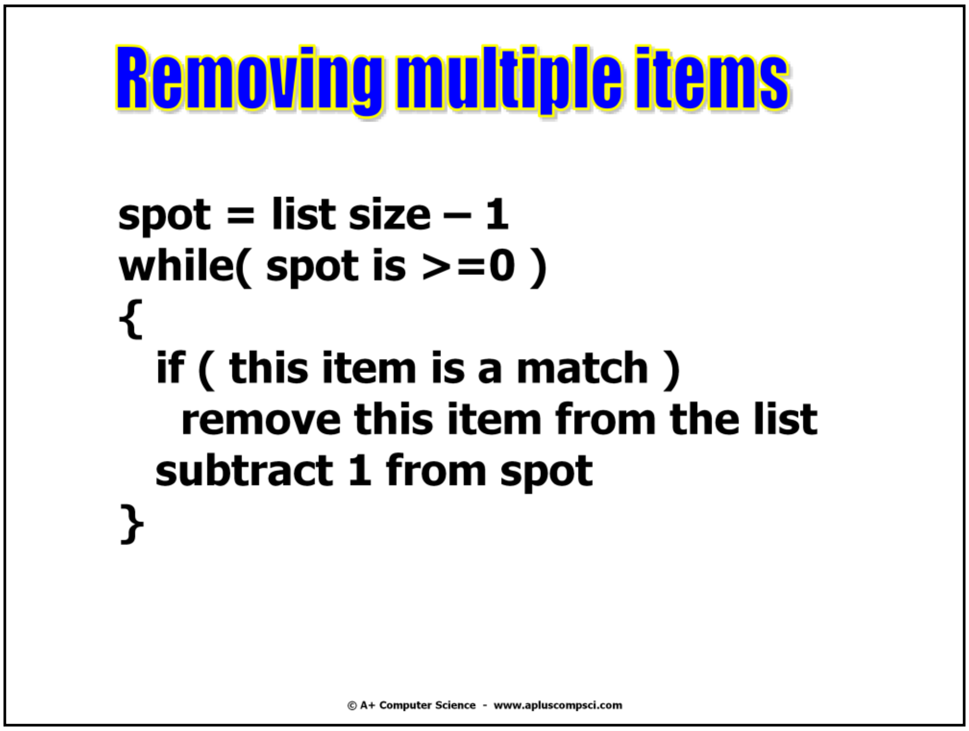
`[a, b]` becomes `[b]`

`[b, c, d]` becomes `[b, c]`



Open removeone.java removetwo.java

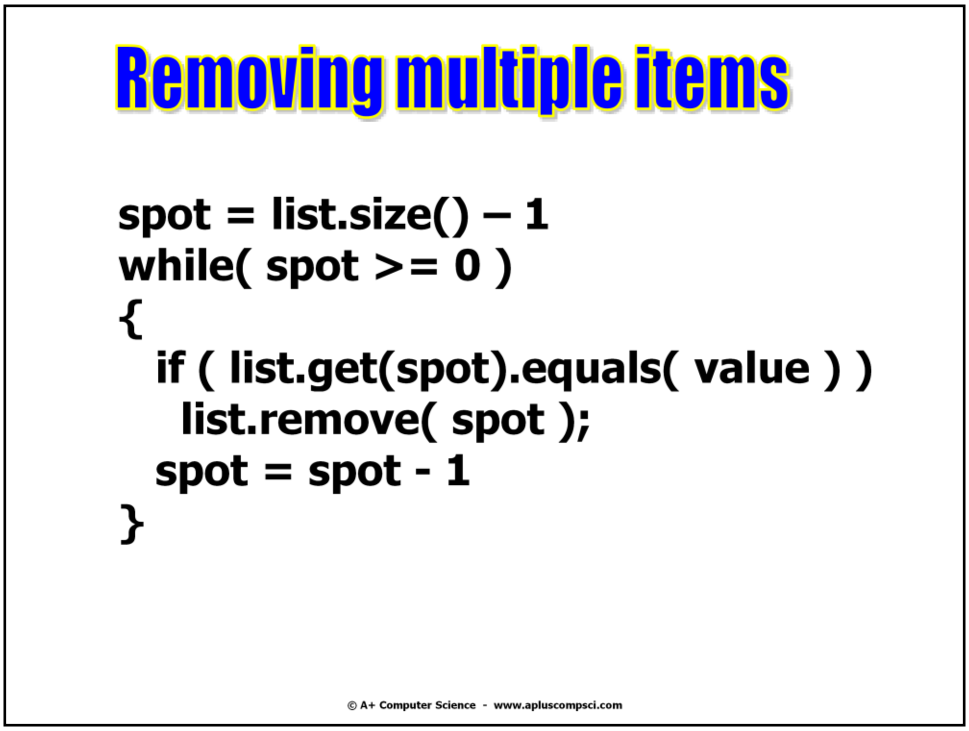
Removing Multiple Items



Removing multiple items

```
spot = list size – 1
while( spot is >=0 )
{
  if ( this item is a match )
    remove this item from the list
  subtract 1 from spot
}
```

In order to remove multiple values from an ArrayList, a loop must be used.
The loop will need an if statement to identify the items to remove.
Keep in mind that the ArrayList shrinks when items are removed.
The items in the ArrayList shift down towards spot 0.
The loop must start at size()-1 and go down in order to account for the shift.

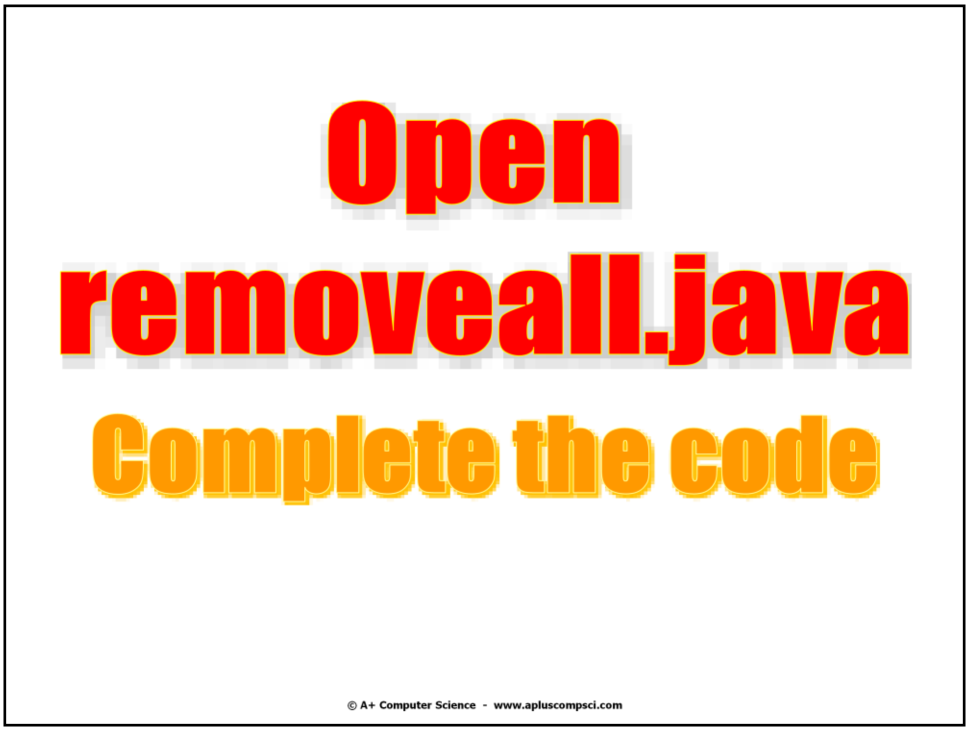


Removing multiple items

```
spot = list.size() – 1
while( spot >= 0 )
{
  if ( list.get(spot).equals( value ) )
    list.remove( spot );
  spot = spot - 1
}
```

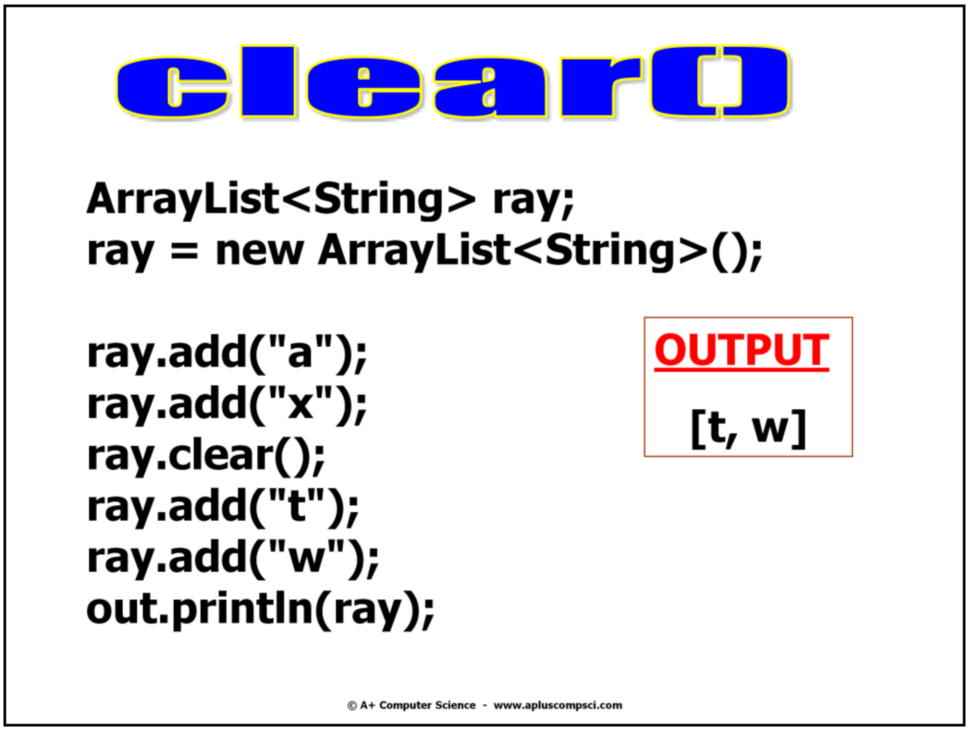
© A+ Computer Science - www.apluscompsci.com

In order to remove multiple values from an ArrayList, a loop must be used.
The loop will need an if statement to identify the items to remove.
Keep in mind that the ArrayList shrinks when items are removed.
The items in the ArrayList shift down towards spot 0.
The loop must start at size()-1 and go down in order to account for the shift.



Open removeall.java

Complete the code



clear()

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

ray.add("a");
ray.add("x");
ray.clear();
ray.add("t");
ray.add("w");
out.println(ray);
```

OUTPUT

```
[t, w]
```

© A+ Computer Science - www.apluscompsci.com

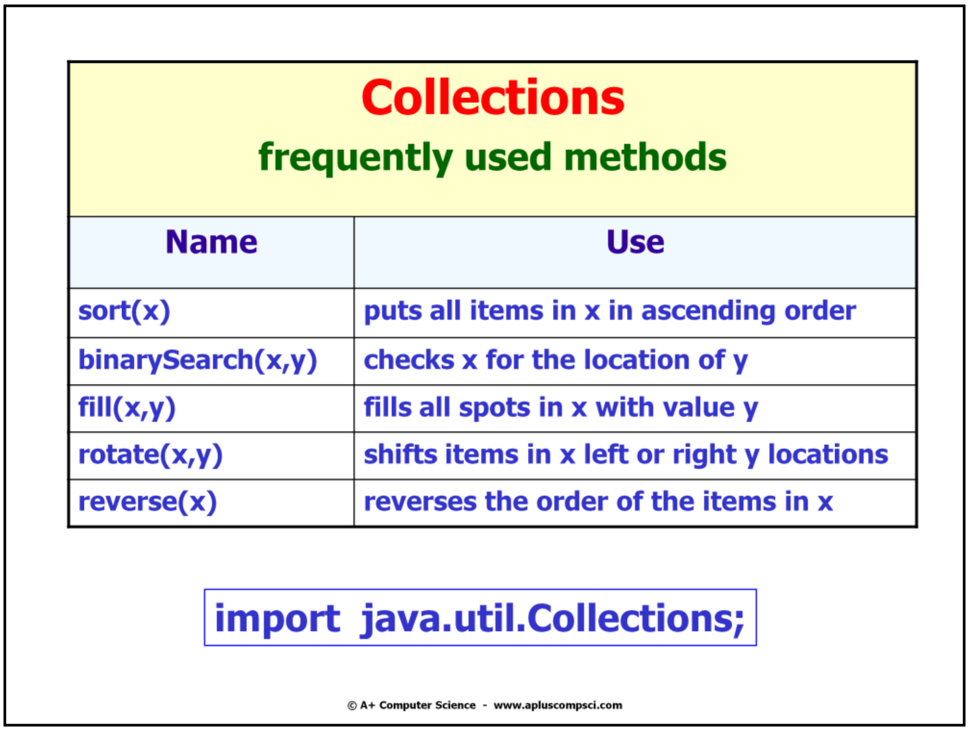
The `clear()` method removes all items from the ArrayList.

The ArrayList becomes an `[]` empty ArrayList with a size() of 0.

The clear() method essentially performs the same operation as instantiating a new ArrayList.

Open clear.java

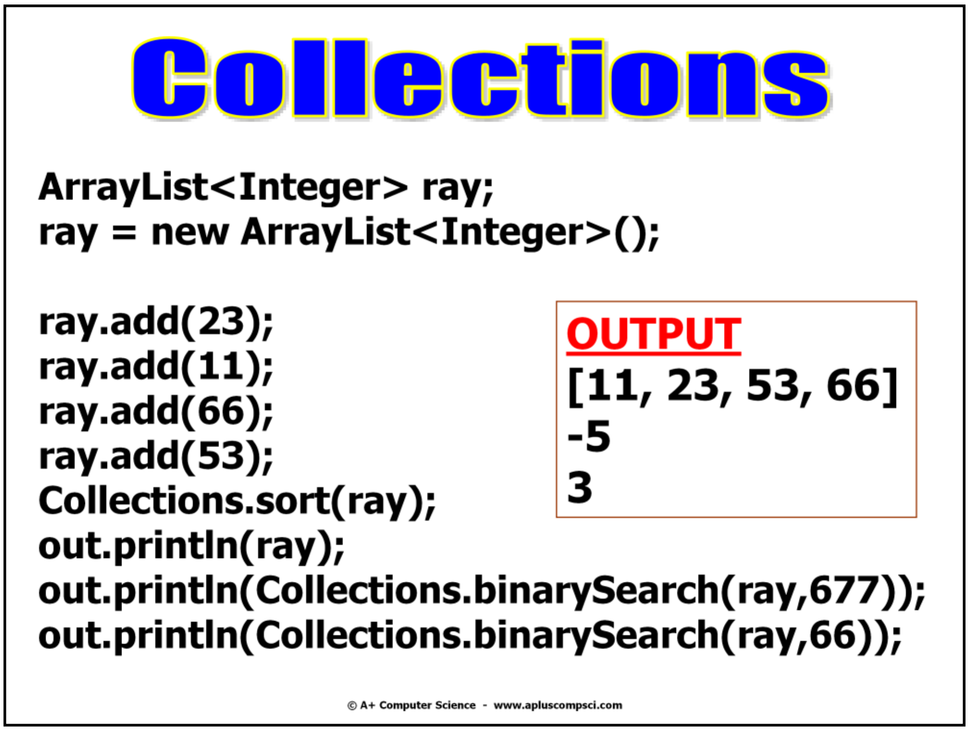
Collections class



Collections
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
fill(x,y)	fills all spots in x with value y
rotate(x,y)	shifts items in x left or right y locations
reverse(x)	reverses the order of the items in x

```
import  java.util.Collections;
```



Collections

```
ArrayList<Integer> ray;
ray = new ArrayList<Integer>();

ray.add(23);
ray.add(11);
ray.add(66);
ray.add(53);
Collections.sort(ray);
out.println(ray);
out.println(Collections.binarySearch(ray,677));
out.println(Collections.binarySearch(ray,66));
```

OUTPUT

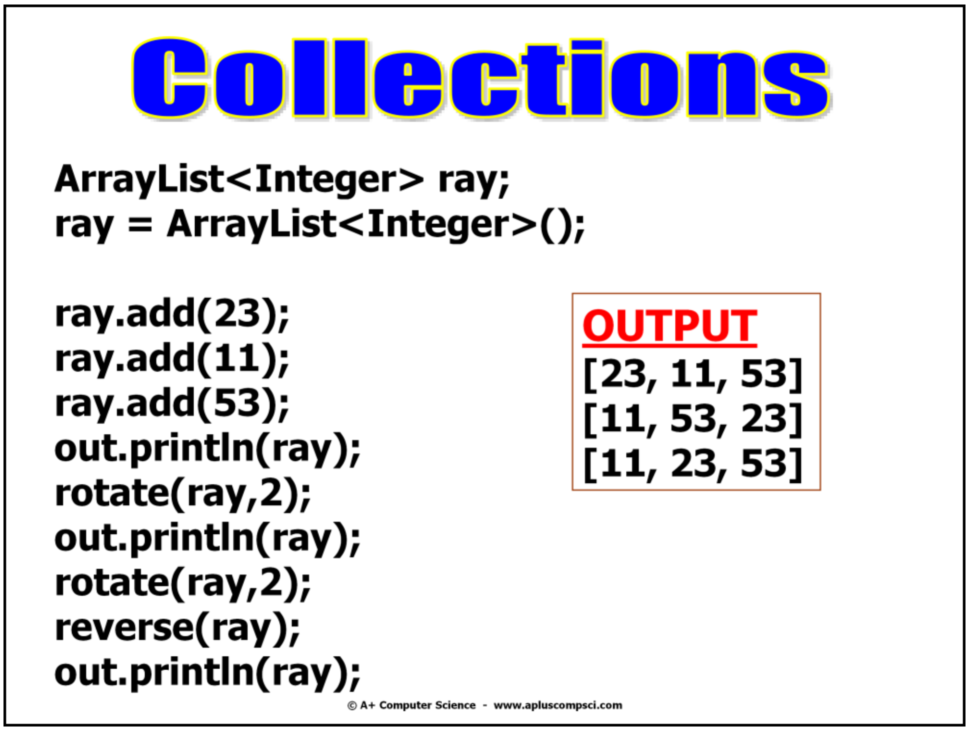
```
[11, 23, 53, 66]
-5
3
```

© A+ Computer Science - www.apluscompsci.com

Collections.sort() will put all items in natural ascending order.

Collectoins.binarySearch() will locate an item. If the item does not exist, binarySearch() will return -1+ -(where the value would be if it was there).

-3 is -1 + -2(2 is the spot where the item would be)



Collections

```
ArrayList<Integer> ray;
ray = ArrayList<Integer>();

ray.add(23);
ray.add(11);
ray.add(53);
out.println(ray);
rotate(ray,2);
out.println(ray);
rotate(ray,2);
reverse(ray);
out.println(ray);
```

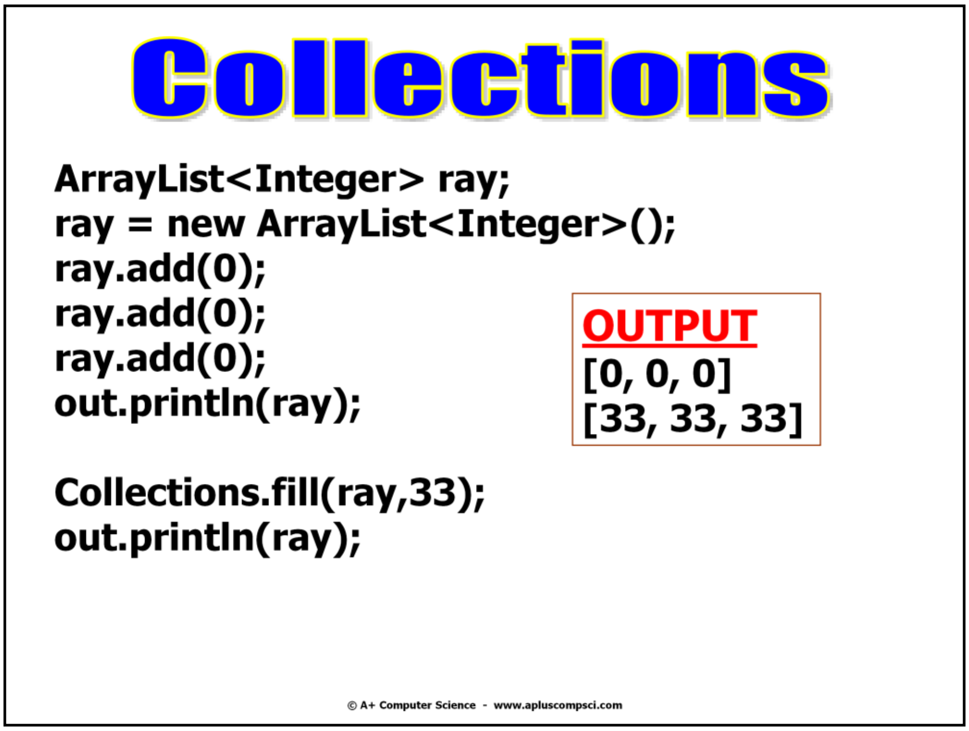
OUTPUT

```
[23, 11, 53]
[11, 53, 23]
[11, 23, 53]
```

© A+ Computer Science - www.apluscompsci.com

Collections.rotate() rotates items to the right or to the left a specified number of spots / positions. A negative number rotates to the left and a positive number rotates to the right.

Collections. reverse() reverses the order of all items.



Collections

```
ArrayList<Integer> ray;
ray = new ArrayList<Integer>();
ray.add(0);
ray.add(0);
ray.add(0);
out.println(ray);

Collections.fill(ray,33);
out.println(ray);
```

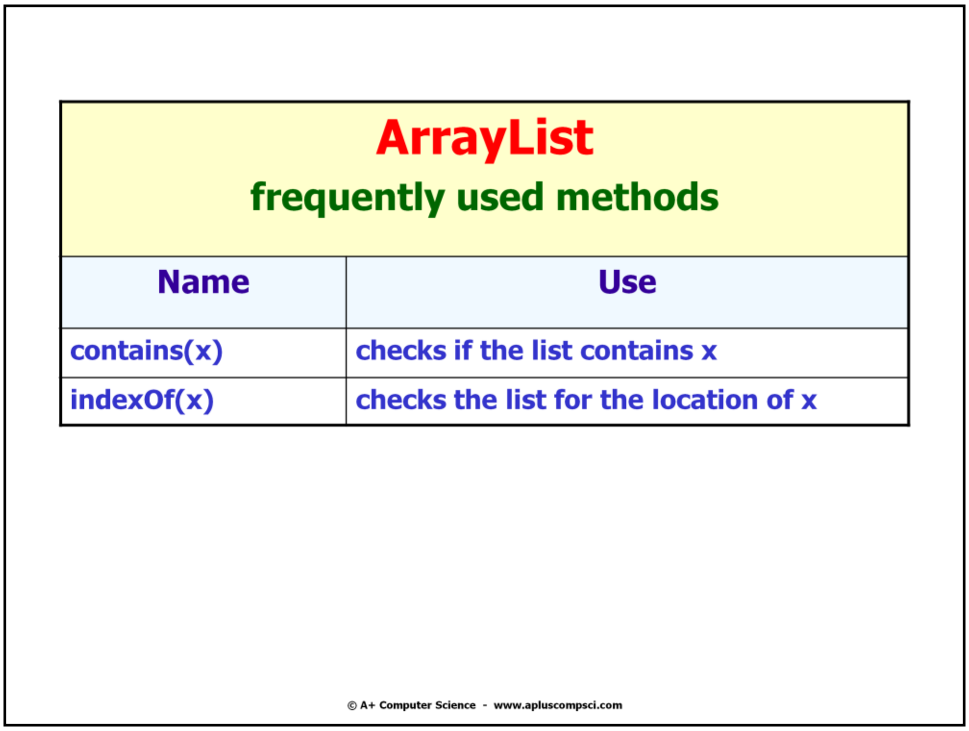
OUTPUT

```
[0, 0, 0]
[33, 33, 33]
```

Collections.fill() will fill in all spots with a specified value.

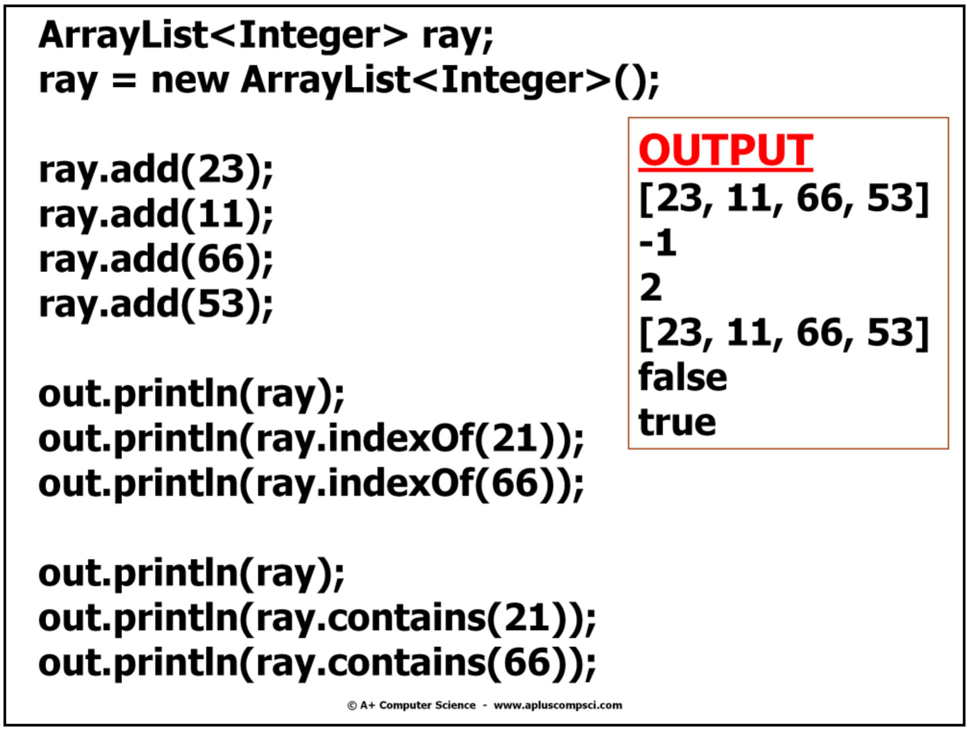
Open
binarysearch.java
rotate.java
fill.java

search methods



ArrayList
frequently used methods

Name	Use
contains(x)	checks if the list contains x
indexOf(x)	checks the list for the location of x



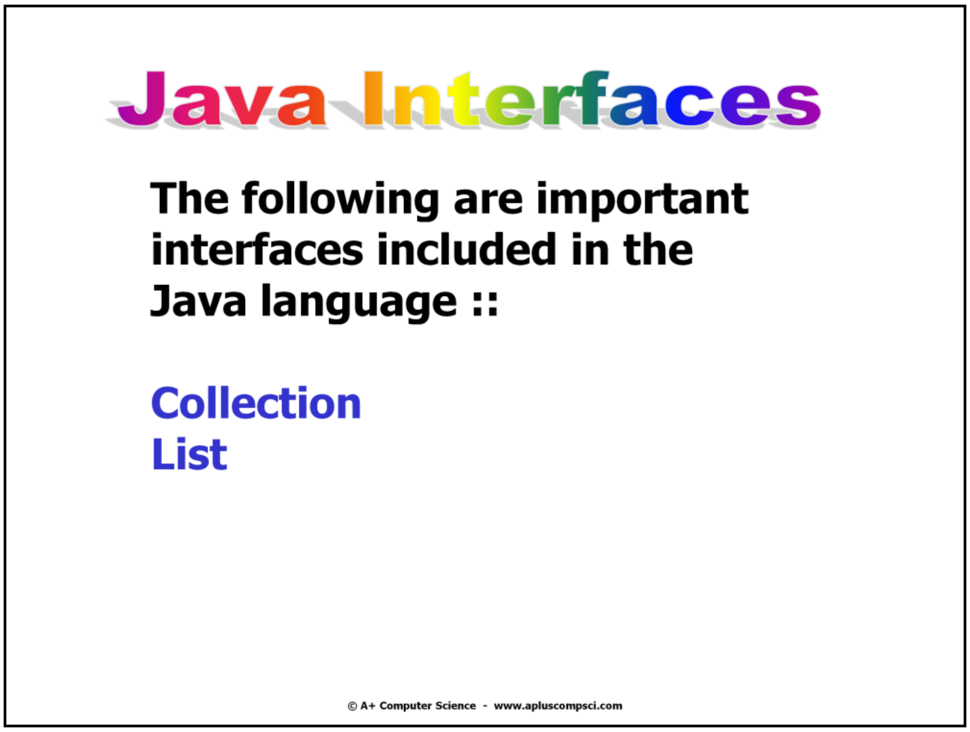
```
ArrayList<Integer> ray;
ray = new ArrayList<Integer>();

ray.add(23);
ray.add(11);
ray.add(66);
ray.add(53);

out.println(ray);
out.println(ray.indexOf(21));
out.println(ray.indexOf(66));

out.println(ray);
out.println(ray.contains(21));
out.println(ray.contains(66));
```

OUTPUT

```
[23, 11, 66, 53]
-1
2
[23, 11, 66, 53]
false
true
```

Open search.java

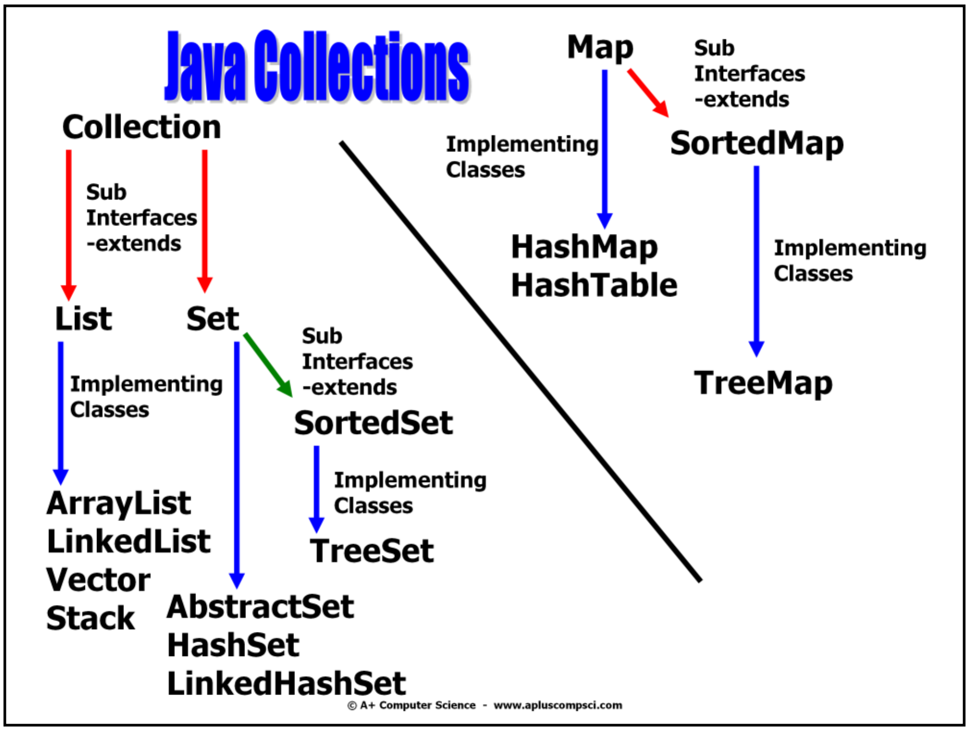
Java Collections



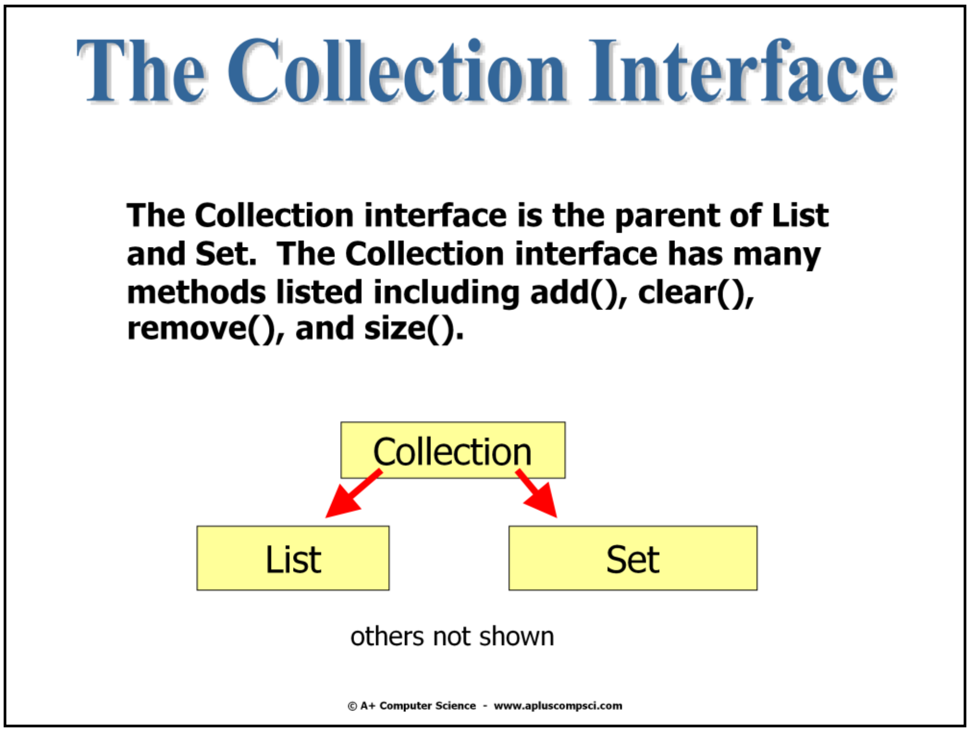
Java Interfaces

The following are important interfaces included in the Java language ::

Collection
List

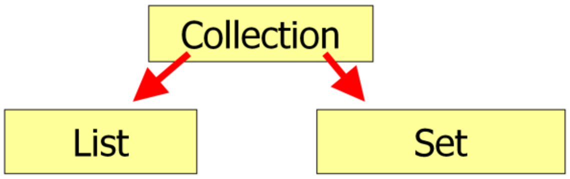


This Collections hierarchy chart is very important. It is a must to know which classes implement which interfaces and which interfaces extend which interfaces.

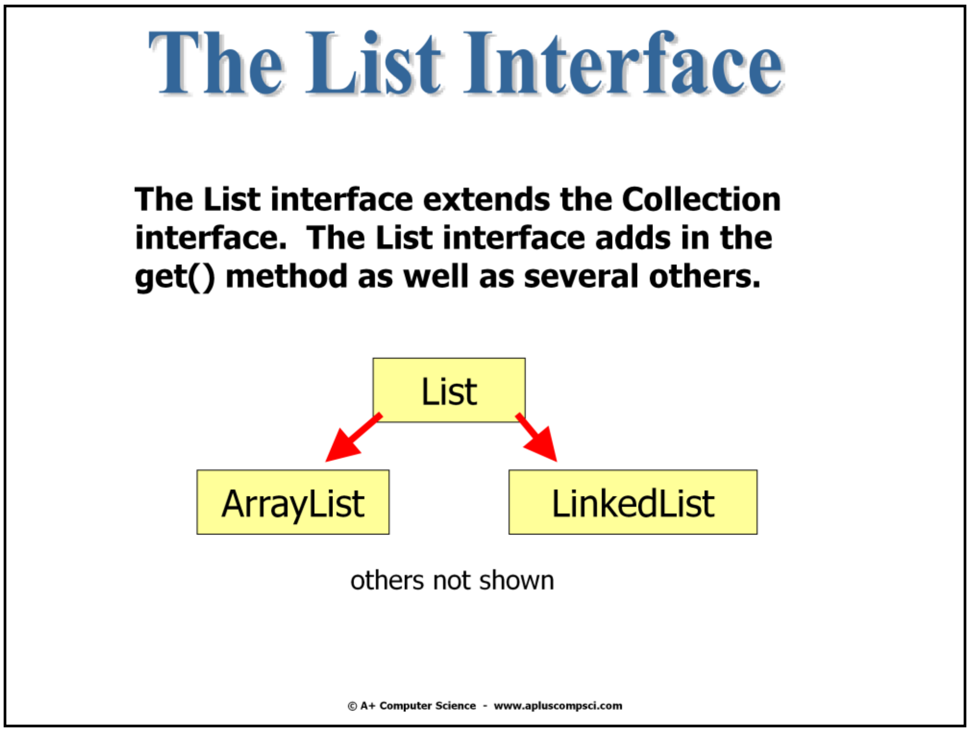


The Collection Interface

The Collection interface is the parent of List and Set. The Collection interface has many methods listed including add(), clear(), remove(), and size().

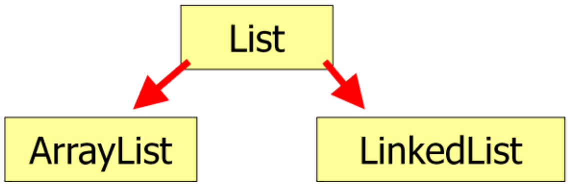


others not shown

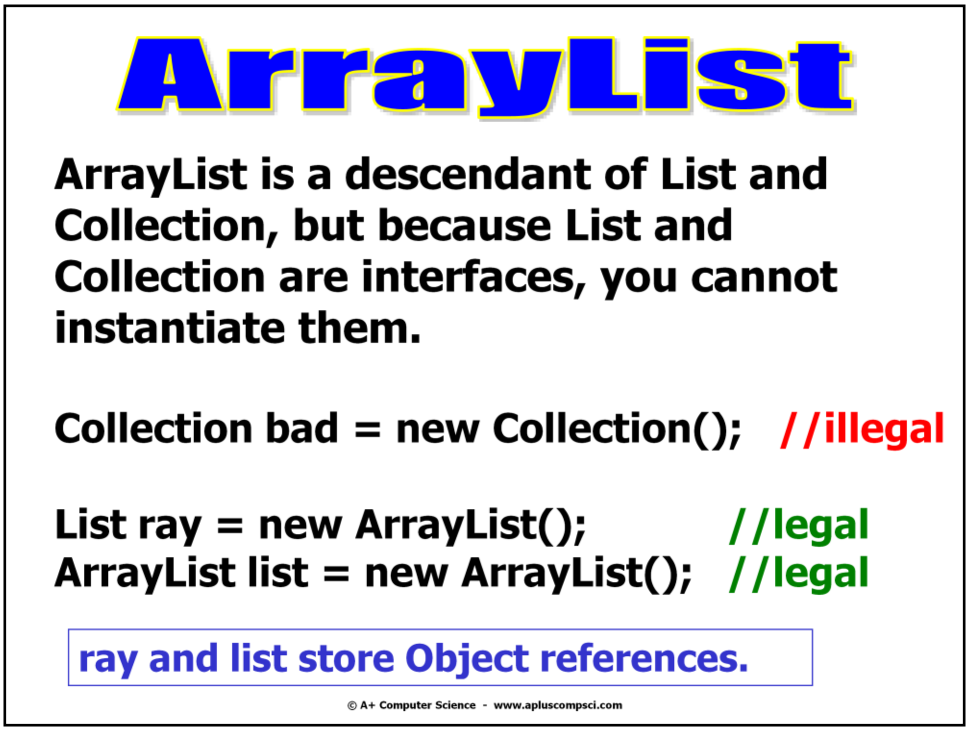


The List Interface

The List interface extends the Collection interface. The List interface adds in the get() method as well as several others.



others not shown



ArrayList

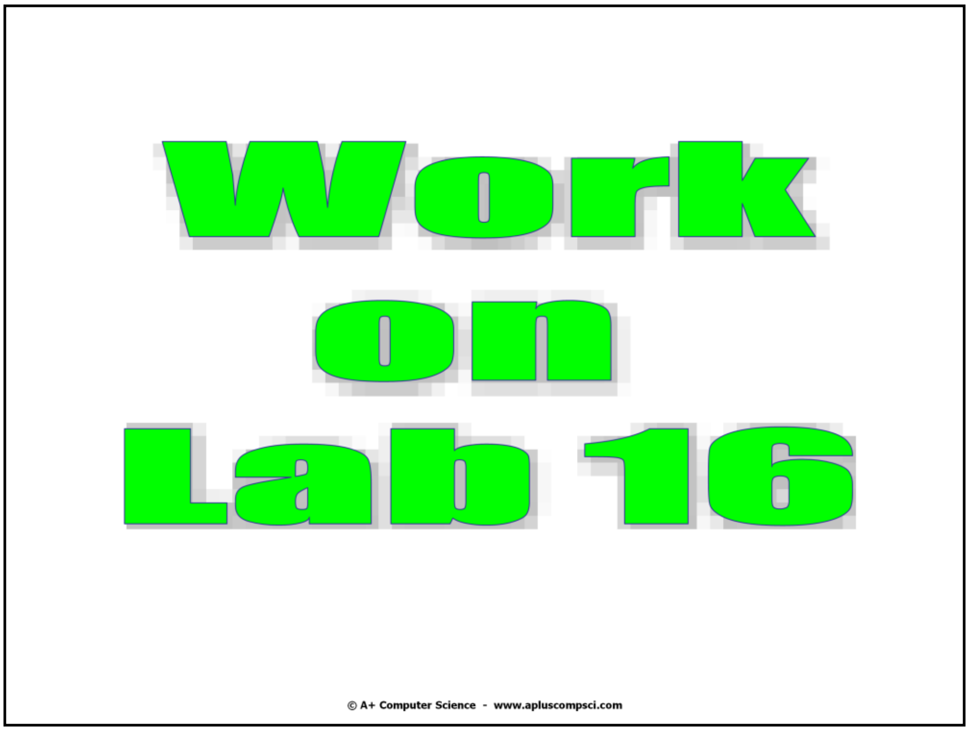
ArrayList is a descendant of List and Collection, but because List and Collection are interfaces, you cannot instantiate them.

```
Collection bad = new Collection();   //illegal

List ray = new ArrayList();          //legal
ArrayList list = new ArrayList();    //legal
```

ray and list store Object references.

In the example above, ray is an ArrayList that stores Object references. In order to call non-Object methods on a spot in ray, casting would be required.

```
ray.add(0,"hello");
out.println(((String)ray.get(0)).charAt(0));
```



Work
on
Lab 16