12

Arrays and Collections

Objectives

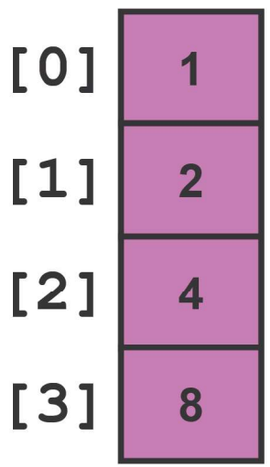
After completing this lesson, you should be able to do the following:

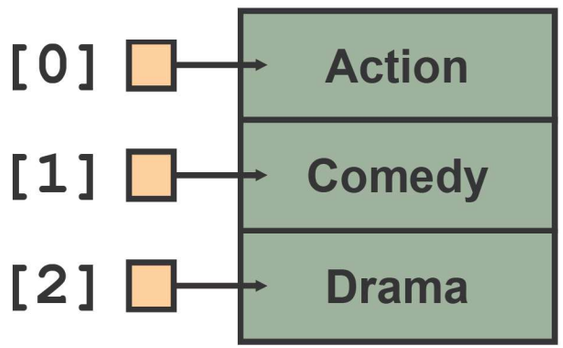
- Describe how to create arrays of primitives and objects
- Process command-line variables
- Work with ArrayLists
- Explore other Java collections such as `Enumerators`, `Iterators`, `ArrayLists`, and `Hashtables`
- Process command-line and system properties

Arrays

An array is a collection of variables of the same type.

- Each element can hold a single item.
- Items can be primitives or object references.
- The length of the array is fixed when it is created.

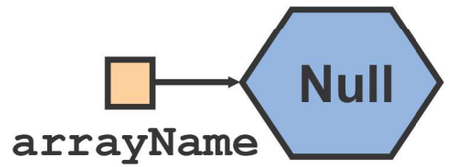




Creating an Array of Primitives

1. Declare the array.

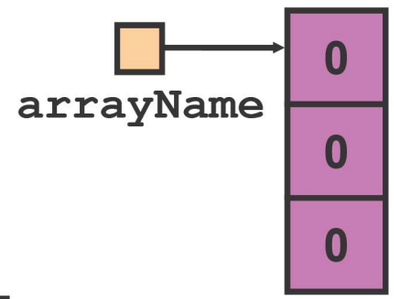
```
type[] arrayName;
… or …
type arrayName[];
```



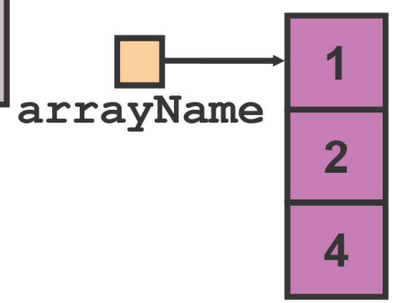
`type` is a primitive, such as `int` and so on.

2. Create the array object.

```
// Create array object syntax
arrayName = new type[size];
```



3. Initialize the array elements (optional).

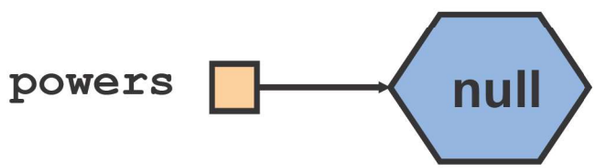


Declaring an Array

- Create a variable to reference the array object:

```
int[] powers;  // Example
```

- When an array variable is declared:
  - Its instance variable is initialized to `null` until the array object has been created



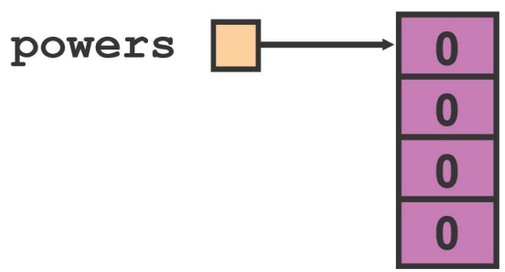
  - Its method variable is unknown until the object is created

Creating an Array Object

- Create an array of the required length and assign it to the array variable:

```
int[] powers;          // Declare array variable
powers = new int[4];   //Create array object
```

  - Create the array object by using the `new` operator.
- The contents of an array of primitives are initialized automatically.

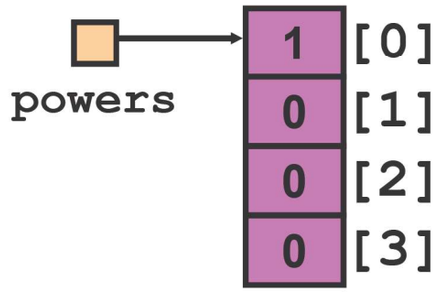


Initializing Array Elements

- Assign values to individual elements:

```
arrayName[index] = value;
```

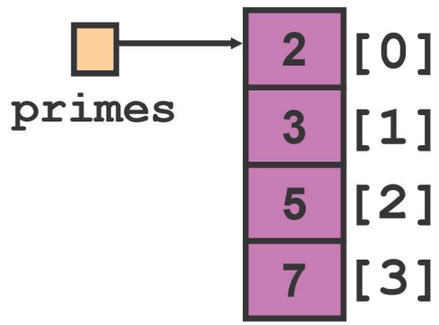
```
powers[0] = 1;
```



- Create and initialize arrays at the same time:

```
type[] arrayName = {valueList};
```

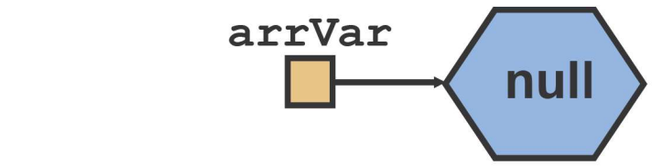
```
int[] primes = {2, 3, 5, 7};
```



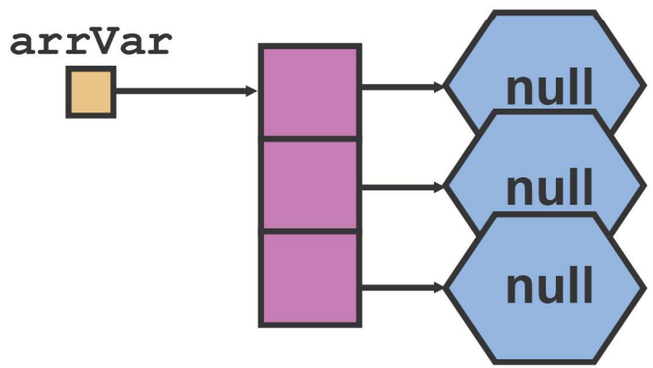
Creating an Array of Object References

1. Declare the array.

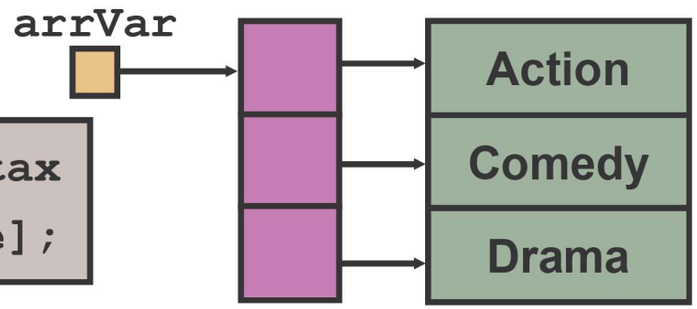
```
ClassName[] arrVar;
… or …
ClassName arrVar[];
```



2. Create the array object.




```
// Create array object syntax
arrVar = new ClassName[size];
```



3. Initialize the objects in the array.

Initializing the Objects in an Array

- Assign a value to each array element:

```
// Create an array of four empty Strings
String[] arr = new String[4];
for (int i = 0; i < arr.length; i++) {
     arr[i] = new String();
}
```

- Create and initialize the array at the same time:

```
String[] categories =
           {"Action", "Comedy", "Drama"};
```

Using an Array of Object References

- Any element can be assigned to an object of the correct type:

```
String category = categories[0];
```

- Each element can be treated as an individual object:

```
System.out.println
      ("Length is " + categories[2].length());
```

- An array element can be passed to any method; array elements are passed by reference.

Going Through the Array Elements

- Use a loop to explore each element in the array:

```
for (int i = 0;i < categories.length; i++){
   System.out.println("Category: "+categories[i]);
}
```

- Java 5.0 provides this alternative enhanced syntax:

```
for (String category: categories) {
     System.out.println ("Category: " +category);
}
```

- `ArrayIndexOutOfBoundsException` occurs when an array index is invalid:

```
String[] list = new String[4];
//The following throws ArrayIndexOutOfBoundsException
System.out.println(list[4]);
```

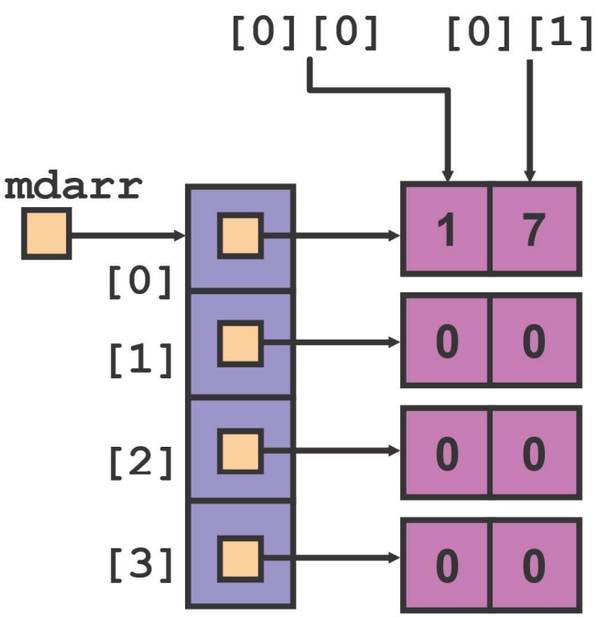
- `NullPointerException` occurs when you try to access an element that has not been initialized:

```
Movie[] movieList = new Movie[3];
// The following will throw NullPointerException
String director = movieList[0].getDirector();
```

Java supports arrays of arrays:

```
type[][] arrayname = new type[n1][n2];
```

```
int[][] mdarr = new int[4][2];
mdarr[0][0] = 1;
mdarr[0][1] = 7;
```



Passing Command-Line Parameters

- `main()` has a single parameter: `args`.
- `args` is an array of `Strings` that holds command-line parameters:

```
C:\> java SayHello Hello World
```

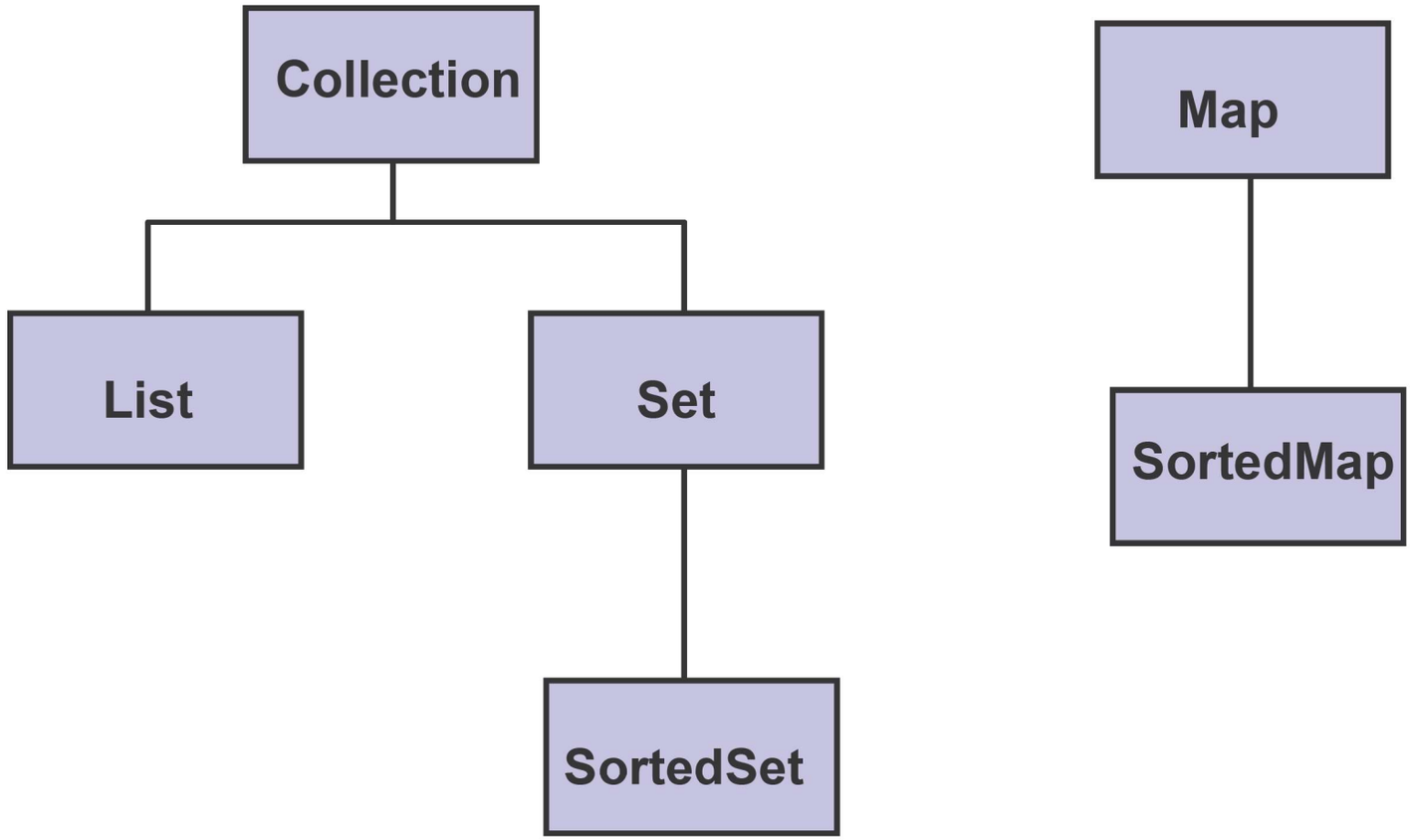
```
public class SayHello {
  public static void main(String[] args) {
    if (args.length != 2)
     System.out.println("Specify 2 arguments");
    else
      System.out.println(args[0]+" "+args[1]);
  } …
```

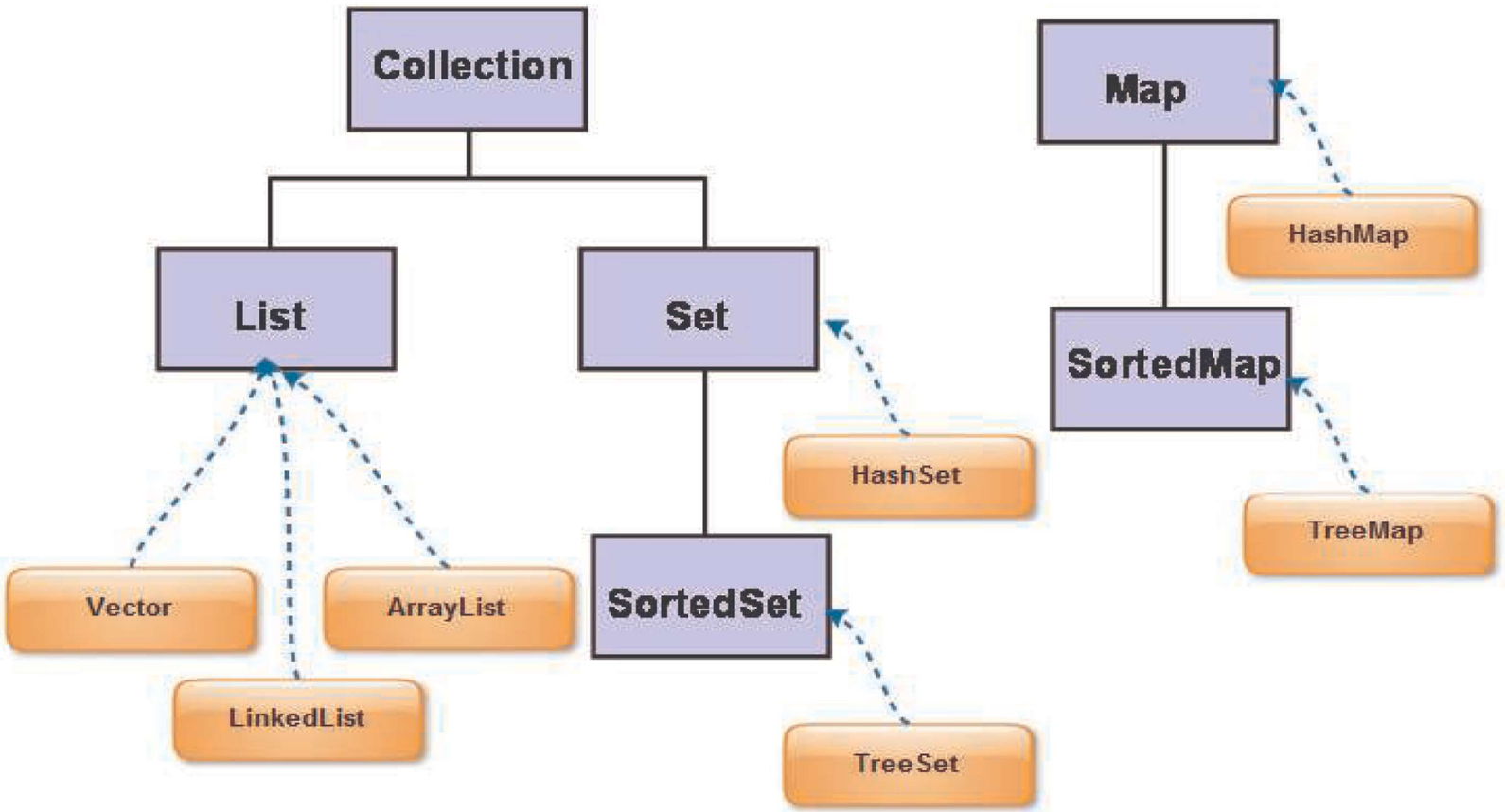
Java Collections Framework

Java Collections Framework is an API architecture for managing a group of objects that can be manipulated independently of their internal implementation. It is:

- Found in the `java.util` package
- Defined by six core interfaces and many implementation classes:
  - `Collection` interface: Generic group of elements
  - `Set` interface: Group of unique elements
  - `List` interface: Ordered group of elements
  - `Map` interface: Group of unique keys and their values
  - `SortedSet` and `SortedMap` for a sorted `Set` and `Map`

Framework Interface Hierarchy




Collection
List
Set
Map
HashMap
SortedMap
HashSet
TreeMap
Vector
ArrayList
SortedSet
LinkedList
TreeSet

Collections Framework Components

Collections Framework is a set of interfaces and classes used to store and manipulate groups of data as a single unit.

- Core interfaces are the interfaces used to manipulate collections and to pass them from one method to another.
- Implementations are the actual data objects used to store collections, which implement the core collection interface.
- Algorithms are pieces of reusable functionality provided by the JDK.

The `ArrayList` class:

- Is a resizable implementation of the `List` interface
- Allows manipulation of the array size
- Has capacity that grows as elements are added to the list
- Creating an empty ArrayList:

```
ArrayList members = new ArrayList();
```

- Creating an ArrayList with an initial size:

```
// Create an ArrayList with 10 elements.
ArrayList members = new ArrayList(10);
```

Modifying an ArrayList

- Add an element to the end of the ArrayList:

```
String name = MyMovie.getNextName();
members.add(name);
```

- Add an element at a specific position:

```
// Insert a string at the beginning
members.add(0, name);
```

- Remove the element at a specific index:

```
// Remove the first element
members.remove(0);
```

Accessing an ArrayList

- Get the first element:

```
String s = members.get(0);
```

- Get an element at a specific position:

```
String s = members.get(2);
```

- Find an object in an ArrayList:

```
int position = members.indexOf(name);
```

- Get the size of an ArrayList :

```
int size = members.size();
```

`Hashtable` Class

The `Hashtable` class:

- Implements the `Map` interface
- Is used to store arbitrary objects that are indexed by another arbitrary object
- Is commonly used with `String` as the key to store objects as values

`Iterator` Interface

The `Iterator` interface, which is part of Java Collections Framework, can be used to process a series of objects. The `java.util.Iterator` interface:

- Implements an object-oriented approach for accessing elements in a collection
- Replaces the `java.util.Enumeration` approach
- Contains the following methods:
  - `hasNext()` returns `true` if more elements exist.
  - `next()` returns the next `Object`, if any.
  - `remove()` removes the last element returned.

Enhancements in Java SE 5.0

Java SE 5.0 provides some enhancements when working with the Java Collections Framework API:

- Generic types:
  - Enable you to define type-specific collections
  - Catch type mismatches at compile time (not run time)
- Enhanced `for`:
  - Simplifies traversing through a collection
  - Removes the need for an iterator
- Autoboxing of primitive types:
  - Simplifies converting between primitive types and their equivalent wrapper types

Summary

In this lesson, you should have learned how to:

- Create Java arrays of primitives
- Create arrays of object references
- Initialize arrays of primitives or object references
- Process command-line arguments in the `main()` method
- Use the `ArrayList` object to implement resizable arrays
- Use the `Hashtable` class and `Iterator` interface

Practice : Overview

This practice covers the following topics:

- Modifying the `DataMan` class
- Creating an array to hold the `Customer`, `Company`, and `Individual` objects
- Adding a method to ensure that the array is successfully created and initialized
- Adding a method to find a customer by an ID value