Learning Java - A Foundational Journey

Session: 5

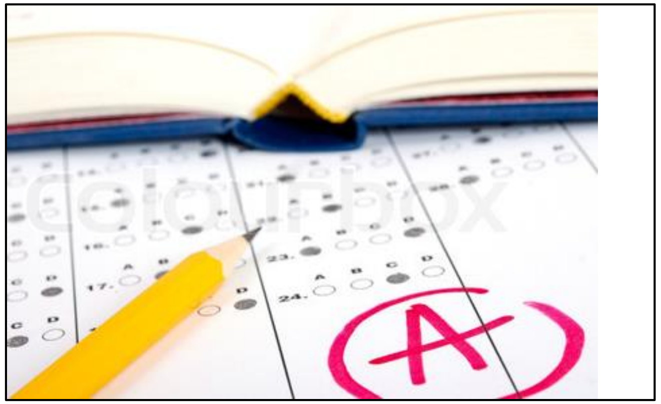
Arrays and Strings

Objectives

- Describe an array
- Explain declaration, initialization, and instantiation of a single-dimensional array
- Explain declaration, initialization, and instantiation of a multi-dimensional array
- Explain the use of loops to process an array
- Describe ArrayList and accessing values from an ArrayList
- Describe String and StringBuilder classes
- Explain command line arguments
- Describe Wrapper classes, autoboxing, and unboxing

Introduction

- Consider a situation where in a user wants to store marks of ten students.
- User can create ten different variables of type integer and store marks in them.
- What if user wants to store marks of hundreds or thousands of students?
- In such a case, one would need to create as many variables.
- This can be a very difficult, tedious, and time consuming task.
- Here, it is required to have a feature that will enable storing of all the marks in one location and access it with similar variable names.
- Array, in Java, is a feature that allows storing multiple values of similar type in the same variable.



Introduction to Arrays

An array is a special data store that can hold a fixed number of values of a single type in contiguous memory locations.

It is implemented as objects.

The size of an array depends on the number of values it can store and is specified when the array is created.

- Arrays have following benefits:

Arrays are the best way of operating on multiple data elements of the same type at the same time.

Arrays make optimum use of memory resources as compared to variables.

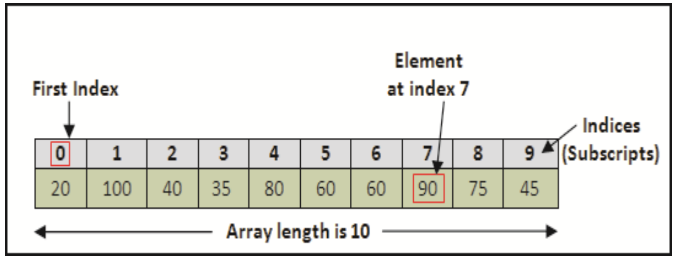
Memory is assigned to an array only at the time when the array is actually used. Thus, the memory is not consumed by an array right from the time it is declared.

- Arrays in Java are of the following two types:

Single-dimensional arrays

Multi-dimensional arrays

Figure displays an array of ten integers storing values such as, 20, 100, 40, and so on.



0	1	2	3	4	5	6	7	8	9
20	100	40	35	80	60	60	90	75	45

Declaring, Instantiating, and Initializing Single-dimensional Array 1-3

A single-dimensional array has only one dimension and is visually represented as having a single column with several rows of data.

- Following figure shows the array named marks and its elements with their values and indices:

marks [4]	
Element	**Value**
marks[0]	65
marks[1]	47
marks[2]	75
marks[3]	50

- Array creation involves following tasks:

Declaring an Array

- Declaring an array notifies compiler that the variable will contain an array of the specified data type. It does not create an array.

Syntax

```
datatype[] <array-name>;
```

Instantiating an Array

- Since array is an object, memory is allocated only when it is instantiated.

Syntax

```
datatype[] <array-name> = new datatype[size];
```

Declaring, Instantiating, and Initializing Single-dimensional Array 2-3

Initializing an Array

During creation:

- To initialize a single-dimensional array during creation, one must specify the values to be stored while creating the array as follows: `int[ ] marks = {65, 47, 75, 50};`
- Notice that while initializing an array during creation, the `new` keyword or size is not required.
- This is because all the elements to be stored have been specified and accordingly the memory gets automatically allocated based on the number of elements.

After creation:

- A single-dimensional array can also be initialized after creation and instantiation. In this case, individual elements of the array must be initialized with appropriate values. For example,

```
int[] marks = new int[4];
marks[0] = 65;
marks[1] = 47;
marks[2] = 75;
marks[3] = 50;
```

- Notice that in this case, the array must be instantiated and size must be specified. This is because, actual values are specified later and to store the values, memory must be allocated during creation of the array.

- Another way of creating an array is to split all three stages:

```
int marks[]; // declaration
marks = new int[4]; // instantiation
marks[0] = 65; // initialization
```

Declaring, Instantiating, and Initializing Single-dimensional Array 3-3

Example of single-dimensional array:

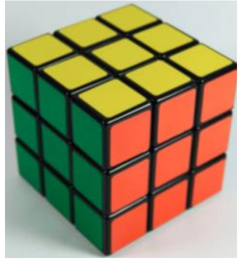
```
package session5;
public class OneDimension {
//Declare a single-dimensional array named marks
int marks[]; // line 1
/**
* Instantiates and initializes a single-dimensional array
*
* @return void
*/
public void storeMarks() {
// Instantiate the array
marks = new int[4]; // line 2
System.out.println("Storing Marks. Please wait...");
// Initialize array elements
marks[0] = 65; // line 3
marks[1] = 47;
marks[2] = 75;
marks[3] = 50;
}
```

```
/**
* Displays marks from a single-dimensional array
*
* @return void
*/
public void displayMarks() {
System.out.println("Marks are:");
// Display the marks
System.out.println(marks[0]);
System.out.println(marks[1]);
System.out.println(marks[2]);
System.out.println(marks[3]);
}
/**
* @param args the command line arguments
*/
public static void main(String[] args) {
//Instantiate class OneDimension
OneDimension oneDimenObj = new OneDimension(); //line 4
//Invoke the storeMarks() method
oneDimenObj.storeMarks(); // line 5
//Invoke the displayMarks() method
oneDimenObj.displayMarks(); // line 6
}
}
```

Declaring, Instantiating, and Initializing Multi-dimensional Array 1-2

A multi-dimensional array in Java is an array whose elements are also arrays. This allows the rows to vary in length.

- The syntax for declaring and instantiating a multi-dimensional array is as follows:



Syntax

```
datatype[][] <array-name> = new datatype [rowsize][colsize];
```

During creation

- While initializing an array during creation, the elements in rows are specified in a set of curly brackets separated by a comma delimiter.

Rows	Columns	
	0	1
0	23	65
1	42	47
2	60	75
3	75	50

After creation

A multi-dimensional array can also be initialized after creation and instantiation.

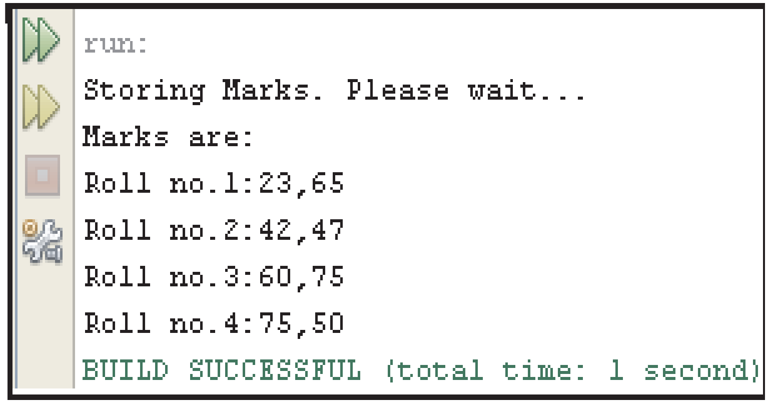
In this case, individual elements of the array need to be initialized with appropriate values.

Each element is accessed with a row and column subscript.

Declaring, Instantiating, and Initializing Multi-dimensional Array 2-2

```
package session5;
public class TwoDimension {
//Declare a two-dimensional array named marks
int marks[][]; //line 1
/**
* Stores marks in a two-dimensional array
*
* @return void
*/
public void storeMarks() {
// Instantiate the array
marks = new int[4][2]; // line 2
System.out.println("Storing Marks. Please wait...");
// Initialize array elements
marks[0][0] = 23; // line 3
marks[0][1] = 65;
marks[1][0] = 42;
marks[1][1] = 47;
marks[2][0] = 60;
marks[2][1] = 75;
marks[3][0] = 75;
marks[3][1] = 50;
}
/**
* Displays marks from a two-dimensional array
*
* @return void
*/
```

```
public void displayMarks() {
System.out.println("Marks are:");
// Display the marks
System.out.println("Roll no.1:" + marks[0][0]+ "," + marks[0][1]);
System.out.println("Roll no.2:" + marks[1][0]+ "," + marks[1][1]);
System.out.println("Roll no.3:" + marks[2][0]+ "," + marks[2][1]);
System.out.println("Roll no.4:" + marks[3][0]+ "," + marks[3][1]);
}
/**
* @param args the command line arguments
*/
public static void main(String[] args) {
//Instantiate class TwoDimension
TwoDimension twoDimenObj = new TwoDimension(); // line 4
//Invoke the storeMarks() method
twoDimenObj.storeMarks();
//Invoke the displayMarks() method
twoDimenObj.displayMarks();
}
}
```

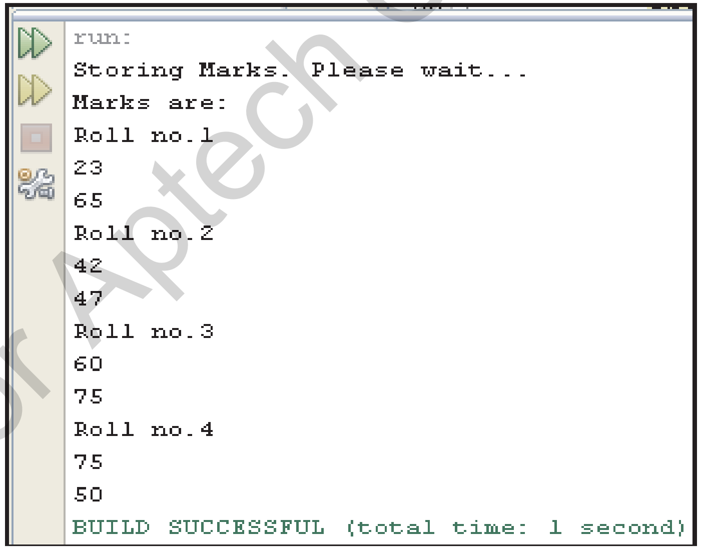


```
run:
Storing Marks. Please wait...
Marks are:
Roll no.1:23,65
Roll no.2:42,47
Roll no.3:60,75
Roll no.4:75,50
BUILD SUCCESSFUL (total time: 1 second)
```

Using Loops to Process and Initialize an Array

- A user can use loops to process and initialize an array.

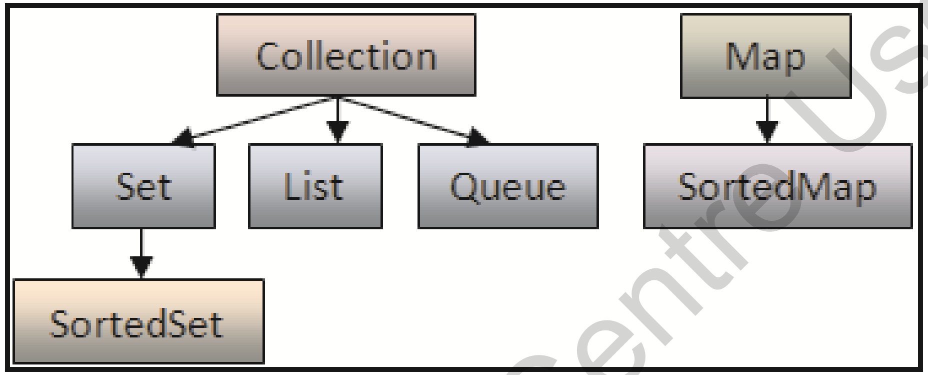
```
...
public void displayMarks() {
   System.out.println(“Marks are:”);

   // Display the marks using for loop
   for(int count = 0; count < marks.length; count++) {
       System.out.println(marks[count]);
   }
}
...
```



```
run:
Storing Marks. Please wait...
Marks are:
Roll no.1
23
65
Roll no.2
42
47
Roll no.3
60
75
Roll no.4
75
50
BUILD SUCCESSFUL (total time: 1 second)
```

Initializing an ArrayList 1-4

A collection is a single object that groups multiple elements into a single unit.



- The general-purpose implementations are summarized in the following table:

Interfaces	Hash table	Resizable array	Tree	Linked list	Hash table + Linked list
Set	HashSet	-	TreeSet	-	LinkedHashSet
List	-	ArrayList	-	LinkedList	-
Queue	-	-	-	-	-
Map	HashMap	-	TreeMap	-	LinkedHashMap

Initializing an ArrayList 2-4

- The `ArrayList` class is a frequently used collection that has the following characteristics:

  - **It is flexible and can be increased or decreased in size as needed.**
  - **Insertion and deletion of data is simpler.**
  - **It can be traversed by using** for **loop, enhanced** for **loop, or other iterators.**
  - **The capacity of an** ArrayList **grows automatically.**
  - **It stores all elements including null.**
  - **The** Arraylist **collection provides methods to manipulate the size of the array.**

Initializing an ArrayList 3-4

Methods that append one or more elements to the end of the list.

Methods that insert one or more elements at a position within the list.

- To traverse an `ArrayList`, one can use one of the following approaches:

→ **A for loop**

→ **An enhanced for loop**

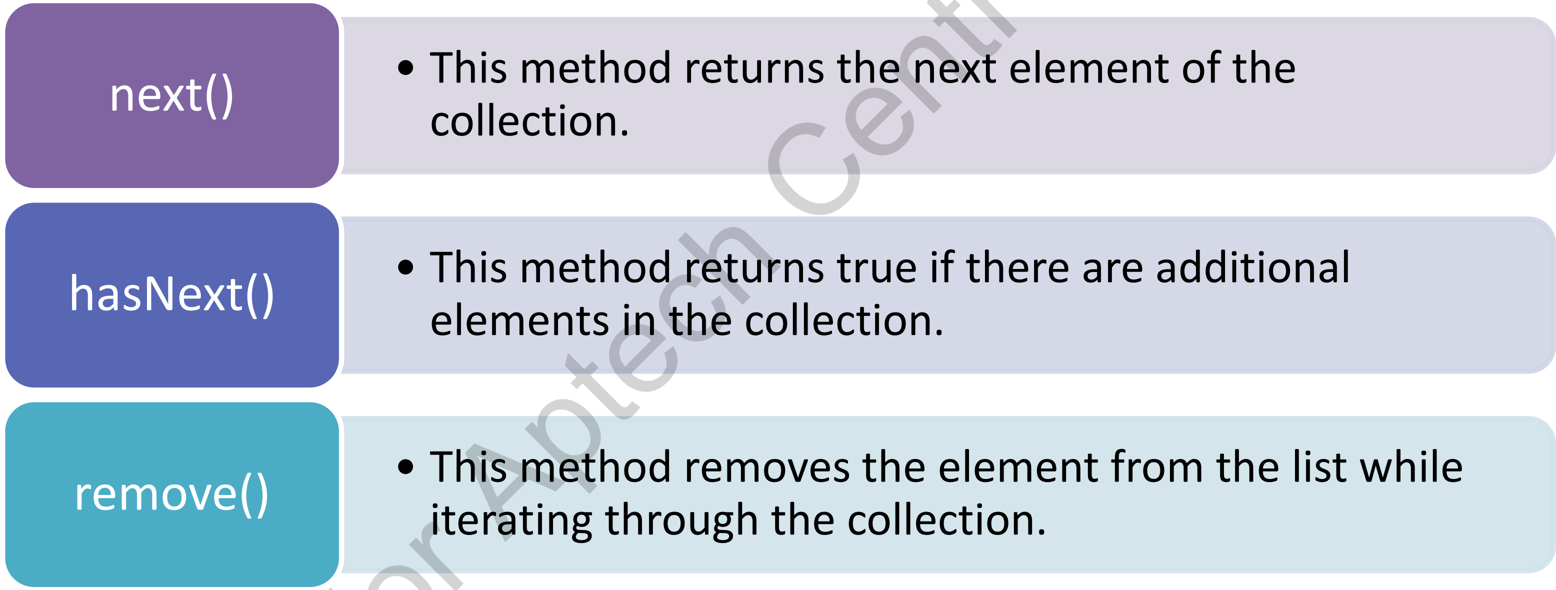
→ **Iterator**

→ **ListIterator**

Initializing an ArrayList 4-4

Iterator interface provides methods for traversing a set of data.

- The `Iterator` interface provides the following methods for traversing a collection:



next()

- This method returns the next element of the collection.

hasNext()

- This method returns true if there are additional elements in the collection.

remove()

- This method removes the element from the list while iterating through the collection.

Accessing Values in an ArrayList 1-2

- An `ArrayList` can be iterated by using the `for` loop or by using the `Iterator` interface.

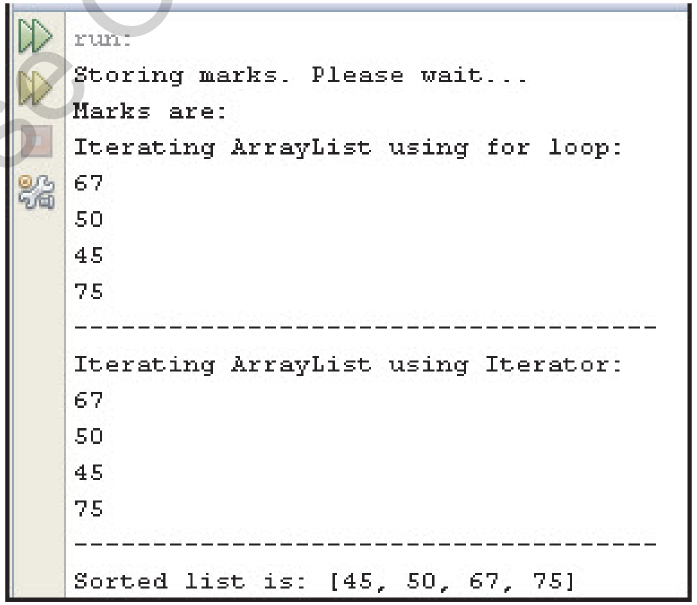
```java
package session5;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Iterator;
public class ArrayListDemo{
// Create an ArrayList instance
ArrayList marks = new ArrayList(); // line 1
/**
* Stores marks in ArrayList
*
* @return void
*/
public void storeMarks(){
System.out.println("Storing marks. Please wait...");
marks.add(67); // line 2
marks.add(50);
marks.add(45);
marks.add(75);
}
/**
* Displays marks from ArrayList
*
* @return void
*/
```

```java
public void displayMarks() {
System.out.println("Marks are:");
// iterating the list using for loop
System.out.println("Iterating ArrayList using for loop:");
for (int i = 0; i < marks.size(); i++) {
System.out.println(marks.get(i));
}
System.out.println("-------------------------------------");
// Iterate the list using Iterator interface
Iterator imarks = marks.iterator(); // line 3
System.out.println("Iterating ArrayList using Iterator:");
while (imarks.hasNext()) { // line 4
System.out.println(imarks.next()); // line 5
}
System.out.println("-------------------------------------");
// Sort the list
Collections.sort(marks); // line 6
System.out.println("Sorted list is: " + marks);
}
/**
* @param args the command line arguments
*/
```

Accessing Values in an ArrayList 2-2

```
public static void main(String[] args) {
//Instantiate the class OneDimension
ArrayListDemo obj = new ArrayListDemo(); // line 7
//Invoke the storeMarks() method
obj.storeMarks();
//Invoke the displayMarks() method
obj.displayMarks();
}
}
```

Output:



```
run:
Storing marks. Please wait...
Marks are:
Iterating ArrayList using for loop:
67
50
45
75
-------------------------------------
Iterating ArrayList using Iterator:
67
50
45
75
-------------------------------------
Sorted list is: [45, 50, 67, 75]
```

The values of an ArrayList **can also be printed by simply writing** System.out.println(“Marks are:”+ marks)**.**

In this case, the output would be: Marks are:[67, 50, 45, 75].

Introduction to Strings

- Consider a scenario, where in a user wants to store the name of a person.

- One can create a character array as shown in the following code snippet:

```
char[] name = {‘J’,’u’,’l’,’i’,’a’}
```

Similarly, to store names of multiple persons, one can create a two-dimensional array.

However, the number of characters in an array must be fixed during creation.

Java provides the String data type to store multiple characters without creating an array.

Strings 1-3

Strings are constant and immutable, that is, their values cannot be changed once they are created.

String buffers allow creation of mutable strings.

```
...
String name = “Mary”;
// This is equivalent to:
char name[] = {‘M’, ‘a’, ‘r’, ’y’};
...
```

- An instance of a `String` class can also be created using the `new` keyword, as shown here:

```
String str = new String();
```

Strings 2-3

- Java also provides special support for concatenation of strings using the plus (+) operator and for converting data of other types to strings as depicted in the following code snippet:

```
...
String str = “Hello”; String str1 = “World”;
// The two strings can be concatenated by using the operator ‘+’
System.out.println(str + str1);

// This will print ‘HelloWorld’ on the screen
...
```

- One can convert a character array to a string as depicted in the following code snippet:

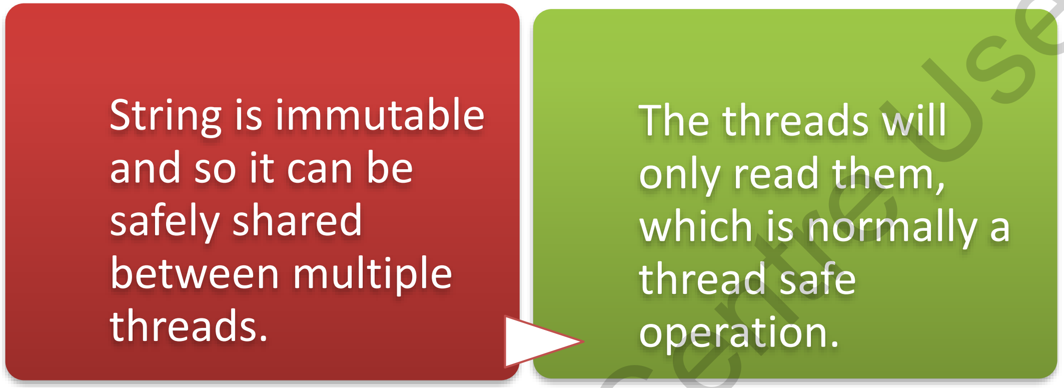
```
char[] name = {‘J’, ‘o’, ‘h’, ‘n’};
String empName = new String(name);
```

The java.lang.String **class is a** final **class, that is, no class can extend it.**

The java.lang.String **class differs from other classes, in that one can use ‘+=’ and ‘+’ operators with** String **objects for concatenation.**

Strings 3-3

- If the string is not likely to change later, one can use the `String` class.
- Thus, a `String` class can be used for the following reasons:



String is immutable and so it can be safely shared between multiple threads.

The threads will only read them, which is normally a thread safe operation.

- The use of `StringBuffer` class ensures that the string is updated correctly.
- However, the drawback is that the method execution is comparatively slower.

It allows modification of the strings without the overhead of synchronization.

Working with String Class 1-5

- Some of the frequently used methods of `String` class are as follows:

length(String str)

- The `length()` method is used to find the length of a string. For example,

```
String str = "Hello";
System.out.println(str.length()); // output: 5
```

charAt(int index)

- The `charAt()` method is used to retrieve the character value at a specific index.
- The index ranges from zero to length() – 1.

```
System.out.println(str.charAt(2));  // output: 'l'
```

concat(String str)

- The `concat()` method is used to concatenate a string specified as argument to the end of another string.

```
System.out.println(str.concat("World"));
  // output: 'HelloWorld'
```

Working with String Class 2-5

compareTo(String str)

- The `compareTo()` method is used to compare two String objects.
- The comparison returns an integer value as the result.
- For example,

```
System.out.println(str.compareTo("World"));
 // output: -15
```

indexOf(String str)

- The `indexOf()` method returns the index of the first occurrence of the specified character or string within a string.
- If the character or string is not found, the method returns -1. For example,

```
System.out.println(str.indexOf("e"));  // output: 1
```

lastIndexOf(String str)

- The `lastIndexOf()` method returns the index of the last occurrence of a specified character or string from within a string.
- For example,

```
System.out.println(str.lastIndexOf("l")); // output: 3
```

replace(char old, char new)

- The replace() method is used to replace all the occurrences of a specified character in the current string with a given new character.
- For example,

```
System.out.println(str.replace('e','a'));
 // output: 'Hallo'
```

Working with String Class 3-5

substring(int beginIndex, int endIndex)

- The substring() method is used to retrieve a part of a string, that is, substring from the given string.
- For example,

```
System.out.println(str.substring(2,5)); // output: 'llo'
```

toString()

- The `toString()` method is used to return a String object.
- It is used to convert values of other data types into strings. For example,

```
Integer length = 5;
System.out.println(length.toString()); // output: 5
```

trim()

- The `trim()` method returns a new string by trimming the leading and trailing whitespace from the current string. For example,

```
String str1 = " Hello ";
System.out.println(str1.trim()); // output: 'Hello'
```

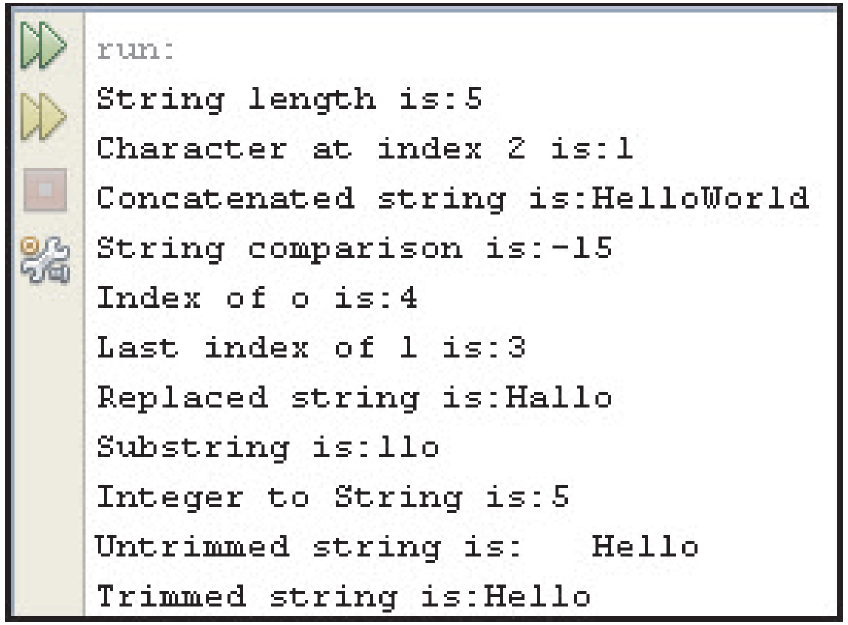
Working with String Class 4-5

```
public class Strings {
String str = "Hello"; // Initialize a String variable
Integer strLength = 5; // Use the Integer wrapper class
/**
* Displays strings using various String class methods
*
* @return void
*/
public void displayStrings(){
// using various String class methods
System.out.println("String length is:"+ str.length());
System.out.println("Character at index 2 is:"+ str.charAt(2));
System.out.println("Concatenated string is:"+ str.concat("World"));
System.out.println("String comparison is:"+ str.compareTo("World"));
System.out.println("Index of o is:"+ str.indexOf("o"));
System.out.println("Last index of l is:"+ str.lastIndexOf("l"));
```

Working with String Class 5-5

```
System.out.println("Replaced string is:"+ str.replace('e','a'));
System.out.println("Substring is:"+ str.substring(2, 5));
System.out.println("Integer to String is:"+ strLength.toString())
;
String str1=" Hello ";
System.out.println("Untrimmed string is:"+ str1);
System.out.println("Trimmed string is:"+ str1.trim());
}
/**
* @param args the command line arguments
*/
public static void main(String[] args) {
//Instantiate class, Strings
Strings objString = new Strings(); // line 1
//Invoke the displayStrings() method
objString.displayStrings();
}
}
```

Following figure shows the output:



```
run:
String length is:5
Character at index 2 is:l
Concatenated string is:HelloWorld
String comparison is:-15
Index of o is:4
Last index of l is:3
Replaced string is:Hallo
Substring is:llo
Integer to String is:5
Untrimmed string is:   Hello
Trimmed string is:Hello
```

Working with StringBuilder Class 1-2

StringBuilder **objects are similar to** String **objects, except that they are mutable and flexible.**

Internally, the system treats these objects as a variable-length array containing a sequence of characters.

The length and content of the sequence of characters can be changed through methods available in the StringBuilder **class.**

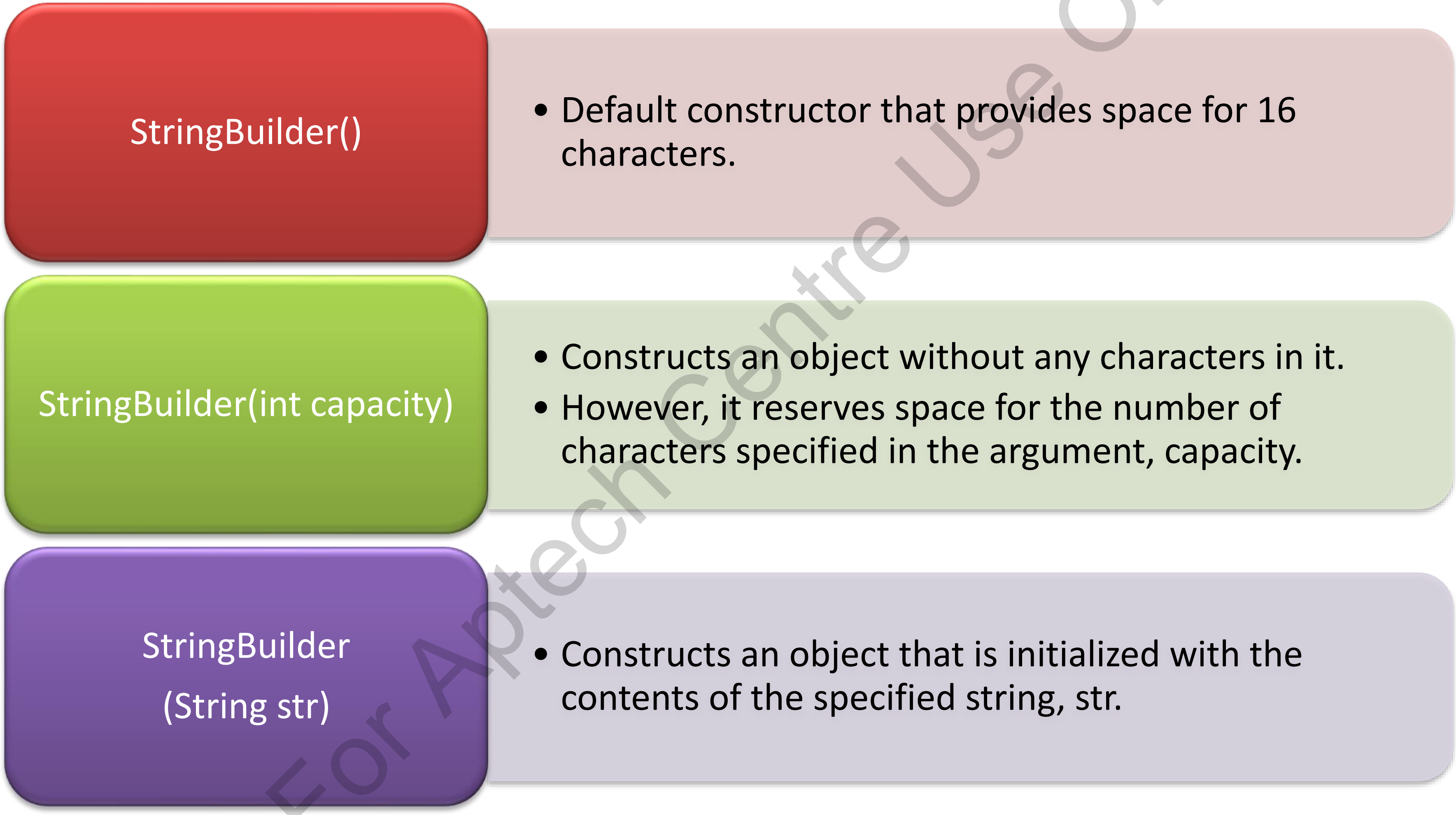
The capacity is returned by the capacity() **method and is always greater than or equal to the length.**

The capacity will automatically expand to accommodate the new strings when added to the string builder.

StringBuilder **object allows insertion of characters and strings as well as appending characters and strings at the end.**

Working with StringBuilder Class 2-2

- The constructors of the `StringBuilder` class are as follows:



StringBuilder()

- Default constructor that provides space for 16 characters.

StringBuilder(int capacity)

- Constructs an object without any characters in it.
- However, it reserves space for the number of characters specified in the argument, capacity.

StringBuilder (String str)

- Constructs an object that is initialized with the contents of the specified string, str.

Methods of StringBuilder Class 1-2

- The StringBuilder class provides several methods for appending, inserting, deleting, and reversing strings as follows:

append

- The append() method is used to append values at the end of the StringBuilder object.

insert()

- The insert() method is used to insert one string into another.
- The new string is inserted into the invoking StringBuilder object.

delete()

- The delete() method deletes the specified number of characters from the invoking StringBuilder object.
- For example,

```
StringBuilder str = new StringBuilder(“JAVA SE 7”);
System.out.println(str.delete(4,7); // output: JAVA 7
```

reverse()

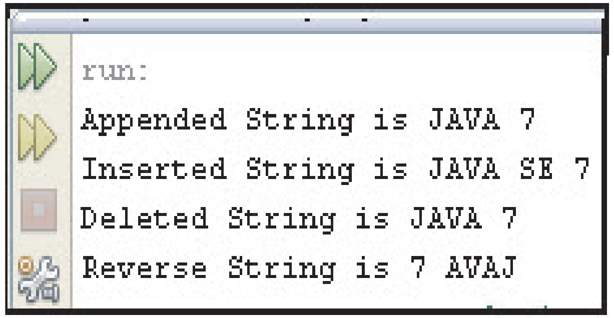
- The reverse() method is used to reverse the characters within a StringBuilder object.
- For example,

```
StringBuilder str = new StringBuilder(“JAVA SE 7”);
System.out.println(str.reverse());
    // output: 7 ES AVAJ
```

Methods of StringBuilder Class 2-2

```
public class StringBuilders {
// Instantiate a StringBuilder object
StringBuilder str = new StringBuilder("JAVA ");
/**
* Displays
strings using various StringBuilder methods
* @return void
*/
public void displayStrings(){
// Use various methods of the StringBuilder class
System.out.println("Appended String is "+ str.append("7"));
System.out.println("Inserted String is "+ str.insert(5, "SE "));
System.out.println("Deleted String is "+ str.delete(4,7));
System.out.println("Reverse String is "+ str.reverse());
}
/**
* @param args the command line arguments
*/
public static void main(String[] args) {
//Instantiate the StringBuilders class
StringBuilders objStrBuild = new StringBuilders(); // line 1
//Invoke the displayStrings() method
objStrBuild.displayStrings();
}
}
```

Following figure shows the output:



```
run:
Appended String is JAVA 7
Inserted String is JAVA SE 7
Deleted String is JAVA 7
Reverse String is 7 AVAJ
```

String Arrays

Sometimes there is a need to store a collection of strings.

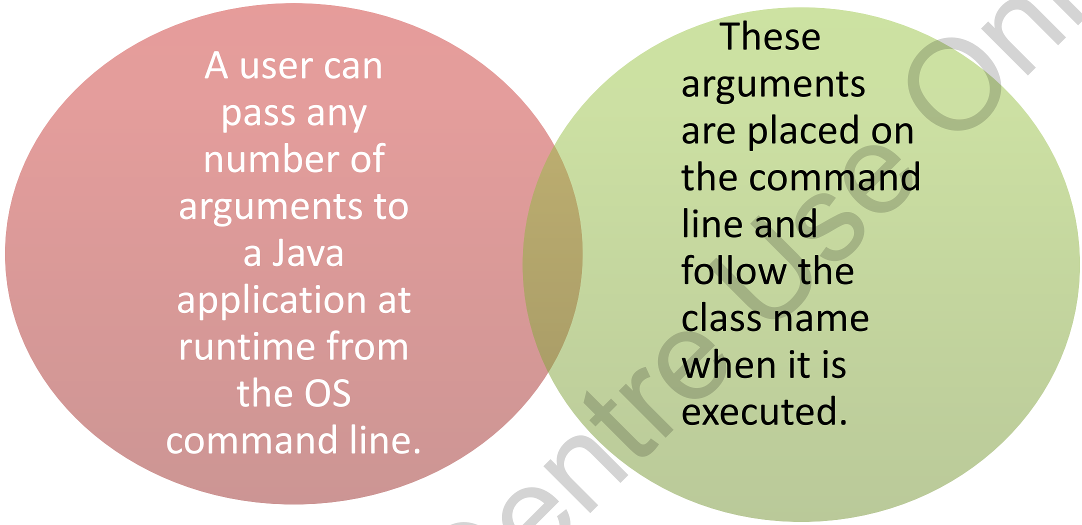
String **arrays can be created in Java in the same manner as arrays of primitive data types.**

For example, String[] empNames = new String[10];

This statement will allocate memory to store references of 10 strings.

However, no memory is allocated to store the characters that make up individual strings.

Command Line Arguments 1-2



A user can pass any number of arguments to a Java application at runtime from the OS command line.

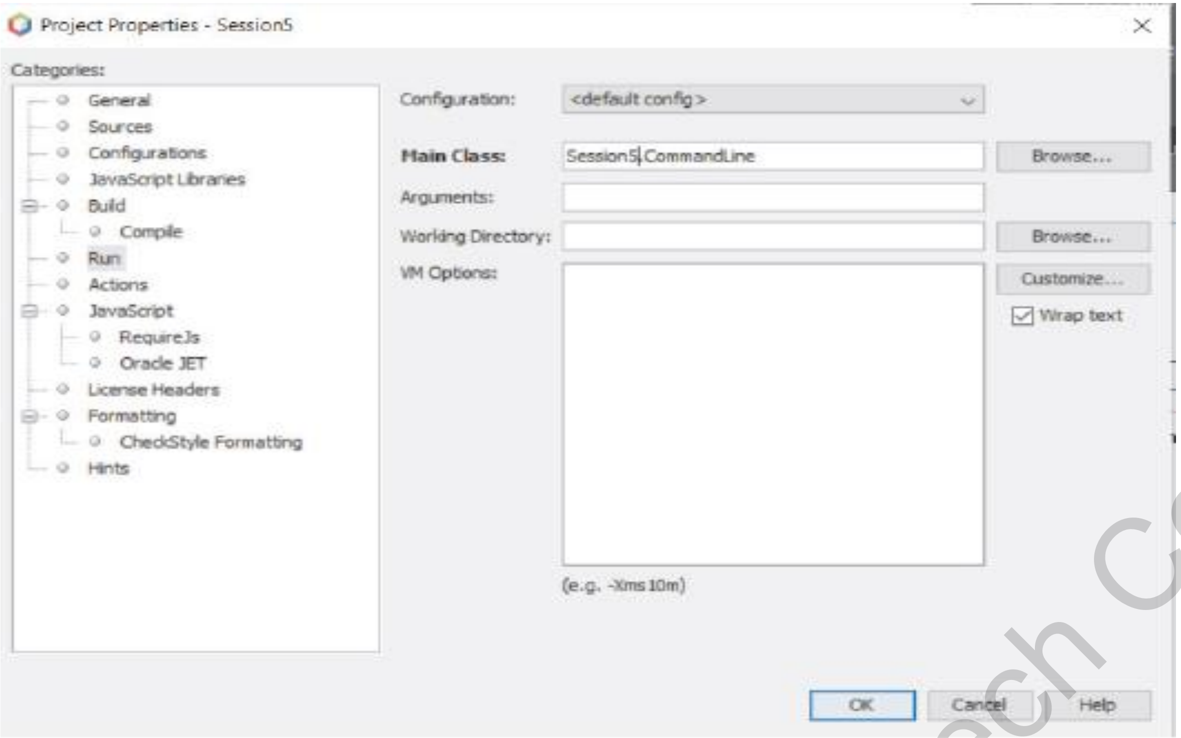
These arguments are placed on the command line and follow the class name when it is executed.

The length of the array is determined from the number of arguments passed at runtime

The arguments are separated by a space

The basic purpose of command line arguments is to specify the configuration information for the application

Command Line Arguments 2-2



Project Properties - Session5

Configuration: <default config>

Main Class: Session5.CommandLine

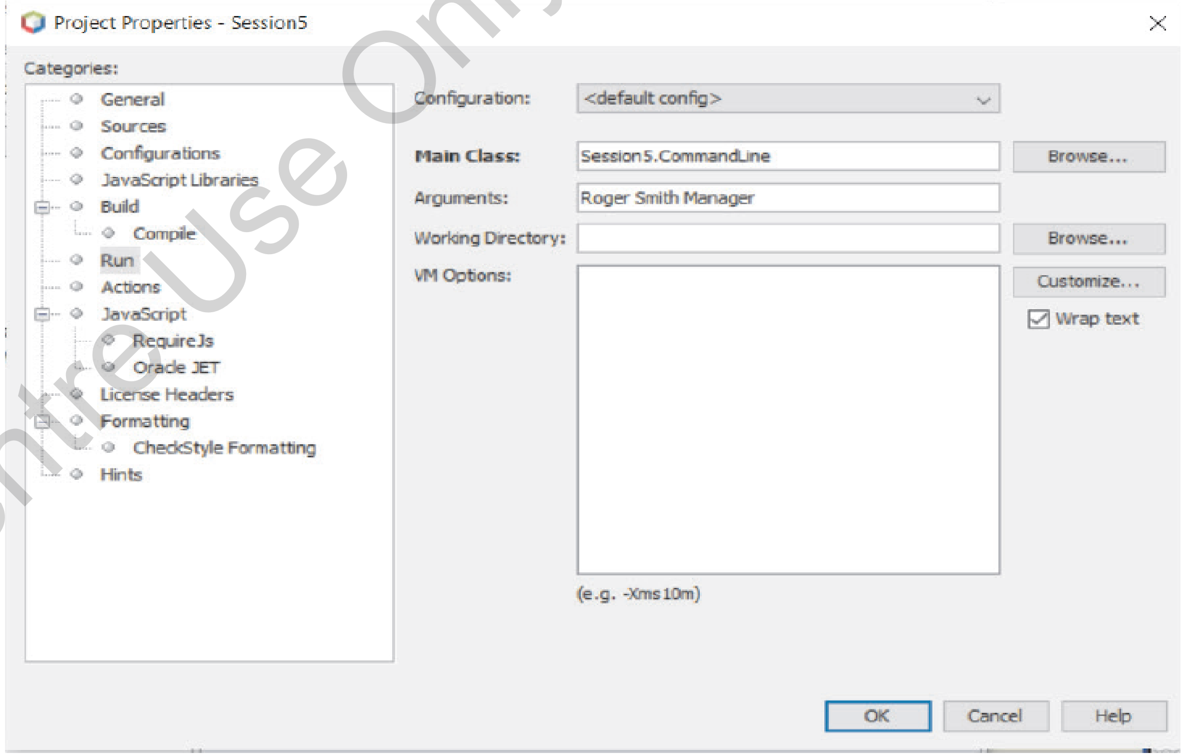
Arguments:

Working Directory:

VM Options:

(e.g. -Xms10m)

☑ Wrap text



Project Properties - Session5

Configuration: <default config>

Main Class: Session5.CommandLine

Arguments: Roger Smith Manager

Working Directory:

VM Options:

(e.g. -Xms10m)

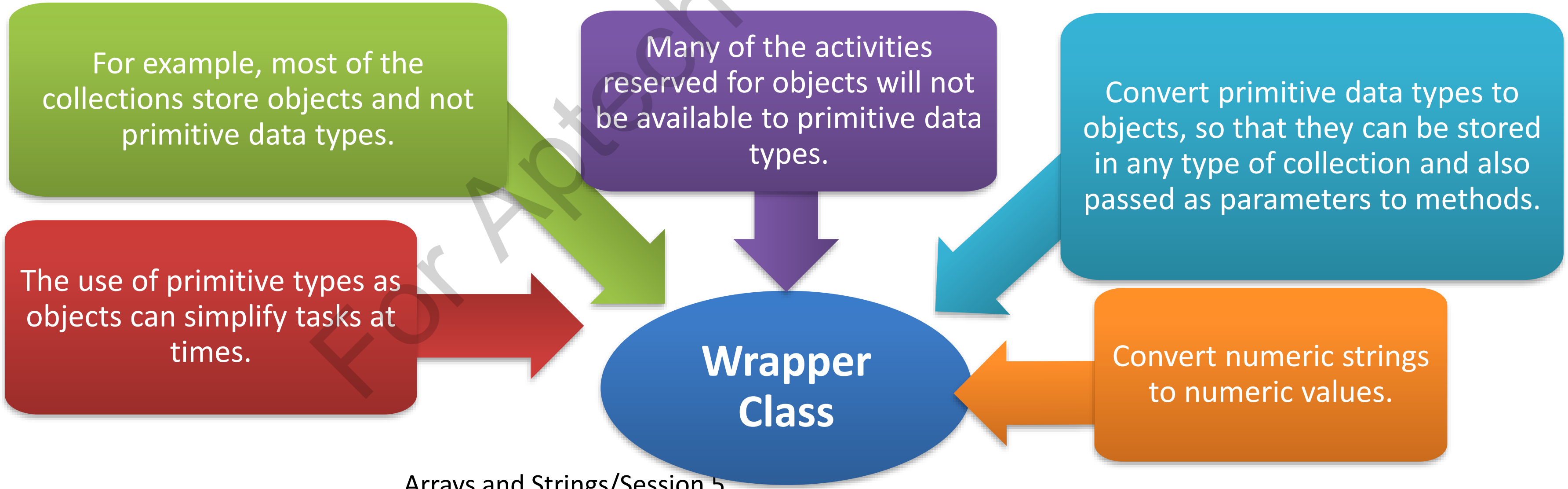
☑ Wrap text

Wrapper Classes

Java provides a set of classes known as wrapper classes for each of its primitive data type that ‘wraps’ the primitive type into an object of that class.

- The primitive types and the corresponding wrapper types are listed in the following table:

Primitive type	Wrapper class
byte	Byte
char	Character
float	Float
double	Double
int	Integer
long	Long
short	Short
boolean	Boolean



For example, most of the collections store objects and not primitive data types.

Many of the activities reserved for objects will not be available to primitive data types.

Convert primitive data types to objects, so that they can be stored in any type of collection and also passed as parameters to methods.

The use of primitive types as objects can simplify tasks at times.

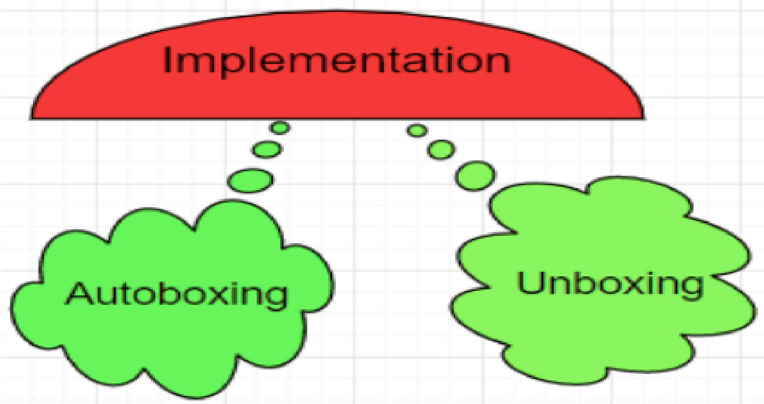
Convert numeric strings to numeric values.

Autoboxing and Unboxing

Autoboxing

- The automatic conversion of primitive data types such as int, float, and so on to their corresponding object types such as Integer, Float, and so on during assignments and invocation of methods and constructors is known as autoboxing.
- For example,

```
ArrayList<Integer> intList = new ArrayList<Integer>();
intList.add(10); // autoboxing
Integer y = 20; // autoboxing
```



Unboxing

- The automatic conversion of object types to primitive data types is known as unboxing.
- For example,

```
int z = y; // unboxing
```

Autoboxing and unboxing helps a developer to write a cleaner code.

Using autoboxing and unboxing, one can make use of the methods of wrapper classes as and when required.

Compact Strings

It is one of the performance enhancements that was introduced in the JVM as part of JDK 9.

Java represented String objects as char[]

Many characters require two bytes to represent them.

Improves the memory consumption and performance.

A final field named coder is used in the internal representation of String with a byte array as follows:

```
private final byte[] value;
/*can be LATIN1 = 0 or UTF16 = 1 */
private final byte coder;
```

Summary

- An array is a special data store that can hold a fixed number of values of a single type in contiguous memory locations.
- A single-dimensional array has only one dimension and is visually represented as having a single column with several rows of data.
- A multi-dimensional array in Java is an array whose elements are also arrays.
- A collection is an object that groups multiple elements into a single unit.
- Strings are constant and immutable, that is, their values cannot be changed once they are created.
- StringBuilder objects are similar to String objects, except that they are mutable.
- Java provides a set of classes known as Wrapper classes for each of its primitive data type that ‘wrap’ the primitive type into an object of that class.
- The automatic conversion of primitive types to object types is known as autoboxing and conversion of object types to primitive types is known as unboxing.
- Compact strings are a new feature in Strings in Java version 9 and higher versions and they improve performance and reduce memory consumption.