TOP SDET JAVA PROGRAMS PREPARED BY

https://www.linkedin.com/in/praveen-bishnoi/

1. How do you reverse a string in Java?

There is no `reverse()` utility method in the `String` class. However, you can create a character array from the string and then iterate it from the end to the start. You can append the characters to a string builder and finally return the reversed string.

The following example code shows one way to reverse a string:

```
public class StringPrograms {

	public static void main(String[] args) {

		String str = "123";

		System.out.println(reverse(str));
	}

	public static String reverse(String in) {
		if (in == null)
			throw new IllegalArgumentException("Null is not
valid input");

		StringBuilder out = new StringBuilder();

		char[] chars = in.toCharArray();

		for (int i = chars.length - 1; i >= 0; i--)
			out.append(chars[i]);

		return out.toString();
	}

}
```

Bonus points for adding `null` check in the method and using `StringBuilder` for appending the characters. Note that the indexing in Java starts from 0, so you need to start at `chars.length - 1` in the `for` loop.

2. How do you swap two numbers without using a third variable in Java?

Swapping numbers without using a third variable is a three-step process that's better visualized in code:

```
b = b + a; // now b is sum of both the numbers
a = b - a; // b - a = (b + a) - a = b (a is swapped)
b = b - a; // (b + a) - b = a (b is swapped)
```

```
public static void main(String[] args) {
        int a = 10;
        int b = 20;

    System.out.println("a is " + a + " and b is " + b);

        a = a + b;
        b = a - b;
        a = a - b;

    System.out.println("After swapping, a is " + a + " and b is " + b);
    }

}
```

The output shows that the integer values are swapped:

```
Output
a is 10 and b is 20
After swapping, a is 20 and b is 10
```

3. Write a Java program to check if a vowel is present in a string.

The following example code shows how to use a regular expression to check whether the string contains vowels:

```
public class StringContainsVowels {

        public static void main(String[] args) {
                System.out.println(stringContainsVowels("Hello"));  //
true
                System.out.println(stringContainsVowels("TV")); // false
        }

        public static boolean stringContainsVowels(String input) {
                return input.toLowerCase().matches(".*[aeiou].*");
        }

}
```

4. Write a Java program to check if the given number is a prime number.

You can write a program to divide the given number `n`, by a number from 2 to `n`/2 and check the remainder. If the remainder is 0, then it's not a prime number. The following example code shows one way to check if a given number is a Prime number:

```
public class PrimeNumberCheck {
```

```
	public static void main(String[] args) {
		System.out.println(isPrime(19)); // true
		System.out.println(isPrime(49)); // false
	}

	public static boolean isPrime(int n) {
		if (n == 0 || n == 1) {
			return false;
		}
		if (n == 2) {
			return true;
		}
		for (int i = 2; i <= n / 2; i++) {
			if (n % i == 0) {
				return false;
			}
		}

		return true;
	}

}
```

Although this program works, it's not very memory and time-efficient. Consider that, for a given number `N`, if there is a prime number `M` between `2` to `√N` (square root of N) that evenly divides it, then `N` is not a prime number.

5. Write a Java program to print a Fibonacci sequence using recursion.

A Fibonacci sequence is one in which each number is the sum of the two previous numbers. In this example, the sequence begins with `0` and `1`. The following example code shows how to use a `for` loop to print a Fibonacci sequence:

```
public class PrintFibonacci {

	public static void printFibonacciSequence(int count) {
		int a = 0;
		int b = 1;
		int c = 1;

		for (int i = 1; i <= count; i++) {
			System.out.print(a + ", ");

			a = b;
			b = c;
			c = a + b;
		}
	}

	public static void main(String[] args) {
	printFibonacciSequence(10);
	}
```

```
}
```

```
Output
0, 1, 1, 2, 3, 5, 8, 13, 21, 34,
```

You can also use recursion to print a Fibonacci sequence, because the Fibonacci number is generated by adding the previous two numbers in the sequence:

```
F(N) = F(N-1) + F(N-2)
```

The following example class shows how to use recursion to calculate a Fibonacci sequence that is 10 numbers long:

```
public class PrintFibonacciRecursive {

    public static int fibonacci(int count) {
                if (count <= 1)
                        return count;

                return fibonacci(count - 1) + fibonacci(count - 2);
        }

        public static void main(String args[]) {
        int seqLength = 10;

        System.out.print("A Fibonacci sequence of " + seqLength + "
numbers: ");

        for (int i = 0; i < seqLength; i++) {
            System.out.print(fibonacci(i) + " ");
        }
        }

}
```

```
Output
A Fibonacci sequence of 10 numbers: 0 1 1 2 3 5 8 13 21 34
```

6. How do you check if a list of integers contains only odd numbers in Java?

You can use a `for` loop and check whether each element is odd:

```
public static boolean onlyOddNumbers(List<Integer> list) {
        for (int i : list) {
                if (i % 2 == 0)
                        return false;
        }

        return true;
}
```

If the list is large, you can use parallel stream for faster processing, as shown in the following example code:

```
public static boolean onlyOddNumbers(List<Integer> list) {
	return list
			.parallelStream() // parallel stream for faster
processing
			.anyMatch(x -> x % 2 != 0); // return as soon as
any elements match the condition
}
```

To learn more about the math behind determining if an integer is odd, refer to the Modulo operation on Wikipedia.

7. How do you check whether a string is a palindrome in Java?

A palindrome string is the same string backwards or forwards. To check for a palindrome, you can reverse the input string and check if the result is equal to the input. The following example code shows how to use the `String charAt(int index)` method to check for palindrome strings:

```
boolean checkPalindromeString(String input) {
	boolean result = true;
	int length = input.length();

	for (int i = 0; i < length/2; i++) {
		if (input.charAt(i) != input.charAt(length - i - 1)) {
			result = false;
			break;
		}
	}

	return result;
}
```

8. How do you remove spaces from a string in Java?

The following example code shows one way to remove spaces from a string using with the `Character.isWhitespace()` method:

```
String removeWhiteSpaces(String input) {
	StringBuilder output = new StringBuilder();

	char[] charArray = input.toCharArray();

	for (char c : charArray) {
		if (!Character.isWhitespace(c))
			output.append(c);
	}

	return output.toString();
}
```

Learn more about removing spaces and other characters from a string in Java.

9. How do you remove leading and trailing spaces from a string in Java?

The `String` class contains two methods to remove leading and trailing whitespaces: `trim()` and `strip()`. The `strip()` method was added to the `String` class in Java 11. The `strip()` method uses the `Character.isWhitespace()` method to check if the character is a whitespace. This method uses Unicode code points, while the `trim()` method identifies any character with a codepoint value less than or equal to `U+0020` as a whitespace character.
The `strip()` method is the recommended way to remove whitespaces because it uses the Unicode standard. The following example code shows how to use the `strip()` method to remove whitespaces:

```
String s = "  abc  def\t";

s = s.strip();

System.out.println(s);
```

Because `String` is immutable, you have to assign the `strip()` output to the string.

10. How do you sort an array in Java?

The `Arrays` utility class has many overloaded `sort()` methods to sort primitive and to object arrays. If you are sorting a primitive array in the natural order, then you can use the `Arrays.sort()` method, as shown in the following example:

```
int[] array = {1, 2, 3, -1, -2, 4};

Arrays.sort(array);

System.out.println(Arrays.toString(array));
```

However, if you want to sort an array of objects, then the object must implement the `Comparable` interface. If you want to specify the sorting criteria, then you can pass the `Comparator` for the sorting logic. Learn more about Comparable and Comparator in Java.

11. How do you create a deadlock scenario programmatically in Java?

Deadlock is a scenario in a multi-threaded Java environment where two or more threads are blocked forever. The deadlock situation arises with at two or more threads. The following example code creates a deadlock scenario:

```
public class ThreadDeadlock {

    public static void main(String[] args) throws InterruptedException
{
        Object obj1 = new Object();
        Object obj2 = new Object();
        Object obj3 = new Object();

        Thread t1 = new Thread(new SyncThread(obj1, obj2), "t1");
        Thread t2 = new Thread(new SyncThread(obj2, obj3), "t2");
        Thread t3 = new Thread(new SyncThread(obj3, obj1), "t3");

        t1.start();
        Thread.sleep(5000);
        t2.start();
        Thread.sleep(5000);
        t3.start();
    }

}

class SyncThread implements Runnable {

    private Object obj1;
    private Object obj2;

    public SyncThread(Object o1, Object o2) {
        this.obj1 = o1;
        this.obj2 = o2;
    }

    @Override
    public void run() {
        String name = Thread.currentThread().getName();

        System.out.println(name + " acquiring lock on " + obj1);
        synchronized (obj1) {
            System.out.println(name + " acquired lock on " + obj1);
            work();
            System.out.println(name + " acquiring lock on " + obj2);
            synchronized (obj2) {
                System.out.println(name + " acquired lock on " + obj2);
                work();
            }
            System.out.println(name + " released lock on " + obj2);
        }
        System.out.println(name + " released lock on " + obj1);
        System.out.println(name + " finished execution.");
    }

    private void work() {
        try {
            Thread.sleep(30000);
        } catch (InterruptedException e) {
```

```
            e.printStackTrace();
        }
    }

}
```

All three threads will be able to acquire a lock on the first object. However, they are using shared resources and are started in such a way that they will keep on waiting indefinitely to acquire the lock on the second object. You can use the Java thread dump to detect the deadlocks. Learn more about deadlock in Java.

12. How can you find the factorial of an integer in Java?

The factorial of an integer is calculated by multiplying all the numbers from 1 to the given number:

```
F(n) = F(1)*F(2)...F(n-1)*F(n)
```

The following example code shows how to use recursion to find the factorial of an integer:

```
public static long factorial(long n) {
        if (n == 1)
                return 1;
        else
                return (n * factorial(n - 1));
}
```

13. How do you reverse a linked list in Java?

`LinkedList descendingIterator()` returns an iterator that iterates over the element in reverse order. The following example code shows how to use this iterator to create a new Linked List with elements listed in the reverse order:

```
LinkedList<Integer> ll = new LinkedList<>();

ll.add(1);
ll.add(2);
ll.add(3);

System.out.println(ll);

LinkedList<Integer> ll1 = new LinkedList<>();

ll.descendingIterator().forEachRemaining(ll1::add);

System.out.println(ll1);
```

Learn more about reversing a linked list from a data structures and algorithms perspective.

14. How do you implement a binary search in Java?

The array elements must be sorted to implement binary search. The binary search algorithm is based on the following conditions:

- If the key is less than the middle element, then you now need to search only in the first half of the array.
- If the key is greater than the middle element, then you need to search only in the second half of the array.
- If the key is equal to the middle element in the array, then the search ends.
- Finally, if the key is not found in the whole array, then it should return `-1`. This indicates that the element is not present.

The following example code implements a binary search:

```
public static int binarySearch(int arr[], int low, int high, int key) {
	int mid = (low + high) / 2;

	while (low <= high) {
		if (arr[mid] < key) {
			low = mid + 1;
		} else if (arr[mid] == key) {
			return mid;
		} else {
			high = mid - 1;
		}
		mid = (low + high) / 2;
	}

	if (low > high) {
		return -1;
	}

	return -1;
}
```

15. Write a Java program that illustrates merge sort.

Merge sort is one of the most efficient sorting algorithms. It works on the principle of “divide and conquer”. It is based on the idea of breaking down a list into several sub-lists until each sub-list consists of a single element, and then merging those sub-lists in a manner that results in a sorted list. The following example code shows one way to use merge sort:

```
public class MergeSort {

	public static void main(String[] args) {
		int[] arr = { 70, 50, 30, 10, 20, 40, 60 };

		int[] merged = mergeSort(arr, 0, arr.length - 1);

		for (int val : merged) {
			System.out.print(val + " ");
```

```
        }
    }

    public static int[] mergeTwoSortedArrays(int[] one, int[] two) {
        int[] sorted = new int[one.length + two.length];

        int i = 0;
        int j = 0;
        int k = 0;

        while (i < one.length && j < two.length) {
            if (one[i] < two[j]) {
                sorted[k] = one[i];
                k++;
                i++;
            } else {
                sorted[k] = two[j];
                k++;
                j++;
            }
        }

        if (i == one.length) {
            while (j < two.length) {
                sorted[k] = two[j];
                k++;
                j++;
            }
        }

        if (j == two.length) {
            while (i < one.length) {
                sorted[k] = one[i];
                k++;
                i++;
            }
        }

        return sorted;
    }

    public static int[] mergeSort(int[] arr, int lo, int hi) {
        if (lo == hi) {
            int[] br = new int[1];
            br[0] = arr[lo];

            return br;
        }

        int mid = (lo + hi) / 2;

        int[] fh = mergeSort(arr, lo, mid);
        int[] sh = mergeSort(arr, mid + 1, hi);

        int[] merged = mergeTwoSortedArrays(fh, sh);

        return merged;
    }
```

```
}
```

16. Can you create a pyramid of characters in Java?

Pattern programs are a very popular interview topic. This type of question is used to understand the logical thinking abilities of the interviewee. Refer to Pyramid Pattern Programs in Java for examples of different ways to create pyramid patterns.

17. Write Java program that checks if two arrays contain the same elements.

To check if two arrays contain the same elements, you need to first create a set of elements from both the arrays, and then compare the elements in these sets to find if there is an element that is not present in both sets. The following example code shows how to check if two arrays only contain common elements:

```
import java.util.Arrays;
import java.util.HashSet;
import java.util.Set;

public class ArraySameElements {

	public static void main(String[] args) {
		Integer[] a1 = {1,2,3,2,1};
		Integer[] a2 = {1,2,3};
		Integer[] a3 = {1,2,3,4};

		System.out.println(sameElements(a1, a2));
		System.out.println(sameElements(a1, a3));
	}

	static boolean sameElements(Object[] array1, Object[] array2) {
		Set<Object> uniqueElements1 = new
HashSet<>(Arrays.asList(array1));
		Set<Object> uniqueElements2 = new
HashSet<>(Arrays.asList(array2));

		// if size is different, means there will be a mismatch
		if (uniqueElements1.size() != uniqueElements2.size())
return false;

		for (Object obj : uniqueElements1) {
			// element not present in both?
			if (!uniqueElements2.contains(obj)) return false;
		}

		return true;
	}

}
```

```
Output
true
false
```

18. How do you get the sum of all elements in an integer array in Java?

You can use a `for` loop to iterate over the array elements and add them to get the final sum:

```
int[] array = { 1, 2, 3, 4, 5 };

int sum = 0;

for (int i : array)
        sum += i;

System.out.println(sum);
```

19. How do you find the second largest number in an array in Java?

There are many ways to solve this problem. You can sort the array in natural ascending order and take the second last value. However, sorting is an expensive operation. You can also use two variables to find the second largest value in a single iteration, as shown in the following example:

```
private static int findSecondHighest(int[] array) {
        int highest = Integer.MIN_VALUE;
        int secondHighest = Integer.MIN_VALUE;

        for (int i : array) {
                if (i > highest) {
                        secondHighest = highest;
                        highest = i;
                } else if (i > secondHighest) {
                        secondHighest = i;
                }

        }
        return secondHighest;
}
```

20. How do you shuffle an array in Java?

The following example code shows how to use the `Random` class to generate random index numbers and shuffle the elements:

```
int[] array = { 1, 2, 3, 4, 5, 6, 7 };

Random rand = new Random();
```

```
for (int i = 0; i < array.length; i++) {
        int randomIndexToSwap = rand.nextInt(array.length);
        int temp = array[randomIndexToSwap];
        array[randomIndexToSwap] = array[i];
        array[i] = temp;
}

System.out.println(Arrays.toString(array));
```

You can run the shuffling code inside another `for` loop to shuffle multiple rounds.

21. How can you find a string in a text file in Java?

The following example code shows how to use the `Scanner` class to read the file contents line by line and then use the `String contains()` method to check if the string is present in the file:

```
boolean findStringInFile(String filePath, String str) throws
FileNotFoundException {
        File file = new File(filePath);

        Scanner scanner = new Scanner(file);

        // read the file line by line
        while (scanner.hasNextLine()) {
                String line = scanner.nextLine();
                if (line.contains(str)) {
                        scanner.close();
                        return true;
                }
        }
        scanner.close();

        return false;
}
```

Note that the example code assumes that the string that you're searching for in the file doesn't contain newline characters.

22. How do you print a date in specific format in Java?

The following example code shows how to use the `SimpleDateFormat` class to format the date string:

```
String pattern = "MM-dd-yyyy";
SimpleDateFormat simpleDateFormat = new SimpleDateFormat(pattern);

String date = simpleDateFormat.format(new Date());
System.out.println(date); // 06-23-2020
```

Lear more about the Java SimpleDateFormat.

23. How do you merge two lists in Java?

The following example code shows how to use the `addAll()` method to merge multiple lists in Java:

```
List<String> list1 = new ArrayList<>();
list1.add("1");
List<String> list2 = new ArrayList<>();
list2.add("2");

List<String> mergedList = new ArrayList<>(list1);
mergedList.addAll(list2);
System.out.println(mergedList); // [1, 2]
```

24. Write a Java program that sorts HashMap by value.

`HashMap` is not an ordered collection. The following example code shows how to sort the entries based on value and store them into `LinkedHashMap`, which maintains the order of insertion:

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;
import java.util.Map.Entry;
import java.util.Set;

public class SortHashMapByValue {

	public static void main(String[] args) {
		Map<String, Integer> scores = new HashMap<>();

		scores.put("David", 95);
		scores.put("Jane", 80);
		scores.put("Mary", 97);
		scores.put("Lisa", 78);
		scores.put("Dino", 65);

		System.out.println(scores);

		scores = sortByValue(scores);

		System.out.println(scores);
	}

	private static Map<String, Integer> sortByValue(Map<String,
Integer> scores) {
		Map<String, Integer> sortedByValue = new
LinkedHashMap<>();

		// get the entry set
		Set<Entry<String, Integer>> entrySet = scores.entrySet();
		System.out.println(entrySet);
```

```
                // create a list since the set is unordered
                List<Entry<String, Integer>> entryList = new
ArrayList<>(entrySet);
                System.out.println(entryList);

                // sort the list by value
                entryList.sort((x, y) ->
x.getValue().compareTo(y.getValue()));
                System.out.println(entryList);

                // populate the new hash map
                for (Entry<String, Integer> e : entryList)
                        sortedByValue.put(e.getKey(), e.getValue());

                return sortedByValue;
        }

}
```

25. How do you remove all occurrences of a given character from an input string in Java?

The `String` class doesn't have a method to remove characters. The following example code shows how to use the `replace()` method to create a new string without the given character:

```
String str1 = "abcdABCDabcdABCD";

str1 = str1.replace("a", "");

System.out.println(str1); // bcdABCDbcdABCD
```

String is immutable in Java. All the string manipulation methods return a new string, which is why you need to assign it to another variable. Learn more about removing characters from a string in Java.

26. How do you get distinct characters and their count in a string in Java?

You can create the character array from the string. Then iterate over it and create a `HashMap` with the character as key and their count as value. The following example code shows how to extract and count the characters of a string:

```
String str1 = "abcdABCDabcd";

char[] chars = str1.toCharArray();

Map<Character, Integer> charsCount = new HashMap<>();

for (char c : chars) {
        if (charsCount.containsKey(c)) {
```

```
                charsCount.put(c, charsCount.get(c) + 1);
        } else
                charsCount.put(c, 1);
}

System.out.println(charsCount); // {a=2, A=1, b=2, B=1, c=2, C=1, d=2,
D=1}
```

27. Can you prove that a `String` object in Java is immutable programmatically?

The following example code shows how to prove that a `String` object is immutable and the comments in the code explain each step:

```
String s1 = "Java"; // "Java" String created in pool and reference
assigned to s1

String s2 = s1; //s2 also has the same reference to "Java" in the pool

System.out.println(s1 == s2); // proof that s1 and s2 have the same
reference

s1 = "Python";
//s1 value got changed above, so how String is immutable?

//in the above case a new String "Python" got created in the pool
//s1 is now referring to the new String in the pool
//BUT, the original String "Java" is still unchanged and remains in the
pool
//s2 is still referring to the original String "Java" in the pool

// proof that s1 and s2 have different reference
System.out.println(s1 == s2);

System.out.println(s2);
// prints "Java" supporting the fact that original String value is
unchanged, hence String is immutable
```

28. Can you write some code to showcase inheritance in Java?

The following example code shows how to use the `extends` keyword to create a subclass of the class `Animal`. The new class `Cat` inherits the variable from the `Animal` class and adds more code that only belongs to the `Cat` class.

```
class Animal {
        String color;
}

class Cat extends Animal {
        void meow() {
                System.out.println("Meow");
        }
```

```
}
```

29. How do you show a diamond problem with multiple inheritance in Java?

The diamond problem occurs when a class inherits from multiple classes and ambiguity occurs when it's unclear which method to execute from which class. Java doesn't allow extending multiple classes to avoid the diamond problem illustrated by the following example:

```
interface I {
	void foo();
}
class A implements I {
	public void foo() {}
}

class B implements I {
	public void foo() {}
}

class C extends A, B { // won't compile
	public void bar() {
		super.foo();
	}
}
```

30. How do you illustrate a try catch example in Java?

The following example code shows an example of try-catch:

```
try {
	FileInputStream fis = new FileInputStream("test.txt");
} catch(FileNotFoundException e) {
	e.printStackTrace();
}
```

From Java 7 onwards, you can also catch multiple exceptions in a single catch block, as shown in the following example. It's useful when you have the same code in all the catch blocks.

```
public static void foo(int x) throws IllegalArgumentException,
NullPointerException {
	// some code
}

public static void main(String[] args) {
	try {
		foo(10);
	} catch (IllegalArgumentException | NullPointerException e) {
```

```
                System.out.println(e.getMessage());
        }
}
```

31. Write a Java program to show a `NullPointerException`

If you are calling a function on `null`, it will throw a `NullPointerException`, as shown in the following example code:

```
public static void main(String[] args) {
        printString(null, 3);

}

static void printString(String s, int count) {
        for (int i = 0; i < count; i++) {
                System.out.println(s.toUpperCase()); // Exception in
thread "main" java.lang.NullPointerException
        }
}
```

You should have null check in place for early validation, as shown in the following example code:

```
static void printString(String s, int count) {
        if (s == null) return;
        for (int i = 0; i < count; i++) {
                System.out.println(s.toUpperCase());
        }
}
```

You can also throw `IllegalArgumentException` based on the project requirements.

32. How do you create a record in Java?

Records was added as a standard feature in Java 16. Records enable you to create a POJO class with minimal code. Records automatically generates `hashCode()`, `equals()`, getter methods, and `toString()` method code for the class. Records are final and implicitly extend the `java.lang.Record` class. The following example code shows one way to cerate a record:

```
import java.util.Map;

public record EmpRecord(int id, String name, long salary, Map<String,
String> addresses) {
}
```

Learn more about records in Java. For details about POJO, refer to Plain old Java object on Wikipedia.

33. How do you create text blocks in Java?

Java 15 added the text blocks feature. You can create multiline strings using text blocks. The multiline string has to be written inside of a pair of triple-double quotes, as shown in the following example:

```
String textBlock = """
                Hi
                Hello
                Yes""";
```

It's the same as creating a string, such as `Hi\\nHello\\nYes`.

34. Show an example of switch expressions and multi-label case statements in Java.

The switch expressions were added as a standard feature in Java 14. The following examples show switch expressions as well as multi-label case statements:

```
int choice = 2;

int x = switch (choice) {
    case 1, 2, 3:
            yield choice;
    default:
            yield -1;
};

System.out.println("x = " + x); // x = 2
```

You can also use lambda expressions in switch expressions.

```
String day = "TH";
String result = switch (day) {
    case "M", "W", "F" -> "MWF";
    case "T", "TH", "S" -> "TTS";

    default -> {
            if (day.isEmpty())
                    yield "Please insert a valid day.";
            else
                    yield "Looks like a Sunday.";
    }
};

System.out.println(result); // TTH
```

35. How do you compile and run a Java class from the command line?

This example refers to the following Java file:

```
public class Test {

public static void main(String args[]) {
                System.out.println("Hi");
        }

}
```

You can compile it using the following command in your terminal:

```
1. javac Test.java
2.
```

To run the class, use the following command in your terminal:

```
1. java Test
2.
```

For the recent releases, the `java` command will also compile the program if the class file is not present. If the class is in a package, such as `com.example`, then it should be inside the folder `com/example`. The command to compile and run is:

```
1. java com/example/Test.java
2.
```

If your class requires some additional JARs to compile and run, you can use the `java -cp` option. For example:

```
1. java -cp .:~/.m2/repository/log4j/log4j/1.2.17/log4j-1.2.17.jar
   com/example/Test.java
2.
```

36. How do you create an enum in Java?

The following example code shows how to create a basic enum:

```
public enum ThreadStates {
        START,
        RUNNING,
        WAITING,
        DEAD;
}
```

`ThreadStates` is the enum with fixed constants fields `START`, `RUNNING`, `WAITING`, and `DEAD`. All enums implicitly extend the `java.lang.Enum` class and implement the `Serializable` and `Comparable` interfaces. Enum can have methods also. Learn more about enums in Java.

37. How do you use the `forEach()` method in Java?

The `forEach()` method provides a shortcut to perform an action on all the elements of an iterable. The following example code shows how to iterate over the list elements and print them:

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

Iterator<String> it = list.iterator();

while (it.hasNext()) {
	System.out.println(it.next());
}
```

You can use the `forEach()` method with a lambda expression to reduce the code size, as shown in the following example code:

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

list.forEach(System.out::print);
```

38. How do you write an interface with `default` and `static` method?

Java 8 introduced default and static methods in interfaces. This bridged the gap between interfaces and abstract classes. The following example code shows one way to write an interface with the `default` and `static` method:

```
public interface Interface1 {

	// regular abstract method
	void method1(String str);

	default void log(String str) {
		System.out.println("I1 logging::" + str);
	}

	static boolean isNull(String str) {
		System.out.println("Interface Null Check");

		return str == null ? true : "".equals(str) ? true :
false;
	}

}
```

Learn more about about `default` and `static` methods in interfaces in Java 8 interface changes.

39. How do you create a functional interface?

An interface with exactly one abstract method is called a functional interface. The major benefit of functional interfaces is that you can use lambda expressions to instantiate them and avoid using bulky anonymous class implementation.
The `@FunctionalInterface` annotation indicates a functional interface, as shown in the following example code:

```
@FunctionalInterface
interface Foo {
	void test();
}
```

40. Show an example of using lambda expressions in Java.

`Runnable` is an excellent example of a functional interface. You can use lambda expressions to create a runnable, as shown in the following example code:

```
Runnable r1 = () -> System.out.println("My Runnable");
```

41. Show examples of overloading and overriding in Java.

When a class has two or more methods with the same name, they are called overloaded methods. The following example code shows as overloaded method called `print`:

```
class Foo {
	void print(String s) {
		System.out.println(s);
	}

	void print(String s, int count) {
		while (count > 0) {
			System.out.println(s);
			count--;
		}
	}

}
```

When a superclass method is also implemented in the child class, it's called overriding. The following example code shows how to annotate the `printname()` method that's implemented in both classes:

```
class Base {
	void printName() {
		System.out.println("Base Class");
	}
}

class Child extends Base {
	@Override
```

```
        void printName() {
                System.out.println("Child Class");
        }
}
```

Learn more about overriding and overloading in Java.

42.-49. Guess the Output

Test yourself by guessing the output of the following code snippets.

```
String s1 = "abc";
String s2 = "abc";

System.out.println("s1 == s2 is:" + s1 == s2);
```

Output

```
false
```

The output of the given statement is `false` because the `+` operator has a higher precedence than the `==` operator. So the given expression is evaluated to `"s1 == s2 is:abc" == "abc"`, which is `false`.

```
String s3 = "JournalDev";
int start = 1;
char end = 5;

System.out.println(s3.substring(start,  end));
```

Output

```
ourn
```

The output of the given statement is `ourn`. The first character is automatically type cast to `int`. Then, since the first character index is 0, it will start from `o` and print until `n`. Note that the `String substring` method creates a substring that begins at index `start` and extends to the character at index `end - 1`.

```
HashSet shortSet = new HashSet();

        for (short i = 0; i < 100; i++) {
    shortSet.add(i);
    shortSet.remove(i - 1);
}

System.out.println(shortSet.size());
```

Output

```
100
```

The size of the `shortSet` is `100`. The autoboxing feature in Java means that the expression `i`, which has the primitive type `short`, converts to a `Short` object. Similarly, the expression `i - 1` has the primitive type `int` and is autoboxed to

an `Integer` object. Since there is no `Integer` object in the `HashSet`, nothing is removed and the size is `100`.

```
try {
        if (flag) {
                while (true) {
                }
        } else {
                System.exit(1);
        }
} finally {
        System.out.println("In Finally");
}
```

Output

No output. This code results in an infinite loop if the flag is `true` and the program exists if the flag is `false`. The `finally` block will never be reached.

```
String str = null;
String str1="abc";

System.out.println(str1.equals("abc") | str.equals(null));
```

Output

```
Exception in thread "main" java.lang.NullPointerException: Cannot
invoke "String.equals(Object)" because "<local1>" is null
```

The given print statement will throw a `java.lang.NullPointerException` because the `OR` logical operator evaluates both the literals before returning the result. Since `str` is `null`, the `.equals()` method will throw an exception. Its always advisable to use short-circuit logical operators, such as `||` and `&&`, which evaluate the literal values from left to right. In this case, since the first literal would return `true`, it would skip the second literal evaluation.

```
String x = "abc";
String y = "abc";

x.concat(y);

System.out.print(x);
```

Output

```
abc
```

The `x.concat(y)` creates a new string but is not assigned to `x`, so the value of `x` is not changed.

```
public class MathTest {

        public void main(String[] args) {
                int x = 10 * 10 - 10;

                System.out.println(x);
        }
```

```
}
```

Output

```
Error: Main method is not static in class MathTest, please define the
main method as:
   public static void main(String[] args)
```

While it might seem like this question is about the order of execution of the mathematical operators, the question is really about noticing that the main method wasn't declared `static`.

```
public class Test {

        public static void main(String[] args) {
                try {
                        throw new IOException("Hello");
                } catch(IOException | Exception e) {
                        System.out.println(e.getMessage());
                }
        }
}
```

Output

```
Test.java:5: error: cannot find symbol
                        throw new IOException("Hello");
                                  ^
  symbol:   class IOException
  location: class Test
Test.java:6: error: cannot find symbol
                }catch(IOException | Exception e) {
                       ^
  symbol:   class IOException
  location: class Test
2 errors
```

This code results in a compile time error. The exception `IOException` is already caught by the alternative `Exception`.

50. Find 5 mistakes in the following code snippet.

```
package com.digitalocean.programming-interviews;

public class String Programs {

        static void main(String[10] args) {
                String s = "abc"
                System.out.println(s);
        }
}
```

Answers

1. The package name can't have hyphens.
2. The class name can't have spaces.

3. The main method is not `public`, so it won't run.
4. The main method argument shouldn't specify the size.
5. The semicolon is missing in the string definition.