1. What is Java and explain features of java.

- Java is high-level, object-oriented, secure, platform independent, robust, high performance and portable programming language.
- James Gosling - 1991.
- It is also known as Platform as it provides its own JRE and API.

Simple:

- Java is a straightforward, clean, and easy-to-understand language. It is simple because:
- Java syntax is based on C++, so it is easier for programmers to learn it after C++.
- Explicit pointers and operator overloading, among other features, have been removed from Java.
- Automatic Garbage Collection in Java.

Object-oriented:

- Java is an object-oriented programming language. Everything in Java is an object.
- Object-oriented software is made up of a variety of distinct types of objects that include both data and behavior.

Platform Independent:

- Java comes with its own platform for executing its code.
- Java doesn't depend upon any operating system to be executed.

Portable:

- Java is Write Once and Run Anywhere.
- Java program (.java) converted into byte code(.class) which can easily run on any machine.

Secured:

- Because it does not use explicit pointers, Java is safe.
- Java also provides the concept of bytecode and exception handling which make it more secured.

Robust:

- Java is a powerful programming language because it makes extensive use of memory management.
- The concepts like automatic garbage collection, exception handling make it more robust.

Architecture-neutral:

- Java is architecture neutral because it has no implementation-dependent features, such as the size of primitive types.
- The int data type occupies two bytes in 32-bit architecture and four bytes in 64-bit architecture in C programming. In Java, however, it takes up 4 bytes of memory on both 32 and 64-bit platforms.

Interpreted:

- Java uses a JIT interpreter along with the compiler for program execution.

High-performance:

- Java bytecode is faster than other traditional interpreted programming languages since it is "near" native code.

Multi-threaded:

- The primary advantage of multi-threading is that each thread consumes no memory. It shares a common memory area.

Dynamic:

- Java is a dynamic language. It supports the dynamic loading of classes. This means that classes are loaded as needed.

Distributed:

- Java is distributed because it makes it easy for users to construct distributed Java applications. Distributed applications are built using RMI and EJB.
- This Java feature allows us to access files from any system connected to the internet by calling methods.

---

2. Is java pure object oriented?

- Java is not a pure Object Oriented Language because it supports primitive data types such as byte, boolean, char, double, float, int, long, short.
- These primitive data types are not object oriented. This is the reason why Java is not a 100% object-oriented language.

---

3. Can class be declared as protected?

- No, only methods can be declared as protected.

---

4. Can source file contain more than one class declaration?

- Yes. Source file can contain any number of class declarations but only one class can be declared as public.

---

5. What is JRE and why do I need it?

- JRE stands for “Java Runtime Environment”. The Java Virtual Machine (JVM), Java platform classes, and support libraries are all part of it.
- Only previously written apps can be run with JRE. We are unable to create new apps or make changes to existing ones.
- As the name suggests, JRE only provides a Runtime Environment.

---

6. What is JDK and why does it need to be installed?

- JDK stands for Java Development Kit. It's a JRE superset.
- We can use JDK to create, compile, and execute (run) new programmes as well as modify old ones.
- JDK contains JRE as well as development tools (environment to develop, debug and monitor Java programs).

---

7. What is the JVM and why is it necessary?

- JVM stands for Java Virtual Machine. JVM drives the java code.

- We can run Java byte code by translating it to the current OS machine language using JVM.
- It turns Java into a portable language (write once, run anywhere)

---

8. Explain public static void main (String[] args)

- The main() is where every Java program begins. JVM always looks for this method signature to start running an application.
- public static void main(string[] args) can also be written as public static void main(String args[]). Don’t get confused.

public:

- public is an access modifier. The scope of public access modifiers is everywhere.
- It has no restrictions. Declared public data members, methods, and classes can be accessible from anywhere.
- If you make a main() method public then it is allowed to be executed by any program globally.
- If you make a main() method non-public then it is not allowed to be executed by any program.

static:

- To access the main method, the JVM cannot build a class object at runtime. By declaring the main() method as static, JVM can invoke it without instantiating the class.
- JVM will not be able to call the main method to run the program if it is not declared as static.

void:

- The term "void" refers to a method that returns no value.
- Every method in Java specifies the return type.
- In Java, the main method does not return anything.
- Java program starts with the main method and terminates once the main method is finished executing.
- If we make the main method return a value, JVM will be unable to do anything with it. As a result, no value must be returned.

main:

- main is the name of Java main method. When you compile and run a Java program, it is the first thing that runs.
- JVM looks for the main method to start the execution of a Java program.

String[] args:

- String[] args is a parameter that accepts inputs. The name of the String array is ‘args‘ but it can be of anything based on users choice.
- The type of args is always String[] because Java's main method accepts only a single argument of the type String array.
- We can change the array name to whatever and it will still work. i.e. String[] xyz or String[] abc

---

9. How many types of memory are allocated by JVM?

- Heap - whenever an object is created.
- Class area
- Static
- Native method stack.
- Program counter register

---

10. What is a platform?

- Platform is a set of hardware and software environments on which pieces of software are executed.
- Java is a software based platform.

---

11. JIT compiler

At compile Time **At Runtime**
Source code . java → compiler → Bytecode → JIT compiler → Native Machine Code

- The Java Runtime Environment (JRE) is responsible for executing source code and it contains a JIT compiler that does the performance optimization.

- It converts the bytecode into native machine code at runtime.

---

12. What gives Java its ‘Write Once Run Anywhere’ nature?

- The bytecode - Java compiler converts java source code file into byte code which is intermediate language between source code and machine code.
- The bytecode is not platform specific and can be executed on any machine.

---

13. Is the empty .java file name valid?

- Yes.

```
Class A {
    // code
 }
```

To compile: javac .java
To run: java A

---

14. Which OOPs Concepts you used in your current project?

1. Inheritance

We establish a Base Class in a standard Page Object Model to initialise the WebDriver interface, Data Source, Excel Reader, Property File or Config File, WebDriver waits, and so on. We extend the Base Class in our Test Class and Utility Class.We do it by using the extends keyword and achieve the Inheritance. This makes the class more reusable, as we won't have to repeat the startup code.

2. Encapsulation

We know that in a POM Project, we construct a distinct class for each page. All of these classes are excellent examples of encapsulation, in which the data of one class is kept separate from that of another. We declare the data members using @FindBy and initialise them using a constructor with initElement() to utilize them in the test methods.

3. Polymorphism

Polymorphism is based on the idea of a single interface that supports numerous methods. WebDriver is a browser interface that supports several methods such as ChromeDriver(), IEDriver(), SafariDriver(), and FirefoxDriver().

Method Overloading In Selenium

- Methods Overloading is the process of employing two methods with the same name but different parameters in the same class.
- We now use Implicit Wait in Selenium to have the page wait for a set amount of time.
- Because we may supply different Timestamps or TimeUnits like SECONDS, MINUTES, and so on, this is the ideal example of Method Overloading.

Method Overriding In Selenium

- In the WebDriver interface, we use two different methods for navigating or accessing any website i.e. driver.get() and driver.navigate().to().
- These two methods are examples of Method Overriding.

---

15. Abstraction vs Encapsulation

Abstraction	**Encapsulation**
Abstraction solves the problem at design level.	Encapsulation solves the problem at implementation level.
Abstraction is used to hide unnecessary material while presenting relevant information.	To secure data from the outside world, encapsulation involves combining code and data into a single unit.
Abstraction allows you to focus on the object's function rather than how it functions.	Encapsulation refers to the process of hiding the internal details or mechanics of how an object works.

For instance, the appearance of a television, which has a display screen and channel buttons for changing channels, explains Abstraction	However, the inner implementation detail of a television explains Encapsulation by showing how the CRT and Display Screen are connected using separate circuits.

16. Compile-time Polymorphism vs Runtime Polymorphism

Compile-time Polymorphism	Runtime Polymorphism
Compile-time Polymorphism means binding is done at compile time.	Runtime Polymorphism means binding is done at Runtime.
Inheritance is not involved.	Inheritance is involved.
Method Overloading.	Method Overriding.
It provides fast execution.	It provides slower execution as compared to CTP.
Less flexible because all things are resolved at compile time.	More Flexible as all things are resolved at runtime.

17. Method Overloading vs Method Overriding

Method Overloading	Method Overriding
Method Overloading relates to Compile-time Polymorphism.	Method Overriding relates to Runtime Polymorphism.
It helps to increase readability of the program.	It helps to grant the specific implementation of a method which is already provided by the parent class.
It can occur within the class.	It is divided into two classes, each of which has an inheritance relationship.
Method Overloading might or might not need inheritance.	Method Overriding always needs inheritance.

Methods must share the same name but have distinct signatures.	The names and signatures of methods must be the same.
The return type can be the same or distinct.	Return type must be the same.

18. What is the default value of the local variable?

- Local variables are not initialized to any value, not primitive or object references.

---

19. What is the initial value of Object reference which is defined as an instance variable?

- All object references are initialized to null.

---

20. What are different access specifiers in java?

- Private: If a variable is designated private, it can only be accessed by other members of the class. Outside of the class, this variable will be unavailable. As a result, outside members are unable to access private members.
- Public: Methods/variables with public modifiers can be accessed by all the other classes in the project.
- Default: If a variable/method is defined without any access modifier keyword, then that will have a default modifier access.
- Protected: If a variable is declared as protected, then it can be accessed within the same package classes and subclass of any other packages.

---

21. When to use interfaces and when to use abstract classes?

Abstract classes:

- The abstract class is a good choice if you want common base class implementation to all derived classes.

- It is a good choice if you want to use non-public members because in an interface all members must be public.
- If we want to add a new method in future then abstract class is a good choice because if we add a new method to the interface then all the classes that implemented this interface need to implement a new method.
- Abstract classes have an advantage of better forward compatibility. Once a client uses the interface, we can still add new behavior without breaking existing code.

Interfaces:

- Interfaces are a good choice when we think API's will not change for a while.
- If we are using a small concise bit of functionality, use interfaces. If we are designing large functional units, use abstract classes.
- We want to use things similar to multiple inheritance.

---

22. Abstract Class vs Interface

Abstract Class	Interface
Both abstract and non-abstract methods are possible in abstract classes.	Interfaces can only have abstract methods.
Abstract classes can have static, non-static, final, non-final variables.	Interfaces can have only static and final variables.
"abstract" keyword is used to declare abstract classes.	The keyword "interface" defines an interface.
Abstract class doesn't support multiple inheritance.	Interface supports multiple inheritance.
Abstract class provides implementation of interface.	Interface can't provide implementation of abstract class
An abstract class can inherit from another abstract class and implement interfaces.	Interface can extend another java interface only.

'extends' keyword can be used to extend an abstract class.	Interface can be implemented using ‘implements’ keyword.
Members of abstract classes can be public, private and protected.	Members of interfaces are by default public.

Example:
public abstract class Shape{
public abstract void draw();
}
	Example:
	public interface Drawable{
	void draw();
	}

23. What is the purpose of static methods and variables in Java?

- The methods or variables that are defined as static are shared among all the objects of the class.
- The static is a part of a class and not the objects.
- The static variables are defined in the class area and we don't need to create an object of the class to access such variables.

---

24. What are the advantages of packages in Java?

- Packages avoid name clashes.
- Packages provide easier access control.
- We can have classes accessed only within the package.

---

25. What is a Constructor?

- Constructor is a special type of method that has the same name as the class name and is used to initialize the state of an object.
- Every time an object is created using a new keyword, default constructor is called.
- It must not return any explicit value.

---

26. How many types of constructor are used in java?

- Default Constructor and parameterized constructor.

27. What is the purpose of the default constructor?

The default constructor is used to provide the default values to the object like 0, null, etc., depending on the type.

```
class Student{
        int id;
        String name;
        Student_Info(){
          System.out.println(id+” ”+name);
        }

        Public static void main(String[] args){
                Student s1 = new Student();
                Student s2 = new Student();
                s1.Student_Info();
            s2.Student_Info();
        }
}
```

Output:
0 null
0 null

28. Does the constructor return any value?

- Instance of a class

29. Can constructor be inherited?

- No

30. Can constructor be overridden?

- No

31. Can you make the constructor final?

- No, constructor can never be declared as final because it is never inherited.
- If you try to do so it will throw a compile-time error.

32. Can constructor be overloaded?

Yes, by changing the number of arguments or by changing data types of arguments.

```
class Test{
        int i;
        public Test(int k){
            i=k;
            System.out.println(i);
        }

        public Test(int k,int m){
            i=k+1;
            m=k+2;
            System.out.println(i);
            System.out.println(m);
        }
}

class Main{
        public static void main(String[] args){
            Test t1 = new Test(10);
            Test t2 = new Test(10,20);
        }
}
```

Output:

```
10
11
12
```

33. Can we use this and super in a constructor?

- No

34. Constructor vs Method

Constructor	Method
Constructor is a block of code that sets up a newly created object.	Method is a collection of statements which returns value upon execution.
Constructors are used to initialize the object.	Method contains java code to be executed.
Constructor name must match the class name.	Method names can be anything.
Constructor doesn’t not have a return type.	Method must have a return type.
Constructor cannot be inherited by subclasses.	Methods can be inherited by subclasses.
Constructor is initialized when an object is created.	Methods need to be called explicitly.

35. Static Variables and Static Methods

- The term "static variable" refers to the property that all objects share.
- Static methods belong to class rather than objects.
- It can be called using class names. Static variables can be accessed and changed by static methods.

36. What are the restrictions applied to java static methods

- Static methods cannot use non-static variables or call non-static methods directly.
- Because this and super are non-static, they can't be utilised in a static context.

37. Why is the main method static?

- Because objects are not required to call the static method.
- If we make java main method non-static then JVM will have to create an object first and then call main method which will lead to extra memory allocation.

---

38. Can we override the static method?

- No

---

39. Can we overload the static method?

- Yes.

```
class Test1 {
    static void display(int a){
        System.out.println(a);
    }
}

class Test2{
    static void display(int a,int b){
        System.out.println(a+" "+b);
}

public static void main(String[] args){
        Test1.display(10);
        Test2.display(10,20);
    }
}
```

Output:
10
10 20

---

40. Can we make the constructor static?

- Constructor is invoked when an object is created. If we try to make the constructor static, it will throw a compiler error.

---

41. Can we override the overloaded methods?

- Yes

---

42. Can we override the private methods?

- No. Because the scope of private methods is limited to the class and we cannot access them outside of scope.

---

43. Can we change the scope of the overridden method in subclass?

Yes, We can change the scope of a method in a subclass. But, we cannot decrease the accessibility.

- Private: Private can be changed to public, private, protected.
- default: default can be changed to public.
- protected: Protected can be changed to public and default.
- public: public will remain public.

---

44. Can we execute a java program without the main() method?

- It is possible before JDK 1.7. Since JDK 1.7 it is not possible.

---

45. Can we make abstract methods static?

- No.
- In Java, if we declare an abstract method as static then it will become part of a class and we can call it directly which is unnecessary.
- Calling undefined method is completely useless therefore it is not allowed.

---

46. Can we declare static methods and variables inside abstract class?

- Yes

---

47. this vs super

this	super
The current instance of a class is represented by ‘this’ keyword.	The 'super' keyword represents the current instance of the parent class.
In order to call the default constructor of the current class, we can use the ‘this’ keyword.	In order to call the constructor of the parent class, we can use the ‘super’ keyword.
It can be invoked from a static context.	It can’t be invoked from the static context.
We can use it to access current class data members and member functions.	We can use it to access data members and member functions of parent class.

48. Which class is the superclass of all classes in java?

- Object class

---

49. Why are pointers not used in Java?

- Because pointers are unsafe and complex to understand.

---

50. Use of final variable

- Stop value change.
- Stop method overriding.
- Stop inheritance.

51. What is the final blank variable?

- Final variable is a variable that is declared as final and is not initialized.

52. Can you declare the main() method as static?

- yes, public static final void main(String[] args){}

53. Can you declare the interface method as static?

- No. Because methods of interfaces are abstract by default and we can't use static and abstract together.

54. Can we override the main() method?

- No, because the main() method is also a static method.

55. Final vs Static

Final	Static
This modifier is only applicable to inner classes, methods, and variables.	This modifier is applicable to both outer and inner classes, variables, methods, and blocks.
The final variable does not need to be initialised at the moment of declaration.	The static variable must be initialised at the moment of its declaration.
Final variable cannot be reinitialized.	Static variables can be reinitialized.
Final method can’t be inherited.	Static methods have access to the class's static members and can only be called by static methods.
No class can inherit from the final one.	It is not possible to build a static class object, as it only contains static members.

No block for initialization of final variables is supported by the final keyword.	The static variables are initialised using the static block.

56. Can we modify the 'throws' clause of superclass method while overriding it in subclass?

- Yes. But -
- If a superclass doesn't declare an exception, a subclass overridden method cannot declare checked exceptions, but can only declare checked exceptions.
- If superclass declares an exception, then subclass cannot declare parent exception of exception declared by superclass.

---

57. Can we declare the interface as final?

- No. You cannot declare an interface as final because the interface must be implemented by some class to provide its definition.
- Therefore, there is no sense to make it final. However if you try to do so it will throw a compile time error.

---

58. What is a Java instance of operator?

The instanceOf operator in java is also known as type comparison operator because it compares instance with type.

```
class Simple{
        public static void main(String[] args){
                Simple s = new Simple();
              System.out. println(s instanceOf sample)
       }
}
```

---

59. Can we achieve runtime polymorphism by data members?

- No, we can override member functions but not data members.

60. What is an abstract class?

- Abstract classes are those that are declared abstract.
- It requires extension and implementation of its method..
- It cannot be instantiated.
- It can have abstract, non-abstract, static methods and constructors.
- It can also have a final method which will force subclass not to change the body of the method.

---

61. Is it possible to have an abstract method without an abstract class?

- No. If there is no abstract method in a class, the class must be abstract.

---

62. Can you use both abstract and final with a method at same time?

- No. We can't override the final method since we need to override the abstract method to provide its implementation.

---

63. How many ways can we create string objects?

- String literal
  - String s1 = “welcome”;
- New keyword
  - String s1 = new String(“welcome”);

---

64. How many objects are created?

String s1 = “India”;

String s2 = “India”;

String s3 = “India”;

- Only one

---

65. How many objects are created?

String s1 = new String("India");

- Two - one in string constant pool and other in non-pool (Heap memory).

---

66. Why are strings immutable in Java?

- String in Java is immutable, which ensures that the string value does not change. String literals are frequently exchanged among several clients.
- If the value of the string changes (from “ABC” to “abc”), it will affect all reference variables and cause severe discrepancies.
- Hence, strings are immutable in Java. Making a string immutable improves the application's security, caching, synchronisation, and performance.

---

67. What is the difference between equals() method and (==) in Java?

equals() method

- It's used to make that two objects specified by business logic have the same contents.
- The Object class provides a public boolean equals(Object o) function.

double equal operator (==)

- It is a binary operator in Java.
- It examines if both objects are pointing to the same memory location by comparing addresses (or references).

- Default implementation uses a double equal operator == to compare two objects.

---

68. How Object is unreferenced?

- By Nulling
  - s1=null;
- By assigning it to another reference.
  - s1=s2;
- By anonymous object.
  - New A();

---

69. String vs StringBuffer

String	StringBuffer
The String class is immutable.	The StringBuffer class is mutable
When you concatenate too many strings, the String becomes slow and requires more memory since it creates a new instance each time.	The StringBuffer is fast and consumes less memory when you concat many strings.
The String class overrides the Object's equals() method.	The StringBuffer class doesn’t override the equals() method of Object class.

70. StringBuffer vs StringBuilder

StringBuffer	StringBuilder
StringBuffer is synchronized	StringBuilder is not synchronized
Thread Safe	Not thread Safe
Less efficient than StringBuilder	More efficient than StringBuffer

71. How can we create immutable classes in java?

- By making the class and all its members final.

---

72. Why is charArray better than String for password storage?

- Until the garbage is cleared, string stays in the string pool. If we store the password into the string then it stays in memory over long periods of time and anyone having access to the memory dump can extract the password as clear text.
- On the other hand, using char array allows us to set it to blank whenever we are done with the password. It avoids the security threats and enables us to save memory.

---

73. What is garbage collection?

- Garbage collection is the process of reclaiming the unused runtime objects. It is performed for memory management.
- In other words, we can say that garbage collection is a process of removing unused objects from memory to free up space and make this space available for JVM.

---

74. What is the System.gc() method?

- The System.gc() method is used to invoke a garbage collector. The System and Runtime classes both have this method.

---

75. What are the advantages of Exception handling?

The advantages are as follows:

- If an exception is handled, the execution flow will not be interrupted.

- Using the catch declaration, we can pinpoint the issue.

---

76. What are the Exception handling keywords in Java?

try:

- When a try block surrounds a potentially dangerous code. A catch block catches an exception that occurs in the try block.
- Try can be followed by catch or finally or both. However, any one of the blocks is required.

catch:

- This is followed by a try block. Exceptions are caught here.

finally:

- finally is followed either by try block or catch block.
- This block gets executed regardless of an exception. So generally clean up codes are provided in this block.

---

77. Explain the 'throw' keyword in Java.

The ‘throw’ keyword is used to explicitly throw an exception from a method or any block of code.

```
class Test{
	static void fun(){
		try{
			throw new NullPointerException("Demo");
		}catch(NullPointerException e){
			System.out.println("Inside catch");
			throw e;
		}
}
	public static void main(String[] args){
		try{
			fun();
```

```
            }catch(NullPointerException e){
                System.out.println("Caught in Main");
            }
        }
}
```

---

78. Explain ‘throws’ keyword in java?

- The ‘throws’ is used to declare an exception. It notifies the programmer that an exception is possible.
- Exception handling is mainly used to handle checked exceptions. Throws keyword is required only for checked exceptions and usage of throws keyword for unchecked exceptions is meaningless.
- By the help of throws keyword provide information to the caller method about the exception.

```
    class Test{
        static void methodA(char arr) throws IndexOutOfBoundsException{
            System.out.println(arr[6]);
        }
        public static void main(String[] args){
            try{
                char arr[] = new char[5];
                methodA(arr);
            }catch(ArithmeticException e){
                System.out.println(e+ "Exception Handled");
            }
        }
    }
```

---

79. throw vs throws

throw	throws
‘throw’ keyword is used to throw an exception explicitly in a program inside a function.	‘throws’ keyword is used in method signature to declare an exception which might get thrown by a function while executing the code.

We cannot throw multiple exceptions with the throw keyword.	We can declare multiple exceptions that could get thrown by function using throws keyword.
throw keyword is followed by an instance of exception.	throws keyword is followed by class name of exception to be thrown.

80. final vs finally vs finalize

final	finally	finalize
final is a keyword	finally is a block	finalize is a method.
Final is used to apply restrictions on variables, methods and class.	finally is used to execute important code. i.e. close file, close DB connection.	finalize() is used to perform cleanup processing just before an object is garbage collected.

81. How many .class files will be created?

```
class Test{
        class Test1{ // code}
       static class Test2{ // code}
       void run(){
            Helper t = new Helper(){
                Int helpMethod(){
                        return 5;
                }
        };
   }
}
```

- This will produce class files
  - Test.class
  - Test $Test1.class
  - Test $Test2.class
  - Test $1.class (for the implementation of the Helper interface)

---

82. What is the meaning of Collections in Java?

- Collection is a framework for storing objects and manipulating the design in order to store them.
- Collections are used to perform the following operations:
  - Searching
  - Sorting
  - Manipulation
  - Insertion
  - Deletion
- A group of objects is known as collections. All the classes and interfaces for collecting are available in the Java util package.

---

83. What are all the Classes and Interfaces that are available in the collections?

Interfaces:

- Collection
- List
- Set
- Map
- Sorted Set
- Sorted Map
- Queue

Classes:
Lists:

- Array List
- Vector
- Linked List

Sets:

- Hash set
- LinkedHashSet
- Tree Set

Maps:

- Hash Map
- TreeMap
- LinkedHashMap

Queue:

- Priority Queue

---

84. Iterator vs ListIterator

Iterator	ListIterator
It can traverse elements present in the collection only in forward direction.	It can traverse elements present in the collection in forward and backward direction.
Helps to traverse list, map and set.	Helps to traverse only lists.
Index of elements cannot be found using an iterator.	Index of elements can be found using an iterator.
Cannot modify or replace elements.	Can modify or replace elements.
Cannot add elements.	Can add elements easily.

85. ArrayList vs Vector

ArrayList	Vector
ArrayList is not synchronized.	Vector is synchronized.
ArrayList is not a legacy class.	Vector is a legacy class.
ArrayList is fast.	Vector is slow.

ArrayList uses an iterator to traverse elements.	Vector uses an iterator and enumeration to traverse elements.
ArrayList increases the size by 50% if it exceeds its capacity.	Vector increases the size by 100% if it exceeds its capacity.

86. ArrayList vs LinkedList

ArrayList	LinkedList
ArrayList uses dynamic arrays.	LinkedList uses a doubly linked list.
ArrayList is not preferred for manipulation.	LinkedList is preferred for manipulation.
ArrayList provides random access.	LinkedList does not provide random access.
ArrayList stores only objects hence it takes less overhead of memory.	LinkedList stores objects as well as addresses of objects, hence it takes extra overhead of memory.
This class implements List interface hence it can act as a list.	LinkedList implements list and DeQueue interface, hence it can act as list and queue.

87. HashMap vs HashTable

HashMap	HashTable
HashMap is not synchronized.	HashTable is synchronized.
It is not thread safe, hence cant be shared between many threads.	It is thread safe, and can be shared between many threads.

HashMap is fast.	HashTable is slow.
HashMap allows for one null key and numerous null values.	HashTable doesn’t allow any null key or value.
HashMap is a new class implemented in JDK1.2	HashTable is a legacy class.
HashMap is traversed by Iterator.	HashTable is traversed by Iterator and Enumerator.
HashMap inherits AbstractMap class.	HashTable inherits Dictionary class.

88. HashMap vs HashSet

HashMap	**HashSet**
HashMap implements Map interface.	HashSet implements Set interface.
In HashMap we can store key-value pairs.	In HashSet we can store Objects.
HashMap is fast.	HashSet is slow.
Duplicate keys are not permitted in HashMap, although duplicate values are permitted.	HashSet does not allow duplicate values.
HashMap allows for one null key and numerous null values.	HashSet allows one null value.
HashMap is faster than HashSet.	HashSet is slower than HashMap.
HashMap uses put() methods to add elements.	HashSet uses add() method to add elements

89. HashSet vs TreeSet

HashSet	TreeSet
It is implemented through a hash table.	TreeSet is a SortedSet implementation that stores data in trees.
It permits the null object.	It does not allow the null object.
It outperforms TreeSet in terms of search, insert, and delete operations.	For these actions, it is slower than HashSet.
It does not keep elements in a logical arrangement.	The elements are sorted.
It compares two objects using the equals() method.	It compares two objects using the compareTo() method.
It does not permit a heterogenous object.	It permits a heterogenous object.

90. Array vs ArrayList

Array	ArrayList
Cannot contain values of different data types.	Can contain values of different data types.
At the time of declaration, the size must be specified.	Size can be dynamically changed.
To add data, you must first define the index.	No need to specify the index.
Arrays are not type parameterized.	Arraylists are type.
Arrays can contain primitive data types as well as objects.	Arraylists can contain only objects, no primitive data types are allowed.

91. How to initialize a map with key as a string and value as integer?

- HashMap<String, Integer> map = new HashMap<String,Integer>();

---

92. How to add and remove elements from HashMap?

- Add Elements:
  - map.put(“Piyush”,10);
  - map.put(“Rahul”,20);
  - map.put(“Ujwal”,30);
  - map.put(“Shubham”,40);

- Remove Elements:
  - map.remove(“Shubham”,40);

---

93. How to iterate through HashMap?

```
for(Map.Entry<String,Integer> e:map.entrySet()){
        System.out.println(“key”+e.getKey()+ ” value”+e.getValue());
}
```

---

94. Does HashMap maintain insertion order?

- No. TreeMap and LinkedHashMap maintain insertion order.

---

95. What are some of the important Java 8 features?

- forEach() method in iterable interface
- Lambda expressions and functional interfaces
- Static and default methods in interfaces
- Java time API
- Stream API for bulk data operations on collections
- Concurrency API improvements
- Collection API improvements
- Java IO improvements
- Core API improvements

96. What is the output?

```
class Test{
	public static void main(String[] args){
		Test t1 = new Test();
		Test t2 = new Test();
		System.out.println(t1.equals(t2));
		System.out.println(t1 == t2);
	}
}
```

Output:

```
false
false
```

---

97. What is the output?

```
void methodA(){
	try{
		System.out.println("In try block");
	}catch(Exception e){
		System.out.println("In catch-1 block");
	}catch(ArithmeticException e){
		System.out.println("In catch-2 block");
	}
}
```

Output:

```
Compile-time error (unreachable code at second catch block)
```

---

98. What is the output?

```
class Test {
	static void methodA(String str)
		char arr[] = str.toCharArray();
	}
	public static void main(String[] args){
		Test.methodA();
	}
}
```

Output:
No compile time error but NullPointerException at runtime.

---

99. What is the output?

```
class Test {
	static void methodA(String str)
		System.out.println(str);
	}
	public static void main(String[] args){
		Test.methodA(null);
	}
}
```

Output:
null

---

100. Can we write return statement after finally?

Yes.

```
int method_Test(){
	try{
		// code
	}catch(Exception e){
		// code
	}finally{
		System.gc();
	}

	return 0;
}
```