JRE (Java RunTime Environment)

The Java Runtime Environment, or JRE, is a software layer that runs on top of a computer's operating system software and provides the class libraries and other resources that a specific Java program needs to run. The JRE is one of three inter-related components for developing and running Java programs. The other two components are JDK, JVM. JRE also known as Java RTE. JRE includes the Java Virtual Machine (JVM), core classes, and supporting files.

The source Java code gets compiled and converted to Java bytecode. If you wish to run this bytecode on any platform, you require JRE. JRE acts as a layer on the top of the operating system.

JRE consists of the following components:

Deployment technologies such as deployment, Java plug-in, and Java Web Start.

User interface toolkits, including Abstract Window Toolkit (AWT), Swing, Java 2D, Accessibility, Image I/O, Print Service, Sound, drag, and drop (DnD) and input methods.

Integration libraries including Interface Definition Language (IDL), Java Database Connectivity (JDBC), Java Naming and Directory Interface (JNDI), Remote Method Invocation (RMI), Remote Method Invocation Over Internet Inter-Orb Protocol (RMI-IIOP) and scripting.

Other base libraries, including international support, input/output (I/O), extension mechanism, Beans, Java Management Extensions (JMX), Java Native Interface (JNI), Math, Networking, Override Mechanism, Security, Serialization and Java for XML Processing (XML JAXP).

Lang and util base libraries, including lang and util, zip, Java Archive (JAR), instrument, reflection, Collections, Concurrency Utilities, management, versioning, Logging, Preferences API, Ref Objects and Regular Expressions.

Java Virtual Machine (JVM), which comprise of Server Virtual Machine and Java HotSpot Client.

What is Method hiding?

if you create a similar method with the same return type and same method arguments in child class then it will hide the superclass method, this is known as method hiding.

JDK (Java Development Kit)

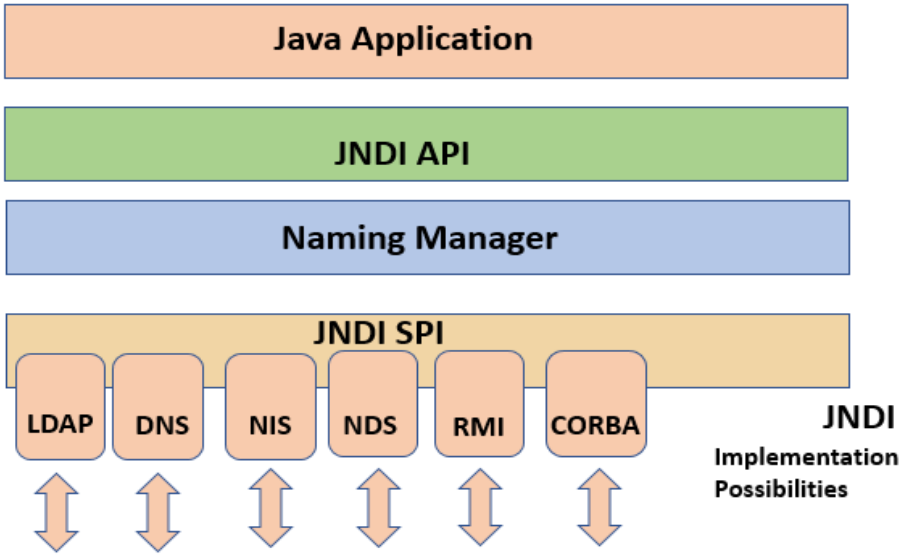
JDK is a set of tools for developing Java applications. Developers choose JDKs by Java version and by package or edition—Java Enterprise Edition (Java EE), Java Special Edition (Java SE), or Java Mobile Edition (Java ME). Every JDK always includes a compatible JRE, because running a Java program is part of the process of developing a Java program.

Java 2D

Java 2D API enables you to easily perform the following tasks: Draw lines, rectangles and any other geometric shape. It comes under JRE.

JNDI (Java Naming and Directory Interface)

JNDI (Java Naming and Directory Interface) enables Java platform-based applications to access multiple naming and directory services. it is Part of the Java Enterprise application programming interface (API) set, JNDI makes it possible for developers to create portable applications that are enabled for a number of different naming and directory services. It comes under JRE.



JNDI comes with two things naming and directory service means

Naming service: It a place where we put objects and methods which is going to be used in our distributed application. And provide them a unique name. So that if we want to use that obj. or method so we can call it through their name directly.

Directory service. It a place where we put objects and methods which is going to be used in our distributed application. And provide them a unique name and a attribute. So that if we want to use that obj. or method so we can call it through their name or directory directly.

If we want to use Naming and Directory service in program we can use JNDI API. Which comes with two thing API & SPI.

JMX (Java Management Extensions)

JMX provide a standard way to managing resource, and monitoring applications and networks. It helps to manage application, objects, devices, service oriented networks.

What is the use of "System.out.printf" in java?

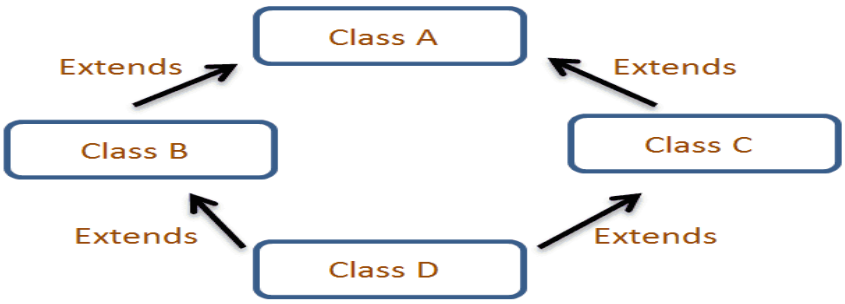
System. out. printf method is used to format the output of numbers and strings.

Example: System.out.printf(Locale.getDefault(),"my name is shubham jadon.",1);

output: my name is shubham jadon.

Why Java doesn't support multiple inheritance?

1) First reason is ambiguity around the Diamond problem, Consider a case where class B extends class A and Class C and both class A and C have the same method display(). Now java compiler cannot decide, which display method it should inherit. To prevent such situation, multiple inheritances is not allowed in java. This is also called the Diamond problem because the structure on this inheritance scenario is similar to 4 edge diamond, see below



2. Second and more convincing reason to me is that multiple inheritances does complicate the design and creates problem during casting, constructor chaining etc. like we call super() from the constructor. if there are two super class then compiler could be confuse which have to choose etc.

A class can implement any number of interfaces but can extend only one class.

The wrapper classes, Byte, Character, Short, Integer, Long, Float and Double are all immutable.

Why interfaces allowed multiple inheritance and does not allowed?

Because interfaces specify only what the class is doing, not how it is doing it.

What is immutable means?

The string is immutable means that we cannot change the object itself, but we can change the reference to the object.

We can make objects of a class immutable with multiple ways: like

- Final keyword.
- Make variable private.

Why does Java not support operator overloading?

Java doesn't supports operator overloading because it's just a choice made by its creators who wanted to keep the language more simple. If you allow programmer to do operator overloading they will come up with multiple meanings for same operator which will make the learning hard and things more confusing and messing.

Why String is final or immutable in Java?

1) The String is immutable in Java because of the security, synchronization and concurrency, caching, and class loading. The reason of making string final is to destroy the immutability and to not allow others to extend it.
2) Immutable string or objects that cannot be modified once it is created. But we can only change the reference to the object.
3) The String objects are cached in the String pool, and it makes the String immutable. The cached String literals are accessed by multiple clients. So, there is always a risk, where action performs by one client affects all other clients. that's why it is immutable. For example, if one client performs an action and changes the string value from Pressure to PRESSURE, all remaining clients will also read that value. For the performance reason, caching of String objects was important, so to remove that risk, we have to make the String Immutable.

What will happen if you put the return statement or System.exit () on the try or catch block? Will finally block execute?

If we use return statement inside try or catch block, finally block will always execute but we use System.exit () inside try or catch block, finally block will not execute.

Note: 1) finally block will always execute even an exception occurred or not in Java.

2) There are few situations where the finally will not be executed like JVM crash, power failure, software crash and etc.

Can you override a private method in Java?

No, we cannot override private or static methods in Java.

Private methods in Java are not visible to any other class which limits their scope to the class in which they are declared.

Can you overload a private method in Java?

Yes, we can overload private methods in Java but, you can access these from the same class due to their scope.

Can you overload static method in Java?

Yes, we can.

Can you override static method in Java?

No, we cannot override static methods because method overriding is based on dynamic binding at runtime and the static methods are bonded using static binding at compile time.

Can we override or overload final method in Java?

No, the Methods that are declared as final cannot be Overridden or hidden but we can override them.

What do the expression 1.0 / 0.0 will return?

The answer to this question is that 1.0 / 0.0 will compile successfully. And it will not throw ArithmeticException. It will just return Double. INFINITY

If a method throws NullPointerException in the superclass, can we override it with a method that throws RuntimeException.

Yes, we can override the method that throws runtimeException. because NullPointerException sub-class of RuntimeException

stringbuffer v/s stringbuilder

- A string buffer is thread-safe whereas string builder is not thread-safe.
- StringBuffer is synchronized. This means that multiple threads cannot call the methods of StringBuffer simultaneously.
- StringBuilder is asynchronized. This means that multiple threads can call the methods of StringBuilder simultaneously.
- StringBuffer was introduced in Java 1.0 whereas string builder was introduced in Java 1.5

Can we access non static variable in static method

Yes, we can throw the obj of class.

```java
public class Snippet
{
   private String instanceVariable;
   private static String staticVariable;

   public String instanceMethod()
   {
      return "instance";
   }

   public static String staticMethod()
   {
      return "static";
   }

   public static void main(String[] args)
   {
      System.out.println(staticVariable); // ok
      System.out.println(Snippet.staticMethod()); // ok

      System.out.println(new Snippet().instanceMethod()); // ok
      System.out.println(new Snippet().instanceVariable); // ok

      System.out.println(Snippet.instanceMethod()); // wrong
      System.out.println(instanceVariable);         // wrong
   }
}
```

ArrayList vs VectorList in java

S.No.	ArrayList	Vector
1.	ArrayList is a resizable or dynamic array.	Vectors are also a form of dynamic array.
2.	ArrayList is not synchronised.	Vector is synchronised.
3.	Syntax of ArrayList: ArrayList<T> al = new ArrayList<T>();	Syntax of Vector: Vector<T> v = new Vector<T>();
4.	If the number of elements overextends its capacity, ArrayList can increase the 50% of the present array size.	If the number of elements overextends its capacity, Vector can increase the 100% of the present array size, which means double the size of an array.
5.	It is not an inheritance class.	It is an inheritance class.
6.	It is faster than Vector.	It is slow as compared to the ArrayList.
7.	It is not a legacy class.	It is a legacy class.
8.	It prefers the Iterator interface to traverse the components.	It prefers an Enumeration or Iterator interface to traverse the elements.

Singleton class in java

Singleton is a design pattern that ensures that a class can only have one object. we can make singleton class with following points.

1) Make a constructor private.

2) Write a static method that has the return type object of this singleton class.

- **Why we create Singleton Class.**

The primary purpose of a Single class is to restrict the limit of the number of object creation to only one.

What will happen if we put a key object in a HashMap which is already there ?

It will replace old mapping because Hash Map doesn't allow duplicate values.

Q11. What makes a HashSet different from a TreeSet?

HashSet	TreeSet
It is implemented through a hash table.	TreeSet implements SortedSet Interface that uses trees for storing data.
It permits the null object.	It does not allow the null object.
It is faster than TreeSet especially for search, insert, and delete operations.	It is slower than HashSet for these operations.
It does not maintain elements in an ordered way.	The elements are maintained in a sorted order.
It uses equals() method to compare two objects.	It uses compareTo() method for comparing two objects.
It does not permit a heterogenous object.	It permits a heterogenous object.

Q12. What are the differences between HashMap and HashTable in Java?

HashMap	Hashtable
It is non synchronized. It cannot be shared between many threads without proper synchronization code.	It is synchronized. It is thread-safe and can be shared with many threads.
It permits one null key and multiple null values.	It does not permit any null key or value.
is a new class introduced in JDK 1.2.	It was present in earlier versions of java as well.
It is faster.	It is slower.
It is traversed through the iterator.	It is traversed through Enumerator and Iterator.
It uses fail fast iterator.	It uses an enumerator which is not fail fast.
It inherits AbstractMap class.	It inherits Dictionary class.

How does Garbage Collection prevent a Java application from going out of memory?

Java Garbage Collector does not prevent a Java application from going out of memory. It simply cleans the unused memory when an object is out of scope and no longer needed. As a result, garbage collection is not guaranteed to prevent a Java app from going out of memory.

Is Java "pass-by-reference" or "pass-by-value"

Java is always "pass-by-value". However, when we pass the value of an object, we pass the reference to it because the variables store the object reference, not the object itself. But this isn't "pass-by-reference." This could be confusing for beginners.

Reflection in Java

Reflection is an API that is used to examine or modify the behavior of methods, classes, and interfaces at runtime. The required classes for reflection are provided under java.lang.reflect package.

Difference between Exception and Error in Java

Exceptions and errors both are subclasses of Throwable class. The error indicates a problem that mainly occurs due to the lack of system resources and our application should not catch these types of problems. Some of the examples of errors are system crash error and out of memory error. Errors mostly occur at runtime that's they belong to an unchecked type.

Exceptions are the problems which can occur at runtime and compile time. It mainly occurs in the code written by the developers. Exceptions are divided into two categories such as checked exceptions and unchecked exceptions.

Note: An exception is an unwanted or unexpected event, which occurs during the execution of a program that disrupts the normal flow of the program's instructions.

- **Checked Exceptions**

They occur at compile time. compiler checks for a these exception. and These exceptions can be handled at the compilation time.

Example of Checked exception- '*File Not Found Exception*'

- **Unchecked Exceptions**

These exceptions occur at runtime. compiler doesn't check for a these kinds of exception. These kinds of exceptions can't be caught or handled during compilation time.

Example of Unchecked Exceptions- *'No Such Element Exception'*

Serialization and Deserialization

Serialization in Java allows us to convert an Object to stream that we can send over the network or save it as file or store in DB for later usage. Deserialization is the process of converting Object stream to actual Java Object to be used in our program. Serialization in Java seems very easy to use at first but it comes with some trivial security and integrity issues.

Serializable in Java

If you want a class object to be serializable, all you need to do it implement the java.io.Serializable interface. Serializable in java is a marker interface and has no fields or methods to implement. It's like an Opt-In process through which we make our classes serializable.

Serialization in java is implemented by *ObjectInputStream* and *ObjectOutputStream*, so all we need is a wrapper over them to either save it to file or send it over the network. Let's see a simple Serialization in java program example.

Points to Note About Serialization in Java?

- Serialization is a marker interface with no method or data member
- You can serialize an object only by implementing the serializable interface
- All the fields of a class must be serializable; otherwise, use the transient keyword
- The child class doesn't have to implement the Serializable interface, if the parent class does
- The serialization process only saves non-static data members, but not static or transient data members
- By default, the String and all wrapper classes implement the Serializable interface.

How to Serialize an Object?

You must use the writeObject() method of the ObjectOutputStream class for serialization and readObject() method of the InputObjectStream class for deserialization purpose.

```
package com.rohitShukla.serialization;

import java.io.Serializable;

public class Employee implements Serializable {

	private String name;

	private int id;

	transient private int salary;

	@Override

	public String toString(){

		return "Employee{name="+name+",id="+id+",salary="+salary+"}";

	}

	public String getName() {

		return name;

	}

	public void setName(String name) {

		this.name = name;

	}

	public int getId() {

		return id;

	}

	public void setId(int id) {

		this.id = id;

	}
```

```
	public int getSalary() {
		return salary;
	}
	public void setSalary(int salary) {
		this.salary = salary;
	}

}
```

Notice that it's a simple java bean with some properties and getter-setter methods. If you want an object property to be not serialized to stream, you can use **transient** keyword like I have done with salary variable.

During serialization, JVM ignores all transient fields. If we need to exclude specific object fields during serialization, mark them as transient.

***Note*:** If we serialize an object which didn't implement the Serializable interface, Java throws java.io.NotSerializableException.

Externalizable interface in Java

Externalization serves the purpose of custom Serialization, where we can decide what to store in stream.

Externalizable interface present in java.io, is used for Externalization which extends Serializable interface. It consist of two methods which we have to override to write/read object into/from stream which are-

Key differences between Serializable and Externalizable

- **Implementation :** Unlike Serializable interface which will serialize the variables in object with just by implementing interface, here we have to explicitly mention what fields or variables you want to serialize.
- **Methods :** Serializable is marker interface without any methods. Externalizable interface contains two methods: writeExternal() and readExternal().
- **Process:** Default Serialization process will take place for classes implementing Serializable interface. Programmer defined Serialization process for classes implementing Externalizable interface.
- **Backward Compatibility and Control:** If you have to support multiple versions, you can have full control with Externalizable interface. You can support different versions of your object. If you implement Externalizable, it's your responsibility to serialize super class.
- **public No-arg constructor:** Serializable uses reflection to construct object and does not require no arg constructor. But Externalizable requires public no-arg constructor.

Below is the example for Externalization-

```
// to read object from stream
```

void readExternal(ObjectInput in)

// to write object into stream

void writeExternal(ObjectOutput out)

Read it for more: https://www.coderscampus.com/java-serialization/

Cloneable in Java

Cloneable is an interface that is used to create the exact copy of an object. It exists in java.lang package. A class must implement the Cloneable interface if we want to create the clone of the class object.

The clone() method of the Object class is used to create the clone of the object. However, if the class doesn't support the cloneable interface, then the clone() method generates the CloneNotSupportedException.

We can also create a copy of an object by using the new keyword, but it will take a lot of processing time. Therefore, using the clone() method is efficient for this purpose. Consider the following example to create a copy of an object using the clone() method.

ClassLoader?

A classloader in Java is a subsystem of Java Virtual Machine, dedicated to loading class files when a program is executed; ClassLoader is the first to load the executable file.

Memory Allocations available in Java?

Java has five significant types of memory allocations.

1. Class Memory
2. Heap Memory
3. Stack Memory
4. Program Counter-Memory
5. Native Method Stack Memory

What are the differences between Heap and Stack Memory in Java?

Stack is generally used to store the order of method execution and local variables. In contrast, Heap memory is used to store the objects. After storing, they use dynamic memory allocation and deallocation.

Check out: https://www.simplilearn.com/tutorials/java-tutorial/java-interview-questions