Quick Work
Page No.:
Date:
M T W T F S S

Difference Between JDK, JRE & JVM

JDK	JRE	JVM
Java Developement kit	Java Runtime environment	Java virtual Machine.
JDK is software developement kit to develop appl" in java	it is a software bundle which provide java class libraries with necessary compo. to run java code.	JVM execute java byte code & provide an environment for executing it.
Platform dependent	Platform dependent	Platform independent
It contain tools for developing debugging and monitoring java code.	It contain class libraries & other supporting files that JVM requires to execute program.	software developement tools are not include in JVM.
It is the superset JRE	It is the subset of JDK	JVM is subset of JRE
JDK enables developer to create java prg that can be executed and run by JRE & JVM	JRE is the part of Java that create JVM	It is the platform component that executes source code
JDK comes with the installer.	JRE contain only environment to execute source code.	

Components of JDK

JDK is is a software derelopement environment which is used to develop java appl & applets. It physically exist. It contain JRE + derelopement tools.

appletviewer - this tool is used to run & debug Java applets without web browser.

apt: annotation-processing tool.

extcheck: It is a utility that detect JAR file conflict

idlj: An IDL to java compiler. This utility generate java bindings from a give java IDL file

jabswitch: It is a Java Acces Bridge. Exposes assistive technologies on ms windows system

java: The loader for Java appl? This tool is an interpreter & can interpret the class file generated by javac compiler. Now single launcher is used for deployment & derelopement jre is replaced by new java loader.

Javac: It specifies the java compiler, which convert source code into java bytecode.

javadoc: documentation generater. which automatically generate documentation from source code comments.

9 What is JIT compiler & its role in JVM ? What is bytecode & why it is important ?

JIT compiler is component of JRE that translate bytecode to machine code at runtime to improve the performance of java appln.

- JIT compiler speeds up java classes, avoid recompilation when code hasn't change.

→ JIT compiler is enabled bydefault & run after prg starts. It analyse the code being execute & identifies part that would benenfit from compilat" o recompilation.

→ Bytecode is an intermediate code that generate by compiler & is made up of numeric code, constant, references. It's designed to be executed efficiently by a software interpreter.

→ Bytecode is fundamental principal of java & it makes java so portable. Java prg compile into bytecode, which interpreted by JVM. JVM is interpreter for each method in Java pr

→ Bytecode is bridge bet? High level code & Low level code.

JVM Architecture

class loader - class loader component is Responsible for loading java class file into JVM at Runtim Locate & read class tyle, verify their bytecode & define classes within the JVM.

Runtime data Area - memory area where the jvm manages data during prg execution.

Method area - store class level data, includin

In jara all the method

bytecode of methods, constant pool, static variable.

Heap - runtime data area where obj allocate
1. young generation - Elden, survivor, other survivor.
2. old generation - long live obj

jara stack - local variable, method argument method specific data stores. It also manage calls & returns.

Native method stack - holds native methods specific data, similar to jara stack. Use for executing native methods.

Program Counter - keeps track of the currently executing bytecode instruction

execution enjine - execute jara bytecode. It can employ different technique for bytecode execution like interpretation, JIT compilation combination of both.

Native method Interface - JNM allow jara prg. to interact with native code written in lang. like C, C++. It provide mechanism for jara code to call native method & access native libraries

JVM Language - JVM architecture support lang. other jara through addition compilers & runtime suppat. eg. Kotlin, scala, Groovy

Role of JVM in Java

Memory Management system of JVM

JVM manages memory automatically through a process that includes garbage collection & compaction.

Garbage collection -

when heap is full, the JVM activates garbage collection to clear out unused objects & free up memory.

Compaction. -

JVM uses compaction to move objects within the heap to free up space. JVM choose external & internal compaction depend on current garbage collect^{n} mode & pos^{n} of compaction area. external compaction near top of heap & internal compaction at bottom.

Memory types -

JVM manage two types of memory heap & stack heap is used for dynamic memory allocation while local variable & method paramenters stack

Memory allocation.

JVM memory management is designed to allocate and manage memory efficiently so that java application run smoothly and without errors.

- It load class file from jre/lib/rt.jar.
- Bootstrap loader is a parent of all other classloader instance.
- It is a part of JVM & it written in native

extension class loader -
→ it is a child of bootstrap class loader
→ take care of loading the extension of the standard core java classes.
→ extension class loader load from jdk extension directory.

System class loader -
→ it is child of extension classloader
→ the system class loader, on the other hand, t care of loading all the appl's level class t
→ It loades files found in the classpath envir variable. -classpath or -cp command line option.

Garbage Collection -

→ In java garbage means unreferenced object.
→ Garbage collection is process of reclaiming th runtime unwed memory ~~allocation~~ automation
→ in other word it is a way to destroy unused obje
→ It make java memory efficient because garbage collector removes the unreference obj from heap memory.
→ It is automatic done by garbage collect (part of jvm) so we don't need to do extr effort. 1. By nulling reference
2. By assigning a reference to anoth
3. By anonymous ob

How will you measure size of obj in java? there is no sizeof() operator in java

Quick Work
Page No.:
Date:
M T W T F S S

	private	protected	default
Access level	limited to the same class	Accessible with same class & subclass	Access within same package
inheritance	Not inherited by subclass	inherited by subclass	inherited by subclasses within the same package
visiblity	visible only within class	visible within the class & subclass	visible within same package & subclasses
encapsulation	Use for encapsulat° & data hiding	Allow limited access to members	provide default access within the same package.

	public
Access level	Access everywhere
inheritance	inherited by subclass
visiblity	visible from any class / subclass
encapsulat°	provide full access to members

Q What happen if you declare variable / function as private in class & try to access it from another class within the same package?

⇒ the variable / fn? we declare private they are accessible only within that class. They are visible only within that class. Other class of same package or different will not able to access these members.

Q Explain the concept of "package-private" or 'default access. How does it affect the visibility of class members?

⇒ If we do not specify any access modifier then the default or package-private is default. They are visible only within same package.

Q can a top level class in Java be declared as protected or private? why or why not?

⇒ private means accessible only within that class and protected means accessible only in same package & its subclasses. Thus top level class or interface are never declared as private.

Q is it possible to make a class private in java? if yes where can be it done & what are the limitations?

→ If we make class private, the code is not accessible outside the class. It doesn't make any sense. There would be no way to access that class members.

Q can you override a method with a different access modifier in a subclass & for ex. can a protected method in a superclass be overriden with a private method in a subclass ? explain.

→ Yes, protected member of superclass can be overridden by a subclass. If the superclass method is protected, the subclass overridden method can have protected or public which means the subclass overridden method cannot have a weaker access specifier.

Access Modifiers in Java

1) Access Modifiers
controls the access/scope level of class, field, method, constructor.

2) Non access modifiers
do not control access level but provide other functionality

There are 4 access modifiers

1. Private :- access within the class in which they declare. Not access outside the class

2. default - access only within same packages. if do not specify any access specifier then it is default. outside package can't access

3. protected - access in same package and its child (subclass) outside the package through inheritance.

4. Public - access everywhere. access within same class, outside the class, within package & outside.
→ widest scope.

* class → public / default.
* Attribute, methods, constructer → private / public / protected / default

Non access specifiers.

* class → final / abstract
* attribute / method → final / static / abstract / transient / volatile / synchronized.

1. final - class cannot be inherited by other class.
attribute / method cannot modify / overridden
2.
3. static - Attribute & method belong to class rather than an obj

Overview of JVM