OOPJ

Tuesday, October 26, 2021 17:39

1. QnA EXPT1

a. What is JAVA?

Ans: JAVA is a general-purpose computer programming language that is concurrent, class based, object oriented and specially designed to have as few implementation dependencies as possible.

b. Features

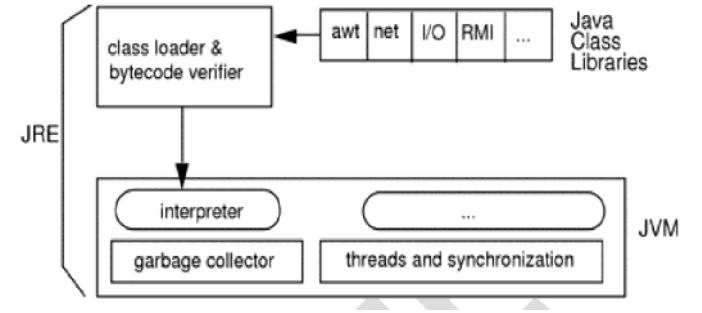
i. rich in API (Application programming interfaces)
ii. open source libraries
iii. platform independent bytecode
iv. free and opensource
v. community support
vi. object-oriented programming language
vii. robust in nature

c. Services are provided by JVM at runtime

i. JVM operates on primitive values.
ii. The JVM provides garbage collector, class loader, bytecode verifier, etc

d. JRE vs. JVM (java runtime env. Vs java virtual machine)

i. JRE is superset of JVM
ii. Source code compiled with java compilers (JAVAC) produces the bytecode (.class files). Source code--->JAVAC-->bytecode
iii. JRE = JVM + many required libraries
iv.

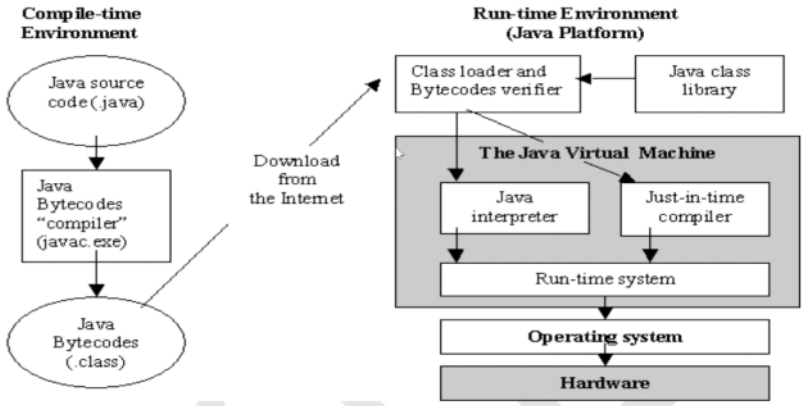


v. JVM runs the bytecode and produces machine code

e. Why we set value for java PATH variable?

i. If we set the path variable - conveniently run the executables such as (javac.exe, java.exe, Javadoc.exe) and so on from any directory without having to type the full path command.
ii. If you don't - specify full path at every time of execution and running

f. JAVA Architecture, Compilation and Execution Phase

i. 

ii. Source code--->JAVAC-->bytecode (Platform independent)
iii. Java compilation time is slower than other languages that's why we have just in time compiler.

g. Theory

i. Source file = compilation unit = text file having class definitions.
ii. all code must reside inside a class
iii. Java is case-sensitive --> name of class should match the name of the file
iv. Cmd line arguments - We are passing the filename to the executables javac and java
   1) C:\>javac filename.java // filename.java, javac is the compiler
   2) C:\>java filename // passing filename to java interpreter as cmd line argument
v. After compilation --> individual class is put into its own output file filename.class
vi. When you execute the Java interpreter as just shown, you are actually specifying the name of the class that you want the interpreter to execute
vii. main() method --> public static void main(String args[]) {
   1) Public --> access specifier --> to control the visibility of class members
      a) If public --> accessed by code outside the class
      b) If private--> prevents a member from being used by code
   2) static --> without having to instantiate a particular instance of the class
      Since main() is called by the Java interpreter before any objects are made.
   3) Void --> main() does not return a value
viii. System.out.println("This is a simple Java program.");
   1) **Built-in println( )** displays the string which is passed to it.
   2) **System** --> predefined class that provides access to the system,
   3) **out** --> output stream that is connected to the console
ix. Command-Line Arguments - pass information into a program to main() when it's running
   1) information that directly follows the program's name on the command line
   2) stored as strings in the String array passed to main( ).

h. Algorithm:

i. Create class Calendar.
ii. Define the main method.
iii. Store command line arguments in particular variables. (First is date, second is month and third is year).
iv. By using switch case and different for loops display particular day for given date.
v. Save the program with the name of the class which consist of main() method.

vi. Compile the program from command prompt by using javac command.
vii. Execute program by passing command line arguments.

2. EXPT 2

a. object-oriented programming =

i. paradigm that provides inheritance, data binding, polymorphism
ii. Aim --> implement real-world entities
iii. Class

b. Class vs Object

	Class	Object
	blueprint or template from which objects are created	instance of class
	Logical entity, group of similar objects Fruits, human	Physical, real world entity Man, Woman, guaha,
How to create	Declared using **class**	Created using **new**
How many times	Declared once	As per requirement
Does it allocate memory when created?	Doesn't allocate	Allocates memory

c. Constructor in JAVA

i. Types of constructors are
   1) Default constructor
   2) Parameterless constructor
   3) Parameterized constructor

d. Garbage Collector

i. Java has no destructor because of garbage collector which frees memory
ii. java.lang.System.gc() method runs the garbage collector
iii. can be called explicitly.

e. Value types and reference data types with sizes

Primitive Datatype	Default Type	Size	Object data types
boolean	False	1 bit	Boolean
char	"\u000"	2 bytes	Character
byte	0	1 byte	Byte
short	0	2 bytes	Short
int	0	4 bytes	Integer
long	01	8 bytes	Long
Float	0.0f	4 bytes	Float
Double	0.0d	8 bytes	Double

f. Wrapper Classes convert **Primitive data type --> Object Data Type**

g. Java is Strongly Type Checked Language

i. every variable must be declared with a data type
ii. variable cannot start off life without knowing the range
iii. once declared, data type cannot change.

h. POJO -

i. Plain Old JAVA Object
ii. class that doesn't need to be a subclass of anything
iii. Doesn't implement specific interfaces or follow specific pattern

i. Practical

i. Array object - collection of indexed(/ identifies by a number) elements of the same type
ii. variable of type array contains a reference to an array object.

3. EXPT3

a. Object Cloning - Create exact copy of an object

Conditions:

i. define clone() method in object class
ii. Implement java.lang.Cloneable interface by the clonable object
    1) Otherwise Clone() generates CloneNotSupportedException.
iii. Syntax of the clone() method -->
    - protected Object clone() throws CloneNotSupportedException
iv. Shallow copy vs Deep copy

	Shallow copy	Deep copy
	bit-wise copy of an object	fully independent copy of an object.
	exact copy of the values in the original object	Entire structure is copied

b. Passing Parameters

Way of Passing parameter	Pass by value	Pass by reference
In java	Present	Not passible
	changes being done in the called method, is not affected in the calling method.	

c. Methods vs functions

	Methods	Functions
	method is on an object	independent of an object
In java	There are only methods in java	Not passible in java
	functions that are to do with an object are called methods	For C++ it would depend on whether or not you're in a class
	In java, all functions are methods as they are all to do with object.	In C++, functions are bits of code that will perform a particular action - but are **not associated with an object**.

d. Passing an array

	Passing an array	Passing an array reference
	Just like passing an object	
	reference to that array is	Passed by value

	copied	
	Any changes in the content of array through that reference will affect the original array.	changing the reference to point to a new array will not change the existing reference in original method.

e. Uses of '**this**' keyword

i. to refer current class instance variable
ii. to invoke current class method (implicitly)
iii. to invoke current class constructor.
iv. to return the current class instance from the method
v. passed as argument in the constructor call
vi. passed as an argument in the method call.

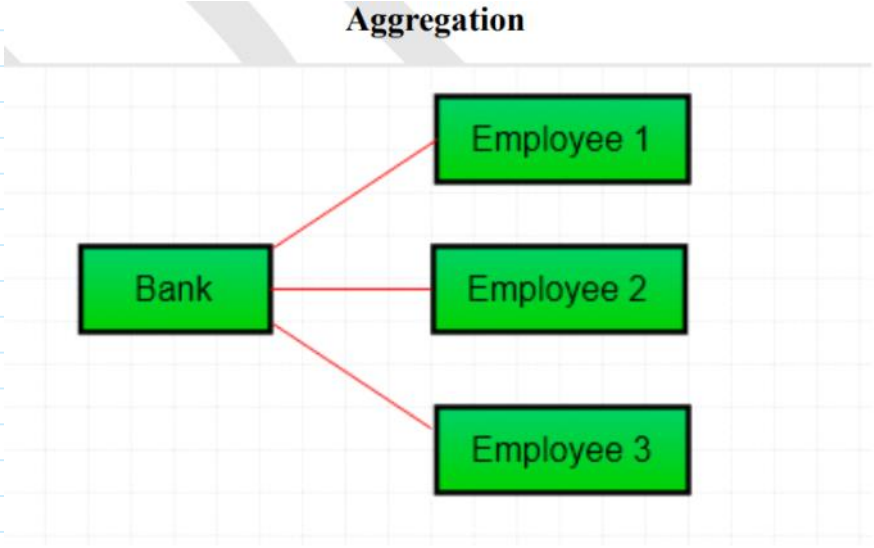
4. EXPT4

a. Inheritance

i. Mechanism -->derive a class from another class for a hierarchy of classes that share a set of attributes and methods.
ii. derived class is called subclass, or child class.
iii. The class from which it is derived is superclass/ parent class

b. association, aggregation and inheritance

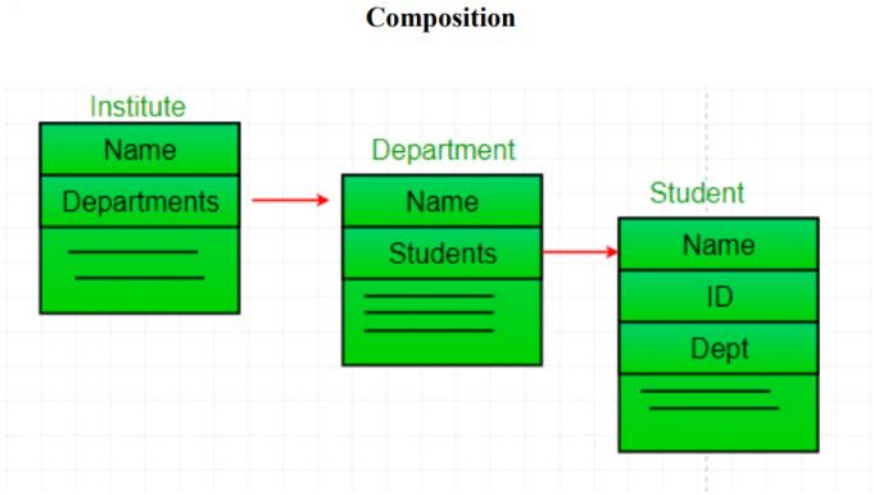
i. Aggregation -

1) 

2) Represents Has-A relationship
3) a unidirectional association i.e. one way relationship
4) both the entries can survive individually
5) ending one entity will not affect the other

ii. Association

iii. Composition

1) 

2) both the entities are dependent on each other
3) composed object cannot exist without the other

c. Super vs this

d. Default object class

i. Java.lang.Object class is the super base class of all Java classes
ii. Every other Java classes descends from Object
iii. If a class is declared without extending another class then it will implicitly extend Object class

e. methods of Object class

toString()	String representation of an Object	to convert an object to String
hashCode()	For every object, JVM generates a unique hash	returns distinct integers for distinct objects
equals(Object obj)	gives a generic way to compare objects for equality	
getClass()	Returns the class object of "this" object	to get actual runtime class of the object
Finalize()	to dispose system resources, perform clean-up activities and minimize memory leaks.	
clone()	returns a new object that is exactly the same as this object	
Wait(), Notify(), NotifyAll()	related to concurrency	

f. Upcasting

1) an object of a sub class can be referred by its super class
2) done during Runtime

i. Benefit of Upcasting
   1) polymorphism or grouping different objects becomes possible
   2) To access the methods in the super class through the object of sub-class

g. @Override annotation

1) To override a method in sub class

2) Using this is a good practice in java
3) Benefits
    a) improves the readability
    b) by using this annotation you instruct compiler that you are overriding this method. If you don't use the annotation then the sub class method would behave as a new method (not the overriding method) in sub class

h. final field, final method, final class

i. Final Field --> constant

1) used only for the values that we want to remain constant throughout the execution
2) A variable when declared with final keyword, its value can't be modified

ii. Final Method --> cannot be overridden

1) e.g. almost all methods of object class

iii. Final Class --> cannot be Inherited to another class

i. Static Field and Static Variable

i. Static Fields

1) A field of a static class can be accessed without an instance of the class
2) good means of storing information

ii. Static Methods vs Instance Methods

	Static Methods	instance methods
	can be called without an instance of that class	instance methods of a class cannot be called from static methods of that class
	don't have any access to instance variables, or instance fields	because there is no instance of the class present in a static method.
	instance variables store information about an instance of a class	
	static methods do not have access to the this keyword because this refers to the current instance of a class, which is not present in the scope of a static method.	However, if an instance of the class is passed to the static method, that instance is free as usual to perform its instance methods in the scope of the static method.
	Used for functionality that is universal to a class	instances of a class can call the public static methods of that class
	do not care about instances of the class or their state	instance of a class can only call its private static methods, like any other private method, from within the class.

1) static method of a class can be called without an instance of that class
2) static methods don't have any access to instance variables, or instance fields because instance variables store information about an instance of a class

j. Runtime Polymorphism or Dynamic method dispatch

i. Mechanism --> a call to an overridden method is resolved at run time, rather than compile time