1. **What is Java? What are benefits of Java language?**

- Java is a high-level, general-purpose and robust object-oriented programming language that is widely used programming language in this internet era. The first Java release happened in May 1995 by Sun Microsystems Inc., and James Gosling is known as the father of Java.
- Java language is platform independent with one of the core principles of WORA (Write Once Run Anywhere), which means the same compiled code can be executed on many operating systems (like Linux, Windows, Mac, etc.) without any source code changes.
- Java supports many commonly needed programming features such as inbuilt automated garbage collection, security, networking and multithreading features, etc.

1. **What is Java Platform? Why it is called a platform?**

- The Java itself is referred to as a platform because of its virtual machine (JVM), where the compiled code is executed. The JVM creates an environment in similar to the operating system for the programs to execute.
- The JVM (Java Virtual Machine) is a specification (JSR336 for Java7) and can be implemented by anyone. The Sun's/Oracle's JDK (Java Development Kit) is the Reference Implementation for a given JVM specification. The specification and reference implementation are governed through JCP (Java Community Process). Please note that any new specification related to Java should go through Java Specification Request (JSR) and there should be Reference Implementation (RI) to show as working specification.
- Java has 3 types of platforms.
- Java SE or J2SE (Standard Edition)
- Java EE/J2EE (Enterprise Edition) Standards/Specification
- Java ME (Micro Edition) → Mobile and devices
- Get more details from Oracle's Java technology site - http://www.oracle.com/technetwork/java/index.html

1. **What are key features of Java?**

- Here is the summary of key features list.
- Java is a simple, high-level and general purpose object-oriented programming language
- Java itself is a platform and technology.
- Java is cross-platform with WORA (Write Once and Run Anywhere) support.
- It is secured and distributed language.
- Java is a network-centric language used for Internet applications.
- It supports multi-threaded coding to achieve high performance.
- Java is a highly robust & reliable platform with automated garbage collection and fewer runtime errors without memory pointers like in C++.
- Java is a static type checking language.
- Java is extensible technology with lot of additional packages available.
- It supports internationalization (i18n).

1. Is Java language cross-platform?

- Yes. One of the key features of Java is this cross platform support. That means once the Java bytecode is generated from the source code on one platform, then it can be executed on other platforms without porting/changing the code.

1. **What is JVM? Describe more on this.**

- JVM stands for Java Virtual Machine.
- The JVM can be expressed as a collection of the specification, implementation, and runtime instance. The specification is about the bytecode specification, and the implementation is the JRE, and the runtime is the instance of VM whenever Java code is executed.
- There are multiple implementations from different vendors.
- HotSpot VM from Oracle(acquired Sun Microsystems, Inc.)
- JRockit VM from Oracle(acquired BEA)
- IBM's J9 VM
- Other VMs
- JVM mainly does the following operations.
- Loads the bytecode using the bootstrap classloader and then apps classloader
- Verifies the bytecode
- Executes the bytecode
- Provides the runtime environment.
- The Java HotSpot JVM is the latest and has the below features.
- Adaptive compiler, which loads and analyzes the code for performance bottlenecks (hotspots) and takes the appropriate action to boost the performance.
- Rapid memory allocation and garbage collection with fast and efficient collectors.
- Thread synchronization with multithreaded code designed to scale with shared memory multi-processors.
- Two types of HotSpot VMs

1. **What is JRE? Describe more on this.**

- JRE stands for Java Runtime Environment.
- It contains the JVM and other core libraries/APIs required for java bytecode/class file execution.
- JRE is a must for any Java program/.class file execution.
- Please note that JRE alone is not sufficient for Java code development.

1. **What is JDK? Describe more on this.**

- JDK stands for Java Development Kit.
- It contains the JRE and other development tools like javac, jar, keytool etc., and other libraries/APIs to provide the java development environment.
- It is required for any java code development (compilation, bundling, using supported libraries, etc.)
- Also include other integration technology APIs/libraries such as JDBC, JAX-WS, etc.

1. **Does JVM know about Java language? If yes, how much or what API it knows? If not, then what it can do?**

- No. JVM doesn't know the Java language itself. It only knows about the particular binary .class format, which contains the instruction bytecode, symbol table, and other ancillary information.
- See more details from Java Virtual Machine specification (Java SE 8) https://docs.oracle.com/javase/specs/jvms/se8/jvms8.pdf

1. **What is the relation between JDK, JRE and JVM?**

- JVM instance is created when java process gets executed (as simple as java command is invoked) and it is always part of JRE. Note that JVM can't be separated out from JRE.
- JRE contains JVM + Core APIs/libraries to get java program runtime environment (JRE=JVM+Core libraries).
- JDK contains the JRE, tools and other integration APIs/libraries (JDK=JRE+Tools+other jar files)

1. **Could you describe in general how is the Java code development process or lifecycle?**

- At a basic level, the Java coding process goes like this.
- Source code (step 1) → Compile (step 2) → Run class (step 3)
- Start with source code writing as a text file with .java extension and it should be coded from top to bottom (in such a way that code is getting executed and in general from top to bottom unless control statements needed).
- Use IDE like Eclipse, NetBeans, IntelliJ, etc. or vi, vim, notepad or any text editor.
- It is recommended to have the filename (<filename>.java) with camelcase style, i.e., capital letter at beginning of the word and lowercase letters in all other places. Example: HelloWorld.java
- Next, do the compilation of the code as below.
- javac <filename>.java
- It generates the .class file with same file name with .class extension (<filename>.class). Example: HelloWorld.class
- Now, ready to run the class with java command, which launches the JVM to execute the class bytecode. Please note that don't supply the .class extension. The below command executes the class bytecode in the launched JVM.
- java <filename-no-class-ext>
- Finally, the developed classes can be bundled as jar file. That means all project related multiple classes can be packaged as .jar (called java archive) using jar tool. Later, these jar libraries can be given to customers or reuse in other projects.

1. **What is meant by compilation? Why do you need to do?**

- The compilation is the process of converting the human-readable machine independent source code into the machine or binary code (instructions to the OS).
- The compilation process is required to create an executable and low-level machine understandable programs. Examples to compiled languages: Java, C++, C, FORTRAN, etc.
- The compiled code is much faster and high performance than interpreted programs.
- The interpreted program means some other program interprets the source code directly without converting to any intermediate binary form (Example interpreted languages: Perl, Python).
- Java uses java program as an intermediate program to create a virtual platform so that cross-platform execution can happen without re-compile/re-create the code.

1. **How do you compile the Java code?**

- The following is the general command syntax for compiling java code
- javac -d <outdir> -cp <classpath> <filename>.java
- This generate the .class file at outdir directory if no compilation errors otherwise prints list of errors/warnings.
- One can use the IDEs to compile java code in the projects.

1. **What is meant by WORA?**

- The "WORA" stands for Write Once Run Anywhere.
- It is one of the core objective of Java technology where cross platform/ portability of code is the key. That means writing of code should not be repeated for every OS/platform, and instead it should work without repeating the developer effort.

1. **What is the Java program's runnable format?**

- The Java program's runnable file is a special binary formatted file called class format. It is a bytecode file with .class extension corresponding to same source .java filename. This binary code is executed by JVM to run on the machine.

1. **How to determine the java and javac versions?**

- Run the the java/javac command with -version option.

```
$ java -version
java version "1.7.0_71"
Java(TM) SE Runtime Environment (build 1.7.0_71-b14)
Java HotSpot(TM) 64-Bit Server VM (build 24.71-b01, mixed mode)

$ javac -version
javac 1.7.0_71
```

1. **How do you run Java programs using the command line?**

- Run the java programs using the command line (CLI) is as below.
- java -cp <classpath> <full-classfilename-no-class-extn>
- The full classname means prefixing the package name (the name used in the code with package statement) and dot (.) to the class like <package>.<classname>
- Examples:
- Or run the program with IDE. Typically during the development time for high productivity.

1. **How do you resolve classes dependency during compilation and run-time? What are different ways?**

- There are two ways to resolve the class dependencies from javac/java commands:
- Add all the dependent class paths (called setting classpath) using an environment variable called "CLASSPATH".
- Or add all dependent class paths using -cp or -classpath option of javac/java command.
- Please note that IDEs might do in different ways from UI by adding jars or classes into the project build settings but would do internally one of the above.

1. **What is a Java package? Why do you use?**

- The Java package is a technique to organize and manage the classes in similar to directories in the filesystem. The package name is followed by reverse domain name (com.jd) notation and specified as "package <pkgname>" statement at beginning of the java program.
- The primary purpose of the package is to resolve the namespace conflict for classes where the same class name (with different code) can be used while developing software(by the same company or different companies).

1. **How do you package the classes into a single file? What is the format of the file?**

- The packaging of multiple Java classes into a single file can be done with jar tool and the jar file format is similar to zip format.
- In general, the jar files are the java libraries with class files and related resources bundled together. There is a manifest file with details about jar is also included.
- Below are some of the key operations with jars.
- To create the jar file from set of files

jar -cvf <filename>.jar <files-or-dirs>

- To extract the jar file contents

jar -xvf <filename>.jar

- To see the jar file contents

jar -tvf <filename>.jar

1. **What do you set in the CLASSPATH (or) -cp (or) -classpath options? Like class file path/class files directory path/jar files directory path/exact jar file path?**

- Set the directory path in CLASSPATH or -cp <classpath> (=<directory>) to include all .class files and <package>.<classfile> classes in the classpath. Please note that jar files if any available in that directory will be not be included. Also, multiple directories can be specified in the classpath with path separator (: on Unix or ; on Windows) to include all classes from all those directories.
- Set specific jar files path in CLASSPATH or -cp <classpath> (=<jarfilepath>), then all classes in the jar files will be included. Also, multiple jar file paths can be specified in the classpath with the path separator (: on Unix or; on Windows platform).
- Set both directory and jar paths together to cover all dependencies (all classes from directories & jar files) in the classpath. Please note that this way in general used in the java command.

1. **What is JVM stack? When does this be created?**

- The JVM stack is the LIFO (Last In First Out) short lived fast memory allocated while executing the java program.
- The stack gets created whenever a method is getting called to store the parameters, local variables (method level), and references to the variables in the heap.
- The stack memory size can be specified using -Xss jvm option while running the java program otherwise uses default size (512k on 32-bit VM on Solaris and 312k on 32-bit VM on windows; 1024k on 64-bit VM).

1. **What is StackOverflowError? When do you get it and how to resolve?**

- The JVM throws java.lang.StackOverflowError, whenever it's stack is full.
- Typically, the stack full can happen in the below situations.
- When the recursive function doesn't have a termination condition and goes into infinite loop.
- When many variables and large size data structure is being used in the default or lower stack sized JVM.
- To resolve the above error, check the following.
- Fix the recursive method termination condition
- Increase the stack size or reduce the size for variables.

1. **What is OutofMemoryError? When do you get this and how to resolve?**

- The java.lang.OutofMemoryError occurs when JVM heap is full during the runtime. Please note that the physical memory can be more than the actual size available to JVM.
- The typical reason for this error could be spikes in the data usage by the java application or memory leaks from the application.
- The memory leaks can happen when leaving behind the allocated memory (not cleaning up) every time the application logic gets executed and thereby filling up all heap in the JVM.
- The error can be resolved after analyzing the objects in the heap and performing the one or all of the below.
- Reduce the data size
- Address the memory leaks
- Increase the heap
- The heap size available to JVM can be customized (increasing/reducing) by specifying -Xms (for initial size) and -Xmx (max size) JVM options while running java program.

1. **What is ClassNotFoundException? When do you get this and how to resolve?**

- The java.lang.ClassNotFoundException is a checked exception in Java and is thrown by the JVM when the given class can't be loaded by its classloader.
- Typically it happens when the application tries to load the named class and can't load in the following.

i) forName() in Class
ii) findSystemClass() in the ClassLoader
iii) loadClass() in the ClassLoader

- The root cause for this exception is that the given class is not in the classpath (not in either CLASSPATH system variable or -cp or -classpath options don't have the class or jar file path or not specified the full classname with package name).
- To resolve the above issue, make sure the classpath is appropriately set in the CLASSPATH environment variable or -cp or -classpath JVM options.

1. **What is JVM heap? When does it gets created?**

- The Java heap is the main memory used by all parts of Java application during execution.
- The heap is going to be created whenever JVM launched/started. That is when java command is issued.
- All Java objects are going to be stored in the heap, and that means whenever an object is created, it is always stored in the heap and is globally accessible across the JVM.
- The heap memory is going live since the start to the end of the application runtime. The heap memory itself is divided into different generations like young generation, old generation, etc., and is cleaned up by automatic Java garbage collector.
- The default JVM heap size is 64MB even when the machine has more RAM.
- The Java runtime throws the java.lang.OutOfMemoryError when the heap is full.
- The heap size available to JVM can be customized (increase/decrease) using -Xms (initial size) and -Xmx (for maximum size) JVM options. Example to set 1G initial and 2GB max JVM heap, then use -Xms1G Xmx2G while running java program.
- If more Java objects and data structures are being constructed in the Java program, then it is required to tune (increase) the heap size of the JVM.

1. **What is Garbage Collection? What is its role in JVM?**

- The Java Garbage collection is the automated process of recovering the unused objects from the Java heap memory. These unused objects are the objects without having any references from other parts/objects of the program.
- The programmer doesn't need to worry about allocating the memory for data structures/variables and also deallocating or to release or clean up those objects from the program itself. Internally, Java will automatically do this process. The deallocation is handled by this Java Garbage Collector (GC).
- It is not guaranteed when GC will do this. The Java will automatically determine the right algorithm based on the goal of the optimized performance of the program execution.
- The GC can be triggered forcibly but again not guaranteed and might impact the program performance. So, it is not recommended to take the GC control in the application program.
- Below are the few Garbage Collector types to understand better on the available options.

i) Serial GC
ii) Parallel GC

- See more details on using the Java options for parallel or serial GC at http://www.oracle.com/webfolder/technetwork/tuto-rials/java/java/index.htmldex.html

1. **What is a variable? Where do you define?**

- Variable is a named memory data pointer to hold data in memory. The values (size and layout) of the variable would be in the range of the defined by the data type.
- Variable has an associated data type like int or object data type such as String.
- Variables are defined either in the scope of class level and method level.
- The variable definition syntax can be seen like <datatype> <variablename>;
- Example: int numberOfItems;

1. **What is meant by variable assignment and initialization?**

- Variable initialization is the definition and assigning value in a single statement.
- Example: int maxNumberOfItems=1000;
- Variable assignment is nothing but setting value after defining the variable in a different statement. <variablename>=<value>.
- Example: maxNumberOfItems=2000;

1. **What are different types of variables?**

- Below are different types of variables.
- Instance or Non-static or Object variables/fields
- Non-instance or Static or Class variables/fields
- Local variables

1. **What is a static variable, and its scope? How do you reference it?**

- The static variable is a variable defined with "static" keyword and defined at class scope level or in a static method of a class.
- These variables belong to the class (not to the specific objects) and thereby called class variables (not instance variables).
- Below is the syntax for variable definition.
- static <datatype> <variablename_id> = <value>;
- Example: static int counter = 1; // the counter value is same irrespective of number of objects created.
- The static variables are referenced using <ClassName>.<variablename> Ex: Hello.counter;
- The static variables get initialized during the class loading and initialized with values.
- All instances share the same copy of the variable.

1. **What is an instance variable and its scope? How do you reference?**

- The instance variables are the variables that are defined at class scope level without static keyword. Also called non-static variables or object variables.
- The instance variables are belong to class instances, i.e., objects. That's why these variables are called instance variables.
- The general syntax for instance variable definition is like:
- <datatype> <variablename/id> = <value>;
- Example: int i = 10;
- The instance variables referenced using <objectreference>.<variablename>; Example: h = new H(); h.i=20;
- The instance variables are initialized by default.

1. **Are all non-static variables are instance variables?**

- No. The non-static variables defined in the method scope called local variables and not instance variables. Also, note that the instance variables should be defined at class level.

1. **What is a local variable, and its scope? How do you reference?**

- The local variables are the variables defined within a method i.e, scope is at method level.
- The local variables should be referenced with the variable name and only within the defined method (local to the method).
- Local non-static variables are not initialized by default, and one has to assign before use. Otherwise, there will be compilation error like "variable x might not have been initialized".

1. **What are the differences between static variable and instance variable?**

- The static variables defined at the class level or within a static method with special keyword 'static' whereas instance variables are the variables without keyword 'static' defined at same class level. Note that both are called fields.
- The static variables are belong to class and thereby called class variables whereas instance variables belong to object and thereby called instance variables.
- The class variables dereferenced with <classname>.<variablename> whereas the instance variables dereferenced with <objectname>.<variablename>.
- If the requirement is to have a single copy of the variable throughout the class, then define as static variable (like a counter to keep track of visited pages) else if each object should have a copy of its own data, then define as an instance variable.

1. **What is a field? How is it same or different (or same) from other types of variables?**

- The field is an attribute or an instance variable or class variable that is defined at the class scope. It is same as the static or instance variable but not the local variable.

1. **What is a data type? Where do you use?**

- In static type programming languages such as Java, the type of the data also must be specified before storing the data. Thereby compiler can do the static type checking to reduce the runtime errors.
- The datatype is used as a part of a variable definition. The supported data types depend on the language.
- The Java data types are as below.
- Primitive data types or primitives
- Object data types
- See more details at https://docs.oracle.com/javase/tutorial/java/nut-sandbolts/datatype-s.html

1. **What are the default values for instance or static variables?**

- Below are the default values assigned automatically when the instance/stat- variables declared with corresponding data types.
- byte : 0
- short : 0
- int : 0
- long : 0L
- float : 0.0f
- double: 0.0d
- boolean: false
- char: '\u0000'
- String or any object type: null
- Examples:

```
int i;  // i=0
String n; //n=null
```

1. **What is meant by Parameter? How is it different from a variable or field?**

- The parameter is the variable used within the method signature as METHOD argument. It is declared same as variables without any initialization. So, parameters can be classified as variables but should not be treated as fields. The similar kind of parameters is used in the constructors.
- Example:

public void main(String [] args) {} // here args is the parameter

1. **What are different techniques to pass the parameter values from caller to callee method?**

- The values of the parameters passed from the caller to the callee is usually determined by the way the language supported. There are two main techniques.

i) Pass by value

ii) Pass by reference

1. **How does the "Pass by value" works?**

- In the "Pass by value" technique, the value of the method parameter to be passed is going to be first copied and then assigned to the actual method's argument value.
- Here, the value is going to be passed than actual memory reference even if it is the object. In this case, its object reference value will be copied and then send to the called method.
- By this, the variable value passed to the method can't be modified through the method. If an object is passed to the method and its object's instance variable changed within the method, then it gets reflected after method execution is completed.
- Example: Java supports this technique.

1. **How does the "Pass by reference" works?**

- In the "Pass by reference", the actual reference of the parameter value is going to be passed to the method than as a copy.
- In this, the variable or object passed to the method can be modified through a method.
- Example: C++ supports but Java doesn't support this technique.

1.What are the differences between "Pass by value" and "Pass by reference"? Please give an example.

- In the "pass by value", the parameter value is going to be copied and then passed to the method whereas memory reference is passed to the method call in the pass by reference.
- See the below example and output to notice the difference between these techniques.

```
Public class Test {
    String x = "Hello";
    method1(x);
    System.out.println(x);
    ...

    Employee e1 = new Employee("Test1");
    method2(e1);
    System.out.println(e1.getName());
    ..

}

public void method1(String s) {

System.out.println(s);
      s = "hi";

System.out.println(s);

}

public void method2(Employee e) {

System.out.println(e.getName());

e.setName("ModifiedName");


e = new Employee("JD1");

e.setName("JD2");
}
```

Console o/p w.r.t "Pass by value" or "Pass by reference".

Hello

hi

hi --> It is not right for "pass by value" but it is correct for "pass by reference".

hello --> displayed with "pass by value" (in Java).

-

Test1

ModifiedName (not "Test1" for "Pass by value" and "JD2" if it the "Pass by reference".)

1. **What is a literal? Any examples?**

- The literal is a fixed value that is specified in the source code and can be assigned to a variable.
- Example:

byte b = 50;

- Integer literals

int n = 20;

- Floating-point literals

float pi1 = 3.1415f;

- Character and String literals

char c = 'x';

String description="This is a test case for feature x";

1. **What is an expression?**

- The expression is a combination of variables, operators, and methods to produce a single value that depends on the data types of the variables and values used in the expression.
- Examples:

int x=10;

int y = 20;

*double z = (x*3.14+y)/10;*

1. **What are the unary & binary operators?**

- The unary means single, and the unary operator is associated with a single operand in the expression.
- Below are the unary operators with simple examples.
- + : makes the expression as positive.
- - : makes expression as negative
- ++ : increment by 1
- -- : decrement by 1
- ! : logical inversion or negation
- Examples: *i++;* (or) *i--;* (or) *++i;* (or) *–i;*
- Binary means two and the binary operator is associated with two operands in the expression.
- Operators : +, -, /, *, %
- Examples: x = y + 20;

1. **What are prefix & suffix unary operators? What are the differences?**

- The prefix unary means the operation (either increment or decrement) performed on the operand first and then evaluated (value is substituted) the value in next. In the expression, the operator going to be specified first and then operand.
- Syntax example: ++i or --i
- The suffix unary means that operand is going to be evaluated (value is substituted) first, and the later operand is going to be incremented or decremented. In the expression, the operand is going be first and then the operator.
- Syntax: i++ or i--
- Examples:
- i=5; system.out.println(i++);//prints 5 and i=6
- i=6; system.out.println(++i);//prints 7 and i=7
- i=7; system.out.println(--i);//prints 6 and i=6
- i=6;system.out.println(i--);//prints 6 and i=5

1. **What is a statement? What are different kinds of statements?**

- A statement is a unit of execution that is expressed in the high-level language with syntax supported by the language.
- At a high level, statements can be classified as below.
- Assignment expression statements
- Prefix or suffix unary expression statements
- Object creation expression statements
- Method invocation statements
- Block statements
- Control statements

1. **Can we have an empty statement? How do you write an empty statement?**

- Yes. Empty statements are ok in Java.
- It is kind of no operation (no-op).
- Simply write open curly braces and close curly braces ({}).

1. **What is a block? Can we have an empty block? Do we need to have ";" at the end for a block?**

- A block is a group of zero or more statements together with statements inside open and close curly braces with appropriate balance (not crossing with other blocks/braces).
- Yes. An empty block is allowed. It is kind of no-operation (nop).
- No ';' after the block statement.
- Example:

```
if (status){ //block1 open
        System.out.println("PASSED");
} //block2 close
else { // block1 open
        System.out.println("FAILED");
} //block2 close
```

1.What are looping statements? Could you briefly explain?

- The "for" loop statement: It is a control flow statement where the block (a group of statements) is going to be iterated through a range of values.
- The general syntax is as below.

```
for (initialization-expression; termination condition; increment/decrement expression) {
  // statements
}
```

- Here is the flow of execution.
- If the termination condition never finishes (when increment/decrement expression is not getting closure to the termination condition), then it will be in the infinite loop.
- Example:

```
//Print 10 hello words
for (int i=0; i<10; i++) {
System.out.println("hello");
}
```

- The "while" loop is a another control flow statement and the while block gets executed until the condition expression is evaluated to false.
- General syntax is as below.

```
while (condition) {
// statements
}
```

- Example:

```
// Print 10 hello words
int i=0;
While (i<10) {
   System.out.println("hello");
}
```

- The "do-while" loop is a control flow statement and the block executed at least once and until while condition is evaluated to false.
- General syntax as below.

```
do {
   // statements
} while (condition);
```

- Example:

```
// Print 10 hello words
int i=0;
do {
System.out.println("hello");
} while (i<10);
```

1. **How do you represent infinite loops with "for", "while" or "do-while" statements?**

- Infinite "for" statement can be done by simply not having any initialization/termi-tion/incre-ment expressions. If required to terminate the infinite loop, then add if condition and use break statement.

```
for (;;) {
//statements
}
```

- Infinite while can be achieved by having true in condition.

```
while (true) {
 //statements
}
```

- Infinite do-while can be achieved by having true in condition.

```
do {
 //statements
}while (true);
```

1.What is meant by branching? What are different branching statements? Where do you use?

- The branching is to get out of the normal execution flow. Below are 3 main statements.
- break
- continue
- return
- break : It is used to come out of the loop or block. Used frequently with a switch statement.
- Unlabeled break - use simply "break"
- Example:

```
int i=0;
for (; ;){
  // statements
  if (i==10) break;
  //statements
  i++;
  //statements
}
```

- Labeled break - use break with label.

```
label:
for ( ; ;){
  //statements
  break label
  //statements
}
```

- continue : It is used to continue next iteration by skipping the current iteration in the loop.
- Unlabeled continue: simply use "continue".

```
int i=0;
for (; ;){
  // statements
  if (i==10) continue;
  //statements
  i++;
  //statements
}
```

Labeled continue: use continue <label>.

```
label:
for ( ; ;){
  //statements
  continue label
  //statements
}
```

return: It is used to come out of the executing method to the caller.

Usage without return value: simply use "return".

Usage with return value: return <variable/value>;

```
public int computeVal(int val) {
  // statements
  if (val>100) {
    return val;
  } else {
    return 100;
  }
}
```

What is the purpose of switch statement? How do you write switch statement?

The switch statement is used when many execution paths arise in the program logic. The same use case can be done with the if-then-else statement, but there will be many if-else conditions.

Switch works with byte, char, int primitive data types and wrapper data types, enumerated data types, String objects.

The switch syntax:

```
switch (data) {
  case <value>: //statements
                break;
  case <value>: //statements
                break;
  case <value>: //statements
                break;
                default: //statements
                      break;
                }
```

- Example as code snippet:

```
/*
  * Guess my vehicle based on given seats
  */
  public String guessVehicleType(int seats) {

String vehicle = null;


switch (seats) {

case 1:


 vehicle = "Bicycle";
```

1. **How do you write an interface? Can you implement multiple interfaces from single class?**

- In general, an interface is for establishing a contract with outside clients.
- The interface is a reference type in similar to class with few restrictions.
- The interface can contain only contain method declarations/ signatures, constants, static methods, default methods, etc.
- The interface should not contain any implementation (there should not be {} for methods) except for default and static methods.
- Interfaces can't be instantiated and can only be implemented by classes or extended by other interfaces.
- Multi-inheritance for interfaces (extending multiple interfaces) in Java is allowed.
- Multiple interfaces can be implemented ("implements" keyword) by a single class.
- Use the interface reference type (i.e., return type) than implementation class while used by the clients.
- Example interface and implementation: Here Vehicle is the interface and Car is the implementation/concrete class. Notice the method differences.

```
/**
* Vehicle.java
*
*/
package jd;

/**
          * Vehicle interface/contract
* @author Jagadesh Babu Munta
*
```

1. **How do you write abstract class? Can you extend from multiple classes?**

- The abstract class is a class that is declared with abstract keyword. At least one of the methods should be declared as abstract, i.e., without an implementation and have only method declaration.
- Example:

```
public abstract class  Graph {
   public abstract void draw();
   void getLocation() {
      ..

   }
}
```

- In Java, multi-inheritance with classes (extending from multiple classes) including abstract classes is not allowed. Java supports only single inheritance for the classes.
- The implementation class must implement all of the abstract methods. Otherwise, the class must also be declared as abstract.
- Implementation class Example:

```
public class  GraphImpl extends Graph {
   void draw() {
      //statements

   }
}
```

1. **What are the differences between an interface and abstract classes?**

- In general, interfaces formulate a contractual agreement with clients, and abstract classes formulate hierarchical relationship.
- In the interfaces, by default all methods are public, fields are static and final whereas, in abstract classes, the fields can be non-static or non-final and can define public/protected/private methods.
- In interface classes, all methods are implicitly abstract and thereby not explicitly added the abstract modifier to the methods. Whereas abstract classes can have 1 or more abstract methods and must be declared as abstract methods.
- Interfaces are defined with interface keyword whereas abstract classes and methods are defined using abstract keyword.
- Also, the abstract class can implement interfaces, but it has to declare as abstract if any method is not implemented.
- Interfaces can support multiple inheritance (extend multiple interfaces) whereas abstract classes support only single inheritance (only one abstract class can be extended). That means a single class can implement multiple interfaces whereas only one abstract class can be extended.
- Both interfaces and abstract classes are similar where these can't be instantiated to create the concrete objects and instead either implemented or extended in creating the concrete objects.

1. **When do you use interfaces and abstract classes?**

- Interfaces can be used in some of the use cases like below.
- Need a specific contract per client without concerning on the implementation.
- Multiple inheritance data type is needed.
- Unrelated classes should be following some contract by implementing those interfaces to have different behaviors.
- Abstract classes can be used in some of the use cases like below.
- Sharing common code (common methods and members) across the closely related classes.
- Declare non-static and nonfinal fields.
- Need a hierarchy among the classes.

1.**How do you write constructors?**

- In OOP, constructors form the basis for creating the objects from the class blueprint.
- Java class constructors can be declared in similar to methods but as a special type of method with name same as the class name and no return type.
- A class can contain any number of constructors either with a different set of arguments or even without any arguments, which is called the default constructor.
- If no constructors defined in class, then Java automatically provides one default constructor. This default constructor calls super class's default constructor automatically. If the class doesn't have direct inheritance, then it will call the very top level Object class's default constructor, which is empty and does nothing.
- Note that if any constructor is defined in class, then the compiler will not provide a default constructor and also it doesn't call superclass's default constructor automatically.
- Example constructors code snippet:

```
public class UserProfile {
   String userName;
   String name;
   String email;

   //No argument constructor
   public UserProfile() {

this.userName="admin";

this.name = "Administrator";
        this.email = "appadmin@gmail.com;
   }
   // Non default constructor
   public UserProfile(String userName, String name, String email) {
```

```
        this.userName = userName;
         this.name = name;
         this.email = email;
    }

    public static void main(String [] args) {
UserProfile p = new UserProfile();
                UserProfile p1 = new UserProfile("jdbabu","Jagadesh Munta,
"jagadesh.munta@gmail.com");
    }
}
```

- The access modifiers can also be specified to the constructors and will have the effect on how other classes can access the class.
- For example, to define a Singleton class, you can define private access modifier to the default constructor so that no other class can construct the objects other than its class itself.

1. **Can you use a default constructor without defining it in a class? Please explain why or why not?**

- Yes. The default constructor is automatically provided by Java when there are no constructors defined. So, one can use the default constructor to create the objects. Also, the superclass's default constructor is called automatically and if no superclass, then top level Object class's default constructor is called.

1. **In class, if you have defined a parametric constructor only, then can you construct the objects with default constructor? In this context, does java provides the default constructor?**

- No. When the class declares any constructor, then Java will not provide the default constructor. Thereby one can't use it unless default constructor is defined within the current class in addition to the parametric constructor.

1.What is an inner class? What it represents in OOP? Describe more on nested classes?

- An inner class is a non-static nested class that is defined within a public class called outer class (enclosing class). That means an outer class can contain one or more inner classes.
- This kind of nested class represents the encapsulation in OOP(Object Oriented Programming).
- Typical use cases of nested classes are as below.
- Needs more closed logical grouping or having local custom type in the class
- Needs more encapsulation
- Needs more readable and maintainable code with logically related functionality together.
- Nested classes are 2 types:
- Static nested classes
- Non-static or Inner classes

- The general syntax/example is as below:

```
public class MyOuterClass {
  // other members

   static class MyNestedClass {
    //statements
   }

   class MyInnerClass {
    //statements
   }

   private void testMethod1() {
       class MyLocalClass {
        // statements
       }

       // Anonymous class

            MyTest mt = new MyTest() {
                public void test1() {
                }
            }
        }

    public static void main(String [] args) {
                        MyOuterClass.MyNestedClass nc = new MyOuter-
Class.MyNestedClass();

            MyOuterClass main = new MyOuterClass();
            MyInnerClass inner = main.new MyInnerClass();
    }
}
```

1. **How does the generated class look like for an inner class?**

- The generated inner class name would be in the below form as <outer-class-name>$<inner-class-name>.class
- Examples:
- Main$UserProfile.class
- MyOuterClass$MyInnerClass.class

1. **What is meant by access modifier? Where do you use? What are different access modifiers?**

- The access modifiers, also called access controls are the keywords that determine whether other classes can access specific fields or methods or members of the current class.
- These are used to achieve encapsulation (data/implementation hiding) in OOP programming for the protection of members.
- List of access modifiers at class level:
- public : this class can be accessed by any class.
- package-friendly or no access modifier keyword: this class can be accessed only by the same package class.
- List of access modifier at member level:
- public : this member is
- protected: this member is
- package-friendly or no access modifier keyword: this member is
- Private: this member is accessible ONLY in the current class

- See the below summary as the table view for easy understanding.

Access Modifier type to a member	Accessible from its own class	Accessible from its Package	Accessible from its Subclass	Accessible from any other or World
public	Yes	Yes	Yes	Yes
protected	Yes	Yes	Yes	No
<no modifier> or package friendly (no keyword for this)	Yes	Yes	No	No
private	Yes	No	No	No

1. **How do you provide access when you have private access members?**

- For this case, provide the public methods so that external classes can use.
- In fact, this is the typical way to do the data encapsulation.
- The recommendation is to use private access modifiers for fields and provide public methods to interact with other classes.

1. **Can you assign multiple access modifiers to a single member?**

- No. Only one access modifier keyword is allowed.

1. **What is final keyword significance? Where do you use?**

- The "final" keyword is to indicate that the value of a variable is fixed and can't be modified. It can be used for variables, constants, methods and classes.
- If the "final" keyword is used for a method, then this method can't be overridden.
- If "final" keyword is used for a class, then this class can't be inherited by any class.
- Example: To define constants, use final as below -

static final int MAX_COUNT=1000;

1. **Can you define a private constructor? If yes, why do you do?**

- Yes. One of the main use case to define a private constructor is create singleton class (design pattern) where only a single instance of a class exits in the JVM. The external access is provided through another public static method or static instance variable than directly creating the objects from constructors.

1. **What is a Java exception? Could you please brief on different types of exception?**

- The Java exception is an event that disrupts the normal flow of execution.
- In general, there are 2 types of events occur in Java.

i) Errors

ii) Exceptions are 2 types

- See the later questions on the differences between these exceptions.
- Example of exceptions handling.

```
try {
  // statements that can raise checked exception.
} catch (Exception e) { // catch with top level Exception class or specific subclass such as FileNotFoundException.
        e.printStackTrace(); //print the stack trace
        //other statements to handle the exception. Or re throw
        // throw new Exception(e);
} finally {
        //statements get executed always, irrespective of exception raised or not.
}
```

1. How is the Exception and Error classes tree?

- java.lang.Object
- java.lang.Throwable

1.How to handle Java exceptions? Which exception to handle first when multiple exceptions to be caught?

- Use try-catch or try-catch-finally block code to handle the raised checked exceptions. The potential exception raising code must be surrounded by try block and then matching exception class must be in the catch block, which gets executed automatically when there is a matched catch block. Otherwise, the compiler can raise an exception if that exception is checked exception.
- (or) Use "throws <exception class>" in the method to pass in the method call stack when the current code is not able to handle the exception directly.
- In the catch arguments, beginning exception classes must be specific class names and later should be generic or super classes. Otherwise, the next Exception classes might not be caught as super classes can match the exception class before child classes.
- For example, FileNotFoundException and some other IOException raised, then it should be like as below (FileNotFoundException must be before IOException or Exception classes).

```
try {
  //..
} catch (FileNotFoundException fnf) { //...
} catch(IOException ioe) { //..
} catch(Exception e) { //..
}
```

- Below is the general syntax for handling exceptions. The finally block gets executed all the time irrespective of the catch blocks executed or not. To simply print the stack trace, use printStackTrace() method on that object.

```
try {
        //statements
} catch (<exception> <variable>) {
        // print stack trace or handle the event
        //<variable>.printStackTrace();
} catch (...) {
        // print stack trace or handle the event or re throw using throw to raise another exception
}
.....
} catch (...) {
        // print stack trace or handle the event
} finally {
        // optional block and always get executed – exception raised or not
        // close resources or so
}
```

- In Java 7 and later versions, multiple types of Java exceptions can be caught in the same catch block instead of duplicating catch block code as in prior versions. Example:

```
catch (Exception1 | Exception2 ex) {
   // print stack trace or handle the event
   ex.printStackTrace();
}
```

1. **What are the differences between checked exceptions and unchecked/ runtime exceptions?**

- Both checked and unchecked exceptions indicate some event happened rather than the normal flow of the program. The following are the differences, and there has been a controversy where one side of programmers preferred checked, and other side preferred unchecked exceptions.

Checked Exceptions	Unchecked/Runtime Exceptions
User code must catch the exception or rethrow the exception. Otherwise, it is a compilation error.	No to need to catch the exception in the user application.
Custom application exception should	RuntimeException class. subclasses.
Use the checked exceptions when there is a reasonable way to recover by the client code.	When there is no way to handle the raised exception, then better to go with unchecked/runtime exceptions.
Example: FileNotFoundException raised when input file doesn't exist at the given path.	Example: NullPointerException raised when a null object is dereferenced.

1. **How to resolve when there is an exception stacktrace?**

- When there is an exception stack, then first see the stack trace and calls stack. Note the exception message and take the action.
- So, the stack trace/stack dump is really required when filing bugs or debugging the exception issues.

1. **How to raise exceptions?**

- Use the throws statement to pass or throw the exception to the called method instead of handling (try-catch) in the same method.
- Use throw statement to raise a new exception.
- Example:

```
public static String search() throws NullPointerException
   return findEmployeeId(0);
}
public static String search2() throws Exception {
   // actual search for employee
   // if not found, then throw exception
   throw new EmployeeNotFoundException(); //user exception
}
```

1. **How to create a new custom or user defined checked exception?**

- To create a new checked Exception
- Inherit the class from Exception
- To create a new unchecked or Runtime
- Inherit the class from RuntimeException
- Here is the sample checked exception class.

```
public class EmployeeNotFoundException extends Exception{
  public EmployeeNotFoundException() {
    //super();
    super("Employee is not found");
  }
  public EmployeeNotFoundException(String mesg) {
  super(mesg);
  }
}
```

- In the application code, use the below statement to raise the exception.

```
throw new EmployeeNotFoundException("Employee "+empId+" is not found");
```

1. **What is a conditional operator, "?:"?**

- A conditional operator is a short form of "if-else" conditional statement.
- General syntax:
- (condition)? val1: val2
- (first)? second: third
- The first argument should return a boolean value.
- Second and third arguments should never return void.
- Example:

Short form:

```
max = (a>b)? a : b

//The above is equivalent to the below if statement.

if (a > b) {
  max = a;
}
else {
  max = b;
}
```

1. **How and where to use "for-each" statement?**

- The "for-each" statement is a short form of "for" loop.
- It is used when a series (of values), each successive value is to be referenced.
- Typically, used in

i) methods of Arrays and Collections.

ii) Iterable (anything that implements Iterable<E>) class.

- The general syntax for for-each is as below.

```
for (type v : array ) {
//body loop and v is going to have successive objects from the array.
}
```

1. **What is "enum" data type and how to use?**

- The "enum" data type a special data type where name and values are referenced from each other instead of using separate mapping (name=value mapping).
- For example, to use the sizes (Large, Medium, Small) or days (week-days as below), then enum will help to represent as is instead of mapping as 1, 2, 3, etc.
- Usually, enum datatype specified as a separate class than a simple variable defined in the current class.
- Examples:

```
// enum type variables
enum WeekDay { MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, SUNDAY };
enum Size {SMALL, MEDIUM, LARGE};

// To reference the above enum variables
for (WeekDay d: EnumSet.range(WeekDay.THURSDAY, WeekDay.SUNDAY)) {
  System.out.println(d); // prints THURSDAY, FRIDAY, SATURDAY, SUNDAY
}
```

1. **How to write a recursion method? What is the core advantage if it? Where to pay attention?**

- Recursion is nothing but a method calling itself (self-calling).
- The advantage is that some of the problems are in recursion nature and easy to solve with recursion way than non-recursion way.
- Below are the core steps to write a recursion method.
- First, form the "Base case"
- Next, process to get closure to the Base Case
- Last, make a recursive call to pass simple problem back
- Examples of recursion in nature
- Factorial of N
- Fibonacci numbers
- The major attention to pay in recursion is that avoid an infinite loop. The stackoverflow error can happen if the proper way of processing to get closure to the base is not done. So, the proper handling of exit condition is important.
- See the sample java code listings at the end of the book covering the recursion code.

1. **What are the different Java I/O classes?**

- The below are the list of basic I/O classes work with files/console/ de-vices.
- Standard I/O (Input/Output)
- Byte Streams
- Character Streams
- Buffered Streams
- Line I/O
- Data Streams
- Object Streams
- See the sample java code listings at the end of the book covering the Java I/O code.

1. **What are Java Collection and Java Collection frameworks?**

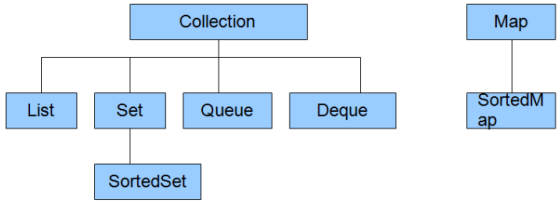
- In simple terms, Collection is a container.
- A collection is an object that groups multiple objects.
- In general, Collection represents a natural group of items.
- The collection has methods that stores, retrieves, manipulates and communicates aggregated data.
- In Collection, the objects are called Elements.
- Collections can be either Ordered or Unordered group of elements.
- The Java Collections Framework is a unification of Interfaces, Imple-mentations and Algorithms that work on a group of objects.

1. **What are the advantages of Java Collections?**

- Some of the key advantages of Java Collections are as below.
- Reduces programming effort
- Increases program speed with quality implementations
- Re-usability

1. **What are the core interfaces in Collection framework?**

- The Collection interface is the top-level interface in the Collections framework.
- Java doesn't directly provide any implementation classes at Collection interface and instead provide the implementations at the sub-interfaces such as List and Set.
- Collection interfaces are generics (type) for compile time check than runtime type check and it can reduce runtime errors.
- public interface Collection<E> extends Iterable<E>{ ...} where E is the type of object contained in the collection.
- Here is the hierarchy of java collection interfaces. Please note that the Map is not in the same class hierarchy as Collection.



1.What are the common methods in the Collection (interface)?

- The Collection interface class contain methods ike size(), isEmpty(), contains(), add(), remove(), clear() etc., for for basic operations; bulk operations like containsAll(), addAll(), retainAll(); and toArray() operations.

```
public interface Collection<E> extends Iterable<E> {
    // Basic operations
    int size(); // to find the number of elements in the collection.
    boolean isEmpty(); // to check if the current collection is empty or not. Return
    true if it is empty.
    boolean contains(Object element);
    // optional
    boolean add(E element);
    // optional
    boolean remove(Object element);
    Iterator<E> iterator();

    // Bulk operations (working with other collections)
    boolean containsAll(Collection<?> c);
    // optional
    boolean addAll(Collection<? extends E> c);
    // optional
    boolean removeAll(Collection<?> c);
    // optional
    boolean retainAll(Collection<?> c);
    // optional
    void clear();

    // Array Operations to convert the collection to an array.
    Object[] toArray();
    <T> T[] toArray(T[] a);
}
```

1. **What is Set interface, and its implementation classes?**

- A Set is a collection that can't contain any duplicates.
- It is an abstraction of a mathematical set.
- The Set interface has the same methods/operations as the Collection except that it is constrained to remove duplicates.
- The Set is used in place of old Vector class (now it is a legacy class after collections framework).
- Set interface extends Collection.
- public interface Set<E> extends Collection<E> { ...}
- The following are some of the Set real world examples.
- Cards in Poker
- Courses in Student's schedule
- Processes running on a machine
- The list of Set implementation classes from Java Collections Framework is as below.
- HashSet (best performing): Elements are stored internally like a hashtable. There is no elements order guarantee in this implementation.
- TreeSet: Elements are stored using the red-black tree and the elements are ordered based on values.
- LinkedHashSet: Elements are similar to HashSet except that it preserves the order in which the elements were added.
- Examples with bulk operations
- Set<Type> union = new HashSet<Type>(s1);
- Set<Type> intersection = new HashSet<Type>(s1);
- Set<Type> difference = new HashSet<Type>(s1);

1.What is List interface, and its implementation classes?

- A List is an ordered collection or sequence of elements container. The insertion order is preserved/maintained.
- The List interface adds more methods on top of Collection for the below functionality.
- Positional access
- Search
- Iterator
- Range view
- The below are the implementation classes
- ArrayList (best performing)
- LinkedList
- Two list objects are equal when same elements are in same order.
- ListIterator extends Iterator to provide forward or backward indexing of elements.
- Below is the List interface and methods include get(), set(), remove(), indexOf(), lastIndexOf() etc.,

```
public interface List<E> extends Collection<E> {

// Positional access
E get(int index);
// optional
E set(int index, E element);
// optional
void add(int index, E element);
// optional
E remove(int index);

// Bulk (optional)
boolean addAll(int index, Collection<? extends E> c);

// Search
int indexOf(Object o);
```

1. **What are different algorithmic methods available in the List collection?**

- Below are the List algorithms.
- Sort (using merge sort/stable sort, and no re-ordering of elements is required)
- Shuffle
- Reverse
- Rotates
- Swap
- ReplaceAll
- Fill
- Copy
- IndexOfSubList
- lastIndexOfSubList

1.Could you explain the Map collection?

- Map collection maps the keys to values. Mainly, it holds key-value pairs.
- The Map can't contain duplicate keys.
- The Map interface is not in the same Collection interface hierarchy.
- It is similar to the old HashTable class.
- The below are implementation classes for Map.
- HashMap
- TreeMap
- LinkedListMap
- Map interface operations are put(), get(), remove() etc. as below.

```
public interface Map<K,V> {
    // Basic operations
    V put(K key, V value);
    V get(Object key);
    V remove(Object key);
    boolean containsKey(Object key);
    boolean containsValue(Object value);
    int size();
    boolean isEmpty();

    // Bulk operations
    void putAll(Map<? extends K, ? extends V> m);
    void clear();

    // Collection Views
    public Set<K> keySet();
    public Collection<V> values();
    public Set<Map.Entry<K,V>> entrySet();

    // Interface for entrySet elements
    public interface Entry {
        K getKey();
        V getValue();
```

1. **How should you write the classes (the rules to be followed) in order to use custom Java beans as elements in a collection?**

- At a minimum, the Java bean (say Employee) should override the below two methods to work as expected otherwise unexpected behavior can happen. This mainly matches the stored objects in the collection.
- equals() : should have the logic to compare the objects.
- hashCode() : override this method whenever equals is overridden. A unique value to be returned. Also, note that don't change the return value once the object is stored in the collection. Example: don't use current timestamp method as the return value.
- Also, implement Comparable to do the automatic sorting like in TreeSet (or) use Comparator class if sorting algorithm is going to be used in that collection.
- See the below code snippet to show required methods to implement in collection element class.

```
/**
* override equals() to say that two employees are equal only when
* employee names and ids are equal.
*/
public boolean equals(Object o) {
        if (o==null) { return false; }
        if (o==this) { return true; }
        if (! (o instanceof Employee)) {
                return false;
        }
        Employee e = (Employee)o;
        if ( (e.getName().equals(this.name))&&(e.getId()==this.id)){
                return true;
        } else {
                return false;
        }
}

/**
* Required whenever equals overridden
* @see java.lang.Object#hashCode()
*/
public int hashCode() {
        return id;
}

/**
* Below method is required for ordering/sorting of elements.
*/
@Override
public int compareTo(Employee o) {
        int nameCompare = name.compareToIgnoreCase(o.getName());
        return (nameCompare!=0 ?nameCompare: new
        Integer(id).compareTo(o.getId()));
}
```

1. **If there is a requirement where a million records should be read from DB and would like to make them unique and sorting. Which collection data structure is appropriate?**

- TreeSet is more appropriate because Set will remove duplicates and Tree will make sure sorting automatically.

1. **If there is a requirement where some kind of paired objects to be stored, then which collection data structure is appropriate?**

- HashMap more appropriate because it is efficient to store key-value pairs.

1. **How to choose a specific Collection data structure? What are different performance characteristics (Big O-complexity)?**

- The following summary table gives each collection and basic purpose. You can check the requirement and pick the appropriate data structure.

(add/remove/contains)

LinkedHashSet

No duplicates + Preserving the insertion order

O(1) constant time for basic operations (add/remove/contains)

Map

HashMap

Key- Value pair + No specific order for keys

O(1) constant time for basic operations (get and put)

1. **What is JDBC? Could you please explain about different JDBC driver types.**

- JDBC stands for Java DataBase Connectivity. It is the platform independent API used to work with different vendor databases from Java code.
- The current API spec version is 4.0 and is called JDBC4.0. The Java packages for JDBC API are java.sql and javax.sql. The API has been included in both Java SE and Java EE editions.
- To work with any database, there is a need to use vendor drivers implemented by installing client or adding to library path or putting the jar in the classpath.
- There are 4 types of JDBC drivers (Reference link)
- Type 1 driver: Drivers that have been implemented the JDBC API by mapping to another database access API such as Microsoft's ODBC (Open DataBase Connectivity used for programming with MSAccess, SQLServer DB). This type of driver is also called JDBC-ODBC bridge. Portability is limited with this driver as this not written in pure Java code. Below is the execution path.
- Type 2 driver: Also called Native drivers. Drivers that have been implemented partly in Java and partly in native code to interact with the native client(platform specific libraries) in order to connect to the DB. Portability is limited with this driver as it is not written in pure java.
- Type 3 driver: Also called Middleware Data Source (written in pure Java). Drivers that have been implemented the JDBC API with Java that communicates with Middleware (with the data source in application servers such as WebLogic/GlassFish) in order to connect to the database. Middleware maintains the data sources with DB connection pool and efficient reuse of expensive DB connections. The JDBC code will have logical data source name, and actual DB configuration will be mapped in the application server.
- Type 4 driver: Also called Thin drivers (written in pure Java).

Drivers that implement the network protocol of specific data source. The client directly connects to the data client. These drivers are simple jar files to put in the classpath and no installation/ setup needed.

1.Could you please explain end-to-end steps in a typical JDBC program?

- The following are the key steps along with code snippets in a typical JDBC program.

(1) Register/load the driver (similar to the code snippet as below).

```
DriverManager.registerDriver (new oracle.jdbc.OracleDriver());
(or)
Class.forName("oracle.jdbc.OracleDriver");
```

Alternatively, you can register the driver when launching the JavaVM (while execute your application), as follows:

```
java -Djdbc.drivers = oracle.jdbc.OracleDriver <ClassName>;
```

(2) Get DB Connection

i) DriverManager approach like below code snippet.

```
Connection
conn=DriverManager.getConnection(con-
nectstring_or_jdbcurl,uname,passwd);

Example: Connection conn = DriverManager.getConnection(
"jdbc:oracle:thin:@<host>:<port>:<sid>","<user>","<passwd>");
```

ii) DataSource approach like below code snippet.

```
OracleDataSource ods = new OracleDataSource();
// set necessary properties (JDBC URL, etc)
java.util.Properties prop = new java.util.Properties();
...
prop.put(....)
ods.setURL(url);

// get the connection
Connection conn = ods.getConnection();
```

iii) JNDI approach like below code snippet.

```
javax.sql.DataSource ds = (javax.sql.DataSource) ctx.lookup("java:comp/
env/my-data-source");
java.sql.Connection conn = ds.getConnection();
```

(3) Create Statement/PreparedStatement/CallableStatement

```
Statement statement = conn.createStatement();
PreparedStatement preparedStatement = conn.prepareStatement();
CallableStatement cs = conn.prepareCall("{call <procedure>}");
```

(4) Execute Query/Update SQL

- There are 3 different statement execution methods

i) executeQuery(String sqlstmt)

```
ResultSet rs = stmt.executeQuery("SELECT empno, ename FROM
emp");
```

ii) executeUpdate(String sqlstmt)

```
int rowcount = stmt.executeUpdate("CREATE TABLE table1
(numcol NUMBER(5,2), strcol VARCHAR2(30)");
int rowcount = stmt.executeUpdate("DELETE FROM emp
WHERE empno = 2354");
```

iii) execute(String sqlstmt)

```
boolean result = stmt.execute(SQLstatement);
if (result) {// was a query - process results
ResultSet r = stmt.getResultSet(); ...
} else { // was an update or DDL - process result
int count = stmt.getUpdateCount();
}
```

(5) Process Results

- The ResultSet is the object that is returned from executing the Statement and maintains a cursor pointing to the current row.
- The next() method in the ResultSet allows iterating through the rows that made the result set, and the getXXX() method is for retrieving the values of the columns where XXX indicate various data types or column types.
- Note that without first having rs.next(), the results can't be retrieved as the cursor doesn't point to the available row.
- Typical ResultSet process loop is as below.

```
while (rs.next() { XXXX obj=rs.getXXXX(<column name>) }
while (rs.next()) {
String ename = rs.getString("ENAME");
int empno = rs.getInt("EMPNO");
...
}
```

(6) Close resources

- Free up the resources with the below steps

i) Close the resultset object

ii) Close statement object

iii) Close connection objects

Use these statements the finally block (in try-catch-finally) so that always get executed even when there are partially allocated resources due to exceptions.

```
rs.close();
stmt.close();
conn.close();
```

1. **What is JDBC URL or Connect string? Can you give examples?**

- JDBC url is a connection string that is used in the Type4 thin driver. The following is the syntax.
- jdbc:<subprotocol>:<subname>//<host>:<port>/[database-name][,-user=xxx,pass-word=zzz]

- Examples:
- Oracle
- MySQL
- SQLServer

1. **How to create a datasource in WebLogic?**

- In WebLogic, create DataSource using Admin Console by navigating to the below.
- Goto http://localhost:7001/console → Services->JDBC->Data-Sources->New
- Then add jdbc code in your program to do the DataSource lookup:

```
Properties props = new Properties();
props.put(Context.INITIAL_CONTEXT_FACTORY,"weblogic.jndi.WLInitialContextFactory");
props.put(Context.PROVIDER_URL,"t3://localhost:7001");
ctx = new InitialContext(props);
ds = (DataSource) ctx.lookup("jdbc/employeedb");
```

← End: Skill#9. Java & JDBC ←

Good!

Keep going and never give up!!

Please re-read again for more clarity and feel free to contact for help!!!

→ Start: Skill#10. ANT →