1)Distinguish between Abstract Class and Interface.

Abstract class	Interface
1) Abstract class can **have abstract and non-abstract** methods.	Interface can have **only abstract** methods. Since Java 8, it can have **default and static methods** also.
2) Abstract class **doesn't support multiple inheritance**.	Interface **supports multiple inheritance**.
3) Abstract class **can have final, non-final, static and non-static variables**.	Interface has **only static and final variables**.
4) Abstract class **can provide the implementation of interface**.	Interface **can't provide the implementation of abstract class**.
5) The **abstract keyword** is used to declare abstract class.	The **interface keyword** is used to declare interface.
6) An **abstract class** can extend another Java class and implement multiple Java interfaces.	An **interface** can extend another Java interface only.
7) An **abstract class** can be extended using keyword "extends".	An **interface** can be implemented using keyword "implements".
8) A Java **abstract class** can have class members like private, protected, etc.	Members of a Java interface are public by default.

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

2. What is a package?

- A **java package** is a group of similar types of classes, interfaces and sub-packages.
- packages act as **containers** for classes.
- Packages are used for:

- Preventing naming conflicts.
- Categorizing the classes and interfaces so that they can be easily maintained.
- Providing controlled access.

- Java Packages are divided into two categories:

1. Built-in Packages (packages from the Java API)
2. User-defined Packages (created by user)

Creating packages

- Java supports a keyword called “package” for creating user-defined packages.
- The package statement must be the first statement in a Java source file followed by one or more interfaces and classes.

Accessing a package

- There are three ways of accessing the classes stored in packages:

1. Using import package.*;
   import mypackage.*;
2. Using import package.classname;
   import mypackage.A;
3. Using fully qualified name.
   mypackage.A

- If you import a package, all the classes and interfaces of that package will be imported excluding the classes and interfaces of the sub packages.

JAVA PROGRAM FOR CREATING AND ACCESSING PACKAGE

```
1. //save by A.java
2. package pack;
3. public class A{
4.   public void msg()
5. {
6. System.out.println("Hello");
7. }
8. }
```

```
1. //save by B.java
2. package mypack;
3. import pack.*;
4.
5. class B{
6.   public static void main(String args[]){
7.    A obj = new A();
8.    obj.msg();
9.   }
10.}
```

```
Output:Hello
```

2. What are User - Defined Packages. Explain with Example

User-defined Packages

User-defined packages are those packages that are designed or created by the developer to

categorize classes and packages. They are much similar to the built-in that java offers. It can be ;we use built-in packages. But If we omit the package statement, the class names are put into the default package, which has no name.
To create a package, we're supposed to use the package keyword.

Syntax:

```
package package-name;
```

EXAMPLE:

Creating user defined package

```
Package pack1;

Public void demos

{

Public void swap(int a,int b)

{

int temp;

System.out.println("before:"+a+""+b);

temp=a;

a=b;

b=temp;

System.out.println("After:"+a+""+b);

}
```

}

```
import pack1.Demos;
class Swap
{
Public static void main(String args[])
{
Demos=new Demos();
d.swap(23,45);
}
}
```

4)What is the use of Abstract Class?

a.An abstract class is a good choice if we are using the inheritance concept since it provides a common base class implementation to derived classes.

b.An abstract class is also good if we want to declare non-public members. In an interface, all methods must be public.

c.If we want to add new methods in the future, then an abstract class is a better choice. Because if we add new methods to an interface, then all of the classes that already implemented that

interface will have to be changed to implement the new methods.

d.If we want to create multiple versions of our component, create an abstract class. Abstract classes provide a simple and easy way to version our components. By updating the base class, all inheriting classes are automatically updated with the change. Interfaces, on the other hand, cannot be changed once created. If a new version of an interface is required, we must create a whole new interface.

e.Abstract classes have the advantage of allowing better forward compatibility. Once clients use an interface, we cannot change it; if they use an abstract class, we can still add behavior without breaking the existing code.

f.If we want to provide common, implemented functionality among all implementations of our component, use an abstract class. Abstract classes allow us to partially implement our class, whereas interfaces contain no implementation for any members.

5. **Explain how multiple inheritance is achieved**
   - Multiple inheritance is not supported by Java using classes, handling the complexity that causes due to multiple inheritances is very

complex. It creates problems during various operations like casting, constructor chaining, etc, and the above all reason is that there are very few scenarios on which we actually need multiple inheritances, so better to omit it for keeping things simple and straightforward.

The above problems can be handled by Default Methods and Interfaces

- Java 8 supports default methods where interfaces can provide a default implementation of methods. And a class can implement two or more interfaces. In case both the implemented interfaces contain default methods with the same method signature, the implementing class should explicitly specify which default method is to be used, or it should override the default method.

Java program to demonstrate Multiple Inheritance

```
Interface PI1 {
    default void show()
    {


        System.out.println("Default PI1");
    }
}

interface PI2 {


    default void show()
    {
```

```
        System.out.println("Default PI2");
    }
}

class TestClass implements PI1, PI2 {


    public void show()
    {

        PI1.super.show();


        PI2.super.show();
    }


    public static void main(String args[])
    {


        TestClass d = new TestClass();
        d.show();
    }
}
```

6. What are wrapper classes? Explain

Wrapper classes

- The wrapper class in Java provides the mechanism to convert primitive data type into object and object into primitive datatype.
- The eight classes of the *java.lang* package are known as wrapper classes in Java.
- The automatic conversion of primitive data type into its corresponding wrapper class is known as **autoboxing.**

- The automatic conversion of wrapper type into its corresponding primitive type is known as **unboxing.** It is the reverse process of autoboxing.

Primitive Type	Wrapper class
boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double

EXAMPLE:

```
Class Test1
{
Public static void main(String args[])
{
int a=5;
double b=5.65;
Integer obj1=Integer.valueof(a);
```

Double obj2=Double.valueof(b);
int i=obj1.intvalue();
double j=obj2.doublevalue();
System.out.println(i+””+j);
}
}

7. what are user defined exceptions? Explain

User defined exceptions

- User defined exceptions in java are also known as Custom exceptions.
- We can create user defined exceptions by extending any exception class.
- Syntax:
- class NewExceptionClass extends Exception
  {
  .....
  }

```
class MyException extends Exception {
    public MyException(String s)
    {

        super(s);
    }
}


public class Main {

    public static void main(String args[])
    {
        try {
            throw new MyException("apple");
```

```
        }
        catch (MyException ex) {
            System.out.println("Caught");
            System.out.println(ex.getMessage());
        }
    }
}
```

o/p:-caught

apple

9. what is the use of Inter Thread Communication? Explain

Cooperation (Inter-thread communication) is a mechanism in which a thread is paused running in its critical section and another thread is allowed to enter (or lock) in the same critical section to be executed.It is implemented by following methods of **Object class**:

- wait()
- notify()
- notifyAll()

1) wait() method

The wait() method causes current thread to release the lock and wait until either another thread invokes the notify() method or the notifyAll() method for this object, or a specified amount of time has elapsed.

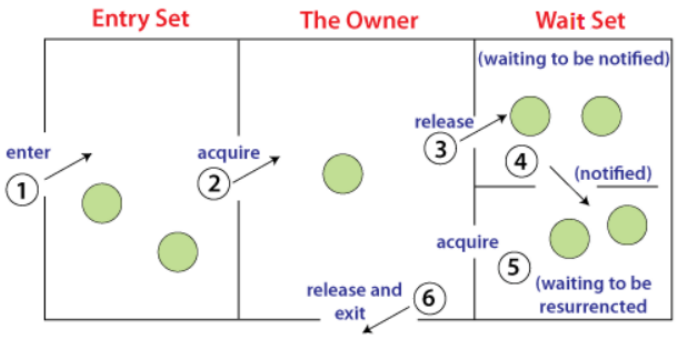
2) notify() method

The notify() method wakes up a single thread that is waiting on this object's monitor. If any threads are waiting on this object, one of them is chosen to be awakened. The choice is arbitrary and occurs at the discretion of the implementation.

3) notifyAll() method

Wakes up all threads that are waiting on this object's monitor.

Understanding the process of inter-thread communication



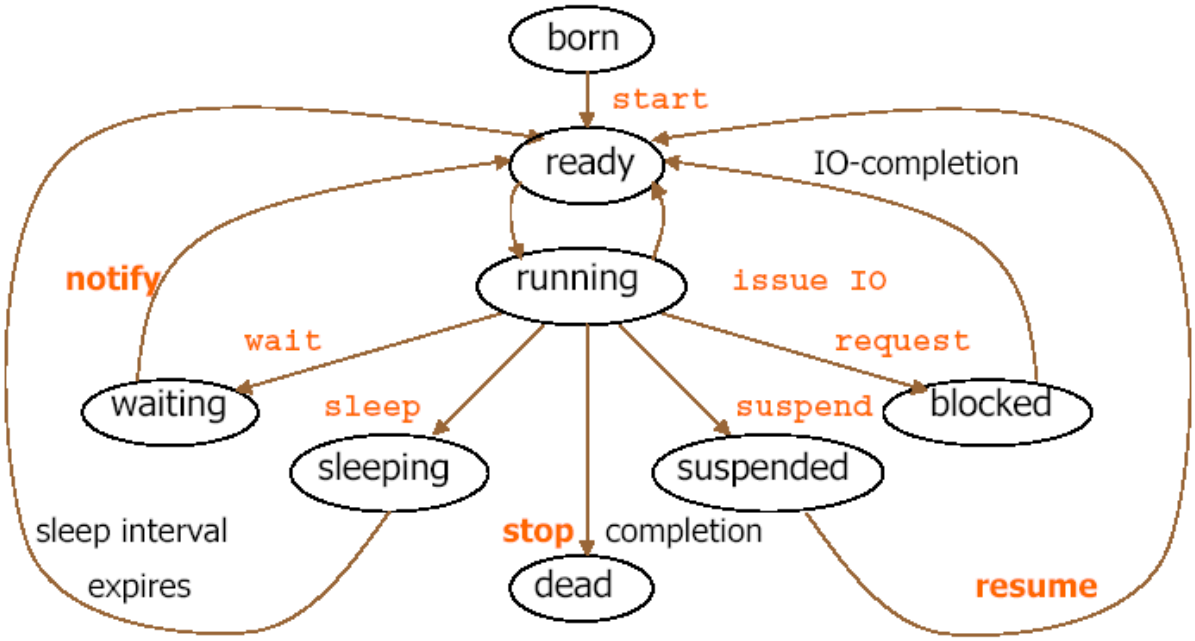
The point to point explanation of the above diagram is as follows:

1. Threads enter to acquire lock.
2. Lock is acquired by on thread.
3. Now thread goes to waiting state if you call wait() method on the object. Otherwise it releases the lock and exits.
4. If you call notify() or notifyAll() method, thread moves to the notified state (runnable state).
5. Now thread is available to acquire lock.
6. After completion of the task, thread releases the lock and exits the monitor state of the object.

9. Explain Thread Life Cycle.

Life cycle of a Thread

A thread goes through various stages in its life cycle.
A thread can be in one of the following states:



- **New/Born state**

-When a new thread is created, it is in the new state.

-When start called, it enters ready state

- **Ready state**

-when a thread is ready for execution it enters ready state.

-Highest-priority ready thread enters running state.

- **Running state**

-System assigns processor to thread (thread begins execution)

- **Dead state**

-When run method completes or terminates, enters dead state.

Once terminated can not be resumed.

- **Blocked state**

-A thread is in the *BLOCKED* state when it's currently not eligible to run.

-Blocked thread cannot use processor, even if available

-Common reason for blocked state - waiting on I/O request

- **Sleeping state**

Entered when sleep method is called

Cannot use processor

Enters ready state after sleep time expires

- **Waiting state**

A thread is in waiting state when it's waiting for some other thread to perform a particular action.

One waiting thread becomes ready when object calls notify.

- **Suspended state**

A thread is in suspended state if its execution stopped when an event occurs.

Can be resumed allowing it to pick up where it left off.

10.Explain the concept of thread synchronization.

Thread Synchronization

- When two or more threads need access to a shared resource sometimes we may get unreliable results. so they need some way to ensure that the resource will be used by only one thread at a time.
- The process by which this is achieved is called **synchronization.**
- The **synchronization** method is used in java to achieve synchronization among threads.
- If one thread is executing synchronized method then all other threads that call this method have to wait until the execution of first thread on this method completes.
- When a thread invokes a synchronized method, it automatically acquires the lock for that object and releases it when the thread completes its task.

11. what is multithreading

Multithreading in Java is a process of executing multiple threads simultaneously.

Java Multithreading is mostly used in games, animation, etc.

Advantages of Java Multithreading

1) It **doesn't block the user** because threads are independent and you can perform multiple operations at the same time.

2) You **can perform many operations together, so it saves time**.

3) Threads are **independent**, so it doesn't affect other threads if an exception occurs in a single thread

```
class MultithreadingDemo extends Thread {
    public void run()
    {
        try {

            System.out.println(
                "Thread " + Thread.currentThread().getId()
                + " is running");
        }
        catch (Exception e) {

            System.out.println("Exception is caught");
        }
    }
}

public class Multithread {
    public static void main(String[] args)
    {
        int n = 8;
        for (int i = 0; i < n; i++) {
            MultithreadingDemo object
                = new MultithreadingDemo();
            object.start();
        }
    }
}
```

Output

```
Thread 15 is running
Thread 14 is running
Thread 16 is running
Thread 12 is running
Thread 11 is running
Thread 13 is running
Thread 18 is running
Thread 17 is running
```

12.explain various access specifiers associated with packages.

☐In java we have four access modifiers:

1. default
2. private
3. protected
4. public

Default access modifier

When we do not mention any access modifier, it is called default access modifier. The scope of this modifier is limited to the package only.

Private access modifier

☐The scope of private modifier is limited to the class only.

☐Private Data members and methods are only accessible within the class

Protected Access Modifier

☐Protected data member and method are only accessible by the classes of the same package and the subclasses present in any package

Public access modifier

☐The members, methods and classes that are declared public can be accessed from anywhere. This modifier doesn't put any restriction on the access.

13. explain various blocks associated with exceptions.

java provides five keywords that are used to handle the exception. The following table describes each.

- **try**

-All statements that are likely to raise an exception are kept in try block.

- **catch**

-This block takes corrective steps when an exception is thrown.

-A try block can follow multiple catch blocks.

- **throw**

-It is used to throw an exception.

- **throws**

-It is used to specify the possible exceptions that are caused by a method. But not handled by the method.

- **finally**

-This block is followed by try-catch block.

-This block is executed irrespective of whether exception occurs or not.

14. what are checked and unchecked exceptions? Explain.

- Checked exceptions are the exceptions that are checked at compile time.
- If some code within a method throws a checked exception, then the method must either handle the exception or it must specify the exception using throws keyword. otherwise the program will give a compilation error.
- All the Exceptions which are direct sub classes of Exception but not inherit RuntimeException are Checked Exceptions.

FileNotFoundException
ClassNotFoundException
InstantiationException
InterruptedException
NoSuchMethodException
NoSuchFieldException

- Unchecked exceptions are not checked at **compile time.**
- Exceptions under *Error* and *RuntimeException* classes are unchecked exceptions.

ArithmeticException

ArrayIndexOutOfBoundsException

NullPointerException

15. What is a deamon thread explain.

Daemon thread

- Thread that runs in the background and come in to execution when the processor is idle.
- Daemon thread is a low priority thread that runs in background to perform tasks such as garbage collection etc.
- We can set a thread to be a Daemon one,by calling **setDaemon(true).**
- A thread must be set to Daemon ,if wanted before it starts(i.e start() is called).
- To determine if a thread is a daemon thread, use the accessor method **isDaemon().**

```
public class DaemonThread extends Thread
{
    public DaemonThread(String name){
        super(name);
    }

    public void run()
    {

        if(Thread.currentThread().isDaemon())
        {
            System.out.println(getName() + " is Daemon thread");
        }

```

```
        else
        {
            System.out.println(getName() + " is User thread");
        }
    }

    public static void main(String[] args)
    {

        DaemonThread t1 = new DaemonThread("t1");
        DaemonThread t2 = new DaemonThread("t2");
        DaemonThread t3 = new DaemonThread("t3");


        t1.setDaemon(true);

        t1.start();
        t2.start();


        t3.setDaemon(true);
        t3.start();
    }
}
```

16. what are thread priorities. how do we set the priorities to threads

- In Java, each thread is assigned priority, which effects the order in which it is scheduled for running.
- The threads have same default priority (NORM_PRIORITY) and they are served using FCFS policy.
- These priorities are defined as final variables within Thread class.
- Java allows users to change priority:

ThreadName.setPriority(int Number)

MIN_PRIORITY = 1

NORM_PRIORITY=5

MAX_PRIORITY=10

```
import java.lang.*;

class ABC extends Thread {


    public void run()
    {

        System.out.println("Inside run method");
    }


    public static void main(String[] args)
    {
        Thread.currentThread().setPriority(6);


        System.out.println(
            "main thread priority : "
            + Thread.currentThread().getPriority());

        ABC t1 = new ABC();
        System.out.println("t1 thread priority : "
                           + t1.getPriority());
    }
}
```

Output

```
main thread priority : 6

t1 thread priority : 6
```

17. Explain how to establish a connection to database using java.

There are 5 steps to connect any java application with the database using JDBC. These steps are as follows:

- Register the Driver class
- Create connection
- Create statement
- Execute queries
- Close connection

1) Register the driver class

The **forName()** method of Class class is used to register the driver class. This method is used to dynamically load the driver class.

```
public static void forName(String className)throws ClassNotFoundException
```

2) Create the connection object

The **getConnection()** method of DriverManager class is used to establish connection with the database.

Syntax of getConnection() method

```
public static Connection getConnection(String url)throws SQLException
```

3) Create the Statement object

The createStatement() method of Connection interface is used to create statement. The object of statement is responsible to execute queries with the database.

Syntax of createStatement() method

```
public Statement createStatement()throws SQLException
```

4) Execute the query

The executeQuery() method of Statement interface is used to execute queries to the database. This method returns the object of ResultSet that can be used to get all the records of a table.

Syntax of executeQuery() method

```
public ResultSet executeQuery(String sql)throws SQLException
```

5) Close the connection object

By closing connection object statement and ResultSet will be closed automatically. The close() method of Connection interface is used to close the connection.

Syntax of close() method

```
public void close()throws SQLException
```

18. explain how to send an sql query to mysql using java

19. Explain how to store the result of a query in java

20. Explain the architecture of JDBC

- The JDBC API Supports two models for database access:

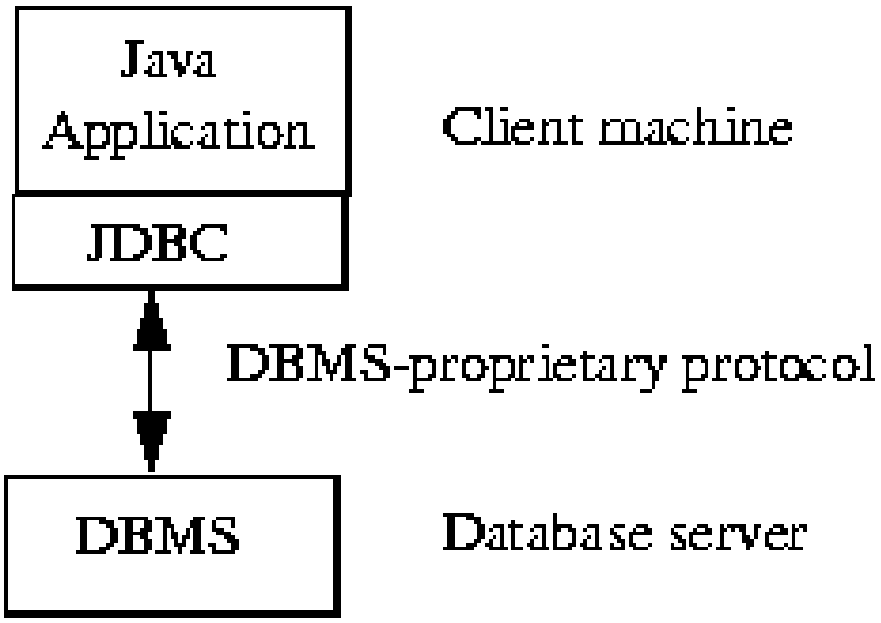
  1.Two tier architecture

  2.Three tier architecture

Two tier architecture

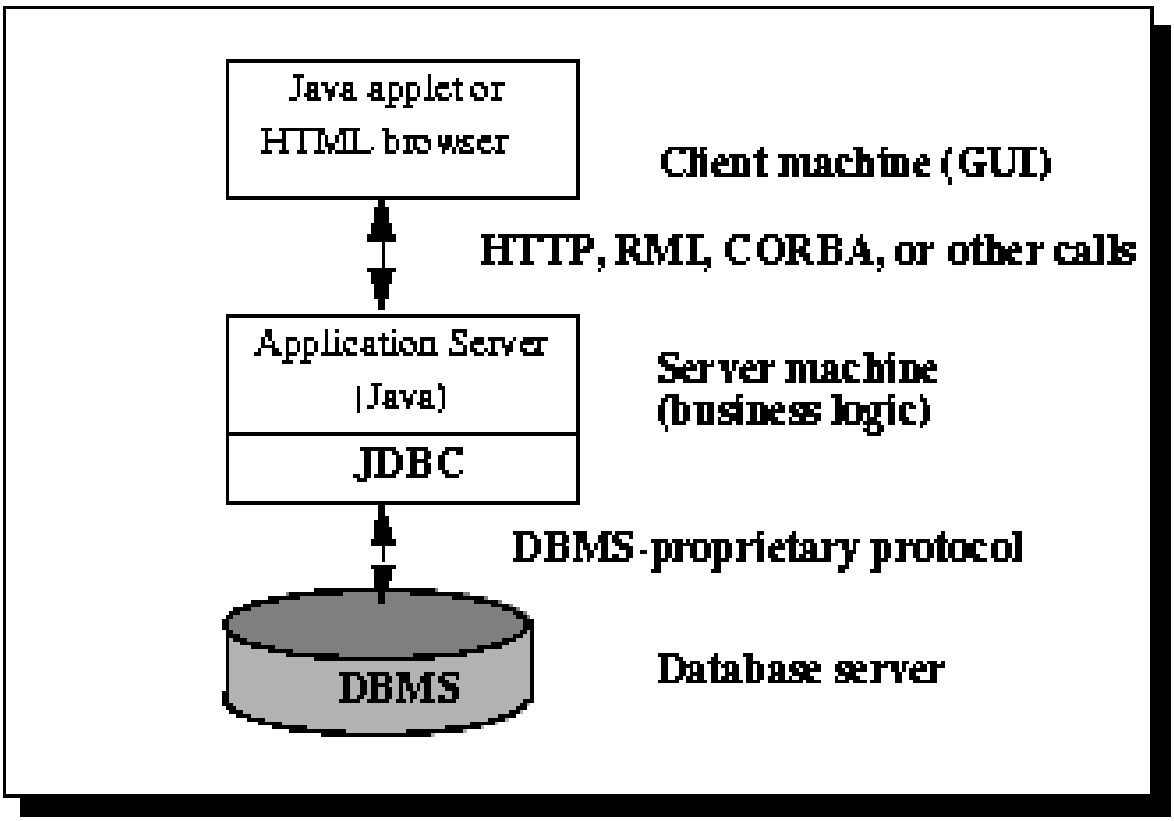
- In this model the java application communicates directly with the data base.
- Both java application and JDBC API are located at client machine and DBMS and database are located at the database server.

Some times both java application and database may reside on the same machine



Three tier architecture

- In this model the user commands are first send to the application server forming the middle tier.
- Application sever containing the JDBC API sends the SQL statements to the database located on the database server.
- The commands are processed and results are sent to the middle tier ,which then sends to the user.
- This model provides better performance and simplifies the deployment of applications.


Java applet or
HTML browser
Client machine (GUI)
HTTP, RMI, CORBA, or other calls
Application Server
(Java)
JDBC
Server machine
(business logic)
DBMS-proprietary protocol
DBMS
Database server