Pending Questions

Q) Static and final difference

Static keyword:

Static keyword is used to do better memory management

You can use static keyword with variable, method, class

If you declare method using static keyword then you don't need to create an object to call that method.

If you declare variable as a static then only once memory is created for that variable in class area and not every time when object is created eg:- in student database you can declare "college name" as static because it is common for all

students.

Final keyword:
Final keyword is used to restrict the user. It means if you declare variable as final then you cannot change its value.
If you declare method as final then you cannot override that method.
If you make any class as final, you cannot extend it.

Q) Java is platform independent and architecture Neutral - Explain / Explain how java achieves platform independence using Bytecodes

Java's compiler generates an architecture-neutral object file format, which makes the compiled code to be executable on many processors, with the presence of Java runtime system Hence Java is architecture neutral.
How java is platform independent just because of Bytecode

(i.e Bytecode is the machine understandable code of the JVM (Java Virtual Machine))
By using Bytecode execution java proves it is a platform independent language.
Here is the process of java bytecode execution:
sample.java → javac(sample.class) → JVM(sample.obj) → display final output.
First source code is used by java compiler and converted that code in .class file and the class file code is in byte code form and that class file is used by JVM and again to convert into object file and then after display final output on the screen.
Java is platform indepenent language (in simple terms we can run (.class file) on any platform. Behavior will be same (on MAC, LINUX and WINDOWS)).
Java compiler converts source code to bytecode and bytecode is not designed for a specific operating system understandable.
JVM reads bytecode and translate

byte code to specific operating system understandable form. Bytecode are also known as JVM understandable code or intermediate code.
Java communicates with JVM and JVM communicates with byte code.
If Bytecode exists then JVM will read and if not exist then JVM will raise exception.

Q. Difference between method overloading and method overriding in java

Method Overloading	Method Overriding
Method overloading is used to increase the readability of the program.	Method overriding is used to provide the specific implementation of the method that is already provided by its super class.
Method overloading is performed within class.	Method overriding occurs in two classes that have IS-A relationship.

In case of method overloading, parameter must be different.	In case of method overriding, parameter must be same.
Method overloading is the example of compile time polymorphism	Method overriding is the example of runtime polymorphism.
In java, method overloading can't be performed by changing return type of the method only. Return type can be same or different in method overloading. But you must have to change the parameter.	Return type must be same or covariant in method overriding.

Method Overloading example

```
class OverloadingExample {
static int add(int a, int b){
  return a+b;}
static int add(int a, int b, int c)
{return a+b+c;}}
```

Method Overriding Example

```
class Animal{
void eat(){
System.out.println("eating...");}
class Dog extends Animal{
void eat(){
System.out.println("eating bread...
");}}
}
```

Q) Program to demonstrate thread lifecycle

```
package com.ciklearn.thread;
public class ThreadDemo extends
Thread{
  public void run(){
    System.out.println("Thread is
running!!");
  }
public static void main(String[]
args){
  ThreadDemo t1 = new ThreadDemo
    ();
  ThreadDemo t2 = new ThreadDemo
    ();
System.out.println("T1 => "+
  t1.getState());
System.out.println("T2 => "+
  t2.getState());
```

```
t1.start();
System.out.println("T1 ==> "+t1.getS-
tata());
System.out.println("T2 ==> "+t2.getStatu
());
t2.start();
System.out.println("T1 ==> "+t1.getS-
tata());
System.out.println("T2 ==> "+t2.getSt-
ata());
  }
}
```

Output:

```
T1 ==> NEW
T2 ==> NEW
T1 ==> RUNNABLE
T2 ==> NEW
T1 ==> RUNNABLE
T2 ==> RUNNABLE
Thread is running!!
Thread is running!!
```

Q) Write a servlet program to count number of page visits using cookie.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class VisitServlet extends HttpS-
```

```
servlet {
static int i=1;
public void doGet(HttpServletRequest
request, HttpServletResponse response)
throws IOException, ServletException
{
response.setContentType("text/html");
PrintWriter out = response.getWriter();
String k = String.valueOf(i);
Cookie c = new Cookie("visit",k);
response.addCookie(c);
int j = Integer.parseInt(c.getValue
());
if(j==1){
 out.println("Welcome");
 }
else{
 out.println("You visited "+i+"
times");
     } i++;
 }
}
```

Web.xml file (servlet entry)

```
<?xml version="1.0" encoding="
ISO-8859-1"?>
<web-app>
<servlet>
<servlet-name>VisitServlet</
```

```
servlet-name>
<servlet-class>VisitServlet</servlet-
class>
</servlet>
<servlet-mapping>
<servlet-name>VisitServlet</servlet-
name>
<url-pattern>/servlet/VisitServlet
</url-pattern>
</servlet-mapping>
</web-app>
```