Q1

The index is incremented incorrectly resulting in the arrayIndexOutOfBounds error. To fix this, the index needs to be incremented after assigning the args[index] value to the list In the while loop.

**while ((index < args.length) && (index < 3)) {**

**list[index] = args[index];**

**index++;**

Q2

The class generates a compiler error because it is attempting to make a static call to a non-static variable.

To fix the problem, the variable needs to be declared static.

public class Demo {

public static String my\_member\_variable = "somedata";

public static void main(String[] args) {

// **TODO** Auto-generated method stub

System.out.println("This generates a compiler error" +

my\_member\_variable );

}

}

Q3

Java stack information for the threads listed above:

===================================================

"t3":

at SyncThread.run(Demo.java:37)

- waiting to lock <0x000000076b5679b8> (a java.lang.Object)

- locked <0x000000076b5679d8> (a java.lang.Object)

at java.lang.Thread.run(Thread.java:748)

"t1":

at SyncThread.run(Demo.java:37)

- waiting to lock <0x000000076b5679c8> (a java.lang.Object)

- locked <0x000000076b5679b8> (a java.lang.Object)

at java.lang.Thread.run(Thread.java:748)

"t2":

at SyncThread.run(Demo.java:37)

- waiting to lock <0x000000076b5679d8> (a java.lang.Object)

- locked <0x000000076b5679c8> (a java.lang.Object)

at java.lang.Thread.run(Thread.java:748)

Found 1 deadlock.

The fix(Updated run method)

public void run() {

String name = Thread.currentThread().getName();

System.out.println(name + " acquiring lock on " + obj1);

synchronized (obj1) {

System.out.println(name + " acquired lock on " + obj1);

work();

}

System.out.println(name + " released lock on " + obj1);

System.out.println(name + " acquiring lock on " + obj2);

synchronized (obj2) {

System.out.println(name + " acquired lock on " + obj2);

work();

}

System.out.println(name + " released lock on " + obj2);

System.out.println(name + " finished execution.");

}

Q4

This problem is caused due to MySQL server timing-out database connections faster than the Hibernate connection pool can notice. This exhausts the Hibernate connection pool as it will keep its connections open while the MySQL side of the connections is already closed.

Resolution:

Hibernate is shipped with a connection test interval default of 10 minutes.

MySQL has its [wait\_timeout](http://dev.mysql.com/doc/refman/5.0/en/server-system-variables.html" \l "sysvar_wait_timeout) variable default value set to 28800 seconds (8 hours).

Therefore, if both sides of the connection still keep the defaults, the problem will never happen, as MySQL will never timeout a connection before Hibernate does it.

In the hibernate.properties file, add the following line and define the number of minutes for the connection timeout:

db.pool.idle.testInterval=1

Q5

The issue is caused by ORB object keeping a thread waiting to connect to a peer that is no longer available. Repeated occurrences of this situation causes threads to needlessly remain occupied, causing the thread pool to exhaust.

Resolution:

While a Singleton ORB object might resolve it, Applying multithreading will resolve the issue of connections building up.