1. “Deadlock is a special type of error that needs to be avoided in multitasking”. Elaborate.

Deadlock is a situation in multitasking where two or more threads are permanently blocked because each thread is holding a resource and waiting for another resource held by another thread.

In a deadlock situation, none of the threads can proceed, resulting in system inefficiency and application freeze. Deadlocks are difficult to detect and debug because no exception is thrown.

Example scenario:
Thread A holds Resource 1 and waits for Resource 2.
Thread B holds Resource 2 and waits for Resource 1.

Thus, deadlock is a serious error in multithreading that must be avoided to ensure smooth execution.

2. “Java's synchronized keyword ensures that only one thread at a time is in a critical region”. Comment.

The synchronized keyword in Java is used to control access to a shared resource. When a method or block is declared synchronized, only one thread can execute it at a time.

This prevents race conditions and ensures data consistency. Other threads attempting to enter the synchronized region are blocked until the current thread exits.

Therefore, synchronized guarantees mutual exclusion and thread safety in critical sections.

3. “Each thread may perform different tasks. Sometimes, it becomes necessary to suspend the execution of a thread for a period of time”. Comment.

In multithreading, threads often need to pause execution temporarily to allow other threads to execute or to wait for resources.

Java provides methods such as sleep() and wait() to suspend a thread for a specified time or until a condition is met. This improves CPU utilization and thread coordination.

Suspending threads helps in implementing delays, scheduling, and synchronization in multithreaded applications.

4. How we can resume from a suspended thread? Explain with the help of an example?

A suspended thread can be resumed using notify() or notifyAll() methods. These methods wake up waiting threads.

Example:

```
class Demo {
synchronized void test() throws InterruptedException {
wait();
System.out.println("Thread resumed");
}
synchronized void resumeThread() {
notify();
}
}
```

Here, wait() suspends the thread and notify() resumes it.

5. With an example explain the different methods of stopping a thread.

Stopping a thread can be done using different techniques:

1. Using a boolean flag (recommended):

```
class MyThread extends Thread {
volatile boolean running = true;
public void run() {
while(running) { }
}
public void stopThread() {
running = false;
}
}
```

2. Using interrupt():

```
Thread t = new Thread();
t.interrupt();
```

The stop() method is deprecated because it is unsafe.

Hence, controlled stopping mechanisms are preferred.

6. Explain the different methods used for avoiding deadlocks

Deadlocks can be avoided using the following methods:

1. Avoid nested locks
2. Lock ordering – acquire locks in a fixed order

3. Use timeout while acquiring locks
4. Use synchronized blocks carefully
5. Use concurrency utilities like java.util.concurrent

By following these practices, deadlocks can be effectively prevented in multithreaded programs.