1. “A thread that is created, must be bound to the run( ) method of an object”. Comment.

In Java, every thread is associated with a run() method because this method contains the code that the thread executes. When a thread is created, either by extending the Thread class or by implementing the Runnable interface, the run() method must be overridden.

The Java Virtual Machine (JVM) always looks for the run() method when a thread is started using the start() method. If run() is not defined, the thread will not perform any task.

Thus, binding a thread to the run() method is mandatory because it defines the execution logic of the thread.

2. “When a thread is alive, it indicates it is in one of its several states”. Justify.

A thread is said to be alive when it has been started and has not yet terminated. During its lifetime, a thread passes through multiple states such as Runnable, Running, Waiting, Blocked, and Sleeping.

Even when a thread is waiting or blocked, it is still considered alive because it has not completed execution. A thread becomes dead only after it finishes execution or is stopped.

Hence, an alive thread may exist in any of its lifecycle states except the terminated state.

3. “The second way that is used to create a thread is, by creating a new class that extends Thread, and then creating an instance of that class”. Elaborate.

In Java, one way to create a thread is by extending the Thread class. In this approach, a new class inherits from Thread and overrides the run() method.

Steps:
1. Create a class that extends Thread
2. Override the run() method
3. Create an object of the class
4. Call the start() method

Example:
```
class MyThread extends Thread {
public void run() {
System.out.println("Thread running");
}
}

MyThread t = new MyThread();
```

```
t.start();
```

This method is simple but limits inheritance because Java does not support multiple inheritance.

4. “Understanding the life cycle of a thread is very important, especially at the time of developing codes using threads”. Elaborate.

Understanding the thread lifecycle helps developers manage thread execution efficiently and avoid issues such as deadlock, starvation, and race conditions.

By knowing thread states, programmers can control thread execution using methods like sleep(), wait(), notify(), and join(). This knowledge ensures proper synchronization and optimal CPU utilization.

Therefore, understanding the thread lifecycle is essential for writing reliable and efficient multithreaded programs.

5. What are the different states of a thread, or what is thread lifecycle?

The thread lifecycle consists of the following states:

1. New: Thread is created but not started.
2. Runnable: Thread is ready to run.
3. Running: Thread is executing.
4. Blocked: Thread is waiting for a resource.
5. Waiting: Thread waits indefinitely for another thread.
6. Timed Waiting: Thread waits for a fixed time.
7. Terminated (Dead): Thread execution is completed.

These states define the complete lifecycle of a thread.

6. With an example explain the different methods of creating a User thread?

User threads can be created in two main ways:

1. By extending Thread class

```
class A extends Thread {
public void run() {
System.out.println("User Thread");
}
}
```

2. By implementing Runnable interface

```
class B implements Runnable {
public void run() {
System.out.println("User Thread");
}
}

Thread t = new Thread(new B());
t.start();
```

The Runnable approach is preferred as it allows multiple inheritance and better design.

7. “There are three ways for the threads to communicate with each other”. Elaborate.

Threads communicate with each other using inter-thread communication mechanisms provided by Java.

1. wait():
Causes the current thread to wait until another thread invokes notify().

2. notify():
Wakes up a single waiting thread.

3. notifyAll():
Wakes up all waiting threads.

These methods are defined in the Object class and are used inside synchronized blocks to ensure proper coordination between threads.