1. Java defines two ways in which we can create a runnable object:

- by implementing the Runnable interface.

- by extending the Thread class.

2. We use Thread class to instantiate, access, and control the thread.

3. run() establishes the entry point for another, concurrent thread of execution within your program.

4. The new thread will start running/executing only when its start() method is called, which is declared by Thread. The start() method causes the JVM to call run().

5. The sleep() method causes the thread from which it is called to suspend execution for the specified period of milliseconds.

6. Inside main(), a new Thread object is created by the following sequence of statements:

// First, construct a MyThread object.

MyThread mt = new MyThread("Child #1");

// Next, construct a thread from that object.

Thread newThrd = new Thread(mt);

// Finally, start execution of the thread.

newThrd.start();

7. When a class extends Thread, it must override the run() method, which is the entry point for the new thread. It must also call start() to begin execution of the new thread.

8. The isAlive() method returns true if the thread on which it is called is still running. It returns false otherwise.

9. To wait for a thread to finish is to call join(). This method waits until the thread on which it is called terminates and the specified thread joins it. Additional forms of join() allow you to specify a maximum amount of time that you want to wait for the specified thread to terminate.

10. When a child thread is started, its priority setting is equal to that of its parent thread. You can change a thread’s priority by calling setPriority(), which is a member of Thread. The value of priority level must be within the range MIN\_PRIORITY and MAX\_PRIORITY. Currently, these values are 1 and 10, respectively. To return a thread to default priority, specify

NORM\_PRIORITY, which is currently 5. These priorities are defined as static final variables within Thread.

You can obtain the current priority setting by calling the getPriority( ) method of Thread.

11. When using multiple threads, the coordination between the activities of two or more threads is achieved by synchronization.

Reasons for synchronisation:

- when two or more threads need access to a shared resource that can be used by only one thread at a time.

- when one thread is waiting for an event that is caused by another thread.

Key to synchronization in Java is the concept of the monitor, which controls access to an object. A monitor works by implementing the concept of a lock. When an object is locked by one thread, access to the object by another thread is restricted. When the

thread exits, the object is unlocked and is available for use by another thread.

12. The code can be synchronised in two ways:

-You can synchronize access to a method by modifying it with the synchronized keyword,i.e., a synchronized method is created by preceding its declaration with 'synchronized'.

-You can put calls to the methods defined by a class inside a synchronized block.

This is the general form of a synchronized block:

synchronized(objref ) {

// statements to be synchronized

}

Here, objref is a reference to the object for which synchronization is needed.

13. -A thread called T is executing inside a synchronized method and needs access to a resource called R that is temporarily unavailable. We can have T temporarily release control of the object, allowing another thread to run. When R becomes available, T can be notified and resume execution. We need some form of interthread communication in which one thread can notify another that it is blocked and later be notified that it can resume execution.

-Java supports interthread communication with the wait(), notify(), and notifyAll() methods.

-These methods are implemented by the Object class (so, all objects support interthread communication) and can be called only from within a synchronized context.

14. When a thread is temporarily blocked from running, it calls wait(). This causes the thread to go to sleep and the monitor for that object to be released, allowing another thread to use the object. At a later point, the sleeping thread is awakened when some other thread enters the same monitor and calls notify(), or notifyAll().A call to notify() resumes one waiting thread. A call to notifyAll() notifies all threads, with the highest priority thread gaining access to the object.

15. Calls to wait() should take place within a loop that checks the condition on which the thread is waiting.

16. -We can pause, restart, or terminate a thread. Typically, this is accomplished by establishing two flag variables: one for suspend and resume, and one for stop.

-For suspend and resume, as long as the flag is set to “running”, the run() method must continue to let the thread execute. If this variable is set to “suspend”, the thread must pause. For the stop flag, if it is set to “stop”, the thread must terminate.

Eg:

-We can define two boolean variables, suspended and stopped, which govern the suspension and termination of a thread.

-If suspended variable is true, the wait() method is invoked to suspend the execution of the thread and suspended is set to true.

-To resume execution, suspended is set to false and notify() is invoked to restart the thread.

-To stop the thread, set stopped to true, suspended to false and then call notify(). These steps are necessary to ensure that a suspended thread is stopped.