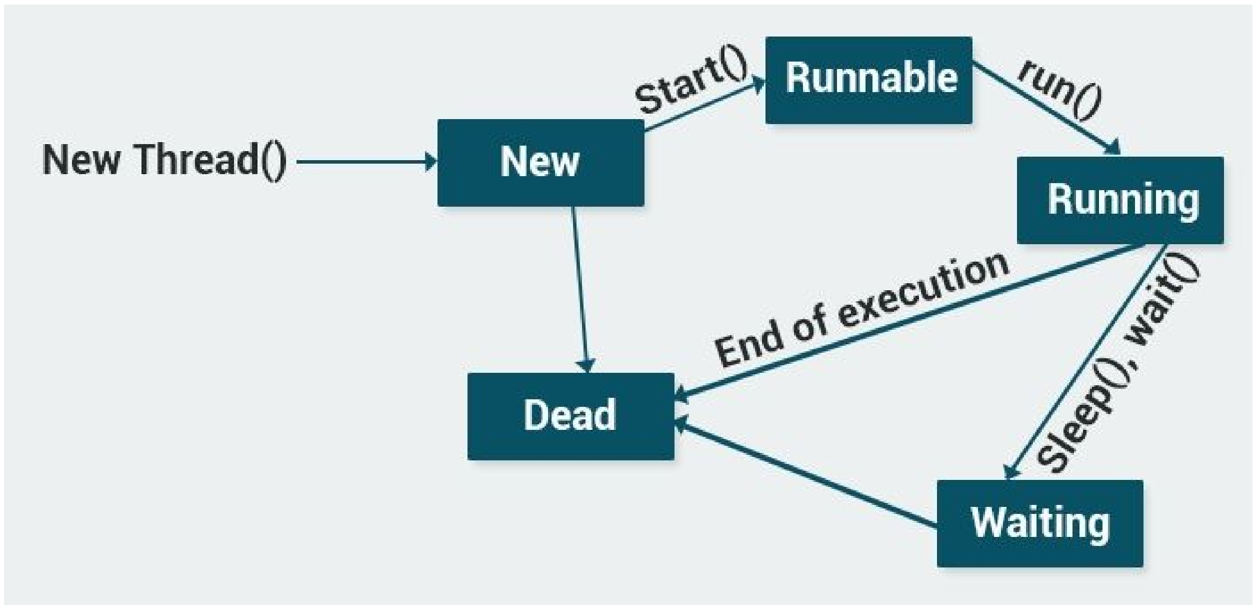
7. Multithreading

Threads

Java is a **multi-threaded** programming language. This means that our program can make optimal use of available resources by running two or more components concurrently, with each component handling a different task.

You can subdivide specific operations within a single application into individual **threads** that all run in parallel.

The following diagram shows the life-cycle of a thread.



There are two ways to create a thread.

1. Extend the Thread class

Inherit from the **Thread** class, override its **run**() method, and write the functionality of the thread in the **run**() method.

Then you create a new object of your class and call it's **start** method to run the thread.

Example:

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

class MyClass {
        public static void main(String[ ] args) {
                Loader obj = new Loader();
                obj.start();
        }
}
```

As you can see, our Loader class extends the Thread class and overrides its **run()** method. When we create the **obj** object and call its **start**() method, the **run**() method statements execute on a different thread.

Note:
Every Java thread is prioritized to help the operating system determine the order in which to schedule threads. The priorities range from 1 to 10, with each thread defaulting to priority 5. You can set the thread priority with the **setPriority**() method.

Q: Fill in the blanks to run the method in a separate thread.

```
class A ________ Thread {

         public void ______ () {
           System.out.println("Hello");
         }

         public static void main(String[ ] args) {
                A object = new A();
                object._______();
         }
}
```

Threads

The other way of creating Threads is **implementing the Runnable interface**.

Implement **the** run() method. Then, create a new Thread object, pass the Runnable class to its constructor, and start the Thread by calling the **start**() method.

Example:

```
class Loader implements Runnable {
        public void run() {
                System.out.println("Hello");
        }
}

class MyClass {
        public static void main(String[ ] args) {
                Thread t = new Thread(new Loader());
                t.start();
        }
}
```

The **Thread.sleep()** method pauses a Thread for a specified period of time. For example, calling **Thread.sleep(1000);** pauses the thread for one second. Keep in mind that **Thread.sleep(**) throws an InterruptedException, so be sure to surround it with a **try/catch** block.

Note:

It may seem that implementing the Runnable interface is a bit more complex than extending from the Thread class. However, implementing the Runnable interface is the preferred way to start a Thread, because it enables you to extend from another class, as well.

Q: Fill in the blanks to implement the Runnable interface and run a new thread.

```
class A ________  __________ {
  public void run() {
  System.out.println("Bye");
  }
}
public class App {
 public static void main(String[ ] args) {
  Thread ob = new Thread(new _________ ());
  ob.________();
  }
}
```