1. **`Timer`**: Diese Klasse steuert den Zeitablauf. Sie legt fest, **wann** und in welchem Intervall eine Aufgabe ausgeführt wird (z.B. alle 2 Sekunden).
2. **`TimerTask`**: Diese Klasse beschreibt **was** gemacht werden soll. Sie enthält den Code, der beim Ablauf der Zeit ausgeführt wird (z.B. eine Nachricht wie "Hallo Welt" ausgeben).

- **`Timer`**: Steuert, **wie oft** und **wann** etwas ausgeführt wird (z.B. jede 2 Sekunden).
- **`TimerTask`**: Definiert, **welche Aufgabe** ausgeführt wird (z.B. "Hallo Welt" ausgeben).

3. `Timer` nutzt intern Threads, um die geplanten Aufgaben auszuführen.

- Wenn du einen `Timer` erstellst, startet er intern **einen neuen Thread**, der dafür sorgt, dass die geplanten Aufgaben (`TimerTask`s) zu den festgelegten Zeiten ausgeführt werden.
- Dieser Thread läuft unabhängig vom Hauptprogramm, was bedeutet, dass dein Programm weiterarbeiten kann, während der Timer im Hintergrund die Aufgaben steuert.
- **`TimerTask` wird von diesem Thread ausgeführt**:
- Der `TimerTask`, der die eigentliche Logik enthält (also das, **was** gemacht werden soll), wird von dem **Thread des Timers** ausgeführt. Das bedeutet, dass die `run()`-Methode des `TimerTask` in diesem separaten Thread läuft, während dein Hauptprogramm parallel weiterläuft.

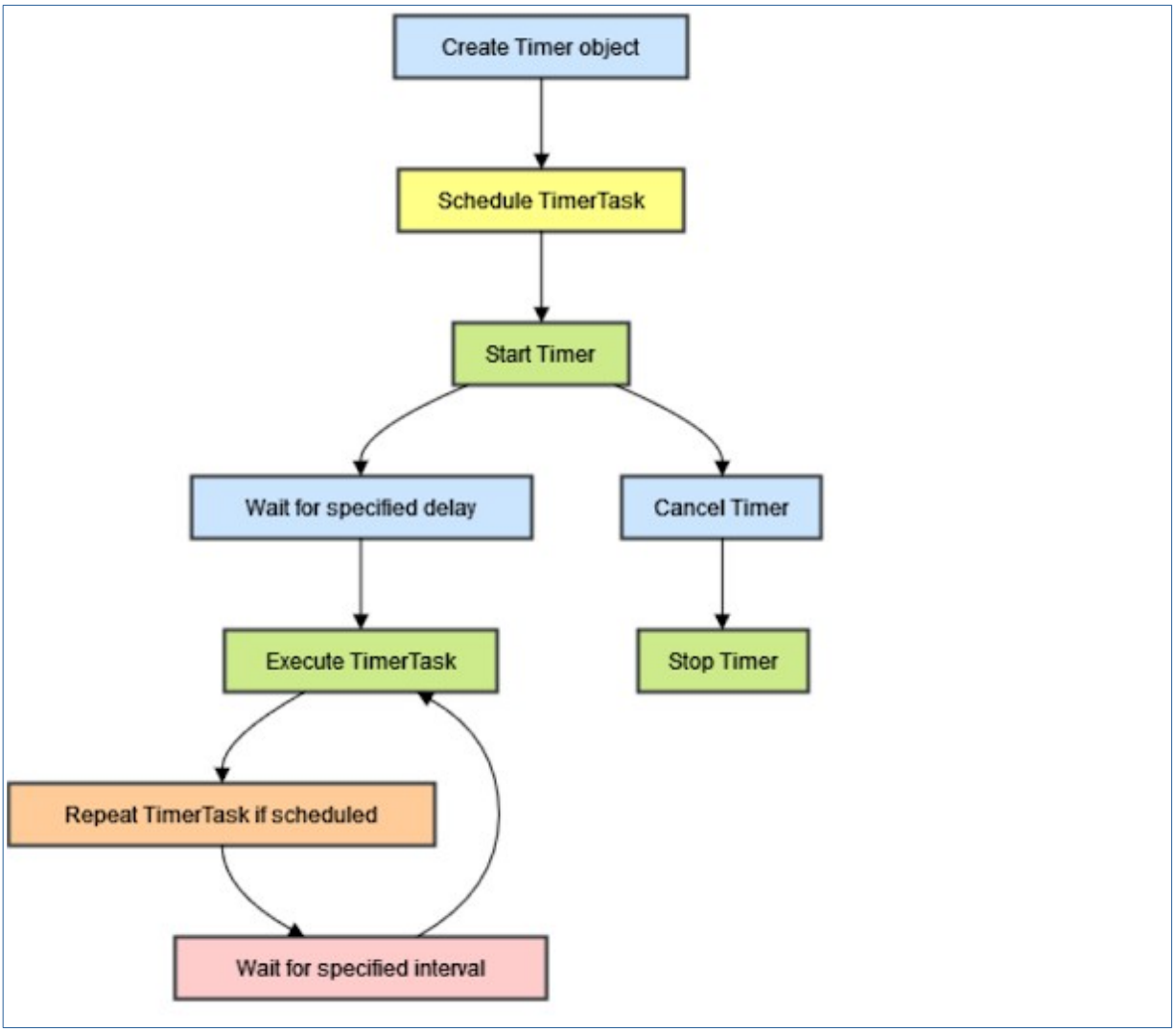
Beziehung zwischen `Timer` und `TimerTask`:

1. **`Timer`**:
    - Der **`Timer`** ist das **Objekt, das die Zeit steuert**. Es plant und verwaltet, wann eine bestimmte Aufgabe (also ein `TimerTask`) ausgeführt werden soll.
    - Der `Timer` an sich hat keine Logik darüber, **was** getan werden soll, sondern nur **wann** etwas getan werden soll. Dafür braucht er eine **Aufgabe**, die er ausführt.
2. **`TimerTask`**:
    - Ein **`TimerTask`** ist eine **definierte Aufgabe** (eine bestimmte Aktion, die ausgeführt werden soll), aber es weiß nicht von alleine, **wann** es ausgeführt werden soll.
    - Das `TimerTask`-Objekt stellt lediglich die Logik bereit, die ausgeführt wird, wenn der `Timer` es auslöst.

Wie arbeiten `Timer` und `TimerTask` zusammen?

- **Der `Timer` braucht den `TimerTask`**, um zu wissen, **was** er tun soll. Ohne eine Aufgabe (also ohne `TimerTask`), die er ausführt, wäre der `Timer` nutzlos, da er keine Aktion durchführen kann.

- **Der `TimerTask` braucht den `Timer`**, um zu wissen, **wann** er ausgeführt wird. Der `TimerTask` alleine hat keinen Mechanismus, um sich selbst zu starten oder in regelmäßigen Abständen ausgeführt zu werden.



Timer Klasse (Methoden):

1. schedule(TimerTask task, long delay):
- führt die Aufgabe **nur einmal** nach der angegebenen Verzögerung (in Millisekunden) aus und nicht wiederholt.
- sagt dem Timer, dass er die Aufgabe nach n **Millisekunden (1 Sekunde)** einmalig ausführen soll

```
public void schedule(TimerTask task, long delay) {
    if (delay < 0)
        throw new IllegalArgumentException("Negative delay.");
    sched(task, System.currentTimeMillis()+delay, 0);
}
```

2. schedule(TimerTask task, long delay, long period):

- **TimerTask task:** Dies ist die Aufgabe, die geplant werden soll. Es handelt sich um eine Instanz von `TimerTask`, die den auszuführenden Code definiert.

- **long delay:** Die Verzögerung in Millisekunden, nach der die Aufgabe zum ersten Mal ausgeführt wird. Wenn `delay = 1000` ist, wird die Aufgabe nach 1 Sekunde ausgeführt.

- **`long period`: Das Wiederholungsintervall in Millisekunden**. Dies gibt an, wie oft die Aufgabe nach ihrer ersten Ausführung wiederholt werden soll. Wenn `period = 1000`, wird die Aufgabe jede Sekunde wiederholt.

```
public void schedule(TimerTask task, long delay, long period) {
    if (delay < 0)
        throw new IllegalArgumentException("Negative delay.");
    if (period <= 0)
        throw new IllegalArgumentException("Non-positive period.");
    sched(task, System.currentTimeMillis()+delay, -period);
}
```

3. schedule(TimerTask task, Date firstTime, long period):

- **`TimerTask task`**: Die Aufgabe, die ausgeführt werden soll. Dies ist eine Instanz von `TimerTask`, welche die `run()`-Methode implementiert, die die eigentliche Aufgabe definiert.

- **`Date firstTime`**: Dies ist der Zeitpunkt, zu dem die Aufgabe zum ersten Mal ausgeführt werden soll. Der Zeitpunkt wird als `Date`-Objekt übergeben. Die Methode ruft `firstTime.getTime()` auf, um den Zeitwert in Millisekunden seit dem 1. Januar 1970 (Epochenzeit) zu erhalten.

(((`System.currentTimeMillis()` gibt die aktuelle Zeit)))

- **`period`:** Das Wiederholungsintervall. In diesem Beispiel wird die Aufgabe alle 2 Sekunden (2000 Millisekunden) nach der ersten Ausführung wiederholt.

4. `cancel()`:
- **Stoppt den Timer**, sodass keine weiteren Aufgaben mehr ausgeführt werden.

5.int purge():
- entfernt alle **abgebrochenen Aufgaben** (also `TimerTask`s, die mit `cancel()` abgebrochen wurden) aus der Aufgabenwarteschlange des `Timer`.

- Wenn eine `TimerTask`-Aufgabe mit der Methode `cancel()` abgebrochen wird, bleibt sie zunächst in der internen Aufgabenwarteschlange des `Timer`, bis die `purge()`-Methode aufgerufen wird.

- Die `purge()`-Methode **bereinigt diese interne Warteschlange und entfernt alle `TimerTask`s**, die abgebrochen wurden, sodass sie nicht mehr Ressourcen oder Speicherplatz belegen.