# LibreOffice Border Customization

### Daemon Threads Explained with a Real-World Scenario:

**Scenario**: Imagine you have a hotel. The hotel has:

1. **Front-desk staff (User threads)**: These are the people who handle the check-ins and check-outs. They are essential for the hotel's operation. As long as a guest is at the front desk, the staff needs to keep working.
2. **Cleaning staff (Daemon threads)**: These people clean the hotel rooms in the background. Their work is important but not critical for running the hotel. If there are no more guests, the hotel can close, and the cleaning staff will stop working.

Now, if all the guests (user threads) leave the hotel, the hotel can close, and the cleaning staff (daemon threads) will stop working too. The cleaning staff's work is not essential for the hotel to stay open, but they work as long as the hotel is open.

### Daemon Thread Example: Real-Time Scenario in a Java Program

Let's break down a similar scenario in a Java program where:

- **User threads** represent the essential tasks (like processing orders).
- **Daemon threads** represent background tasks (like auto-saving data or logging).

### Java Code Example:

```java
public class DaemonThreadExample {
    public static void main(String[] args) {
        // 1. Create a user thread
        Thread userThread = new Thread(new UserTask(), "User Thread");
        System.out.println(userThread.getName() + " is daemon: " + userThread.isDaemon());
// false by default

        // 2. Create a daemon thread
        Thread daemonThread = new Thread(new DaemonTask(), "Daemon Thread");
        daemonThread.setDaemon(true);  // Mark this thread as a daemon thread
        System.out.println(daemonThread.getName() + " is daemon: " +
daemonThread.isDaemon());  // true

        // 3. Start both threads
        userThread.start();
        daemonThread.start();

        // 4. Main thread finishes here, but JVM will wait for the user thread to finish
        System.out.println("Main thread ends.");
    }
}

// 5. Simulating an important user task
class UserTask implements Runnable {
    @Override
    public void run() {
            System.out.println(Thread.currentThread().getName() + " is running.");
            Thread.sleep(5000);  // Simulate work by sleeping for 5 seconds
            System.out.println(Thread.currentThread().getName() + " finished its task.");
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}

// 6. Simulating a background daemon task
class DaemonTask implements Runnable {
    @Override
    public void run() {
        while (true) {
            try {
                System.out.println(Thread.currentThread().getName() + " is running (daemon
thread).");
                Thread.sleep(1000);  // Daemon thread runs every second
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

### Breaking Down the Code with Explanation:

1. **Create a User Thread**:

```java
Thread userThread = new Thread(new UserTask(), "User Thread");
```

   - We create a **user thread** (just like a front-desk staff member at a hotel).
   - This thread is responsible for doing an essential task, and the JVM will not shut down until this thread finishes.

2. **Check if the Thread is Daemon**:

```java
System.out.println(userThread.getName() + " is daemon: " + userThread.isDaemon());
// false by default
```

   - By default, `userThread` is not a daemon thread (`false`). User threads are essential and keep the application running.

3. **Create a Daemon Thread**:

```java
Thread daemonThread = new Thread(new DaemonTask(), "Daemon Thread");
daemonThread.setDaemon(true);  // Mark this thread as a daemon thread
```

   - We create a **daemon thread** (like the cleaning staff at a hotel).
   - Daemon threads work in the background, and the JVM can shut down when all user threads

4. **Check if Daemon Thread**:

```java
System.out.println(daemonThread.getName() + " is daemon: " +
daemonThread.isDaemon());  // true
```

    - The `daemonThread` is explicitly marked as a daemon (`true`), meaning it's a background thread that won't prevent the JVM from exiting when user threads finish.

5. **Start the Threads**:

```java
userThread.start();
daemonThread.start();
```

    - Both the **user thread** and the **daemon thread** are started. The user thread does some important work, and the daemon thread works in the background continuously.

6. **Main Thread Ends**:

```java
System.out.println("Main thread ends.");
```

    - The main thread will exit, but the JVM waits for the **user thread** to finish before shutting down. It won't wait for the daemon thread.

### How the Threads Work:

- **User Thread (`UserTask`)**: This simulates an important task (e.g., processing an order or handling a request). It runs for 5 seconds and then finishes.

```java
public void run() {
    try {
        System.out.println(Thread.currentThread().getName() + " is running.");
        Thread.sleep(5000);  // Simulate work by sleeping for 5 seconds
        System.out.println(Thread.currentThread().getName() + " finished its task.");
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
```

- **Daemon Thread (`DaemonTask`)**: This simulates a background task (e.g., saving logs or cleaning up temporary files). It runs in an infinite loop, but it won't keep the JVM alive. Once the **user thread** finishes, the JVM will terminate the **daemon thread**.

```java
public void run() {
    while (true) {
        try {
            System.out.println(Thread.currentThread().getName() + " is running 
(daemon thread).");
            Thread.sleep(1000);  // Daemon thread runs every second
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

### Real-Time Analogy:

- **User Thread**: Think of the **user thread** as the front-desk staff in a hotel. The hotel stays open as long as there are guests (user threads). If the guest (user thread) leaves, the hotel can close.
- **Daemon Thread**: The **daemon thread** is like the cleaning staff. They work in the background to clean the hotel, but if the hotel has no guests (user threads), the hotel closes, and the cleaning staff stops working.

### Output Explanation:

```plaintext
User Thread is daemon: false
Daemon Thread is daemon: true
User Thread is running.
Daemon Thread is running (daemon thread).
Daemon Thread is running (daemon thread).
Daemon Thread is running (daemon thread).
Daemon Thread is running (daemon thread).
User Thread finished its task.
Main thread ends.
```

1. **User Thread** starts running, and after 5 seconds, it completes its task.
2. The **Daemon Thread** keeps running in the background, printing a message every second.
3. Once the **User Thread** finishes, the JVM terminates, and the **Daemon Thread** stops running.

This shows that daemon threads only run as long as user threads are alive and will be terminated automatically when the JVM exits.