University of Ruhuna
Faculty of Technology
Department of Informtion and Communication Technology

Advanced Programming Practicum
(ICT3122)

Lab Sheet 7 19th Feb 2024

Understanding Thread Pausing Methods

Objective: The objective of this practical session is to deepen your understanding of various methods through which threads can pause or be blocked in Java. You will explore different scenarios and implement code examples to demonstrate each thread pausing method.

Instructions:

- Read the description of each thread pausing method carefully.
- Study the provided code examples.
- Implement and run each code example to observe its behavior.
- Take notes on your observations and any questions or difficulties you encounter.
- Ask for assistance if needed.

Thread Pausing Methods:

1. **Blocking on I/O:**

Threads can block while waiting for input/output operations to complete. This commonly occurs when reading from or writing to external sources such as files or network sockets.

```
import java.io.*;

public class IOBlockingExample {
    public static void main(String[] args) {
        Thread ioThread = new Thread(new IOThread());
        ioThread.start();
    }
}

class IOThread implements Runnable {
    @Override
    public void run() {
        try {
            BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
            String input = reader.readLine(); // This will block until user enters some input
            System.out.println("User input: " + input);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

This code demonstrates how a thread can block while waiting for user input from the console.

2. **Blocking on a Synchronized Object:**

Threads can block while waiting to acquire a lock on a synchronized object. This allows synchronized access to shared resources and prevents race conditions.

```
public class SynchronizedBlockingExample {
    public static void main(String[] args) {
        Object lock = new Object();
        Thread syncThread = new Thread(new SyncThread(lock));
        syncThread.start();

        // Simulating another thread notifying the waiting thread after some time
        try {
            Thread.sleep(2000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        synchronized (lock) {
            lock.notify();
        }
    }
}

class SyncThread implements Runnable {
    private Object lock;

    public SyncThread(Object lock) {
        this.lock = lock;
    }

    @Override
    public void run() {
        synchronized (lock) {
            try {
                lock.wait(); // This will block until another thread calls notify() or notifyAll()
                System.out.println("Thread resumed after waiting on lock.");
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

This code illustrates how a thread can wait indefinitely until another thread notifies it after acquiring the lock on a shared object.

3. **Going to Sleep:**

Threads can pause by invoking the **sleep()** method, causing them to sleep for a specified duration. This is useful for introducing delays in program execution.

```
public class SleepExample {
    public static void main(String[] args) {
        Thread sleepThread = new Thread(new SleepThread());
        sleepThread.start();
    }
}

class SleepThread implements Runnable {
    @Override
    public void run() {
        try {
            System.out.println("SleepThread is going to sleep for 3 seconds.");
            Thread.sleep(3000); // Sleep for 3 seconds
```

```
            System.out.println("SleepThread woke up after 3 seconds.");
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

This code showcases how a thread can pause for a specified duration using the **sleep()** method.

4. **Joining Another Thread:**

Threads can pause by invoking the **join()** method on another thread, causing them to wait until the specified thread completes its execution.

```
public class JoinExample {
    public static void main(String[] args) {
        Thread mainThread = Thread.currentThread();
        Thread joinThread = new Thread(new JoinThread(mainThread));
        joinThread.start();
        try {
            joinThread.join(); // Main thread will wait for joinThread to finish
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("Main thread continues after JoinThread completes.");
    }
}

class JoinThread implements Runnable {
    private Thread mainThread;

    public JoinThread(Thread mainThread) {
        this.mainThread = mainThread;
    }

    @Override
    public void run() {
        try {
            System.out.println("JoinThread is running.");
            Thread.sleep(2000); // Simulating some work
            System.out.println("JoinThread is done.");
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

This code demonstrates how a main thread can wait for another thread to complete using the **join()** method.

5. **Yielding:**

Threads can voluntarily yield their current execution to allow other threads of the same priority to run. This is useful for cooperative multitasking.

```
public class YieldExample {
    public static void main(String[] args) {
        Thread yieldingThread = new Thread(new YieldThread());
        yieldingThread.start();
        for (int i = 0; i < 5; i++) {
            System.out.println("MainThread: " + i);
        }
    }
}
```

```
class YieldThread implements Runnable {
    @Override
    public void run() {
        for (int i = 0; i < 5; i++) {
            System.out.println("YieldingThread: " + i);
            Thread.yield(); // Yielding to allow other threads to run
        }
    }
}
```

This code shows how a thread can yield to allow other threads to execute.

6. **Waiting on an Object:**

Threads can pause by invoking the **wait()** method on an object, causing them to wait for notification from another thread. This is commonly used for inter-thread communication.

```
public class WaitExample {
    public static void main(String[] args) {
        Object lock = new Object();
        Thread waitThread = new Thread(new WaitThread(lock));
        waitThread.start();

        // Simulating another thread notifying the waiting thread after some time
        try {
            Thread.sleep(2000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        synchronized (lock) {
            lock.notify();
        }
    }
}

class WaitThread implements Runnable {
    private Object lock;

    public WaitThread(Object lock) {
        this.lock = lock;
    }

    @Override
    public void run() {
        synchronized (lock) {
            try {
                System.out.println("WaitThread is waiting on lock.");
                lock.wait(); // Wait indefinitely until another thread calls notify() or notifyAll()
                System.out.println("WaitThread resumed.");
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

This code illustrates how a thread can pause and wait for notification from another thread using the **wait()** method.

7. **Being Preempted by a Higher-Priority Thread:**

Threads can be preempted by higher-priority threads, meaning they are temporarily paused to allow execution of the higher-priority thread.

```
public class PriorityExample {
    public static void main(String[] args) {
        Thread highPriorityThread = new Thread(new HighPriorityThread());
        highPriorityThread.setPriority(Thread.MAX_PRIORITY); // Set high priority
        highPriorityThread.start();

        Thread lowPriorityThread = new Thread(new LowPriorityThread());
        lowPriorityThread.setPriority(Thread.MIN_PRIORITY); // Set low priority
        lowPriorityThread.start();
    }
}

class HighPriorityThread implements Runnable {
    @Override
    public void run() {
        for (int i = 0; i < 5; i++) {
            System.out.println("HighPriorityThread: " + i);
            try {
                Thread.sleep(1000); // Simulating some work
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}

class LowPriorityThread implements Runnable {
    @Override
    public void run() {
        for (int i = 0; i < 5; i++) {
            System.out.println("LowPriorityThread: " + i);
            try {
                Thread.sleep(1000); // Simulating some work
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

This code demonstrates how a thread with lower priority can be preempted by a higher-priority thread.

Conclusion: By completing this practical session, you will have gained practical experience in implementing and understanding various thread pausing methods in Java. These concepts are fundamental to writing efficient and responsive multithreaded applications. Make sure to experiment with the provided code examples and ask questions if you encounter any difficulties.