

Lab Thread

Exercise 1

Creating a thread

1. The first approach for creating threads is implement Runnable interface (**recommended**)

```
YourTask.java
1 package session01.mthread;
2 /**
3  * Task thread sample
4  * @author VoVanHai
5  */
6 public class YourTask implements Runnable{
7     private String taskName;
8     private int counter;
9
10    public YourTask(String taskName, int counter) {
11        this.taskName = taskName;
12        this.counter = counter;
13    }
14
15    @Override
16    public void run() {
17        for (int i = 0; i < counter; i++) {
18            System.out.println(taskName+ "#" + i);
19        }
20    }
21 }
```

```
YourTask.java TaskExecute.java
1 package session01.mthread;
2
3 public class TaskExecute {
4     public static void main(String[] args) {
5         Runnable r1=new YourTask("Print Task", 20);
6         Runnable r2=new YourTask("Distribute Task", 23);
7         Thread t1=new Thread(r1);
8         Thread t2=new Thread(r2);
9         t1.start();
10        t2.start();
11    }
12 }
```

Run and make a review for the result

2. The second approach for creating a thread by extending the Thread class (**not recommended**)

```
AnotherTask.java
1 package session01.mthread.ex02;
2
3 public class AnotherTask extends Thread{
4     private String taskName;
5     private int counter;
6
7     public AnotherTask(String taskName, int counter) {
8         this.taskName = taskName;
9         this.counter = counter;
10    }
11
12    @Override
13    public void run() {
14        for (int i = 0; i < counter; i++) {
15            System.out.println(taskName+ "#"+i);
16        }
17    }
18 }
```

```
AnotherTask.java TaskRun.java
1 package session01.mthread.ex02;
2
3 public class TaskRun {
4     public static void main(String[] args) {
5         Runnable r1=new AnotherTask("Collect Task", 15);
6         Runnable r2=new AnotherTask("Process Task", 19);
7         Thread t1=new Thread(r1);
8         Thread t2=new Thread(r2);
9         t1.start();
10        t2.start();
11    }
12 }
```

Run and make a review for the result. Explain why this way was not a recommendation.

3. The third approach is implemented Callable interface

```
ComputationTask.java
2
3 import java.util.concurrent.Callable;
4
5 public class ComputationTask implements Callable<Long>{
6     private String taskName;
7     public ComputationTask(String taskName) {
8         this.taskName = taskName;
9     }
10
11     @Override
12     public Long call() throws Exception {
13         Long result=0L;
14         for (int i = 0; i < 1000; i++) {
15             result+=i;//simple for testing purpose
16             System.out.println(taskName + " #" +i);
17             Thread.sleep(10);
18         }
19         return result;
20     }
21 }
```

```
ComputationTask.java ComputationExecutor.java
3 import java.util.concurrent.Callable;
4 import java.util.concurrent.FutureTask;
5
6 public class ComputationExecutor {
7     public static void main(String[] args) throws Exception{
8         Callable<Long> call=new ComputationTask("long-last-computaion");
9         FutureTask<Long> task = new FutureTask<>(call);
10        new Thread(task).start();
11
12        //Waits if necessary for the computation to complete,
13        //and then retrieves its result.
14        long result=task.get();
15        System.out.println("Result:"+result);
16    }
17 }
```

Run and make a review for the result.

Exercise 2

Manipulate methods of thread

1. Using *join()* method
Waits for this thread to die.

```

AnotherTask.java  YourTask.java  TestJoinThread.java
1 package session01.mthread.ex04;
2
3 public class AnotherTask implements Runnable{
4     private String taskName;
5     private int counter;
6
7     public AnotherTask(String taskName, int counter) {
8         this.taskName = taskName;
9         this.counter = counter;
10    }
11
12    @Override
13    public void run() {
14        for (int i = 0; i < counter; i++) {
15            System.out.println(taskName+ "#"+i);
16        }
17    }
18 }

```

```

AnotherTask.java  YourTask.java  TestJoinThread.java
1 package session01.mthread.ex04;
2
3 public class YourTask implements Runnable{
4
5     @Override
6     public void run() {
7         try {
8             Thread t=new Thread(
9                 new AnotherTask("Another task",10));
10            t.start();//start another task
11            for (int i = 0; i < 8; i++) {
12                System.out.println("Your Task #"+i);
13                if(i==5)
14                    t.join();//join thread
15            }
16        } catch (InterruptedException e) {
17            e.printStackTrace();
18        }
19    }
20 }

```

```

AnotherTask.java  YourTask.java  TestJoinThread.java
1 package session01.mthread.ex04;
2
3 public class TestJoinThread {
4     public static void main(String[] args) throws Exception{
5         new Thread(new YourTask()).start();
6     }
7 }

```

Run and make a review for the result.

2. Using yield() method

The **java.lang.Thread.yield()** method causes the currently executing thread object to temporarily pause and allow other threads to execute.

```

ThreadDemoUsingYieldMethod.java
1 package session01.mthread.ex05;
2
3 public class ThreadDemoUsingYieldMethod implements Runnable {
4     private Thread t;
5
6     public ThreadDemoUsingYieldMethod(String str) {
7         t = new Thread(this, str);
8         t.start();
9     }
10    public void run() {
11
12        for (int i = 0; i < 5; i++) {
13            // yields control to another thread every 5 iterations
14            if ((i % 5) == 0) {
15                System.out.println(Thread.currentThread().getName() + "yielding control...");
16                /* causes the currently executing thread object to temporarily
17                 pause and allow other threads to execute */
18                Thread.yield();
19            }
20        }
21        System.out.println(Thread.currentThread().getName() + " has finished executing.");
22    }
23
24    public static void main(String[] args) {
25        new ThreadDemoUsingYieldMethod("Thread 1");
26        new ThreadDemoUsingYieldMethod("Thread 2");
27        new ThreadDemoUsingYieldMethod("Thread 3");
28    }
29 }

```

Run program and make a review.

3. Using daemon thread

```

DaemonThread.java
1 package session01.mthread.ex06;
2
3 public class DaemonThread extends Thread {
4     public void run() {
5         System.out.println("Entering run method");
6         try {
7             System.out.println("In run Method: currentThread() is"
8                 + Thread.currentThread());
9             while (true) {
10                 try {
11                     Thread.sleep(500);
12                 } catch (InterruptedException x) {
13                 }
14                 System.out.println("In run method: woke up again");
15             }
16         } finally {
17             System.out.println("Leaving run Method");
18         }
19     }
20    public static void main(String[] args) throws Exception{
21        System.out.println("Entering main Method");
22        DaemonThread t = new DaemonThread();
23
24        t.setDaemon(true); //turn t to daemon thread
25
26        t.start();
27        Thread.sleep(3000);
28        System.out.println("Leaving main method");
29    }
30 }

```

Run the program and observe.

Comment line 24 and run again. Make an explanation about this case.

4. *** Using the 'wait - notify' mechanism

Create three classes: Storage, Counter and Printer.

The Storage class should store an integer.

The Counter class should create a thread that starts counting from 0 (0, 1, 2, 3 ...) and stores each value in the Storage class.

The Printer class should create thread that keeps reading the value in the Storage class and printing it.

Create a program that creates an instance of the Storage class, and sets up a Counter and a Printer object to operate on it.

(*) *Modify the program to ensure that each number was printed exactly once, by adding suitable synchronization.*

a. Wrong solution

```
class MyQueue {
    int n;
    synchronized int get() {
        System.out.println("Got: " + n);
        return n;
    }
    synchronized void put(int n) {
        this.n = n;
        System.out.println("Put: " + n);
    }
}

class Producer implements Runnable {
    MyQueue q;
    Producer(MyQueue q) {
        this.q = q;
        new Thread(this, "Producer").start();
    }
    public void run() {
        int i = 0;
        while(true) { q.put(i++);}
    }
}

class Consumer implements Runnable {
    MyQueue q;
    Consumer(MyQueue q) {
        this.q = q;
        new Thread(this, "Consumer").start();
    }
    public void run() {
        while(true) { q.get();}
    }
}

public class Producer_Consumer_Demo {
    public static void main(String args[]){
        MyQueue q = new MyQueue();
        new Producer(q);
        new Consumer(q);
    }
}
```

Run, observe and explain why it was a incorrect version.

b. Correct solution

```

class MyQueue {
    int n;
    boolean valueSet = false;
    synchronized int get() {
        if(!valueSet)
            try { wait(); } catch (InterruptedException e) {}
        System.out.println("Got: " + n);
        //assume that our work take a time to execute
        try{Thread.sleep(300);}catch(Exception x){}
        valueSet = false;
        notify();
        return n;
    }
    synchronized void put(int n) {
        if(valueSet)
            try { wait(); } catch (InterruptedException e) {}
        this.n = n;
        valueSet = true;
        System.out.println("Put: " + n);
        //assume that our work take a time to execute
        try{Thread.sleep(500);}catch(Exception x){}
        notify();
    }
}

```

```

class Producer implements Runnable {
    MyQueue q;
    Producer(MyQueue q) {
        this.q = q;
    }
    public void run() {
        int i = 0;
        while(true) {
            q.put(i++);
        }
    }
}

```

```

class Consumer implements Runnable {
    MyQueue q;
    Consumer(MyQueue q) {
        this.q = q;
    }
    public void run() {
        while(true) {
            q.get();
        }
    }
}

```

```

public class Producer_Consumer_Demo_Fixed {
    public static void main(String args[]) {
        System.out.println("Press Control-C to stop.");
        ExecutorService service = Executors.newFixedThreadPool(2);

        MyQueue q = new MyQueue();

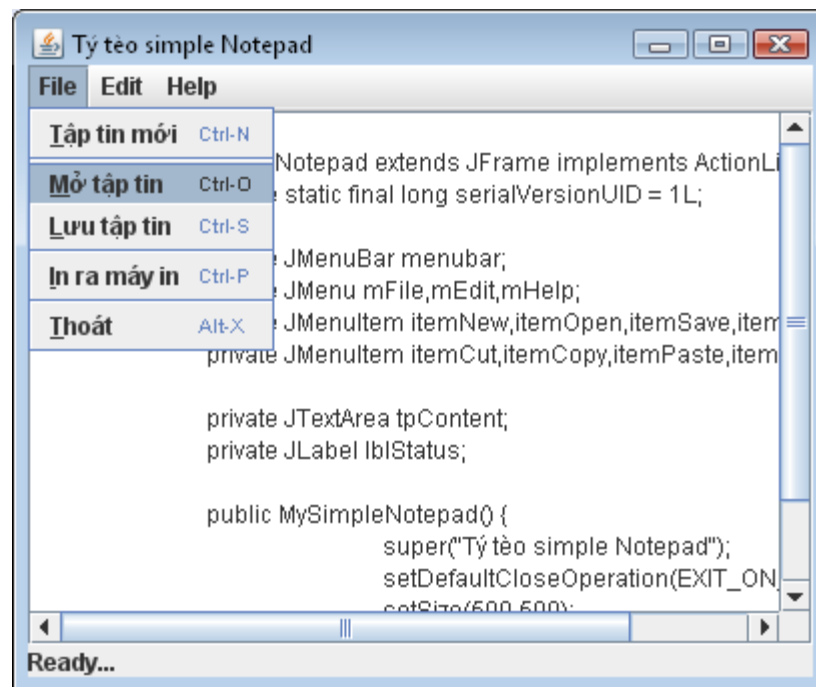
        service.execute(new Producer(q));
        service.execute(new Consumer(q));
    }
}

```

Run and explain the result.

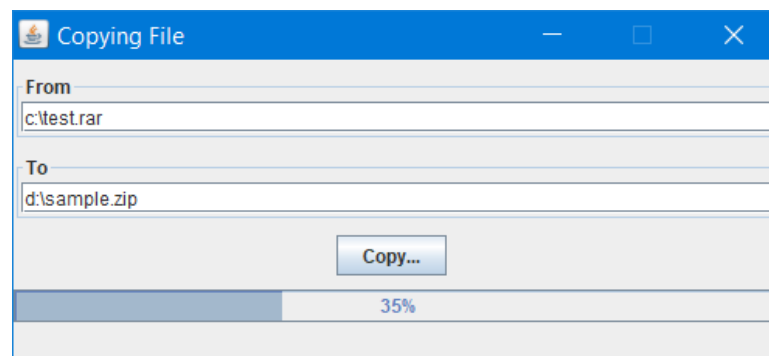
Exercise 3

Write an application simulates simple notepad (use multithreading loading file) Main GUI as follow



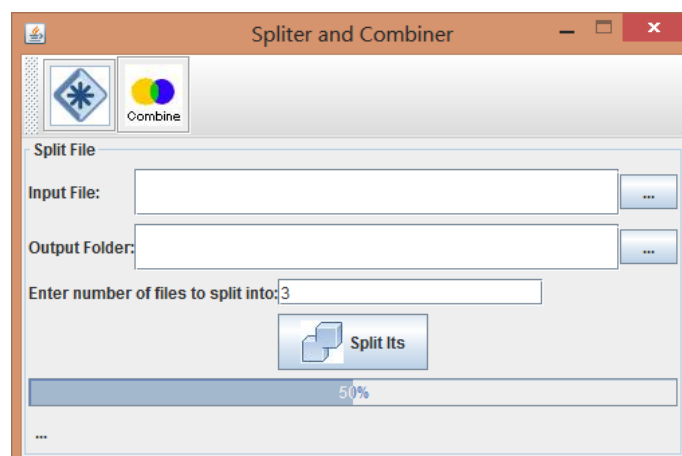
Exercise 4

Write a GUI application that copies files. A progress bar is used to display the progress of the copying operation, as shown in following figure.



Exercise 5

Suppose you wish to back up a huge file (e.g., a 10-GB AVI file) to a CD-R. You can achieve it by splitting the file into smaller pieces and backing up these pieces separately. Write a utility program that splits a large file into smaller ones. (Display the percentage of work done in a progress bar, as shown in following figure)



Read more

1. <https://docs.oracle.com/javase/8/docs/api/java/lang/Thread.html>
2. <https://docs.oracle.com/javase/tutorial/essential/concurrency/>