**Quiz for Java Programming** (Tutor: cyd@bupt.edu.cn)

Unit07-Multi-Threading

1. To create a thread, we can 1) generate a subclass of thread , 2) provide a runnable implement

object

1. With each of the above two methods to create threads, we need to override run

method

1. Write a program to create 2 threads with different method. Thread1 prints character 'a' to 'z', and Thread2 prints number 1-26

// 写一个函数PrintChar 打印'a'-'z'，在 main 中调用

// 写一个函数PrintNum 打印 1-26，在 main 中调用

// 改造PrintChar，让它变成一个线程(继承Thread)，让它运行

// 改造PrintNum，让它变成一个线程(实现Runnable)，让它运行

package school.unit7**;**import java.lang.Thread**;**public class TestPrint {  
 public static void main(String[] args) {  
 //创建 PrintChar 和 PrintNum 的线程对象  
 var tChar = new PrintChar()**;** var rNum = new PrintNum()**;** // 这是一个 runnable 接口的实现类  
 var tNum = new Thread(rNum)**;** // 启动两个线程对象  
 tChar.start()**;** tNum.start()**;** }  
}  
// 创建 PrintChar 类，继承 Thread  
class PrintChar extends Thread{  
 @Override  
 public void run() {  
 printChar()**;** }  
 // 将下面两个函数的代码放到 run() 中  
 public void printChar() { //打印 'a' - 'z'  
 for(int i = 0**;** i<26**;** i++) {  
 System.*out*.println((char)('a'+i))**;** // 对 ASCII 进行加法操作  
 try {  
 Thread.*sleep*(100)**;** } catch (InterruptedException e) {  
 // *TODO Auto-generated catch block* e.printStackTrace()**;** }  
 }  
 }  
  
}  
  
// 创建 PrintNum 类，实现 Runnable 接口  
class PrintNum implements Runnable{  
 @Override  
 public void run() {  
 printNum()**;** }  
 public void printNum() { // 打印 1-26  
 for(int i = 0 **;** i < 26**;** i++){  
 System.*out*.println(i+1)**;** try {  
 Thread.*sleep*(100)**;** } catch (InterruptedException e) {  
 // *TODO Auto-generated catch block* e.printStackTrace()**;** }  
 }  
 }  
}

1. Modify Quiz 3. After thread1 prints 'o', it ***yield***s to thread2; After thread2 prints number 20, it waits to ***join*** thread1.

// 在什么地方调用 yield()? 如何调用 yield()? 在内部线程代码逻辑中,使用Thread静态方法直接调用

// 在什么地方调用 join()? 如何调用 join()? 在实现的线程逻辑代码中,获取current thread来调用join方法

// 怎样才能满足join() 调用时的语句所需的对象？ 执行到该线程

package school.unit7**;**import java.lang.Thread**;**public class TestPrint {  
 public static void main(String[] args) {  
  
 new Thread(new PrintChar()).start()**;** new Thread(new PrintNum()).start()**;** }  
}  
// 创建 PrintChar 类，继承 Thread  
class PrintChar implements Runnable{  
 @Override  
 public void run() {  
 printChar()**;** }  
 // 将下面两个函数的代码放到 run() 中  
 public void printChar() { //打印 'a' - 'z'  
 for(int i = 0**;** i<26**;** i++) {  
 System.*out*.println((char)('a'+i))**;** if( (char)('a'+i) == 'o' ){  
 Thread.*yield*()**;** }  
 try {  
 Thread.*sleep*(100)**;** } catch (InterruptedException e) {  
 // *TODO Auto-generated catch block* e.printStackTrace()**;** }  
 }  
 }  
  
}  
  
// 创建 PrintNum 类，实现 Runnable 接口  
class PrintNum implements Runnable{  
 @Override  
 public void run() {  
 printNum()**;** }  
 public void printNum() { // 打印 1-26  
 for(int i = 0 **;** i < 26**;** i++){  
 System.*out*.println(i+1)**;** if(i == 20)  
 {  
 //调用join方法  
 try {  
 Thread.*currentThread*().join()**;** } catch (InterruptedException e) {  
 e.printStackTrace()**;** }  
 }  
 try {  
 Thread.*sleep*(100)**;** } catch (InterruptedException e) {  
 // *TODO Auto-generated catch block* e.printStackTrace()**;** }  
 }  
 }  
}

1. Create 2 threads. Thread1 adds 1 to an Integer object x and then print out x; the addition loops for 20 times; Thread2 subtracts 1 from the object x and then prints out x; the subtraction loops for 20 times.

Use "synchronize" to insure the printed results are correct.

// 写一个 add()函数，完成加法和输出。尝试在 add()内部循环 20 次

// 写一个 subtract()函数，完成减法和输出。尝试在 subtract()内循环 20 次

// Synchronize 怎么用

// - 函数改东西就同步函数

// - 对象有变化就同步对象

package school.unit7**;**public class TestSynchronized {  
// 写一个 add()函数，完成加法和输出。尝试在 add()内部循环 20 次  
// 写一个 subtract()函数，完成减法和输出。尝试在 subtract()内循环 20 次  
 Integer x = Integer.*valueOf*(0)**;** Object anObject = new Object()**;** public static void main(String[] args) {  
 // 创建 inner class 的实例比较特殊  
 var ts = new TestSynchronized()**;** var tAdd = ts.new Add()**;** var tSubtract = ts.new Subtract()**;** tAdd.start()**;** tSubtract.start()**;** }  
  
// Synchronize 怎么用  
// - 函数改东西就同步函数  
// - 对象有变化就同步对象  
 //inner class  
 public class Add extends Thread {  
 @Override  
 public void run() {  
 add(x)**;** // 参数该如何处理？在 TestSynchronized 类中创建一个Integer 对象  
 }  
 public void add(Integer x) {  
 synchronized(anObject) {  
 for(int i =0**;** i < 20**;** i++) {  
 int t = x.intValue()**;** t++**;** // x = new Integer(t); // 已弃用  
 x = Integer.*valueOf*(t)**;** // valueOf() 是静态函数  
 System.*out*.print("a"+x.intValue()+ " ")**;** }  
 }  
 }  
 }  
  
 public class Subtract extends Thread {  
 @Override  
 public void run() {  
 subtract(x)**;** }  
 public synchronized void subtract(Integer x) {  
 synchronized(anObject) {  
 for(int i = 0**;** i < 20**;** i++) {  
 int t = x.intValue()**;** t--**;** x = Integer.*valueOf*(t)**;** System.*out*.print("s"+x.intValue() + " ")**;** }  
 }  
 }  
 }  
}

1. Modify Quiz3. When thread starting, thread1 sleep 1 second and thread2 wait() thread1. When thread1 finish sleeping, it notify() thread2 to go on. You need to create a synchronized object, which has wait() and notify()

// 线程同步函数的调用方法

// - 自己睡 Thread.sleep()

// - 自己让 Thread.yield()

// - 别人完蛋 OtherThread.join()

// - 有对象才能等通知 anObj.wait(); anObj.notify()

// - 自己可以做对象吗？不正常啊

package school.unit7**;**import java.lang.Thread**;**public class TestSynchronized02 {  
 private int count = 1**;** private final Object lock = new Object()**;** public void printNumbers() {  
 Thread thread1 = new Thread(() -> {  
 synchronized (lock) {  
 while (count <= 100) {  
 try {  
  
 Thread.*sleep*(1000)**;** // Thread 1 sleeps for 1 second  
  
 System.*out*.println("Thread 1: " + count++)**;** lock.notify()**;** // Notifies waiting thread (Thread 2)  
 lock.wait()**;** // Releases the lock and waits for notification  
 } catch (InterruptedException e) {  
 e.printStackTrace()**;** }  
 }  
 lock.notify()**;** // Notifies the other thread in case it's still waiting  
 }  
 })**;** Thread thread2 = new Thread(() -> {  
 synchronized (lock) {  
 try {  
 while (count <= 100) {  
 lock.wait()**;** // Thread 2 waits for notification  
 System.*out*.println("Thread 2: " + count++)**;** Thread.*sleep*(100)**;** // Optional delay for clarity  
 lock.notify()**;** // Notifies the other thread (Thread 1)  
 }  
 } catch (InterruptedException e) {  
 e.printStackTrace()**;** }  
 }  
 })**;** thread2.start()**;** thread1.start()**;** try {  
 thread1.join()**;** thread2.join()**;** } catch (InterruptedException e) {  
 e.printStackTrace()**;** }  
 }  
  
 public static void main(String[] args) {  
 TestSynchronized02 numberPrinter = new TestSynchronized02()**;** numberPrinter.printNumbers()**;** }  
}

//不可以先start t1然后再start t2否则会引发类似死锁一样的状态(两个线程都陷入wait状态)



1. When two or more threads need to acquire the locks on several shared objects, the program would probably get into thread conflict
2. We use lock to prevent the program running into the above circumstances.