高级语言程序设计I实验报告

**实验九 多线程机制**

班 级： 22软件工程

姓 名： 王玉升

学 号： 22206091012

成 绩：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 实验名称 | | **实验九 多线程机制** | | | |
| 实验日期 | | | 2023.12.18 | 实验课时 | 2 |
| 实  验  目  的  及  要  求 | （给出本次实验内容及要求的具体描述）   1. 实验目的：   1．了解多线程的概念  2．掌握线程的创建和使用  3．掌握线程的同步控制   1. 实验要求： 2. 安JDK及Ｅclipse开发环境，并使用Ｅclipse开发平台进行java语言程序的编辑、编译和运行。 3. 在D盘或E盘以自己的学号末尾四位数字为名创建文件夹，并在其下面为每个实验项目创建子文件夹，程序相关文件存储在相应的文件夹中（如学号末尾是”1023”，实验八则应创建目录结构“E:\1023\sy09”，实验八程序文件放在“…\sy09”）。 4. 实验报告中附上程序清单和程序运行结果截图（控制台中输出结果后用自己的真实姓名注释）。 5. 每题程序代码中至少有两处用自己的真实姓名进行注释。   5．实验结束后进行总结。 | | | | |
| 实  验  环  境 | （列出本次实验所使用的平台和相关软件）  地点：  开发环境：JDK1.8、eclipse | | | | |
| 实  验  过  程  及  实  验  结  果 | 1. **创建模拟乌龟和兔子生存（出生、成长、死亡）的两个线程，理解线程的四种状态。将【代码X】替换为Java程序代码（注释不能删除）：**   **/\*本题重点是掌握使用Thread子类创建线程，掌握线程的创建、启动、死亡、休眠的方法\*/**   * 1. 程序清单：   **ThreadExample.java**  package 一**;** class Tortoise extends Thread{  int sleepTime=**0,**liveLength=**0;** // sleepTime每次休眠时间,liveLength模拟寿命  Tortoise(int sleepTime**,**String name**,**int liveLength) {  this.sleepTime=sleepTime**;** this.liveLength=liveLength**;** setName(name)**;** //setName（）是继承Thread的方法，设置线程的名字  }   */\*\*\*  \* 王玉升  \*/* public void run() {  while(true) {  liveLength--**;** System.*out*.print("@")**;** try {  *sleep*(sleepTime)**;** //【代码1】让线程调用 sleep方法进入中断状态，需进行异常捕获，sleepTime毫秒后线程重新排队，等待CUP资源。  }  catch(InterruptedException e) { }  if(liveLength<=**0**) {  System.*out*.print(getName()+"进入死亡状态\n")**;** //getName()方法是继承Thread的方法，获取线程的名字  return**;**//【代码2】结束run方法的语句。  }  }  } }  package 一**;** class Rabbit extends Thread{  int sleepTime=**0,**liveLength**;** Rabbit(int sleepTime**,**String name**,**int liveLength) {  this.sleepTime=sleepTime**;** this.liveLength=liveLength**;** setName(name)**;** }   */\*\*\*  \* 王玉升  \*/* public void run() {  while(true) {  liveLength--**;** System.*out*.print("\*")**;** try {  Thread.*sleep*(sleepTime)**;** //【代码3】让线程调用 sleep方法进入中断状态，sleepTime毫秒后线程重新排队  // 等待CUP资源。  }  catch(InterruptedException e) {  }  if(liveLength<=**0**) {  System.*out*.print( Thread.*currentThread*().getName()+"进入死亡状态\n")**;** // 【代码4】获取线程的名字  return**;** // 【代码5】结束run方法的语句。  }  }  } }  package 一**;** public class ThreadExample{    */\*\*\*  \* 王玉升  \** ***@param*** *args  \*/* public static void main(String args[ ]) {  Rabbit rabbit**;** rabbit = new Rabbit(**1000,**"兔"**, 5**)**;** //【代码6】新建线程rabbit，休眠1秒，名字“兔”，寿命5。  Tortoise tortoise**;** tortoise =new Tortoise(**1000,**"龟"**,10**)**;** //【代码7】新建线程tortoise，休眠1秒，名字“龟”，寿命10。  rabbit.start()**;** //【代码8】启动线程tortoise。  tortoise.start()**;** //【代码9】启动线程rabbit。  } }   * 1. 运行程序和运行结果（截图）：      1. **使用Thread的子类Student创建创建两个线程A和B，A线程打印输出你的学号5次，每次输出之间睡眠1秒，B线程打印输出你的姓名5次，每次输出之间睡眠1秒。**   **/\*本题重点是掌握使用Thread的子类创建线程，掌握线程的创建、启动、休眠等\*/**   * 1. 程序清单：   package 二**;** public class Student extends Thread {  private String name**;** private int id**;** public Student(String name**,** int id) {  this.name = name**;** this.id = id**;** }   */\*\*\*  \* 王玉升  \*/* public void run() {  // A线程打印学号  for (int i = **0;** i < **5;** i++) {  System.*out*.println("学号: " + id)**;** try {  Thread.*sleep*(**1000**)**;** // 睡眠1秒  } catch (InterruptedException e) {  e.printStackTrace()**;** }  }   // B线程打印姓名  for (int i = **0;** i < **5;** i++) {  System.*out*.println("姓名: " + name)**;** try {  Thread.*sleep*(**1000**)**;** // 睡眠1秒  } catch (InterruptedException e) {  e.printStackTrace()**;** }  }  }   public static void main(String[] args) {  Student studentA = new Student("王玉升"**, 609**)**;** Student studentB = new Student("王"**, 1012**)**;** studentA.start()**;** // 启动A线程  studentB.start()**;** // 启动B线程  } }   * 1. 运行程序和运行结果（截图）：      1. **编程：直接使用Thread类创建线程对象实现第2题（使用Runnable接口）**   **/\*本题重点是掌握使用实现Runnable接口类的实例作为目标对象创建线程\*/**   * 1. 程序清单：   package 三**;** public class Student implements Runnable {  private String name**;** private int id**;** public Student(String name**,** int id) {  this.name = name**;** this.id = id**;** }   @Override  public void run() {  // A线程打印学号  for (int i = **0;** i < **5;** i++) {  System.*out*.println("学号: " + id)**;** try {  Thread.*sleep*(**1000**)**;** // 睡眠1秒  } catch (InterruptedException e) {  e.printStackTrace()**;** }  }  for(int i=**0;**i<**5;**i++){  System.*out*.println("姓名: "+name)**;** try{  Thread.*sleep*(**1000**)**;** }catch(Exception e){  e.printStackTrace()**;** }  }  }   */\*\*\*  \* 王玉升  \** ***@param*** *args  \*/* public static void main(String[] args) {  Thread threadA = new Thread(new Student("王玉升"**, 1012**))**;** Thread threadB = new Thread(new Student("李四"**, 9999**))**;** // 启动线程A和B  threadA.start()**;** threadB.start()**;** } }   * 1. 运行程序和运行结果（截图）：      1. **“马家河子”和“赵庄”共饮一口水井的水， Village类实现了Runnable接口，中有一个int变量waterAmount，刻画水井的水量，主类Land创建了两个线程，模拟两个村庄在同一口井中饮水。将【代码X】替换为Java程序代码（注释不能删除）**   **/\*本题重点是掌握线程资源共享的方法、线程的同步与通信\*/**   * 1. 程序清单：   **Village.java:**  public class Village implements Runnable{ //【代码1】定义目标对象类，实现Runnable接口  static int *waterAmount***;** //模拟水井的水量   static void setWaterAmount(int m) { //设置水井的水量  if(m>**0**)  *waterAmount* = m**;** }  public void run(){ //【代码2】重写run方法  while(true) {  String name=Thread.*currentThread*().getName()**;** //获取线程名  if(name.equals("赵庄")) {  drinkWater(name**,200**) **;** //赵庄一次用水200  }  else if(name.equals("马家河子")){  drinkWater(name**,100**)**;**//【代码3】马家河子一次用水100  }  System.*out*.println(" 剩 "+*waterAmount*)**;** try{  Thread.*sleep*(**200**)**;**//【代码4】线程休眠200毫秒  }  catch(InterruptedException e){}  if(*waterAmount*<=**0**) {  return**;** }  }  }   */\*\*\*  \* 王玉升  \** ***@param*** *name  \** ***@param*** *n  \*/* public synchronized void drinkWater(String name**,** int n){ // 【代码5】 定义同步方法drinkWater，模拟从水井取水，  if( *waterAmount*-n>=**0**) {  *waterAmount* = *waterAmount*-n**;** System.*out*.print(name+"喝了"+n+"升水,")**;** }  else{  System.*out*.print(name+"喝了"+*waterAmount*+"升水,")**;** *waterAmount* = **0;** }  } }  **Land.java**  package 四**;** public class Land {  */\*\*\*  \* 王玉升  \** ***@param*** *args  \*/* public static void main(String args[]) {  Village village = new Village()**;** //创建目标对象  village.*setWaterAmount*(**2000**)**;** //设置共享数据水量  Thread zhaoZhuang **,**maJiaHeZhi**;** zhaoZhuang=new Thread(village)**;** //创建线程  maJiaHeZhi = new Thread(village) **;** //【代码6】 创建线程maJiaHeZhi  zhaoZhuang.setName("赵庄")**;** //设置线程名字为"赵庄"  maJiaHeZhi.setName("马家河子")**;**//【代码7】设置线程名字为"马家河子"  zhaoZhuang.start()**;** maJiaHeZhi.start()**;**//【代码8】启动线程maJiaHeZhi ,进入就绪状态  } }   * 1. }运行程序和运行结果（截图）：      1. **设计两个线程， 一个充当电子表， 每隔 1 秒显示下一系统时间； 另一个充当闹钟， 每到整点就开始报时， 即显示 5 次整点提示， 同时将第一个线程挂起，报时完毕再将第一个线程恢复运行。 (选做)**   **/\*本题重点是掌握线程资源共享的方法、线程的同步与通信\*/**   * 1. 程序清单：   package 五**;** public class AlarmClockDemo {  */\*\*\*  \* 王玉升  \** ***@param*** *args  \*/* public static void main(String[] args) {  ElectronicTimerThread electronicTimer = new ElectronicTimerThread()**;** AlarmClockThread alarmClock = new AlarmClockThread(electronicTimer)**;** electronicTimer.start()**;** alarmClock.start()**;** } }  package 五**;** import java.util.concurrent.TimeUnit**;** class ElectronicTimerThread extends Thread {  private boolean isRunning = true**;** */\*\*\*  \* 王玉升  \*/* @Override  public void run() {  while (isRunning) {  System.*out*.println("当前系统时间: " + System.*currentTimeMillis*())**;** try {  TimeUnit.*SECONDS*.sleep(**1**)**;** } catch (InterruptedException e) {  e.printStackTrace()**;** }  }  }   public void stopRunning() {  isRunning = false**;** } }  package 五**;** import java.util.concurrent.TimeUnit**;** class AlarmClockThread extends Thread {  private ElectronicTimerThread electronicTimerThread**;** private static final int *CLOCK\_CHIME\_INTERVAL* = **60;** private static final int *ALARM\_CHIME\_COUNT* = **5;** */\*\*\*  \* 王玉升  \** ***@param*** *electronicTimerThread  \*/* public AlarmClockThread(ElectronicTimerThread electronicTimerThread) {  this.electronicTimerThread = electronicTimerThread**;** }   */\*\*\*  \* 王玉升  \*/* @Override  public void run() {  while (true) {  try {  TimeUnit.*SECONDS*.sleep(*CLOCK\_CHIME\_INTERVAL*)**;** if (System.*currentTimeMillis*() % **60** == **0**) {  for (int i = **0;** i < *ALARM\_CHIME\_COUNT***;** i++) {  System.*out*.println("整点报时")**;** electronicTimerThread.stopRunning()**;** try {  TimeUnit.*SECONDS*.sleep(**1**)**;** } catch (InterruptedException e) {  e.printStackTrace()**;** } finally {  electronicTimerThread.start()**;** }  }  } else {  electronicTimerThread.start()**;** }  } catch (InterruptedException e) {  e.printStackTrace()**;** }  }  } }   * 1. 运行程序和运行结果（截图）： | | | | |
| 总  结 | 实验心得体会： | | | | |