**深 圳 大 学 实 验 报 告**

**课程名称：Java程序设计**

**实验项目名称： 必实验4 线程应用**

**学院： 计算机与软件学院**

**专业： 计算机科学与技术**

**指导教师：卢亚辉**

**报告人： 林宪亮 学号：2022150130 班级：国际班**

**实验时间： 2023年11月23日（周四）~2023年11月29日（周三）**

**实验报告提交时间： 2023年11月27号**

**教务部制**

|  |
| --- |
| **实验目的与要求：**  **实验目的：** 掌握Java程序设计中的线程同步等技术。  **实验要求：**  (1). 运行以下三个程序（每个程序运行10次），并对输出结果给出分析。在报告中附上程序截图和详细的文字说明。（15分）  **程序1:**  **程序2:**  **程序3:**  (2). 编写Java应用程序实现如下功能：第一个线程输出数字1,2,..,12，第二个线程输出英文单词数字和月份One January, Two February, …, Twelve December，输出的顺序和格式为1OneJanuary2TwoFebruary...12TwelveDecember，即每1个数字紧跟着2个英文单词的方式。要求线程间实现通信。要求采用实现Runnable接口和Thread类的构造方法的方式创建线程，而不是通过Thread类的子类的方式。在报告中附上程序截图、运行结果截图和详细的文字说明。（15分）  (3). 编写Java应用程序实现如下功能：创建工作线程，模拟银行现金账户取款操作。多个线程同时执行取款操作时，如果不使用同步处理，会造成账户余额混乱，要求使用syncrhonized关键字同步代码块，以保证多个线程同时执行取款操作时，银行现金账户取款的有效和一致。要求采用实现Runnable接口和Thread类的构造方法的方式创建线程，而不是通过Thread类的子类的方式。在报告中附上程序截图、运行结果截图和详细的文字说明。（25分）  (4). 有一座东西向的桥，只能容纳一个人，桥的东边有20个人（记为E1,E2,…,E20）和桥的西边有20个人（记为W1,W2,…,W20），编写Java应用程序让这些人到达对岸，每个人用一个线程表示，桥为共享资源，在过桥的过程中输出谁正在过桥（不同人之间用逗号隔开）。运行10次，分别统计东边和西边的20人先到达对岸的次数。要求采用实现Runnable接口和Thread类的构造方法的方式创建线程，而不是通过Thread类的子类的方式。在报告中附上程序截图、运行结果截图和详细的文字说明。（25分）  报告写作。要求：主要思路有明确的说明，重点代码有详细的注释，行文逻辑清晰可读性强，报告整体写作较为专业。（20分）  **说明：**  （1）本次实验课作业满分为100分，占总成绩的比例（待定）。  （2）本次实验课作业截至时间2022年11月29日（周三）21:59。  （3）报告正文：请在指定位置填写，本次实验**需要单独提交源程序文件**（源程序单独打包在Blackboard中上传，不要包含外部导入的包）。  （4）个人信息：WORD文件名中的“姓名”、“学号”，请改为你的姓名和学号；实验报告的首页，请准确填写“学院”、“专业”、“报告人”、“学号”、“班级”、“实验报告提交时间”等信息。  （5）提交方式：截至时间前，请在Blackboard平台中提交。  （6）发现抄袭（包括复制&粘贴整句话、整张图），**抄袭者和被抄袭者的成绩记零分（含抄袭往届同学的作业）。**  （7）延迟提交，不得分；如有特殊情况，请于截至日期之后的48小时内发邮件到luyahui@szu.edu.cn，并在邮件中注明课程名称、作业名称、姓名、学号等信息，以及特殊情况的说明，我收到后会及时回复。  （8）期末考试阶段补交无效。 |

|  |
| --- |
| (1). 运行以下三个程序（要求每个程序运行10次），并对输出结果给出分析。在报告中附上程序截图和详细的文字说明。（15分）  **程序1:**  **程序截图**：  PrintChar类：    PrintNum类：    TaskThreadDemo类：    **结果（每行太长，就不采用截图）：**  第一次： abbbbbbbbbbbbbaaaaaaaaaaabbbbbbbbbbbbbbbbabbaaaaaaaaaaabbbbbaaaaaaaaabbbbbbbaaaabbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbaaaaaaaaaaabbbbaaabbbbaaabbbbaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100  第二次：  abbbaaaaaaaaaaabbbbbbbbbbaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaabbbbaaaaaabbbbbaaaaaaabbbbbbbaaaaaaaaaabbbbbbbbbaaaaaaaaaaabbbbbbbbbbbbbabbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100  第三次：  bbbaaaaaaaaaaaabbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbaaabbbbbbbbbbbbbaaaaaaaaaaaaaaaaabbbbbbbbbbbbbbbbbbaaaaabbbbbbaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaabbbbbbbbbbbbbbbbbbbbbbbbbbbbaaaa 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100  这里只展示三次结果，省略剩余的七次。  **说明：**  这个Java程序演示了使用线程同时打印字符和数字。  PrintChar类：这个类实现了Runnable接口，这意味着它可以用来创建线程。它有两个私有字段，charToPrint和times，分别表示要打印的字符和要打印的次数。run方法包含一个循环，打印指定次数的指定字符。  PrintNum类：类似于PrintChar，这个类也实现了Runnable接口。它有一个私有字段lastNum，表示要打印的最后一个数字。run方法包含一个循环，打印从1到指定最后一个数字的数字。  TaskThreadDemo类：这是包含main方法的主类，程序从这里开始。它创建了PrintChar和PrintNum的实例，并使用它们创建了三个线程（thread1、thread2和thread3）。  在主方法中，它启动了这三个线程，使它们并行运行。这样，就可以同时打印字符 'a' 和 'b'，以及从1到100的数字。在上面展示的结果中，我们可以发现，程序会同时打印字符‘a’和‘b’，然后再打印数字，并且这个与编程时语句的书写顺序没有关系。  **程序2:**  **程序截图：**  PrintChar类：    PrintNum类：  ExecutorDemo类：  **结果（每行太长，就不采用截图）：**  第一次：  aabbbaaaaaaabbbaaaaabbbbbbbbbbbbbbbbbaaaaaaaaaaaaaaaaabbbbbbbbbbbaaaaaaaaaaabbbbbbbbbbbbbbbbbbaaaaaaaaabbbbbbbbbbbbbbbbbaaabbbbbbbbbbbbbbbbbbbbbbbbaaaaaaaabbbbbbbaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100  第二次：  aaaaaaaaabbbbbbbbbaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaabbbbbbaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaabbbbbbbbbbbbbbbaaaaaaabbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100  第二次：  baaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaabbbbbbbbbbbaaaaaaaabbbbbaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaabbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100  第三次：  aaaaaaaaaaaaaaaaaabbbbbbbbbbbbbbbbbbbbbbbbbbaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaabbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaabbbbbbbbbbbbbb 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100  这里只展示三次结果，省略剩余的七次。  **说明：**  程序2与程序1类似。都是使用线程同时打印字符和数字。  ExecutorDemo 类说明:  ExecutorDemo 类是程序的主类，演示了如何使用 Executor 框架来管理线程的执行。  在 main 方法中，首先创建了一个具有固定大小（3个线程）的线程池。  线程池创建:  ExecutorService executor = Executors.newFixedThreadPool(3);  通过 Executors.newFixedThreadPool(3) 创建了一个固定大小的线程池，其中包含3个线程。  任务提交和执行:  executor.execute(new PrintChar('a', 100));创建并执行了一个打印字符 'a' 的任务，该任务由 PrintChar 类实现。  executor.execute(new PrintChar('b', 100));创建并执行了一个打印字符 'b' 的任务，同样由 PrintChar 类实现。  executor.execute(new PrintNum(100));创建并执行了一个打印数字序列的任务，由 PrintNum 类实现。  线程池关闭:  executor.shutdown();调用 shutdown() 方法关闭线程池。这表示线程池不再接受新的任务，同时会等待已提交的任务执行完成。  在上面展示的结果中，我们可以发现，程序会同时打印字符‘a’和‘b’，然后再打印数字，并且这个与编程时语句的书写顺序没有关系。  **程序3:**  **程序截图：**  AccountWithoutSync：  AddAPennyTask：  Account：  **结果：**  第一次：  第二次：  第三次：  第四次：  第五次：  第六次：  第七次：  第八次：  第九次：  第十次：  **说明：**  创建线程池：  ExecutorService executor = Executors.newCachedThreadPool();  创建一个可缓存的线程池，它会根据需要创建新线程。  提交任务到线程池：  for (int i = 0; i < 100; i++) { executor.execute(new AddAPennyTask()); }  向线程池提交了100个任务，每个任务都是 AddAPennyTask 类的实例，它的 run 方法调用 account.deposit(1); 来向账户中存入1元。  关闭线程池并等待任务完成：  executor.shutdown();  关闭线程池，表示不再接受新的任务。  while (!executor.isTerminated()) {}  在这个循环中，主线程等待线程池中的所有任务执行完毕。这是通过检查 executor.isTerminated() 的返回值来完成的。  输出最终账户余额：  System.out.println("What is the balance? " + account.getBalance());  输出最终的账户余额。  AddAPennyTask类：  private static class AddAPennyTask implements Runnable { ... }  AddAPennyTask 类是一个实现 Runnable 接口的任务，其 run 方法调用 account.deposit(1);，模拟存入1元钱的操作。  Account类：  private static class Account { ... }  Account 类表示银行账户，包含了一个余额字段和存款方法。  存款方法 deposit 用于模拟存款过程，其中包含了一个短暂的延迟（Thread.sleep(5);）。  由于这个程序没有同步机制，多个线程可能同时执行 deposit 方法，导致最终的账户余额不是我们预期的100，而是1或者2。  (2). 编写Java应用程序实现如下功能：第一个线程输出数字1,2,..,12，第二个线程输出英文单词数字和月份One January, Two February, …, Twelve December，输出的顺序和格式为1OneJanuary2TwoFebruary...12TwelveDecember，即每1个数字紧跟着2个英文单词的方式。要求线程间实现通信。要求采用实现Runnable接口和Thread类的构造方法的方式创建线程，而不是通过Thread类的子类的方式。在报告中附上程序截图、运行结果截图和详细的文字说明。（15分）  **程序截图：**  **结果截图：**  **说明：**  这段代码实现了两个线程交替打印数字和月份的功能，格式为 "OneJanuary, TwoFebruary, ..., TwelveDecember"。以下是对代码的详细说明：  TwoThread 类：  实现了 Runnable 接口，用于创建两个线程，一个用于打印数字，另一个用于打印月份。  包含一个字符串数组 months 存储月份的英文表示，以及一个表示最后一个数字的变量 lastNum。  run 方法：  使用 synchronized (this) 确保线程安全，避免两个线程同时访问共享资源。  通过 Thread.currentThread().getName().equals("Thread-1") 判断当前是打印字符的线程还是打印数字的线程。  第一个线程打印字符，第二个线程打印数字，通过 wait() 和 notify() 实现线程的交替执行。  notify() 用于唤醒等待的线程，wait() 用于使当前线程等待。  main 方法：  创建 TwoThread 类的实例 twoThread。  创建两个线程，分别用于执行 twoThread 实例的 run 方法。  启动两个线程，使其并发执行。  InterruptedException 处理：  在 wait() 和 notify() 方法中，捕获了 InterruptedException 异常，但是并没有具体处理。线程的命名：  通过 Thread.currentThread().getName() 判断当前线程的名称。在 main 方法中，创建的两个线程分别被命名为 "Thread-0" 和 "Thread-1"。  如此，由于我们的Thread1会先开始执行run方法，而它的名字是”thread-0”,所以进行判断之后，会开始输出数字信息，输出一个数字后会使用notify（）方法使得Thread2线程开始执行，打印月份信息，然后使用wait（）方法使自己进入等待状态，对于Thread2也是一样的，它打印一次月份信息就会使用notify（）方法唤醒Thread1线程，然后使用wait（）方法使自己进入等待。依次交替，实现数字和月份信息的交替输出。Notify（）方法唤醒的线程取决于线程调度器，而这里只有两个线程，所以只会唤醒另一个线程。  (3). 编写Java应用程序实现如下功能：创建工作线程，模拟银行现金账户取款操作。多个线程同时执行取款操作时，如果不使用同步处理，会造成账户余额混乱，要求使用syncrhonized关键字同步代码块，以保证多个线程同时执行取款操作时，银行现金账户取款的有效和一致。要求采用实现Runnable接口和Thread类的构造方法的方式创建线程，而不是通过Thread类的子类的方式。在报告中附上程序截图、运行结果截图和详细的文字说明。（25分）  **程序截图：**  Bank：  主函数:  Withdraw线程类：  **结果截图：**  根据我们的取款记录：  三个账号的余额都是正确的。  **说明：**  这是一个Java程序，模拟了银行现金账户的取款操作，并使用了synchronized关键字确保多个线程同时执行取款操作时的安全性。以下是代码的详细文字说明：  Bank 类：  类中包含私有成员变量 balance 和 name，分别表示银行账户余额和名称。  构造方法 public Bank(int initialBalance, String name) 用于初始化银行账户余额和名称。  withdraw 方法是一个同步方法，使用了synchronized关键字，确保在同一时刻只有一个线程能够执行该方法。该方法模拟取款操作，打印取款信息和余额，或者在余额不足时打印失败信息。  Withdraw 类：  实现了 Runnable 接口，用于创建线程来模拟取款操作。  构造方法 public Withdraw(Bank account, int amount) 用于初始化银行账户和取款金额。  run 方法包含了对 account 对象调用 withdraw 方法的操作，即执行取款操作。  main 方法：  在 main 方法中，创建了三个银行账户实例：account1、account2、account3。  创建了九个线程，分别对这三个账户进行取款操作。  使用 start() 方法启动这九个线程，并使用 join() 方法等待所有线程执行完成。  在所有线程执行完成后，输出最终的账户余额信息。  同步方法的使用：  withdraw 方法中使用了 synchronized 关键字，确保了多个线程在同时执行取款操作时，对账户余额的访问是安全的，避免了竞争条件和数据不一致的问题。  线程的命名：  每个线程在创建时都被命名为 "Thread-X"，其中 X 为线程的编号。  最终输出：  在所有线程执行完成后，通过 System.out.println 输出每个账户的最终余额信息。  根据上面的输出结果，我们不难发现，同一个账户，只有当上一次取钱操作结束之后，下一次取钱操作才会开始，而对于不同的账户，取钱操作则可以同时发生。如果我们去掉了synchronized 关键字，那么就会是以下的结果：  同一个账户会同时进行多个取钱操作，这就会导致一些不安全性，造成账户混乱。  (4). 有一座东西向的桥，只能容纳一个人，桥的东边有20个人（记为E1,E2,…,E20）和桥的西边有20个人（记为W1,W2,…,W20），编写Java应用程序让这些人到达对岸，每个人用一个线程表示，桥为共享资源，在过桥的过程中输出谁正在过桥（不同人之间用逗号隔开）。运行10次，分别统计东边和西边的20人先到达对岸的次数。要求采用实现Runnable接口和Thread类的构造方法的方式创建线程，而不是通过Thread类的子类的方式。在报告中附上程序截图、运行结果截图和详细的文字说明。（25分）  **代码截图：**  BridgeProblem：  主函数：  Bridge：  Person：  **结果截图：**  省略后面的五次运行结果。  记录东边和西边的20人先到达对岸的次数：  多运行多几次结果：  说明：  BridgeProblem 类：  初始化共享资源和统计变量：  Bridge bridge = new Bridge();  创建了 Bridge 对象作为共享资源，用于模拟桥。  eastCount、westCount、westFirst 和 eastFirst 分别用于记录东边和西边到达对岸的人数，以及西边和东边先到达对岸的次数。  循环运行模拟桥过河的逻辑：  for (int j = 1; j <= 10; j++) {  使用 for 循环运行10次模拟，每次模拟桥上的人过桥的情况。  输出模拟次数：  System.out.println("NO " + j + " time:");  输出当前模拟的次数。  重置统计变量：  eastCount = 0;  westCount = 0;  在每次模拟开始前，重置东边和西边到达对岸的人数。  创建人和线程：  Thread[] eastPerson = new Thread[21];  Thread[] westPerson = new Thread[21];  创建数组存储20个东边和20个西边的人对应的线程。  for (int i = 1; i <= 20; i++) {  eastPerson[i] = new Thread(new Person("E" + i, bridge, true));  westPerson[i] = new Thread(new Person("W" + i, bridge, false));  // 启动线程  eastPerson[i].start();  westPerson[i].start();  }  使用循环创建20个东边和20个西边的人，每个人通过 Person 类的线程表示，传入人名、桥对象和方向参数。  启动每个人的线程，使他们并发执行。  等待线程执行完成：  for (int i = 1; i <= 20; i++) {  try {  eastPerson[i].join();  westPerson[i].join();  } catch (InterruptedException e) {  e.printStackTrace();  }  }  使用 join 方法等待每个人的线程执行完成，确保每个模拟步骤都完成后再进行下一次。  输出每次模拟的人数统计：  System.out.println("\nEast side count: " + eastCount);  System.out.println("West side count: " + westCount);  输出当前模拟的东边和西边人数统计。  输出最终统计结果：  System.out.println("\nEast side first count: " + eastFirst);  System.out.println("West side first count: " + westFirst);  输出最终统计的东边和西边先到达对岸的次数。  Bridge 类：  crossBridge 方法是一个同步方法，确保桥只能同时容纳一个人。  模拟人通过桥的情况，输出人名和过桥信息，每经过5人换行。  统计到达对岸的人数，若有20个人都到达对岸，则使该边的先到达对岸次数加一。  crossBridge 方法使用了 synchronized 关键字，确保桥只能同时容纳一个人，防止并发访问问题。  模拟过桥的情况：  在每次人通过桥的过程中，输出人名和过桥信息，每经过5人换行（为了方便截图与查看信息），通过这种方式模拟了桥上人的情况。  Person 类：  run 方法实现了 Runnable 接口，模拟人过桥的情况。  通过 bridge.crossBridge 调用桥的同步方法，表示人通过桥。  通过结果我们不难看出，绝大部分情况下，东边的人都要比西边的人先到达对岸。  ++++++++++++++++++++++++++++++++++++++++++++++++++++++  **其他（例如感想、建议等等）。**  本次实验，我初步认识学习了多线程以及它的作用，多线程可以运行程序的多个部分一起执行，这样可以提高程序的并发性，这样可以提高用户体验，就如我们的电脑，如果一次只能运行一个app，那么功能就会十分差劲，用户体验很差。  多线程丰富了程序的功能，但是也使程序的不达预期变得更加难以查找错误，尤其是现在刚开始学习的阶段，对于这种机制还是不熟悉，第一个任务还好，有老师提供的代码，只需要抄一次运行得到结果，然后搞清楚程序做了什么，是怎么个并行法就行了。但是到了后面自己写的程序，总是会受到同步机制的影响，产生许多错误使得程序的结果，输出和预期很不一样。就比如Bridge那题，在记录东西岸先到达对岸的次数时，开始程序会输出东边一次，西边两次，这很明显和我们的预期不一致，东西边先到达的次数的和应该是10才可能正确，后来把判断是否到达次数加一的语句也写到那个同步机制的函数中，才避免了这个错误，正确的输出了结果，所以，为了避免同步机制产生一些错误，最好把所有run的内容都变成同步的。 |

深圳大学学生实验报告用纸

|  |
| --- |
| 指导教师批阅意见：  成绩评定：  指导教师签字：  2021年 月 日 |
| 备注： |

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。

2、教师批改学生实验报告时间应在学生提交实验报告时间后10日内。