哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

2018 年春季学期

计算机学院大二软件构造课程

Lab 6 实验报告

姓名	穆添愉
学号	1160301008
班号	1603010
电子邮件	1417553133@qq.com
手机号码	15636094072

目录

1 实验目标概述

本次实验训练学生的并行编程的基本能力,特别是 Java 多线程编程的能力。根据一个具体需求,开发两个版本的模拟器,仔细选择保证线程安全(threadsafe)的构造策略并在代码中加以实现,通过实际数据模拟,测试程序是否是线程安全的。另外,训练学生如何在 threadsafe 和运行性能之间寻求较优的折中,为此计算吞吐率等性能指标,并做仿真实验。

(1)Java 多线程编程

(2)面向线程安全的 ADT 设计策略选择、文档化

(3)模拟仿真实验与对比分析

(4)基本的 GUI 编程

2 实验环境配置

仓库链接如下:

https://github.com/ComputerScienceHIT/Lab6-1160301008

3 实验过程

3.1 ADT 设计方案

ADT 设计如下:

(1) Ladder 类

作用: 表示一个梯子

属性:

```
private final String name;
private final int rung;
private final List<Monkey> rungs = Collections.synchronizedList(new ArrayList<>());
```

方法:

```
public Ladder(String name, int rung);

public int getLength();

public String getName();

public List<Monkey> getRungs();

public synchronized boolean empty();

public synchronized boolean setMonkey(int location, Monkey m);

public synchronized int getTotalNumber();
```

specification:

首先有如下的 AF, RI 等

```
//Rep Invariant
//    assert rung >= 1
//    name != null;
//Abstraction Function
//    represent the ladder.
//Safety form rep exposure
//    name, rung, rungs are all private and final.
//Thread safety argument
//    all accesses to rungs are protected by the Ladder lock.
```

各个方法的 spec 如下:

```
/**
 * the constructor of class Ladder
 * @param name, the name.
 * @param rung, requires >= 1.
 */
```

```
/**
 * get the length of the ladder.
 * @return the length.
 */
```

```
/**
 * get the name of the ladder.
 * @return the name.
 */
```

```
/**
 * return the monkey and location.
 * @return rungs.
 */
```

```
/**
 * test the ladder is empty or not.
 * @return true if there is no monkey on it, otherwise false.
 */
```

```
/**
 * set a monkey on the ladder.
 * @param location, the location we hope the monkey at.
 * @param m, the monkey.
 * @return true if we can set the monkey on the ladder successfully, otherwise false.
 */
```

```
/**
 * get the total number of the monkeys on the ladder.
 * @return the total number.
 */
```

(2) Monkey 类

作用：表示一只猴子

属性：

```
private final String direction;
private final int name;
private final int bornTime;
private int speed;
private int speedTime;
private int location;
private Ladder ladder;
private List<Ladder> allLadders;
private Logger logger;
private CountDownLatch doneSignal;
```

方法：

```
public Monkey(String direction, int name, int bornTime, int speed, List<Ladder> allLadders);

public int getName();

public int getSpeed();

public String getDirection();

public int getBornTime();

public int getTime();

public int getTotalTime();

public void setLog(Logger logger);

public void setCountDownLaunch(CountDownLatch doneSignal);

public void run();
```

spec：

首先有如下的 AF RI

```
//Abstraction Function
//    represent the monkey.
//Rep Invariant
//    assert name >= 1.
//    direction can only be "L->R" or "R->L".
//    assert bornTime >= 0.
//Safety from exposure
//    all fields are private and final.
//Thread safety argument
//    when the monkey modify the ladder, use the lock, so all operations
//    are protected by the lock.
```

```
/**
 * the constructor of the class Monkey.
 * @param direction, the direction of the movement, which can only be "L->R" or "R->L".
 * @param name, the name of the monkey, which must be an Integer.
 * @param bornTime, the born time of the monkey.
 * @param speed, the speed of the monkey, requires 1 <= speed <= MV.
 * @param allLadders, requires >= 1.
 */
```

```
/**
 * get the name of the monkey.
 * @return the name.
 */
```

```
/**
 * get the speed of the monkey.
 * @return the speed.
 */
```

```
/**
 * get the direction of the monkey.
 * @return the direction.
 */
```

```
/**
 * get the born time of the monkey.
 * @return the bornTime.
 */
```

```
/**
 * get the time from the monkey born to leaving the ladder.
 * @return the speedTime.
 */
```

```
/**
 * count the number of the monkeys which have already leave the ladder.
 * @param doneSignal
 */
```

(3) MonkeyGenerator 类

作用：生成猴子

属性：

```
private final int timeInterval;
private final int k;
private final int maxNum;
private final int maxSpeed;
private List<Monkey> monkeys = Collections.synchronizedList(new ArrayList<>());
private List<Ladder> ladders;
private Logger logger;
```

方法：

```
public MonkeyGenerator(int timeInterval, int k, int maxNum,
int maxSpeed);
public void setLadders(List<Ladder> ladders);
public void setLog(Logger logger);
public void run();
```

spec：

首先有如下的 AF RI

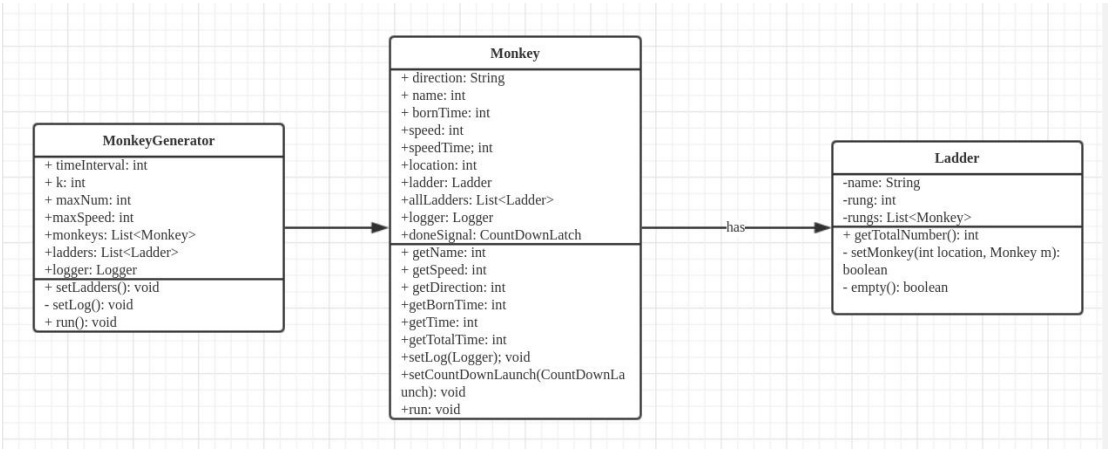
```
//Abstraction Function
//    represents the monkey generator.
//Rep Invariant
//    assert the timeInterval, k, maxNum and maxSpeed > 0.
//Safety from rep exposure
//    all fields are private and final.
//Thread safety argument
//    all operations are protected by the lock.
```

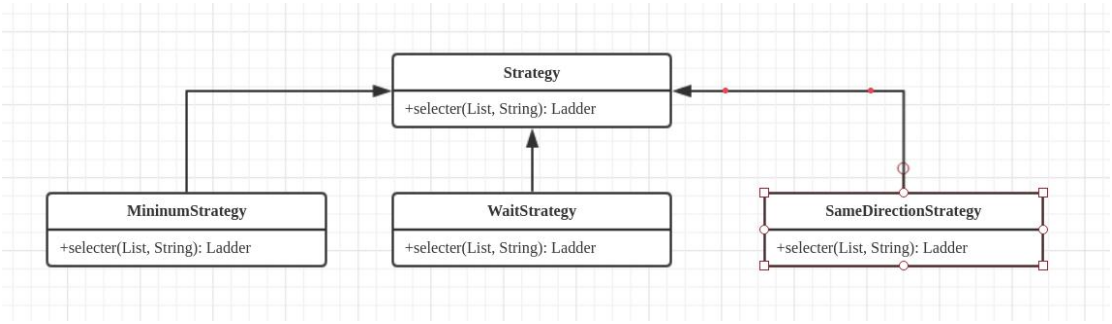
```
/**
 * the constructor of the class MonkeyGenerator.
 * @param timeInterval
 * @param k
 * @param maxNum
 * @param maxSpeed
 */
```

```
/**
 * set the ladders.
 * @param ladders
 */
```

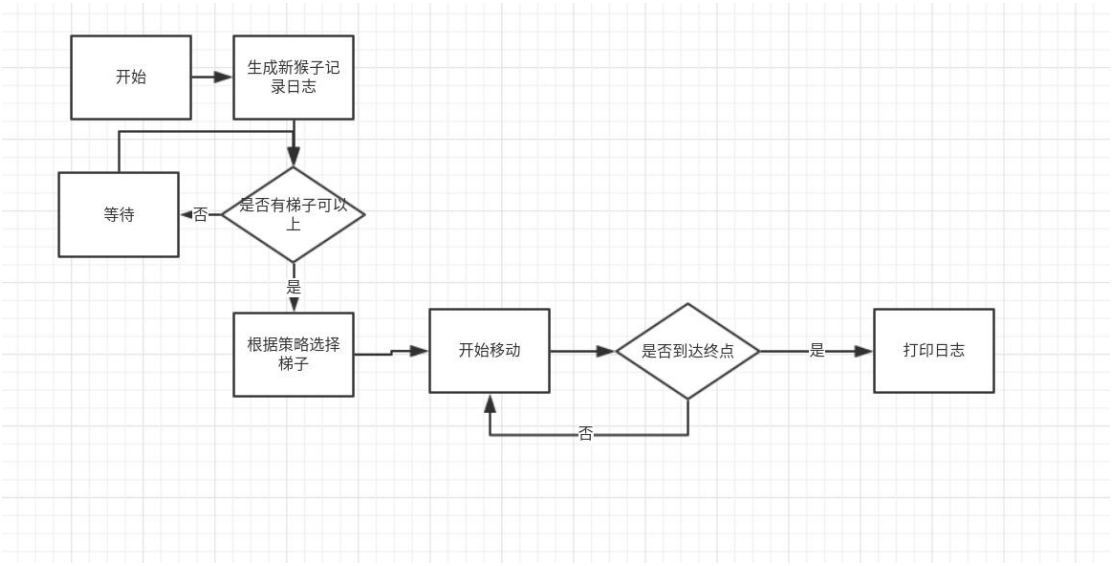
```
/**
 * set the logger.
 * @param logger
 */
```

（可选）以类图形式给出多个类之间的关系。





3.2 Monkey 线程的 run()的执行流程图



3.3 至少两种“梯子选择”策略的设计与实现方案

3.3.1 策略 1

优先选择没有猴子的梯子，若所有梯子上都有猴子,则优先选择没有与我对向而行的猴子的梯子;若满足该条件的梯子有很多,则随机选择;

只需要遍历所有梯子，如果有梯子为空，这直接选择这个梯子，否则就去选方向一致的梯子.

代码如下:

```
public class SameDirectionStrategy implements Strategy{
  @Override
  public Ladder selecter(List<Ladder> ladders, String direction) {
    synchronized (ladders) {
      for(int i = 0; i < ladders.size(); ++i) {
        if(ladders.get(i).empty())
          return ladders.get(i);
      }
      for(int i = 0; i < ladders.size(); ++i) {
        List<Monkey> rungs = ladders.get(i).getRungs();
        for(int j = 0; j < rungs.size(); ++j) {
          if(rungs.get(j) != null && rungs.get(j).getDirection().equals(direction))
            if(direction.equals("L->R") && rungs.get(0) == null)
              return ladders.get(i);
            if(direction.equals("R->L") && rungs.get(ladders.get(i).getLength() - 1
              return ladders.get(i);
          }
        }
      }
    }
    return null;
  }
}
```

3.3.2 策略 2

优先选择整体推进速度最快的梯子(没有与我对向而行的猴子、其上的猴子数量最少、梯子离我最近的猴子的眞实行进速度最快);

代码如下:

```
public class MininumStrategy implements Strategy {
  @Override
  public Ladder selecter(List<Ladder> ladders, String direction) {
    synchronized (ladders) {
      for(int i = 0; i < ladders.size(); ++i) {
        if(ladders.get(i).empty())
          return ladders.get(i);
      }
      List<Ladder> list = new ArrayList<>();
      for(int i = 0; i < ladders.size(); ++i) {
        List<Monkey> rungs = ladders.get(i).getRungs();
        for(int j = 0; j < rungs.size(); ++j) {
          if(rungs.get(j) != null && rungs.get(j).getDirection().equals(direction)) {
            if(direction.equals("L->R") && rungs.get(0) == null) {
              list.add(ladders.get(i));
            }
            if(direction.equals("R->L") && rungs.get(ladders.get(i).getLength() - 1) == null) {
              list.add(ladders.get(i));
            }
          }
        }
      }
      Ladder l = null;
      int num = 1000;
      for(Ladder ladder: list) {
        if(ladder.getTotalNumber() < num) {
          num = ladder.getTotalNumber();
          l = ladder;
        }
      }
      return l;
    }
  }
```

3.3.3 策略 3（可选）

优先选择没有猴子的梯子，也就是空梯子，如果没有空梯子，再选择之前的两种

```
public class WaitStrategy implements Strategy {
  @Override
  public Ladder selecter(List<Ladder> ladders, String direction) {
    synchronized (ladders) {
      for(int i = 0; i < ladders.size(); ++i) {
        if(ladders.get(i).empty())
          return ladders.get(i);
      }
    }
    return null;
  }
}
```

3.4 “猴子生成器”MonkeyGenerator

首先看构造器，接受四个参数，timeInterval，k，maxNum，maxSpeed

timeInterval 为生成每一批猴子的时间间隔，k 为每一次生成猴子的个数，maxNum 为一共的猴子个数，maxSpeed 为每只猴子的最大速度.

有如下两个方法：

```
/**
 * set the ladders.
 * @param ladders
 */
public void setLadders(List<Ladder> ladders) {
  this.ladders = ladders;
}

/**
 * set the logger.
 * @param logger
 */
public void setLog(Logger logger) {
  this.logger = logger;
}
```

分别用来进行接收所有的梯子和设置 logger 用来进行记录日志.

接下来就是重写的 run()方法

有一个内部计数器，用来判断生成的猴子数是不是已经达到了最大的数字，并且可以根据这个计数器来为猴子进行命名，满足按照时间顺序递增的要求，每生成一只猴子就进行记录日志

```
while (num + k <= maxNum) {
  for(int i = 0; i < k; ++i) {
    ++num;
    int speed = random.nextInt(maxSpeed) + 1;
    String direction = speed % 2 == 0 ? "R->L" : "L->R";
    Monkey monkey = new Monkey(direction, num, bornTime, speed, ladders);
    monkey.setLog(logger);
    monkey.setCountDownLaunch(doneSignal);
    monkeys.add(monkey);
    (new Thread(monkey)).start();
  }
  try {
    bornTime += timeInterval;
    Thread.sleep(timeInterval * 1000);
  } catch (Exception e) {
    e.printStackTrace();
  }
}
```

接下来是进行日志记录的部分，计算吞吐率，公平性

根据给出的公式进行计算即可，代码如下：

```
    logger.info("total time: " + totalTime);
    logger.info("Throughput rate: " + maxNum / totalTime);
    int fair = 0;
    for(int i = 0; i < monkeys.size() - 1; i++) {
      for(int j = i + 1; j < monkeys.size(); j++) {
        if((monkeys.get(j).getTime() - monkeys.get(i).getTime()) * (monkeys.get(j).getBornTime() - monkeys.get(i).getBornTime()) >
          fair += 1;
        } else {
          fair -=1;
        }
      }
    }
    int x = maxNum  * (maxNum - 1) / 2;
    logger.info("fairness: " + (double)fair / x);
```

3.5 如何确保 threadsafe？

首先确认竞争的来源，就是 Monkey 之间对于 Ladders 的竞争，所以只需要考虑对于梯子进行操作的时候加上锁就好.

(1) Monkey 对于梯子的竞争

Monkey 在进行选择梯子的时候，要对梯子进行加锁，这样子就可以防止多线程带来的一系列弊端.

```
    while(ladder == null) {
      synchronized (this) {
        ladder = strategy.selecter(allLadders, direction);
        if(ladder == null) {
          try {
            Thread.sleep(1000);
            ++time;
            System.out.println("name: " + name + ", wait time: " + time +
                ", direction: " + direction + ", strategy: " + strategyName);
            logger.info("name: " + name + ", wait time: " + time +
                ", direction: " + direction + ", strategy: " + strategyName);

          } catch (Exception e) {
            e.printStackTrace();
          }
        } else {
          synchronized (ladder) {
            if(direction.equals("L->R"))
              ladder.setMonkey(0, this);
            else ladder.setMonkey(ladder.getLength() - 1, this);
          }
        }
      }
    }
```

(2) 多只猴子在梯子上过河时候，由于速度不同，对于踏板的竞争

```
  synchronized (ladder) {
    List<Monkey> rungs = ladder.getRungs();
    for(int i = location - 1; i >= 0; --i) {
      if(rungs.get(i) != null && rungs.get(i).getSpeed() < speed) {
        synchronized (this) {
          speed = rungs.get(i).getSpeed();
        }
        break;
      }
    }
    if(location - speed > 0) {
      ladder.setMonkey(location, null);
      ladder.setMonkey(location - speed, this);
      location -= speed;
    } else {
      ladder.setMonkey(location, null);
      String message = "name: " + name + ", direction: "
          + direction + ", spend time: " + time + ", strategy: "
          + strategyName;
      System.out.println(message);
      logger.info(message);
      this.speedTime = time;
      doneSignal.countDown();
      break;
    }
  }
```

只要猴子在梯子上，就将其即将要进行走的踏板加上锁.

3.6 系统吞吐率和公平性的度量方案

可以根据实验手册中给出的公式来进行计算

吞吐率表示每秒过河的猴子的数目，用公式 N/T 表示

公平性$F = \Sigma(A,B) \in \Theta\ F(A,B)/n/(n-1)*2$，$\Theta = \{(A, B)|A \neq B, (B, A) \notin \Theta\}$，

计算方法：

吞吐率：求出找出所有猴子的最大到达时间 longestTime，这个时间就是猴子们过桥的总时间，然后用 N/(double)longestTime 即可求得吞吐率

公平性：对于每一个猴子，让其跟 id 大于它的猴子比较时间，如果这两个猴子的 BirthTime 和 ArriveTime 不满足公平性的要求，那么 adjustment 减一，否则加一，最后除以即可

- “吞吐率”是指：假如N只猴子过河的总耗时为T秒，那么每只猴子的平均耗时为$X = \frac{T}{N}$秒，则吞吐率$Th = \frac{N}{T}$表征每秒钟可过河的猴子数目。
- “公平性”是指：如果 Monkey 对象 A 比 Monkey B 出生得更早，那么 A 应该比 B 更早抵达对岸，则为“公平”；若 A 比 B 晚到对岸，则为“不公平”。设 A 和 B 的产生时间分别为Y_a和Y_b，抵达对岸的时间分别为Z_a和Z_b，那么公平性$F(A,B) = \begin{cases} 1, & if\ (Y_b - Y_a) * (Z_b - Z_a) \geq 0 \\ -1, & otherwise \end{cases}$。对$N$只猴子两两计算其之间的公平性并综合到一起，得到本次模拟的整体公平性$F = \frac{\sum_{(A,B)\in\Theta} F(A,B)}{C_N^2}, \Theta = \{(A,B)|A \neq B, (B,A) \notin \Theta\}$，其取值范围为$[-1,1]$。

3.7 输出方案设计

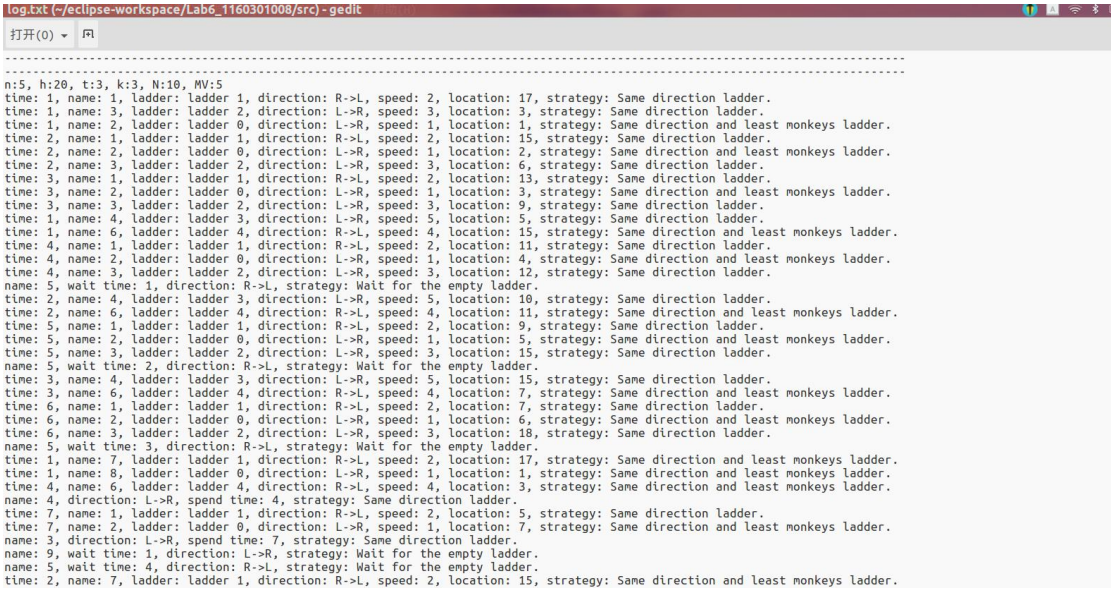
选择日志输出

代码如下：

```
1 package v1;
2
3 import java.util.logging.Formatter;
5
6 public class LogFormat extends Formatter{
7   @Override
8   public String format(LogRecord lr) {
9     return lr.getMessage() + "\r\n";
10  }
11 }
12
```

每一步操作都进行日志记录

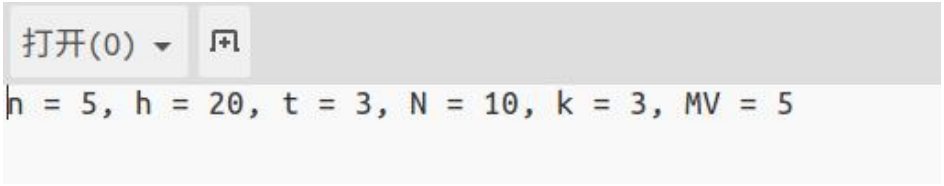
最终日志文件如下：



3.8 猴子过河模拟器 v1

3.8.1 参数如何初始化

采用从文件中读取的形式来进行参数的初始化，参数文件如下：



其中，采取正则表达式进行字符串的匹配，具体代码如下：

```
BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(new FileInputStream(filePath)));
String string = bufferedReader.readLine();
Matcher matcher = null;
if ((matcher = Pattern.compile(
    "^n\\s*=\\s*(\\d+),\\s*h\\s*=\\s*(\\d+),\\s*t\\s*=\\s*(\\d+),\\s*N\\s*=\\s*(\\d+),\\s*k\\s*=\\s*(\\d+),\\s*MV\\s*=\\s*(\\d-
    .matcher(string)).find()) {
  int ladderNum = Integer.parseInt(matcher.group(1));
  int rung = Integer.parseInt(matcher.group(2));
  int t = Integer.parseInt(matcher.group(3));
  int k = Integer.parseInt(matcher.group(5));
  int maxNum = Integer.parseInt(matcher.group(4));
  int maxSpeed = Integer.parseInt(matcher.group(6));
```

这样只需要修改文件中的参数，就可以进行多次的实践.

3.8.2 使用 Strategy 模式为每只猴子随机选择决策策略

首先设计 Strategy 接口，其中只有一个方法用来选择梯子，代码如下：

```java
package v1;

import java.util.List;

public interface Strategy {
  /**
   * according different strategies, select the ladder in all ladders.
   * @param ladders, all ladders waiting to be chosen
   * @param direction, the direction that monkey moves, which can only be "L->R" or "R->L".
   * @return a ladder chosen according to the strategy.
   */
  public abstract Ladder selecter(List<Ladder> ladders, String direction);
}
```

然后有三种 Strategy 类分别用不同的实现策略去实现 select 方法，代码如下：

```java
import java.util.List;
import java.util.PrimitiveIterator.OfDouble;

public class WaitStrategy implements Strategy {
  @Override
  public Ladder selecter(List<Ladder> ladders, String direction) {
    synchronized (ladders) {
      for(int i = 0; i < ladders.size(); ++i) {
        if(ladders.get(i).empty())
          return ladders.get(i);
      }
    }
    return null;
  }
}
```

```java
import java.util.ArrayList;

public class MininumStrategy implements Strategy {
  @Override
  public Ladder selecter(List<Ladder> ladders, String direction) {
    synchronized (ladders) {
      for(int i = 0; i < ladders.size(); ++i) {
        if(ladders.get(i).empty())
```

```java
public class SameDirectionStrategy implements Strategy{
  @Override
  public Ladder selecter(List<Ladder> ladders, String direction) {
    synchronized (ladders) {
      for(int i = 0; i < ladders.size(); ++i) {
        if(ladders.get(i).empty())
```

3.9 猴子过河模拟器 v2

在不同参数设置和不同“梯子选择”模式下的“吞吐率”和“公平性”实验结果

及其对比分析。

3.9.1 对比分析：固定其他参数，选择不同的决策策略

基础参数设置如下:

```
打开(0)
n = 5, h = 20, t = 3, N = 10, k = 3, MV = 5
```

Strategy1:

```
total time: 9.0
Throughput rate: 1.1111111111111112
fairness: 0.24444444444444444
```

Strategy2:

```
total time: 9.0
Throughput rate: 1.1111111111111112
fairness: 0.37777777777777777
```

Strategy3:

```
total time: 9.0
Throughput rate: 1.1111111111111112
fairness: 0.4222222222222222
```

3.9.2 对比分析：变化某个参数，固定其他参数

(1)首先固定其他参数，只改变 N

当 N = 20 时:

```
total time: 18.0
Throughput rate: 1.1111111111111112
fairness: 0.08421052631578947
```

当 N = 40 时:

```
total time: 39.0
Throughput rate: 1.0256410256410255
fairness: -0.16923076923076924
```

(2)固定其他参数，只改变 n

当 n = 1 时：

```
total time: 30.0
Throughput rate: 1.0
fairness: 0.32873563218390806
```

当 n = 2 时：

```
total time: 30.0
Throughput rate: 1.0
fairness: 0.0022988505747126436
```

当 n = 4 时：

```
total time: 28.0
Throughput rate: 1.0714285714285714
fairness: 0.04367816091954023
```

3.9.3 分析：吞吐率是否与各参数/决策策略有相关性？

根据上述结果，可以看出来吞吐率和参数之间的依赖性还是比较大的，比如说和 n 还有 N 都有依赖性，而且影响较大，但是对于策略来说，有影响，但是没有参数的影响大.

3.9.4 压力测试结果与分析

(1) 压力测试 1：设计一种参数配置，使得产生的猴子数量非常多、非常密集，而梯子数量有限。观察此时你的程序的吞吐率和公平性表现如何。参数如下：

```
n = 4, h = 20, t = 3, N = 100, k = 3, MV = 5
```

实验结果如下：

```
total time: 99.0
Throughput rate: 1.0101010101010102
```

(2) 压力测试 2：设计一种参数配置，使得各猴子的速度差异非常大。观察此时

你的程序的吞吐率和公平性表现如何。这里调整最大速度特别大，这样造成猴子之间差距特别大，参数如下：

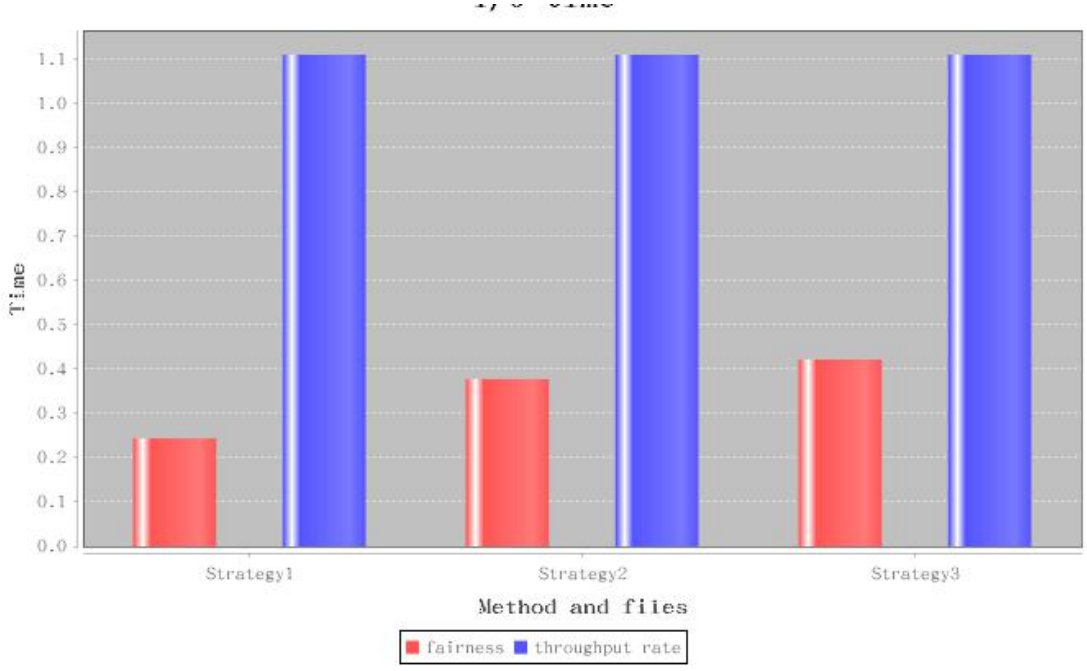
```
n = 4, h = 20, t = 3, N = 20, k = 3, MV = 30
```

结果如下：

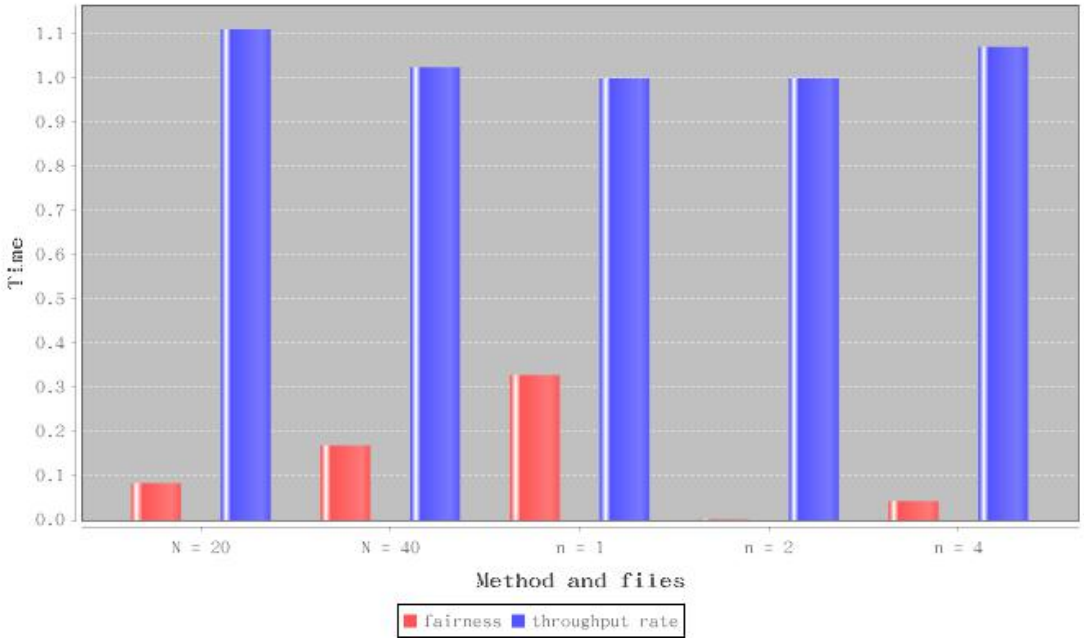
```
total time: 18.0
Throughput rate: 1.1111111111111112
fairness: 0.031578947368421054
```

采用 JFree Chart 进行对比

(1)不同策略下对于公平性和吞吐率的影响



(2)不同参数下对于吞吐率和公平性的影响



4 实验进度记录

请尽可能详细的记录你的进度情况。

日期	时间段	计划任务	实际完成情况

5 实验过程中遇到的困难与解决途径

对于多线程相关的知识一直都不太懂，上课没太听懂，所以首先看了<Java 编程思想>，自学了多线程，然后才开始写实验.

6 实验过程中收获的经验、教训、感想

本节除了总结你在实验过程中收获的经验和教训，也可就以下方面谈谈你的

感受（非必须）：

(1) 多线程程序比单线程程序复杂在哪里？你是否能体验到多线程程序在性能方面的改善？

(2) 你采用了什么设计决策来保证 threadsafe？如何做到在 threadsafe 和性能之间很好的折中？

(3) 你在完成本实验过程中是否遇到过线程不安全的情况？你是如何改进的？

(4) 关于本实验的工作量、难度、deadline。

(5) 到此为止你对《软件构造》课程的意见和建议。