プログラミングA 第9回・宿題

宿題の提出は Moodle で月曜日までに行ってください。各宿題ごとに提出ファイルを zip 等で一つのファイルにまとめて該当する Moodle の課題に提出しなさい。この資料や関係するコードをインターネットなどに公開することは著作権上、禁止されています。

1 宿題 1

以下を満たすことを目指して、クラス Scanner と Fax を作成した。

- Scanner が data を持つ。
- Scanner は、data が null の場合に data に非 null 値（null ではない値）を設定して、その非 null 値を出力する。これを繰り返す。
- Fax は、data が null ではない場合にその非 null 値を出力して、data に null を設定する。これを繰り返す。

プログラムの潜在的な問題が解決されるようにプログラムを修正し、修正後の全体を提出せよ。

参考資料：Scanner.java

```
public class Scanner extends Thread {
  String data = null;

  public void run() {
    while(true) {
      if(data == null) {
        data = "値が設定されています";
        System.out.println("scan: " + data);
      }
    }
  }

  public static void main(String[] args) {
    Scanner scanner = new Scanner();
    Fax fax = new Fax(scanner);
    scanner.start();
    fax.start();
  }
}
```

参考資料：Fax.java

```
class Fax extends Thread {
  Scanner scanner = null;

  public Fax(Scanner s) {
    super();
    scanner = s;
```

```
  }

  public void run() {
    while(true) {
      if(scanner.data != null) {
        System.out.println("fax : " + scanner.data);
        scanner.data = null;
      }
    }
  }
}
```

2 宿題 2-1

添付の Person クラスは immutable なクラスではなく、インスタンスの生成後にフィールドの値を変えることができる。PersonClient 中の「// HERE」の箇所にコードを追記して、実行すると以下を標準出力に得られるようにし、PersonClient の全体を提出せよ。ここでは、Person や Address を編集しないこと。

```
Living in Tokyo
Living in Paris
```

参考資料：Person.java

```
public class Person {

  final private Address address;

  public Person(String s) {
    address = new Address(s);
  }

  public Address getAddress() {
    return address;
  }

  public String toString() {
    return "Living in " + address.getText().toString();
  }
}
```

参考資料：PersonClient.java

```
public class PersonClient {

  public static void main(String[] args) {
    Person p = new Person("Tokyo");
    System.out.println(p);
```

```
      // HERE

      System.out.println(p);
    }

}
```

参考資料：Address.java

```
public class Address {

   private final StringBuffer text;

   public Address(String s) {
      text = new StringBuffer(s);
   }

   public StringBuffer getText() {
      return text;
   }
}
```

3 宿題 2-2

Person や Address を修正して、Person クラスについてインスタンスの生成後にフィールドの値を変えられないようにし、修正した全てのクラスの定義を提出せよ。なお結果として、前の設問において改変した PersonClient は動作しなくなる。また、Person が Address を通じて現住所を持つという機能は維持すること。

4 宿題 3

Read-Write Lock パターンが適用された添付のクラス群について以下に回答せよ。

(1) DataBuffer を実行した際に得られる結果を参照しながら、ReadWriteLock クラスの仕組みを説明せよ。
(2) DataBuffer 内部において、ReadWriteLock クラスの代わりに ReadWriteLock2 クラスを用いると、実行結果はどのよう変化するか？ その変化した理由を併せて説明せよ。

ヒント:

- WriterThread よりも、ReadThread のインスタンスを多く用意して実行している。
- 同時に複数の ReaderThread インスタンスが読むことができる。その間、WriterThread インスタンスは書けない。
- 同時にただ一つの WriterThread インスタンスが書くことができる。その間、他の WriterThread インスタンスは書けず、ReaderThread インスタンスも読めない。

- ReaderThread と WriterThread による読み書きを公平に、できるだけ交互に実施できることが望ましい（生存性の観点）。

参考資料：DataBuffer.java

```
public class DataBuffer {

  private final ReadWriteLock lock = new ReadWriteLock();
  private final StringBuffer value = new StringBuffer();

  public String read() throws InterruptedException {
    lock.readLock();
    try {
      Thread.sleep(100);
      return value.toString();
    } finally {
      lock.readUnlock();
    }
  }

  public void append(char c) throws InterruptedException {
    lock.writeLock();
    try {
      value.append(c);
    } finally {
      lock.writeUnlock();
    }
  }

  public static void main(String[] args) {
    DataBuffer data = new DataBuffer();
    new ReaderThread(data).start();
    new ReaderThread(data).start();
    new ReaderThread(data).start();
    new ReaderThread(data).start();
    new WriterThread(data).start();
    new WriterThread(data).start();
  }
}
```

参考資料：ReadWriteLock.java

```
/*
  Copyright (C) 2002,2006 Hiroshi Yuki.
http://www.hyuki.com/dp/dp2.html
hyuki@hyuki.com

This software is provided 'as-is', without any express or implied warranty.
In no event will the authors be held liable for any damages
```

```
arising from the use of this software.

Permission is granted to anyone to use this software for any purpose,
including commercial applications, and to alter it and redistribute it freely,
subject to the following restrictions:

1. The origin of this software must not be misrepresented; you must not claim
that you wrote the original software. If you use this software in a product,
an acknowledgment in the product documentation would be appreciated but is not
required.

2. Altered source versions must be plainly marked as such, and must not be
misrepresented as being the original software.

3. This notice may not be removed or altered from any source distribution.
 */

public final class ReadWriteLock {
   private int readingReaders = 0;
   private int writingWriters = 0;
   private int waitingWriters = 0;
   private boolean preferWriter = true;

   public synchronized void readLock() throws InterruptedException {
      while (writingWriters > 0 || (preferWriter && waitingWriters > 0)) {
         wait();
      }
      readingReaders++;
   }

   public synchronized void readUnlock() {
      readingReaders--;
      preferWriter = true;
      notifyAll();
   }

   public synchronized void writeLock() throws InterruptedException {
      waitingWriters++;
      try {
         while (readingReaders > 0 || writingWriters > 0) {
            wait();
         }
      } finally {
         waitingWriters--;
      }
      writingWriters++;
   }

   public synchronized void writeUnlock() {
```

```
      writingWriters--;
      preferWriter = false;
      notifyAll();
   }
}
```

参考資料：ReaderThread.java

```
public class ReaderThread extends Thread {
   private final DataBuffer data;

   public ReaderThread(DataBuffer d) {
      data = d;
   }

   public void run() {
      try {
         while (true) {
            System.out.println(Thread.currentThread().getName() + " reads " + data.read());
         }
      } catch (InterruptedException e) {
      }
   }
}
```

参考資料：WriterThread.java

```
import java.util.Random;

public class WriterThread extends Thread {
   private final DataBuffer data;
   private static final Random random = new Random();

   public WriterThread(DataBuffer d) {
      data = d;
   }

   public void run() {
      try {
         for(int i = 0; i < 10; i++) {
            char c = (char)('0' + i);
            data.append(c);
            System.out.println(Thread.currentThread().getName() + " appends " + c);
            Thread.sleep(random.nextInt(3000));
         }
      } catch (InterruptedException e) {
      }
   }
}
```

参考資料：ReadWriteLock2.java

```
/*
   Copyright (C) 2002,2006 Hiroshi Yuki.
http://www.hyuki.com/dp/dp2.html
hyuki@hyuki.com

This software is provided 'as-is', without any express or implied warranty.
In no event will the authors be held liable for any damages
arising from the use of this software.

Permission is granted to anyone to use this software for any purpose,
including commercial applications, and to alter it and redistribute it freely,
subject to the following restrictions:

1. The origin of this software must not be misrepresented; you must not claim
that you wrote the original software. If you use this software in a product,
an acknowledgment in the product documentation would be appreciated but is not
required.

2. Altered source versions must be plainly marked as such, and must not be
misrepresented as being the original software.

3. This notice may not be removed or altered from any source distribution.
 */

public final class ReadWriteLock2 {
   private int readingReaders = 0;
   private int writingWriters = 0;

   public synchronized void readLock() throws InterruptedException {
      while (writingWriters > 0) {
         wait();
      }
      readingReaders++;
   }

   public synchronized void readUnlock() {
      readingReaders--;
      notifyAll();
   }

   public synchronized void writeLock() throws InterruptedException {
      while (readingReaders > 0 || writingWriters > 0) {
         wait();
      }
      writingWriters++;
```

```
  }

  public synchronized void writeUnlock() {
    writingWriters--;
    notifyAll();
  }
}
```

5 宿題 4

添付の MessageHost を実行すると、MessageHost クラスの request メソッドの呼び出し毎に以下のように $10^N + 1 \sim 10^N + 10$（N=1,2,3）の出力が N を固定したまま連続し、呼び出しのたびに呼び出し側が長く待たされてしまう。

```
Handled: 11
Handled: 12
Handled: 13
Handled: 14
Handled: 15
Handled: 16
Handled: 17
Handled: 18
Handled: 19
Handled: 20
Handled: 101
Handled: 102
Handled: 103
Handled: 104
Handled: 105
Handled: 106
Handled: 107
Handled: 108
Handled: 109
Handled: 110
Handled: 1001
Handled: 1002
Handled: 1003
Handled: 1004
Handled: 1005
Handled: 1006
Handled: 1007
```

```
Handled: 1008
Handled: 1009
Handled: 1010
```

MessageHost クラスの request メソッドの内部のみを修正して、request メソッドを呼び出すと直ちに応答が得られるようにし、修正したコードを提出せよ。以下のように $10^N + 1$ 〜 $10^N + 10$ の出力が N について混ざり合った結果を標準出力に得ること（ただし出力の詳細は環境によって異なる）。

ヒント: 無名インナークラスを用いること。

```
Handled: 1001
Handled: 11
Handled: 101
Handled: 1002
Handled: 12
Handled: 102
Handled: 103
Handled: 1003
Handled: 13
Handled: 104
Handled: 14
Handled: 1004
Handled: 105
Handled: 15
Handled: 1005
Handled: 106
Handled: 16
Handled: 1006
Handled: 107
Handled: 17
Handled: 1007
Handled: 108
Handled: 1008
Handled: 18
Handled: 109
Handled: 1009
Handled: 19
Handled: 110
Handled: 20
Handled: 1010
```

参考資料：MessageHost.java

```
public class MessageHost {

   private final MessageHelper helper = new MessageHelper();

   public void request(final int number) {
      for(int i = 1; i <= 10; i++) {
         helper.handle(number + i);
      }
   }

   public static void main(String[] args) {
      MessageHost host = new MessageHost();
      host.request(10);
      host.request(100);
      host.request(1000);
   }
}
```

参考資料：MessageHelper.java

```
public class MessageHelper {
   public void handle(int v) {
      try {
         Thread.sleep(100);
         System.out.println("Handled: " + v);
      } catch (InterruptedException e) {
      }
   }
}
```

6 宿題 5

Future パターンが適用された添付のクラス群について、Main を実行した際に得られる結果を参照しながら、プログラム全体の仕組みを説明せよ。

参考資料：Data.java

```
public interface Data {
   public abstract String getContent();
}
```

参考資料：FutureData.java

```
public class FutureData implements Data {
   private RealData realdata = null;
   private boolean ready = false;
   public synchronized void setRealData(RealData r) {
```

```
    if (ready) {
      return;
    }
    realdata = r;
    ready = true;
    notifyAll();
  }
  public synchronized String getContent() {
    while (!ready) {
      try {
        wait();
      } catch (InterruptedException e) {
      }
    }
    return realdata.getContent();
  }
}
```

参考資料：Host.java

```
public class Host {
  public Data request(final int number) {
    final FutureData future = new FutureData();
    new Thread() {
      public void run() {
        RealData realdata = new RealData(number);
        future.setRealData(realdata);
      }
    }.start();
    return future;
  }
}
```

参考資料：Main.java

```
public class Main {
  public static void main(String[] args) {
    Host host = new Host();
    Data data1 = host.request(10);
    Data data2 = host.request(20);
    Data data3 = host.request(30);
    try {
      Thread.sleep(3000);
    } catch (InterruptedException e) {
    }
    System.out.println(data1.getContent());
    System.out.println(data2.getContent());
    System.out.println(data3.getContent());
  }
```

```
}
```

参考資料：RealData.java

```
public class RealData implements Data {
   private final String content;

   public RealData(int number) {
      StringBuffer temp = new StringBuffer();
      for(int i = 0; i < number; i++) {
         try {
            Thread.sleep(100);
            temp.append('a');
         } catch (InterruptedException e) {
         }
      }
      System.out.println("RealData has been made: " + temp);
      content = temp.toString();
   }

   public String getContent() {
      return content;
   }
}
```