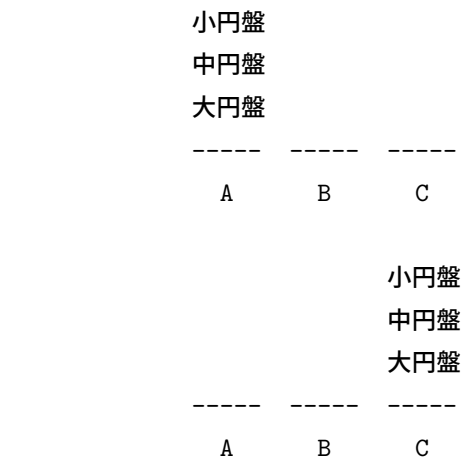
プログラミングA 第10回・宿題

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

大きさの全て異なる N 枚の円盤が、大きいものを一番下に台 A に積み上がっている。ここで、台 B を利用して、すべての円盤を台 C に移動する問題を考える。ただし、円盤は一度に一枚しか移動できず、小さい円盤の上に大きい円盤は乗せられないものとする。



円盤が 2 枚以下の場合は、以下の考え方を定義した添付の DiscTower クラスの disc1 メソッドを用いて、必要な円盤の移動を出力できる。ただし disc1 メソッドにおいては再帰呼び出しを用いていない。

- N=1 のときは、A にある円盤を C に移動すればよい。
- N=2 のときは、A の一番下の円盤以外をいったん B に移動して退避させて、A に残った円盤を C に移動し、最後に B に退避しておいた円盤を C に移動すればよい。

上記の考え方を一般化して、任意の N 枚（ただし $N > 0$）である場合に必要な円盤の移動を出力するように disc2 メソッドを完成させて、その内容提出せよ。再帰呼び出しを用いること（つまり、disc2 メソッドの中で disc2 メソッドを呼び出すこと）。

DiscTower クラスを実行して以下を標準出力に得ること。

```
Move 1 Disc from A to C
A -> C
Move 2 Discs from A to C
A -> B
A -> C
B -> C
Move 3 Discs from A to C
```

```
A -> C
A -> B
C -> B
A -> C
B -> A
B -> C
A -> C
```

参考資料：DiscTower.java

```
public class DiscTower {

   public static void disc1(char start, char mid, char target, int n) {
      if(n == 1) {
         move(start, target);
      } else if(n == 2) {
         move(start, mid);
         move(start, target);
         move(mid, target);
      }
   }

   public static void disc2(char start, char mid, char target, int n) {

   }

   public static void move(char start, char target) {
      System.out.println(start + " -> " + target);
   }

   public static void main (String[] args) {
      System.out.println("Move 1 Disc from A to C");
      disc1('A', 'B', 'C', 1);
      System.out.println("Move 2 Discs from A to C");
      disc1('A', 'B', 'C', 2);
      System.out.println("Move 3 Discs from A to C");
      disc2('A', 'B', 'C', 3);
   }
}
```

2　宿題 2

指定したファイルやディレクトリを「深く」コピーするクラス DeepCopy を完成させて、その全体を提出せよ。つまり、src がファイルの場合は dst としてコピーし、src がディレクトリの場合は配下の全要素をそのままの構成で dst としてコピーすること。

利用方法は java DeepCopy [src] [dst] とする。例えば以下の構成の場合に、java DeepCopy c:
temp c:
copy とすると、

```
c:\temp
c:\temp\a.doc
c:\temp\data
c:\temp\data\b.txt
```

以下を得る。

```
c:\copy
c:\copy\a.doc
c:\copy\data
c:\copy\data\b.txt
```

ヒント:

- File インスタンスに対して getAbsolutePath メソッドを呼び出すと、当該ファイル（またはディレクトリ）の絶対パスを得る。
- あるディレクトリ（例えば絶対パスが C:
  temp というディレクトリ）の配下に、あるファイルやディレクトリ（例えば test.doc ファイル）を作成したい場合は、絶対パスと
  およびファイル・ディレクトリ名を連結した文字列で新たな File インスタンスを作成すればよい（例えば new File(”C:
  temp
  test.doc”))。ただしそのディレクトリ（この例では C:
  temp ）が存在している必要があり、もし存在していなければ当該ディレクトリを表す File インスタンスの mkdir メソッドを呼び出せばよい。

参考資料：DeepCopy.java

```
import java.io.*;
public class DeepCopy {

  public static void main(String[] args) {
    copyFileOrDirectory(args[0], args[1]);
  }

  public static void copyFileOrDirectory(String srcName, String dstName) {
    File srcFile = new File(srcName);
    if( ... /* srcFileがファイルの場合*/ ) {
      copyFile(srcFile.getAbsolutePath(), dstName);
    } else if( ... /* srcFileがディレクトリの場合 */ ) {
      File dstDir = new File(dstName);
      dstDir.mkdirs();
```

```
        /*
         * srcFile から配下のファイルやディレクトリの一覧を得て、
         * それぞれについて copyFileOrDirectoryの再帰呼び出し。
         */
        ...
      }
    }

    public static void copyFile(String srcName, String dstName) {
      /*
       * FileCopy を参考にして、srcNameで指定されるファイルの内容を
       * dstName で示されるファイルに書き出す。
       */
    }
}
```

3 宿題 3

講義資料中の以下の全ての実行結果を回答欄に貼りつけるとともに、各結果を得た仕組みを簡潔に説明せよ。

(3-1) DisplayURL について何らかの URL を指定して実行した結果、および、その結果を得た仕組み
(3-2) DaytimeTCPClient について time-b.timefreq.bldrdoc.gov を接続先として指定して実行した結果、、および、その結果を得た仕組み
(3-3) DaytimeUDPServer を実行したままの状態で、同じマシン上で DaytimeUDPClient について localhost（つまり同じマシン）をサーバとして指定実行した結果、および、その結果を得た仕組み

4 宿題 4-1

クラス EchoUDPServer を完成させて、受信データを送り返すエコーサービスのサーバを UDP 通信で実装し、その全体を提出せよ。各送受信におけるデータサイズの上限は 256 バイトとする。クライアントにはクラス EchoUDPClient を利用し、サーバ側のポート番号として 7 を指定すること。

サーバを起動した状態江同一マシン上でクライアントを以下のように用いると、以下を標準出力に得ること。

```
>java EchoUDPClient localhost

Client sent    : This is test
Client received: This is test
```

サーバを起動していない状態でクライアントを以下のように用いると、以下を標準出力に得ること。

```
>java EchoUDPClient localhost
```

```
Client sent    : This is test
Error: java.net.SocketTimeoutException: Receive timed out
```

ヒント: 講義資料における DaytimeUDPServer を参考にするとよい。

参考資料：EchoUDPClient.java

```java
import java.net.*;
public class EchoUDPClient {
  public static void main(String[] args) {
    DatagramSocket socket = null;
    try {
      byte[] outData = "This is test".getBytes();
      byte[] inData = new byte[256];
      InetAddress address = InetAddress.getByName(args[0]);
      socket = new DatagramSocket();
      DatagramPacket outPacket = new DatagramPacket(outData, outData.length, address, 7);
      DatagramPacket inPacket = new DatagramPacket(inData, inData.length);
      socket.setSoTimeout(5000);
      socket.send(outPacket);
      System.out.print("Client sent : ");
      System.out.write(outData);
      System.out.println();
      socket.receive(inPacket);
      System.out.print("Client received: ");
      System.out.write(inData);
      System.out.println();
    } catch (Exception e) {
      System.out.println("Error: " + e);
    } finally {
      if(socket != null) {
        socket.close();
      }
    }
  }
}
```

参考資料：EchoUDPServer.java

```java
import java.net.*;
import java.io.IOException;
public class EchoUDPServer {
  public static void main(String args[]) {
    DatagramSocket socket = null;
    try {
      socket = new DatagramSocket(7);
      while (true) {
        // Prepare data.

        // Make DatagramPacket for input.
```

```
            // Receive packet.

            // Make DatagramPacket for output.

            // Send packet.

          }
        } catch (IOException ioe) {
          System.err.println(ioe);
        } finally {
          if (socket != null) socket.close();
        }
    }
}
```

5 宿題 4-2

クラス EchoUDPServer について、クライアントからのリクエスト毎にスレッドを生成して当該スレッドに対応させるように拡張したクラス EchoUDPServer2 を作成し、その全体を提出せよ。ただし、挙動を見えやすくするために、パケットの送受信時にサーバ側の標準出力に送受信データおよびそれを扱ったスレッドの情報を出力すること。従って、サーバ側では、複数または単一クライアントからの複数の接続に応じて以下を標準出力に得る。

```
>java EchoUDPServer
Thread[main,5,main] received: This is test
Thread[Thread-0,5,main] sent: This is test
Thread[main,5,main] received: This is test
Thread[Thread-1,5,main] sent: This is test
```

ヒント: Thread-Per-Message パターンを用いればよい。ただし、無名インナークラス内からその外側のクラス（EchoUDPServer2）のフィールドや引数を参照したい場合には final 宣言されている必要があることに留意すること。添付の雛形を用いて構わない。

参考資料：EchoUDPServer.java

```
import java.net.*;
import java.io.IOException;
public class EchoUDPServer2 {
  public static void main(String args[]) {
    final DatagramSocket socket;
    DatagramPacket inPacket = null;
    try {
      socket = new DatagramSocket(7);
      try {
        while (true) {
```

```
            // Prepare data.

            // Make DatagramPacket for input.

            // Receive packet.

            // Print data received.

            // Obtain socket address.
            final SocketAddress inpacketSocketAddress = inPacket.getSocketAddress();

            // Make Thread-Per-Message.
            new Thread() {
              public void run() {
                try {
                  // Make DatagramPacket for output.

                  // Send packet.

                  // Print data sent.

                } catch(IOException se) {
                }
              }
            }.start();
          }
        } finally {
          if (socket != null) socket.close();
        }
      } catch (IOException ioe) {
        System.err.println(ioe);
      }
    }
}
```

6 宿題 5

以下のそれぞれについて、この小テスト（宿題）や講義で扱うプログラムソースコードを参照する形で説明せよ.

(5-1) バイトストリームと文字ストリームの共通点と相違点
(5-2) java.io パッケージにおける Decorator パターンの目的と仕組み
(5-3) Java により TCP 通信の実現と UDP 通信の実現の共通点と相違点（用いるクラスなど）