プログラミングA 第10回・宿題　回答

1　宿題 1

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) {
      if(n > 0) {
         disc2(start, target, mid, n - 1);
         move(start, target);
         disc2(mid, start, target, n - 1);
      }
   }

   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

copyFileOrDirectory() において、コピー元の種別を確認し、ファイルの場合は copyFile() を呼び出してバイトストリームによりファイルをコピーする。ディレクトリの場合は配下の全要素を取得して、各要素につい

て再帰的に copyFileOrDirectory() を呼び出す。

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.isFile()) {
      copyFile(srcFile.getAbsolutePath(), dstName);
    } else if(srcFile.isDirectory()) {
      String[] srclist = srcFile.list();
      File dstDir = new File(dstName);
      File dstFile = null;
      dstDir.mkdirs();
      for(int i = 0; i < srclist.length; i++) {
        dstFile = new File(dstDir.getAbsolutePath() + "¥¥" + srclist[i]);
        copyFileOrDirectory(srcFile.getAbsolutePath() + "¥¥"
            + srclist[i], dstFile.getAbsolutePath());
      }
    }
  }
  public static void copyFile(String srcName, String dstName) {
    int data = 0;
    InputStream fis = null;
    OutputStream fos = null;
    try {
      fis = new BufferedInputStream(new FileInputStream(srcName));
      fos = new BufferedOutputStream(new FileOutputStream(dstName));
      while((data = fis.read()) != -1) {
        fos.write(data);
      }
    } catch(FileNotFoundException fnfe) {
    } catch(IOException ioe) {
    } finally {
      try {
        if(fis != null) fis.close();
        if(fos != null) fos.close();
      } catch(IOException ioe) {
      }
    }
  }
}
```

3　宿題 3

(3-1)
- 実行時の引数で指定した URL から入力バイトストリームを取得して (URL クラスより) 文字ストリームに変換し (InputStreamReader による)、さらにバッファリング機能を付与したうえで (BufferedReader による)、最後まで 1 行ずつ読みとって標準出力に出力している。

(3-2)
- 実行時の引数で指定したホストの 13 番ポートについて TCP クライアントソケットを作成して入力バイトストリームを取得し、文字ストリームに変換し、さらにバッファリング機能を付与したうえで、最後まで 1 行ずつ読みとって標準出力に出力している。対象サーバにおいて Daytime プロトコルに基づいたサービスが 13 番ポートで実行されていれば、上記の接続に対して自動的に時刻を返してくる。

(3-3)
- DaytimeUDPClient は、実行時の引数で指定された接続先サーバのホスト名からインターネットアドレスを取得し、サーバに 256 バイトを送る UDP の送信 (出力) 用パケットを作成する。また、サーバから 256 バイトを受け取る UDP の受信 (入力) 用パケットを作成する。UDP 通信のソケットを作成し 5 秒をタイムアウト時間として設定のうえ、送信用パケットを送り、受信用パケットを用いてデータを受信、最後にその受信データを標準出力に出力する。

4　宿題 4-1

注意: 1 度の送信で 256byte(128 文字) を超えるデータは失われる

参考資料：EchoUDPServer.java

```
import java.net.*;
public class EchoUDPServer {
  public static void main(String args[]) {
    DatagramSocket socket = null;
    try {
      socket = new DatagramSocket(7);
      byte[] inData = new byte[256];
      DatagramPacket inPacket = null;
      DatagramPacket outPacket = null;
      while (true) {
        inData = new byte[256];
        inPacket = new DatagramPacket(inData, inData.length);
        socket.receive(inPacket);
        outPacket = new DatagramPacket(
             inData, inData.length, inPacket.getSocketAddress());
        socket.send(outPacket);
      }
    } catch (Exception e) {
      System.out.println("エラー: " + e);
    } finally {
      if (socket != null) socket.close();
    }
```

```
  }
}
```

5 宿題 4-2

参考資料：EchoUDPServer2.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) {
        final byte[] inData = new byte[256];
        inPacket = new DatagramPacket(inData, inData.length);
        socket.receive(inPacket); System.out.print(Thread.currentThread() + " received: ");
        System.out.write(inData); System.out.println();
        final SocketAddress inpacketSocketAddress = inPacket.getSocketAddress();
        new Thread() {
          public void run() {
            try {
              DatagramPacket outPacket =
                new DatagramPacket(inData, inData.length, inpacketSocketAddress);
              socket.send(outPacket);
              System.out.print(Thread.currentThread() + " sent: ");
              System.out.write(inData); System.out.println();
            } catch(IOException se) {
            }
          }
        }.start();
      }
      } finally {
        if (socket != null)
          socket.close();
      }
    } catch (IOException ioe) {
      System.err.println(ioe);
    }
  }
}
```

6 宿題 5

(5-1)
- 共通点: 入出力として、データの連続的な流れを扱う。ファイルやネットワークの違いに対して透過的なプログラム作成が可能である。
- 相違点: バイトストリームはバイト単位の入出力を扱い、任意のテキスト以外のデータの扱いに適する。対して文字ストリームは文字単位の入出力を扱い、テキストのデータの扱いに適する。

(5-2)
- Decorator デザインパターンは、オブジェクト (インスタンス) の役割を実行時に追加可能とする設計上のパターンである。
- java.io パッケージにおいて同パターンの適用により、入出力ストリームを扱うオブジェクトに対して様々な機能を柔軟に追加および拡張することを可能としている。
- 具体的には、FilterInputStream および FilterOutputStream は、読み書き処理にあたり内包するストリームにおけるもともとの読み書きに処理に加えて、独自の追加処理を行う。バッファリング機能や先読み機能を追加する様々なストリームクラスを FilterInput/OutputStream のサブクラスとすることで、それらの様々な機能を柔軟に組み合わせて追加することを可能としている。

(5-3)
- Java において TCP 通信と UDP 通信は、java.net パッケージを用いて異なる計算機間のトランスポート層における通信として実現される点は共通している。
- TCP 通信は、Socket クラスを用いてコネクションを設定したうえで、入出力ストリームを取得し、必要に応じて java.io パッケージの種々のストリームクラスで機能拡張したうえで、ストリームへのデータの読み書きにより実現する。
- 対して UDP 通信では、コネクションを設定せず、DatagramSocket クラスによりソケットを作成のうえ送受信 (入出力) 用のパケットを DatagramPacket クラスにより用意してパケットを送受信する。