独習Java第3版

12.1 インターネットアドレス
12.2 サーバーソケットとソケット
12.3 データグラムソケット
　　とデータグラムパケット
12.4 URL

インターネットアドレス

- インターネットアドレスとは？
  - 32ビットの長さを持ち、インターネットに接続されたマシンの識別のために用いられる
  - アドレスはピリオドで区切られた4つの番号からなる
  - ピリオドで区切られたトークンの並びで表現されることもある
- DNS（Domain Name System）とは？
  - インターネットアドレスをドットストリング表記からどっと10進表記に変換する役割を持つ

InetAddressクラス

- java.netパッケージのInetAddressクラスは
  インターネットアドレスをカプセル化している

InetAddressクラスに定義されている主なインスタンスメソッド

メソッド	説明
byte[] getAddress()	アドレス情報を含むバイトの配列を返す。 データはネットワークバイトオーダー （最初の要素が上位バイト）で格納される
String getHostAddress()	アドレス情報を表す文字列を返す
String getHostName()	ホスト名を表す文字列を返す

InetAddressのメソッド(1/2)

- getByName()メソッド
  static InetAddress getByName(String *hostName*) throws UnknownHostException
  - DNSによって提供される情報を使って名前からアドレスへの変換を実行する
- getAllByName()メソッド
  static InetAddress getAllByName(String *hostName*) throws UnknownHostException
  - ホストが複数のアドレスもっている場合使用する。InetAddressオブジェクトの配列を取得できる。

InetAddressのメソッド(2/2)

- どちらのメソッドも*hostName*には
  インターネットホストの名前を指定

- getLocalHost()メソッド
  static InetAddress getLocalHost() throws
  UnknownHostException
  - ローカルホストの情報をカプセル化した
    InetAddressオブジェクトが返される

例

```
import java.net.*;
class InetAddressDemo {
  public static void main(String[] args) {
  try{
    // アドレスを取得する
    InetAddress ias[] =
          InetAddress.getAllByName(args[0]);
    for (int i=0; i<ias.length;i++){
      System.out.println(ias[i].getHostName());
      System.out.println(ias[i].getHostAddress());
      byte bytes[] = ias[i].getAddress();
      for(int j=0;j<bytes.length;j++){
        if(j>0)
          System.out.print(".");
        if(bytes[j] >=0)
          System.out.print(bytes[j]);
        else
          System.out.print(bytes[j]+256);
        }
          System.out.println("");
        }
      }
      catch(Exception e){
        e.printStackTrace();
      }
    }
}
```

実行結果:

```
C:¥java>java InetAddressDemo 127.0.0.1
localhost
127.0.0.1
127.0.0.1
```

サーバーソケットとソケット

- ソケットとは？
  - 2つのマシン間の双方向通信経路の一端
  - 2つのアプリケーションが、信頼性のある順次データ交換を行うためのメカニズムを提供
    - これはソケットがTCP(Transmission Control Protocol)とIP（Internet Protocol）を使用することによって実現されている
- ServerSocketクラス、Socketクラスとは？
  - クライアント/サーバーアプリケーションを作成するのに使用

ServerSocketクラス

- ServerSocketクラスは
  - サーバーアプリケーションを作成するのに使用
  - クライアントから送られてくる要求を監視する
- ServerSocketコンストラクタ

ServerSocket(int *port*) throws IOException

  - *port*はクライアントからの要求を監視するための
    ソフトウェアポート
  - 他の形式のコンストラクタは送られてくる要求待ちの
    行列を制限したり、特定のアドレスをバインド(監視)する

メソッド

- accept()メソッド

Socket accept() throws IOException

  - クライアントから送られてくる要求を監視する
  - 要求が到着するまで待機する
  - Socketオブジェクト(クライアントとの通信に使用)を返す
- close()メソッド

void close() throws IOException

  - サーバーソケットをクローズするのに使用する

Socketクラス

- クライアントとサーバーのデータ交換には Socketクラスを使用する
- Socketクラス

Socket(String *hostName*, int *port*) throws UnknownHostException, IOException

  - *hostName*はサーバーホストの名前（ドットストリング表記or 10進表記）
  - portはそのサーバーのソフトウェアポートでソケットの接続先

メソッド(1/2)

- ソケットの作成後は通信に使う入力ストリームと出力カストリームを取得する必要がある
  - getInputStream()メソッド
    getOutputStream()メソッドを使用

InputStream getInputStream() throws IOException

OutputStream getOutputStream() throws IOException

メソッド(2/2)

- InputStreamオブジェクトと
  OutputStreamオブジェクトは
  通常、それぞれのDataInputStreamオブジェクト、
  DataOutputStreamオブジェクトの作成に使用

- ソケットをクローズするにはclose()メソッドを使用

  void close() throws IOException

例(1/2)

- サーバーソフトウェア

```
import java.io.*;
import java.net.*;
import java.util.*;

public class ServerSocketDemo {
 public static void main(String[] args) {
  try{
   // ポートを取得する
   int port = Integer.parseInt(args[0]);
   // 乱数ジェネレータを作成する
   Random random = new Random();
   // サーバーソケットを作成する
   ServerSocket ss = new ServerSocket(port);
   // 無限ループを作成する
    while(true){
     // クライアントからの要求を受け取る
     Socket s = ss.accept();
     //結果をクライアントに書き込む
     OutputStream os = s.getOutputStream();
     DataOutputStream dos = new
DataOutputStream(os);
     dos.writeInt(random.nextInt());
     // ソケットをクローズする
     s.close();
    }
   }
   catch(Exception e){
    System.out.println("Exception :"+e);
   }
  }
}
```

例(2/2)

- クライアントソフトウェア

```
import java.io.*;
import java.net.*;

class SocketDemo {
 public static void main(String[] args) {
  try{
   // サーバーとポートを取得する
   String server = args[0];
   int port = Integer.parseInt(args[1]);
   // ソケットを作成する
   Socket s = new Socket(server, port);
   // サーバーから乱数を読み取る
   InputStream is = s.getInputStream();
   DataInputStream dis =
                  new DataInputStream(is);
   int i = dis.readInt();
   // 結果を表示する
   System.out.println(i);
   // ソケットをクローズする
   s.close();
  }
  catch(Exception e){
   System.out.println("Exception: " + e);
  }
 }
}
```

実行例

- サーバーソフトウェア起動の例
  - C:¥java>java ServerSocketDemo 4321
    - 4321は送られてくる要求の到着先となるソフトウェアポート
    - 4321でなくても構わないが、1024より下の番号は避ける
- クライアントソフトウェア起動の例
  - C:¥java>java SocketDemo 127.0.0.1 4321
    - 127.0.0.1 はローカルマシンを表す
    - 2番目の引数はサーバーアプリケーションの時と同じポート番号を指定しなければならない
- SocketDemoアプリケーションは乱数を表示すると終了

データグラムソケットとデータグラムパケット

- TCP(Transmission Control Protocol)
  - ソケットでは、信頼性ある順次データ交換を実現するために用いられる
  - 高い信頼性の確保と引き換えに高負荷
- UDP(User Datagram Protocol)
  - TCPの代わりに用いられるプロトコル
  - 順次データの交換は保証されない
  - TCPに比べ負荷が低い（高速に通信できる）

クラス

- java.netライブラリにはデータグラムを使ったクライアント/サーバーアプリケーションの作成に使用できる2つのクラスが用意されている
  - DatagramPacketクラス
  - DatagramSocketクラス

DatagramPacket(1/3)

- DatagramPacketコンストラクタ

DatagramPacket(byte *buffer*[], int *size*)
DatagramPacket(byte *buffer*[], int *size*,
InetAddress *ia*, int *port*)

  - DatagramPacketクラスは
    データグラムパケットをカプセル化する
  - 前者は着信データグラムを受け取るのに使用する
    DatagramPacketオブジェクトの作成
  - buffer :着信データが書き込まれるバイト配列
  - size :bufferのサイズ数

DatagramPacket(2/3)

- 後者は発信データグラムを送信するのに
  使用するDatagramPacketオブジェクトを作成
- buffer　:バイト配列でここから発信データを読み取る
- size　:bufferから読み取るバイト数
- ia, port　:受信側のインターネットアドレスとポートを指定

DatagramPacket(3/3)

- DatagramPacketクラスに定義されている主なインスタンスメソッド

メソッド	説明
InetAddress getAddress()	着信データグラムの場合、発信もとのマシンのアドレスを返す 発信データグラムの場合、宛先のマシンのアドレスを返す
byte[] getData()	データが含むバイトの配列を返す
int getLength()	パケット内のバイト数を返す
int getPort()	ポートを返す
void setAddress(InetAddress *ia*)	*ia*にアドレスをセットする
foid setData(byte *buffer*[])	*buffer*にデータをセットする
void setLength(int *size*)	*size*で指定したサイズのパケットをセットする
void setPort(int *port*)	*port*にポートをセットする

DatagramSocket(1/3)

- データグラムの送信と受信に使用
- DatagramSocketコンストラクタ

  DatagramSocket() throws SocketException

  DatagramSocket(int *port*) throws SocketException
  - 前者は利用可能なポートにデータグラムソケットをバインドする
  - 後者はソケットを*port*にバインドする

DatagramSocket(2/3)

- receive()メソッド

  void receive(DatagramPacket *dp*) throws IOException
  - 着信データグラムを監視する
  - データグラムが到着するまで待機する
  - *dp*は着信データが格納される DatagramPacketオブジェクト

DatagramSocket(3/3)

- send()メソッド

  void send(DatagramPacket *dp*) throws IOExcption
  - データグラムを送信する
  - *dp*は送信されるデータと宛先のホストとポートが格納されるDatagramPacketオブジェクト
- データグラムソケットはclose()メソッドを使用してクローズする
- close()メソッド

  void close()

例(1/2)

- 受信側

```
import java.net.*;

class DatagramReceiver {
  private final static int BUFSIZE = 20;

  public static void main(String args[]) {
    try {
      // ポートを取得する
      int port = Integer.parseInt(args[0]);

      // ポートのDatagramSocketオブジェクトを作成
       する
      DatagramSocket ds = new
       DatagramSocket(port);

      // 着信したデータを保持するバッファを作成する
      byte buffer[] = new byte[BUFSIZE];

    // 無限ループを作成する
      while(true) {
        // データグラムパケットを作成する
        DatagramPacket dp =
          new DatagramPacket(buffer,
            buffer.length);

// データを受け取る
        ds.receive(dp);

        // データグラムパケットからデータを得る
        String str = new String(dp.getData());
        // データを表示する
        System.out.println(str);
      }
    }
    catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

例(2/2)

- 送信側

```
import java.net.*;

class DatagramSender {
  public static void main(String args[]) {
    try  {
      // 宛先のインターネットアドレスを作成する
      InetAddress ia =
        InetAddress.getByName(args[0]);
      // 宛先ポートを取得する
      int port = Integer.parseInt(args[1]);

      // データグラムソケットを作成する
      DatagramSocket ds = new DatagramSocket();

      // データグラムパケットを作成する
      byte buffer[] = args[2].getBytes();
      DatagramPacket dp =
        new DatagramPacket(buffer, buffer.length,
          ia, port);
      // データグラムパケットを送信する
        ds.send(dp);
      }
      catch (Exception e) {
        e.printStackTrace();
      }
    }
}
```

実行結果

受信側:

C:¥java>java DatagramReceiver 50000

送信側:

C:¥jv>java DatagramSender 127.0.0.1 50000 Dokusyu-Java

受信側:

Dokusyu-Java

URL

- URL(Uniform Resource Locator)は
  Webリソースを識別するもの

  protocol://*host*:*port*/*file*

  - *protocol*はリソースを取得するときに使う
    プロトコルの名前（例：ftp, http）
  - *host*パラメータはリソースの取得先マシンを識別
  - *port*パラメータはサーバーのソフトウェアポートを識別
    （省略した場合は各プロトコル既定の値が使用される）
  - *file*パラメータはサーバー上のファイルを識別

URLクラス

- URLについての情報をカプセル化
  - URLコンストラクタ

URL(String *protocol*, String *host*, int *port*, String *file*) throws MalformedURLExcption

URL(String *protocol*, String *host*, String *file*) throws MalformedURLException

URL(Sting *urlString*) throws MalformedURLException

  - protocol, host, port, fileの意味は前スライドと同じ
  - urlStringは上の4つのパラメータを1つにまとめたもの

メソッド

- openStream()メソッド

  InputStream OpenStream() throws IOException
  - URLの入力ストリームをオープンし、そのストリームを返す
- リソースの内容は次の入力ストリームから読み取ることができる

  getFile()メソッド、getHost()メソッド、getPort()メソッド、getProtocol()メソッド
  - String getFile()
  - String getHost()
  - int getPort()
  - String getProtocol

例

- ファイル名：URLDemo.java

```
import java.io.*;
import java.net.*;

class URLDemo {
  public static void main(String args[]) {
    try {

      // URLを取得する
      URL url = new URL(args[0]);

      // 入力ストリームを取得する
      InputStream is = url.openStream();

      // URLからデータを読み取り、表示する
      byte buffer[] = new byte[1024];
      int i;
      while((i = is.read(buffer)) != -1) {
        System.out.write(buffer, 0, i);
      }
    }
```

```
    catch (Exception e) {
        e.printStackTrace();
    }
  }
}
```

実行例１：
C:¥java>java URLDemo file:URLDemo.java
実行例２：
C:¥jv>java URLDemo http://www.shoeicha.com

課題

- 引数で指定された回数データグラムを受信し、受信した順番とは逆に、受信したデータグラムを表示するプログラムを作りなさい