ネットワークプログラミング
21005 2号館10階
第7回
2014/11/10

岩井将行

(もう2015年終わりじゃ無い。。。
Threadを少々やりたい。)

課題提出方法

- 課題提出ネットワークフォルダ第2回のフォルダにファイルを提出してください。
- Javaファイル、Classファイルを両方提出すること

Xmasプレゼントを渡そう

- XmasTCPServ.java
- XmasTCPClient.java
- XmasPresent.java

- java serializable＝直列化可能というマーキング

前回の課題

- :XmasPresentサーバクライアントを改造し、Serializable,ITaskが実装されたTaskObjectをサーバに実行させるサーバクライアントを完成させよ。ITaskインターフェースには public void exec() , public int getResult()がメソッドとして存在していることとする。サーバクライアントはTaskServer,TaskClientという名前にすること。

TCPとUDP

- TCPとUDPの違い

	TCP	UDP
信頼性	高信頼	低信頼
転送速度	低速	高速
転送形式	コネクション型	コネクションレス型
その他	端末間同士のデータ転送	上位レイヤからの送信要求が簡潔

TCP v.s. UDP

- TCPはトランスポート層で信頼性のある通信を実現する必要がある場合に利用される。TCPはコネクション指向で順序制御や再送制御を行なうためアプリケーションに信頼性のある通信を提供することができる。

- UDPは高速性やリアルタイム性を重視する通信などに用いられる。
  - 例としてリアルタイムのストリーミングを挙げる。
  - もしTCPを利用した場合、パケットが途中で失われた場合に再送処理を行なうため、その間画像や音が停止するなどの不具合が生じてしまう。これに対して、UDPは再送処理を行なわないのでパケットは送信され続ける。もし多少のパケットが失われていたとしても、一時的に画像や音声が乱れるだけである。よって、ストリーム配信サービスではUDPの方が優れているといえる。

ソケット通信

プログラム同士の通信は

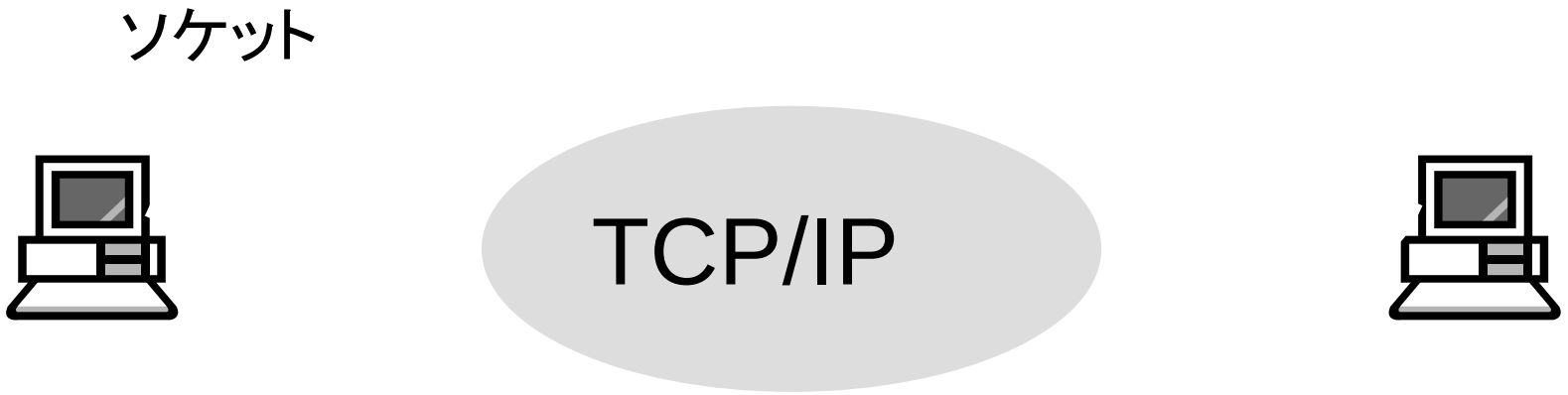
- ソケットを使ってデータの送受信
- ソケットを使った通信

ソケット通信

ソケット

意味:「接続の端点」

コンピュータとTCP/IPを
つなぐ出入り口



ソケット通信

- ソケットを使って通信を行うには
  2つのプログラムが必要

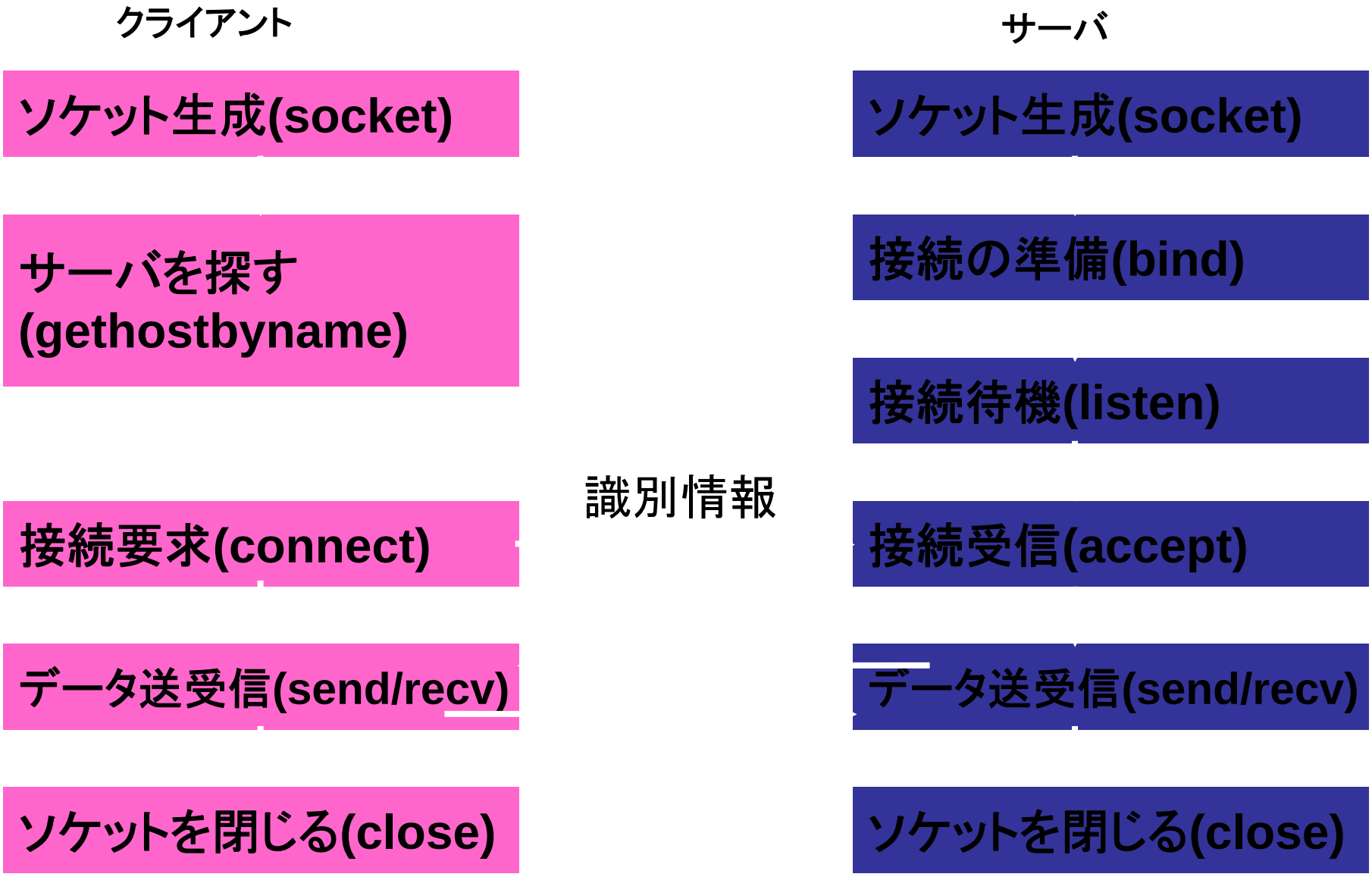
クライアントプログラム

ソケットを用意して
サーバに接続要求を行う

サーバプログラム

ソケットを用意して接続要求を待つ

ソケット通信の全体の流れ



識別情報

- 正しくデータを受け渡しするために
  通信する相手を識別する

IPアドレス

コンピュータを識別

コンピュータのアドレス

ポート番号

プログラムを識別

プログラムの識別番号

ウェルノウン　ポート

よく使われているプログラムの
ポート番号は決まっている

ポート番号	プログラム
21	ftp
22	ssh
23	telnet
80	http(web)

1024番以下は全て決められている

クライアントプログラム（Java）

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

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

        try{
            //ソケットを作成
            String host="localhost";
            Socket socket = new Socket( host, 10000 );

            //入力ストリームを作成
            DataInputStream is =  new DataInputStream
                        new BufferedInputStream(
                            socket.getInputStream()));
```

```
            //サーバ側から送信された文字列を受信
            byte[] buff = new byte[1024];
            int a = is.read(buff);
            System.out.write(buff, 0, a);

            //ストリーム，ソケットをクローズ
            is.close();
            socket.close();

        }catch(Exception e){
                                System.out.println(e.getMessage());
            e.printStackTrace();
        }
    }
}
```

サーバプログラム（Java）

```
//Server.java

import java.net.*;
import java.lang.*;
import java.io.*;

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

            try{
                //ソケットを作成
                ServerSocket svSocket = new ServerSocket(10000
                //クライアントからのコネクション要求受付
                Socket cliSocket = svSocket.accept();

                //出力ストリームを作成
                DataOutputStream os = new DataOutputStream(
                        new BufferedOutputStream(
                            cliSocket.getOutputStream()));

                //文字列を送信
                String s = new String("Hello World!!¥n");
                byte[] b = s.getBytes();
                os.write(b, 0, s.length());
```

```
//ストリーム，ソケットをクローズ
                os.close();
                cliSocket.close();
                svSocket.close();

            }catch( Exception e ){
                System.out.println(e.getMessage());
                e.printStackTrace();
            }
        }
    }
```

サーバプログラム（Java）

ソケット作成，コネクション要求受付待機

```
ServerSocket svSocket =
  newServerSocket(10000);
Socket cliSocket = svSocket.accept();
```

出力ストリーム作成

```
DataOutputStream os =
  new DataOutputStream(
    new BufferedOutputStream(
      cliSocket.getOutputStream());
```

クライアントプログラム（Java）

ソケットを作成

```
Socket socket = new Socket(
            “hoge.com”, 10000 );
```

入力ストリームを作成

```
DataInputStream is =
  new DataInputStream (
    new BufferedInputStream(
      socket.getInputStream()));
```

クライアントプログラム（Java）

サーバ側から送信された文字列を受信

```
n = is.read(buff);
System.out.write(buff, 0, n);
```

ストリーム，ソケットをクローズ

```
is.close();
socket.close();
```

サーバプログラム（Java）

ソケット作成，コネクション要求受付待機

```
ServerSocket svSocket =
  newServerSocket(10000);
Socket cliSocket = svSocket.accept();
```

出力ストリーム作成

```
DataOutputStream os =
  new DataOutputStream(
    new BufferedOutputStream(
      cliSocket.getOutputStream());
```

サーバプログラム（Java）

文字列を送信

```
String s = new String("Hello World!!¥n");
byte[] b = s.getBytes();
os.write(b, 0, s.length());
```

ストリーム，ソケットをクローズ

```
os.close();
cliSocket.close();
svSocket.close();
```

InetAddress

- http://docs.oracle.com/javase/jp/6/api/java/net/InetAddress.html
- **IPアドレスを扱うクラス**
- **java.net**
  クラス InetAddress
- java.lang.Object **java.net.InetAddress すべての実装されたインタフェース**
  :Serializable**直系の既知のサブクラス**
  :Inet4Address, Inet6Address

InetAddress

- static String **getHostName**()
  この IP アドレスに対応するホスト名を取得。
- static InetAddress**getByAddress**(String host, byte[] addr)
  指定されたホスト名および IP アドレスに基づいて InetAddress を作成します。

- static InetAddress**getLocalHost**()
  ローカルホストを返します。

InetAddress

- static InetAddress**getByName**(String host)
  指定されたホスト名を持つホストの IP アドレスを取得します。

- boolean**isReachable**(int timeout)
  そのアドレスに到達可能かどうかをテストします

- String**toString**()
  この IP アドレスを String に変換します。

Objectの配列

Objectの配列：FaceObjKadaiAns.java

- FaceObjAns**[]** fobjs = **new FaceObjAns[9];**

```
for (int j = 0; j < 3; j++) {//行
    yStart = j * 220 + 50;
    for (int i = 0; i < 3; i++) {//列
        xStart = i * 220 + 40;
        fobjs[i + 3 * j] = new FaceObjAns(xStart, yStart);
    }
}
```

描画

```
// 9個数の顔を書く
for (int i = 0; i < fobjs.length; i++) {
   fobjs[i].makeFace(g);
}
```

FaceObjectのコンストラクタ

```
class FaceObjAns {
// コンストラクタ
int h;
int w;
int xStart = 0;
int yStart = 0;
public FaceObjAns(int x, int y) {
        h = 200;
        w = 200;
        this.xStart = x;
        this.yStart = y;
}
// 個々にメソッドを追加
public void makeFace(Graphics g) {
        makeRim(g);
        makeEyes(g, 20);
        makeNose(g, 40);
        makeMouth(g, 80);
}
```

Interface

- 内容に抽象メソッドしか持たないクラスのようなもの(バールのようなもの)をインタフェースと呼びます。
- クラスと並んで、パッケージのメンバーとして存在します。
- インタフェースはクラスによって実装(implements) され、
- 実装クラスはインタフェースで宣言されてい抽象メソッドを実装します。

インタフェースの複数実装

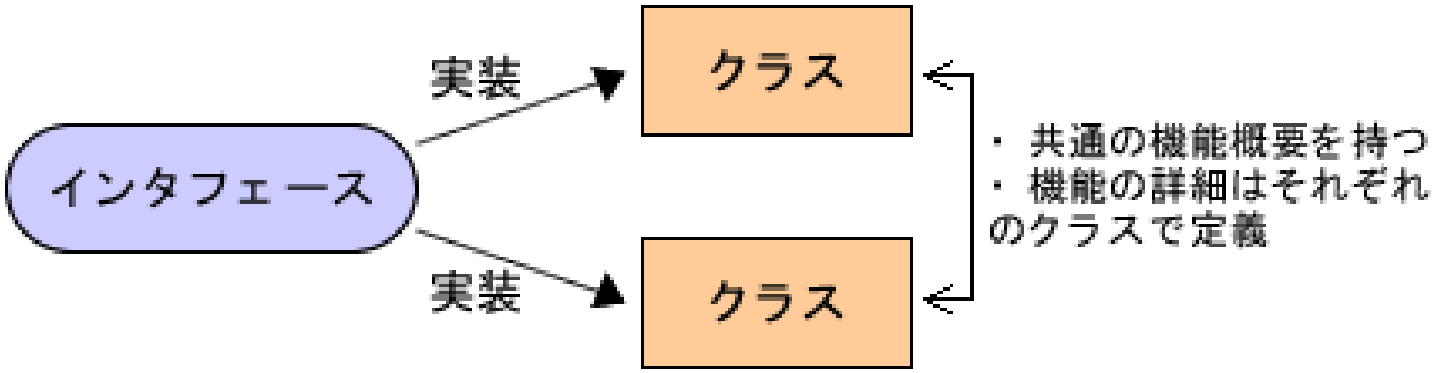
- クラスの場合は、単一のクラスしか継承 (extends) できせんが、インタフェースの場合は、複数のインタフェースを実装 (implements) することができます。

class InterfaceImpl ***implements*** Interface1, interface2, interface

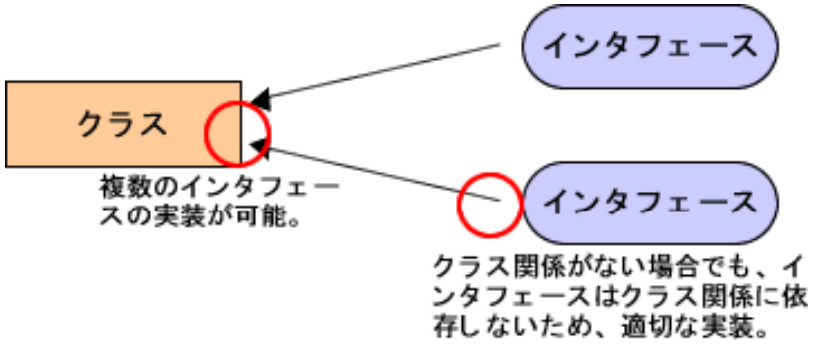
... }

interface の修飾子は public のみ。

インタフェースのメリット



- Javaは複数のスーパークラスを継承することはできません。Javaでは複数のスーパークラスの継承（多重継承）が認められていないため。



Plugin hybrid車は災害時にバッテリとして利用可能

ICar.javaインタフェース

- 車輪in getNumOfTiers()がある。
- スピートを設定する
  - void setSpeed (int sp)
  - int getSpeed()
  - void printCarName()

IElectricCharge.javaインタフェース

- void chargeBattery(int b)
- int getAllBattery()
- int consumeBattery(int b)

HybridCarImpl.java

- を実装してください。
- Yourには自分の名前を入れてください。
- 例　MasaHybridCarImpl.java

呼び出しのMainCall.javaを実装しよう。

- Hint
- MasaHybridCarImpl masaCar= new MasaHybridCarImpl();

- ICar car=(ICar) masaCar;
- car.setSpeed(); car.printCarName();
- IElectricCharge charger =(IElectricCharge) masaCar
- carger.chargeBattery(100);

Thread

Thread,Runnable

MovingBall

ThreadSleep　停止

- 300ミリ秒処理を停止する。

```
try{
   Thread.sleep(3000);
   //3000ミリ秒Sleepする
}catch(InterruptedException e){}
```

Threadの2種類の作りかた

- 1)**implements Runnable**

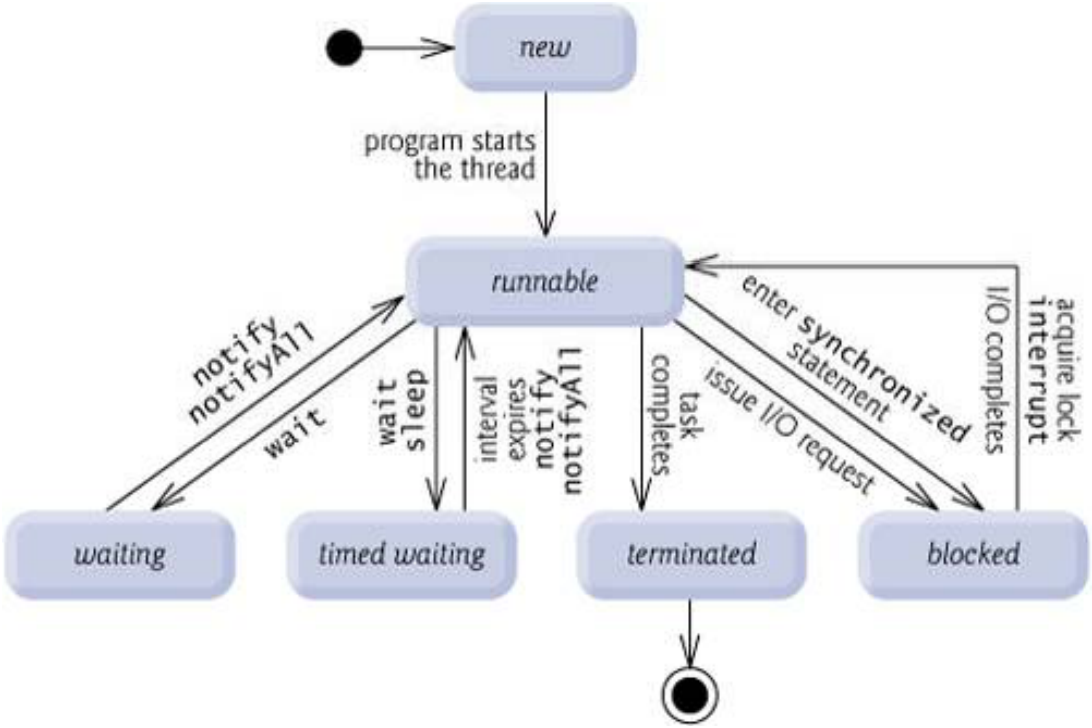
Runnableを実装したクラスをThreadクラスのコンストラクタとして渡す。start()で開始。

```
CountTenRunnable ct = new CountTenRunnable();
Thread th = new Thread(ct);
th.start();
```

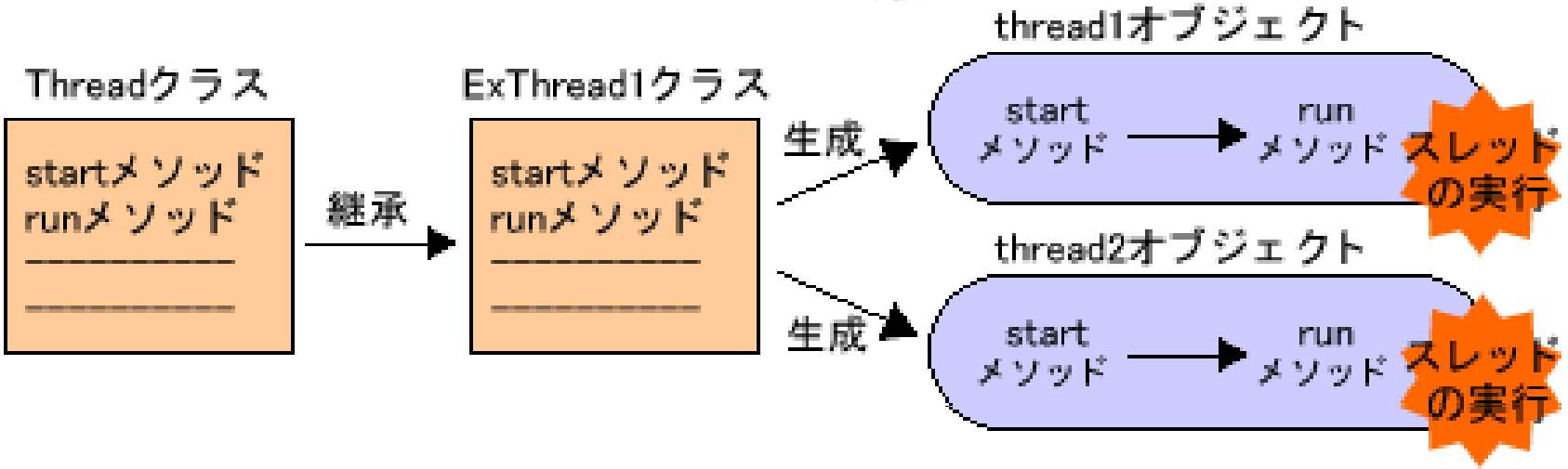
- 2)**extends Thread**

Threadを拡張したクラスをnew してstart()メソッドを呼び出す。

- (1)


new
program starts the thread
runnable
notify
notifyAll
wait
wait
sleep
interval expires
notify
notifyAll
task completes
issue I/O request
enter synchronized statement
I/O completes
acquire lock
interrupt
waiting
timed waiting
terminated
blocked

- (2)


Threadクラス
startメソッド
runメソッド
継承
ExThread1クラス
startメソッド
runメソッド
生成
生成
thread1オブジェクト
start メソッド
run メソッド
スレッドの実行
thread2オブジェクト
start メソッド
run メソッド
スレッドの実行

```
class MainThread{
     public static void main(String args[]){
      /* 別スレッドとして動作させるオブジェクトを作成 */
     SubThread sub = new SubThread(); /* 別のスレッドを作
    成し、スレッドを開始する */
     Thread thread = new Thread(sub) thread.start();
     }
    }
    class SubThread implements Runnable{
          public void run(){ }
    }
```

- CountTest.java
- CountTenRunnable.java
- CountTesterTweThreads.java

MovingBall

```
MovingInnerFFrame f = new MovingInnerFFrame();
Thread th = new Thread(f);
th.start();


class MovingInnerFFrame extends Frame
    implements Runnable {

    public void run() {}
}
```