

pi-14. イベント処理とネットワーク通信: Javaによるイベントハンドラとソケットプログラミング

トピックス: イベント, イベントハンドラ, タイマーイベント, ソケット通信

URL: <https://www.kkaneko.jp/pro/pi/index.html>

(Java の基本, スライド資料とプログラム例)

金子邦彦



今回の内容



- イベントの処理

イベントハンドラ、イベントの登録のプログラム、
マルチスレッドの仕組み

- 通信の基礎

ソケットのバインド, ソケットの接続要求

番号	項目
	復習
14-1	イベントとイベントハンドラ
14-2	Java でのタイマーイベント
14-3	タイマーイベントのプログラム例
14-4	ソケット通信
14-5	Java でのソケット通信

各自、資料を読み返したり、課題に取り組んだりも行う

この授業では、**Java** を用いて基礎を学び、マスターする



```
Main.java
1 public class Main
2 {
3     public static void main(String[] args) {
4         int x = 100;
5         int y = 200;
6         System.out.printf("%d\n", x + y);
7     }
8 }

input
300

...Program finished with exit code 0
Press ENTER to exit console.
```

Java などのプログラミング言語の体験，演習ができるオンラインサービス

GDB online

<http://www.pythontutor.com/>

オンラインなので、「秘密にしたいプログラム」を扱うには十分な注意が必要

GDB online で Java を動かす手順



① ウェブブラウザを起動する

② 次の URL を開く

<https://www.onlinegdb.com>



https://www.onlinegdb.com

③ 「Language」 のところで, 「Java」 を選ぶ

SPONSOR Slack — Bring your team together with Slack, the collaboration hub for work.

Run Debug Stop Share Save { } Beautify Language -- select --

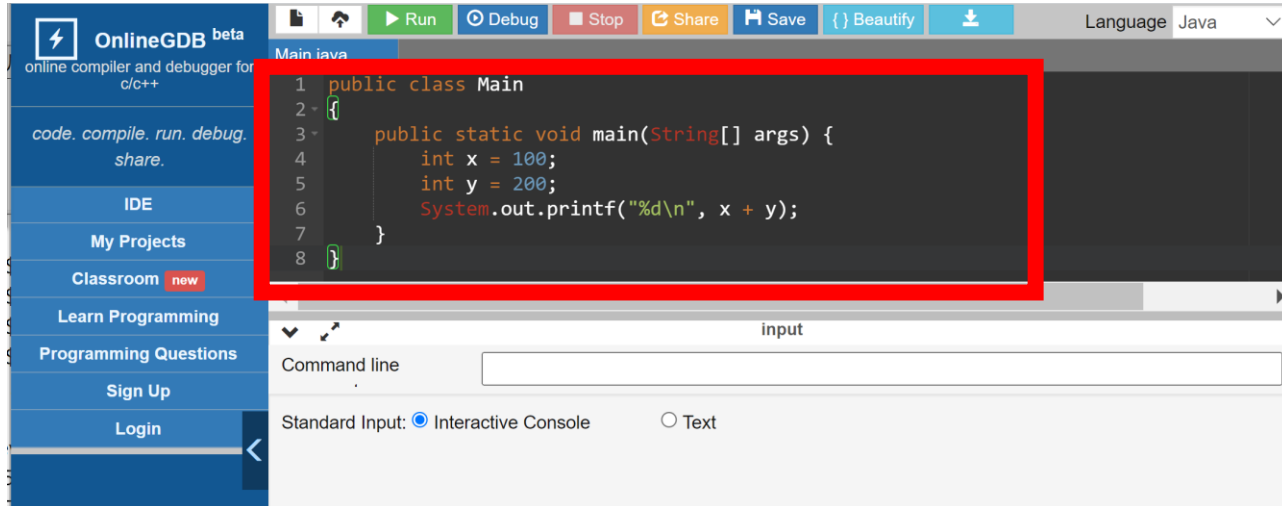
source code

```
1 /*****  
2  
3 Welcome to GDB Online.  
4 GDB online is an online compiler and debugger tool for C, C++, Python  
5 C#, VB, Perl, Swift, Prolog, Javascript, Pascal, HTML, CSS, JS  
6 Code, Compile, Run and Debug online from anywhere in world.  
7  
8 *****/  
9 #include <stdio.h>  
10  
11 int main()  
12 {  
13     printf("Hello World");  
14  
15     return 0;  
16 }  
17
```

-- select --

- C
- C++
- C++ 14
- C++ 17
- Java**
- Python 3
- PHP
- C#
- VB
- HTML,JS,CSS
- Ruby
- Perl
- Pascal
- R
- Fortran
- Haskell
- Assembly(GCC)
- Objective C
- SQLite

④ ソースコードを入れる

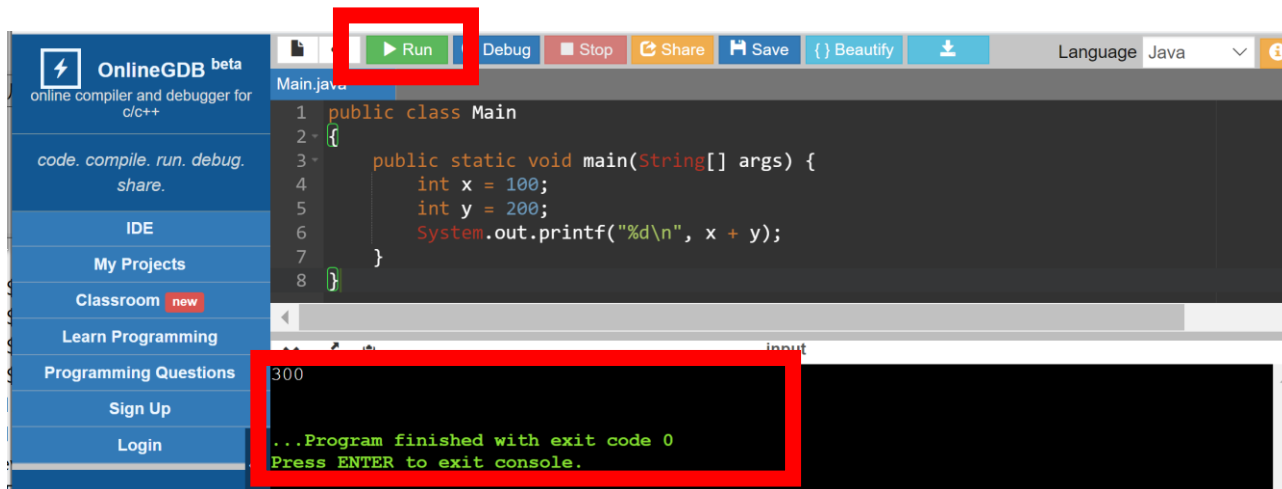


The screenshot shows the OnlineGDB IDE interface. The left sidebar contains navigation links: OnlineGDB beta, code.compile.run.debug.share, IDE, My Projects, Classroom new, Learn Programming, Programming Questions, Sign Up, and Login. The main editor area displays a Java file named 'Main.java' with the following code:

```
1 public class Main
2 {
3     public static void main(String[] args) {
4         int x = 100;
5         int y = 200;
6         System.out.printf("%d\n", x + y);
7     }
8 }
```

A red rectangular box highlights the entire code block. Below the editor, there is an 'input' section with a 'Command line' text box and 'Standard Input' options: 'Interactive Console' (selected) and 'Text'.

⑤ 実行. 実行結果を確認 「Run」をクリック.



The screenshot shows the OnlineGDB IDE interface after clicking the 'Run' button. The 'Run' button in the top toolbar is highlighted with a red box. The main editor area still displays the same Java code. Below the editor, the 'output' section shows the execution result:

```
300
...Program finished with exit code 0
Press ENTER to exit console.
```

A red rectangular box highlights the output text. The 'Standard Input' options are still visible below the output.

14-1. イベントとイベントハンドラ

イベントとイベントハンドラ



イベントの例

- タイマー
- 外部操作（キーボード、マウスの操作）

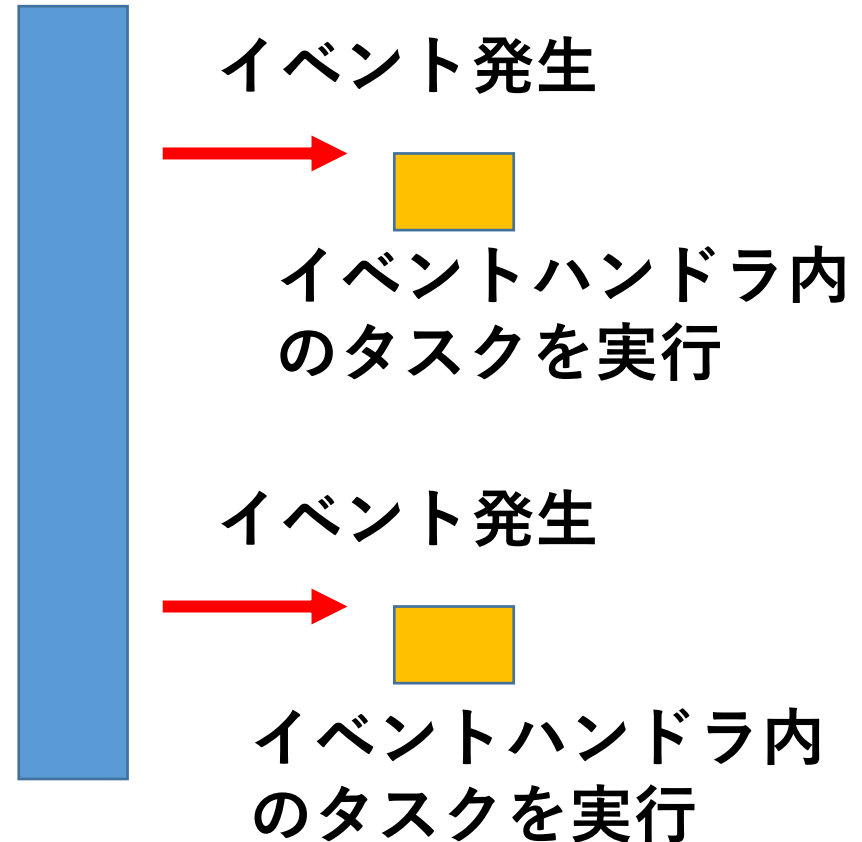
イベントハンドラ

イベントが発生したときの処理のこと。特定のイベントごとに、事前に、行うべき処理を登録しておく。

通常のプログラム実行



イベント発生の際に
マルチスレッド実行



14-2. Java でのタイマーイベント

演習

資料 : 13 ~ 18

【トピックス】
・ タイマーイベント

処理を止める



① 次のソースコードを入れる

```
1  import java.util.*;
2
3  public class Main {
4      public static void main(String[] args) {
5          System.out.println("start");
6          try {
7              Thread.sleep(100000);
8          } catch (InterruptedException e) {}
9      }
10 }
```

100秒止まる
「**100000**」
とあるのは
ミリ秒単位

② 実行結果の確認 （あとで使うので、ブラウザを閉じないこと）

「start」と表示される



```
1 import java.util.*;
2
3 public class Main {
4     public static void main(String[] args) {
5         System.out.println("start");
6         try {
7             Thread.sleep(100000);
8         } catch (InterruptedException e) {}
9     }
10 }
11
```

input

start

Java でのタイマーイベントとイベントハンドラ



イベントハンドラ

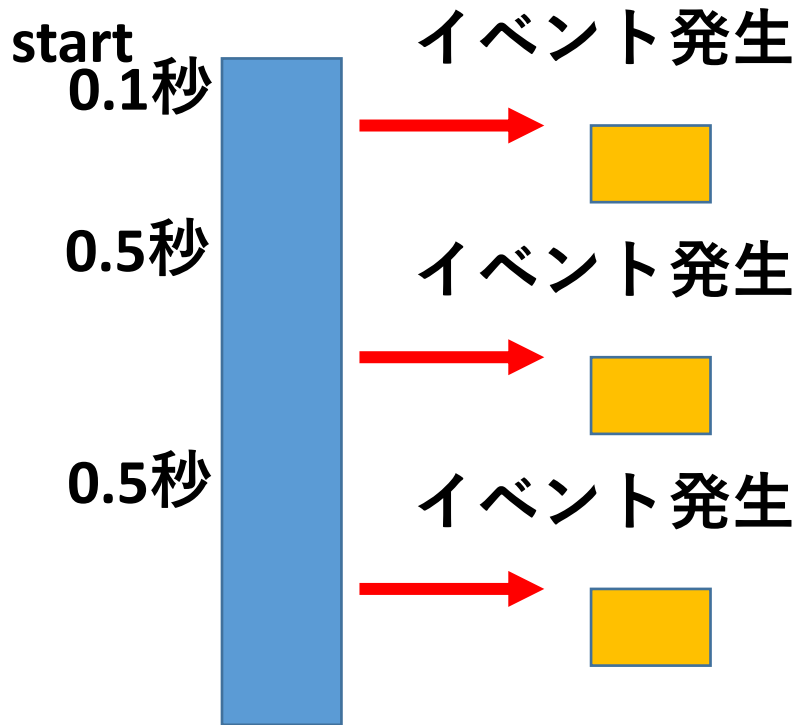
- 「TimerTask」は Java の標準ライブラリのクラス

```
1 import java.util.*;
2
3 class MyTask extends TimerTask {
4     public void run() {
5         System.out.println("hello");
6     }
7 }
8
9 public class Main
10 {
11     public static void main(String[] args) {
12         Timer timer = new Timer();
13         timer.schedule(new MyTask(), 100, 500);
14         System.out.println("start");
15         try {
16             Thread.sleep(100000);
17         } catch (InterruptedException e) {}
18     }
19 }
```

イベントハンドラ の登録

1. 最初 0.1 秒経過したらイベント発生
2. その後, 0.5 秒間隔でイベント発生

マルチスレッドでの実行



メインのスレッド（青）は動き続ける

タイマーイベント

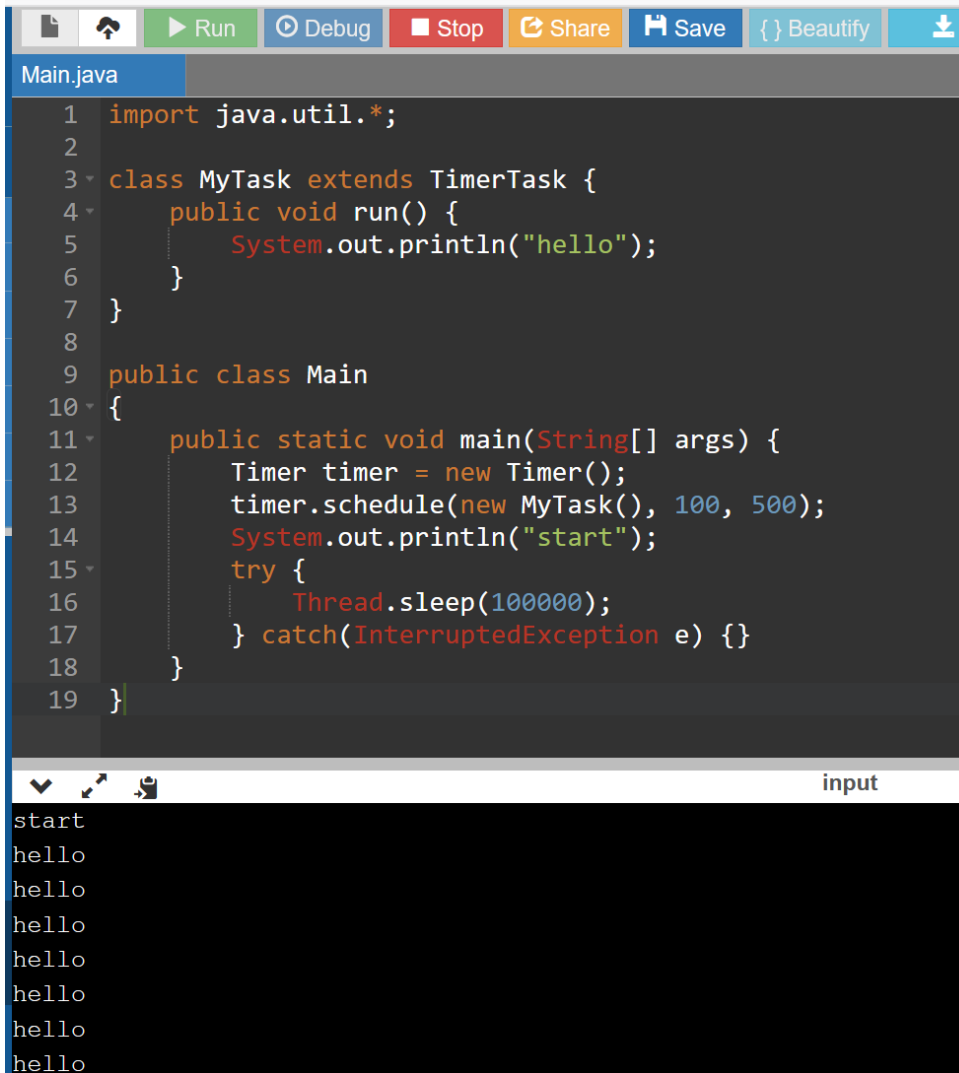


③ 次のソースコードを入れる

```
1  import java.util.*;
2
3  class MyTask extends TimerTask {
4      public void run() {
5          System.out.println("hello");
6      }
7  }
8
9  public class Main
10 {
11     public static void main(String[] args) {
12         Timer timer = new Timer();
13         timer.schedule(new MyTask(), 100, 500);
14         System.out.println("start");
15         try {
16             Thread.sleep(100000);
17         } catch (InterruptedException e) {}
18     }
19 }
```

④ 実行結果の確認 （あとで使うので、ブラウザを閉じないこと）

「start」と表示される



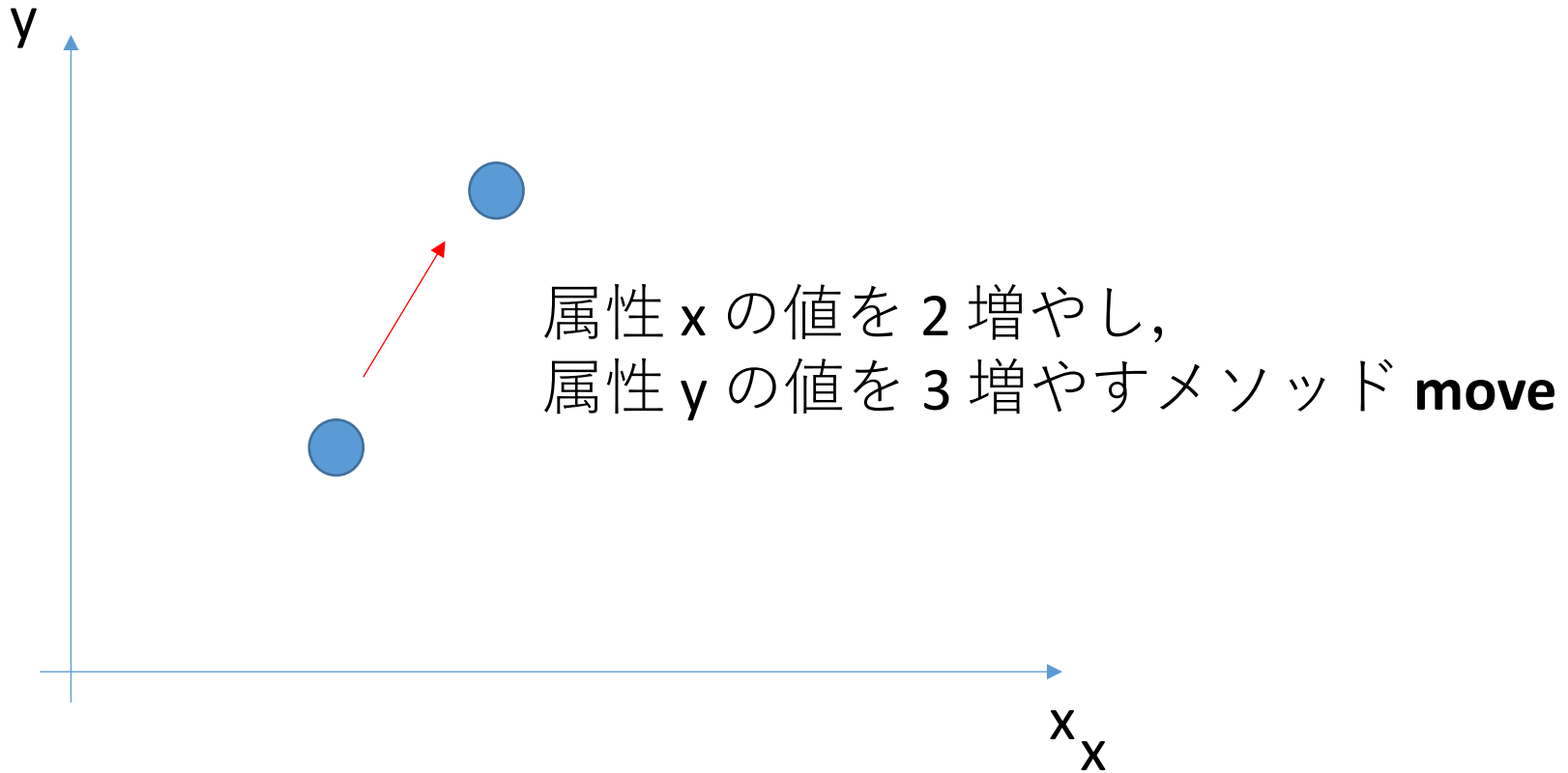
```
1 import java.util.*;
2
3 class MyTask extends TimerTask {
4     public void run() {
5         System.out.println("hello");
6     }
7 }
8
9 public class Main
10 {
11     public static void main(String[] args) {
12         Timer timer = new Timer();
13         timer.schedule(new MyTask(), 100, 500);
14         System.out.println("start");
15         try {
16             Thread.sleep(100000);
17         } catch (InterruptedException e) {}
18     }
19 }
```

input

start
hello
hello
hello
hello
hello
hello
hello
hello

14-3. タイマーイベントのプログラム例

Ball クラスのオブジェクト



演習

資料 : 22 ~ 27

【トピックス】
・ タイマーイベント

① 次のソースコードを入れる



```
1 class Ball {
2     double x;
3     double y;
4     public Ball(double x, double y) {
5         this.x = x;
6         this.y = y;
7     }
8     public void move() {
9         this.x = this.x + 2;
10        this.y = this.y + 3;
11    }
12    public void printout() {
13        System.out.printf("%f %f\n", this.x, this.y);
14    }
15 }
```

次のページに続く

```
16
17 public class Main
18 {
19     public static void main(String[] args) {
20         Ball b = new Ball(0, 0);
21         b.printout();
22         b.move();
23         b.printout();
24     }
25 }
```

② 実行結果の確認

```
0.0000000 0.0000000
2.0000000 3.0000000
```

- **0.5 秒ごとに動かす**

0.5 秒ごとにイベント発生

③ 次のように書き換える



```
1  import java.util.*;
2
3  class Ball {
4      double x;
5      double y;
6      public Ball(double x, double y) {
7          this.x = x;
8          this.y = y;
9      }
10     public void move() {
11         this.x = this.x + 2;
12         this.y = this.y + 3;
13     }
14     public void printout() {
15         System.out.printf("%f %f\n", this.x, this.y);
16     }
17 }
```

次のページに続く



イベント ハンドラ

```
19 class MyTask extends TimerTask {
20     Ball b;
21     public MyTask() {
22         super();
23         this.b = new Ball(0, 0);
24     }
25     public void run() {
26         b.move();
27         b.printout();
28     }
29 }
```

```
30
31 public class Main
32 {
33     public static void main(String[] args) {
34         Timer timer = new Timer();
35         timer.schedule(new MyTask(), 100, 500);
36         System.out.println("start");
37         try {
38             Thread.sleep(100000);
39         } catch (InterruptedException e) {}
40     }
41 }
```

イベント
ハンドラ
の登録と、
メインの
スレッド

④ 実行結果の確認



```
start
2.000000 3.000000
4.000000 6.000000
6.000000 9.000000
8.000000 12.000000
10.000000 15.000000
12.000000 18.000000
14.000000 21.000000
16.000000 24.000000
18.000000 27.000000
```

0.5 秒ごとに動く

14-4. ソケット通信

ソケットとは

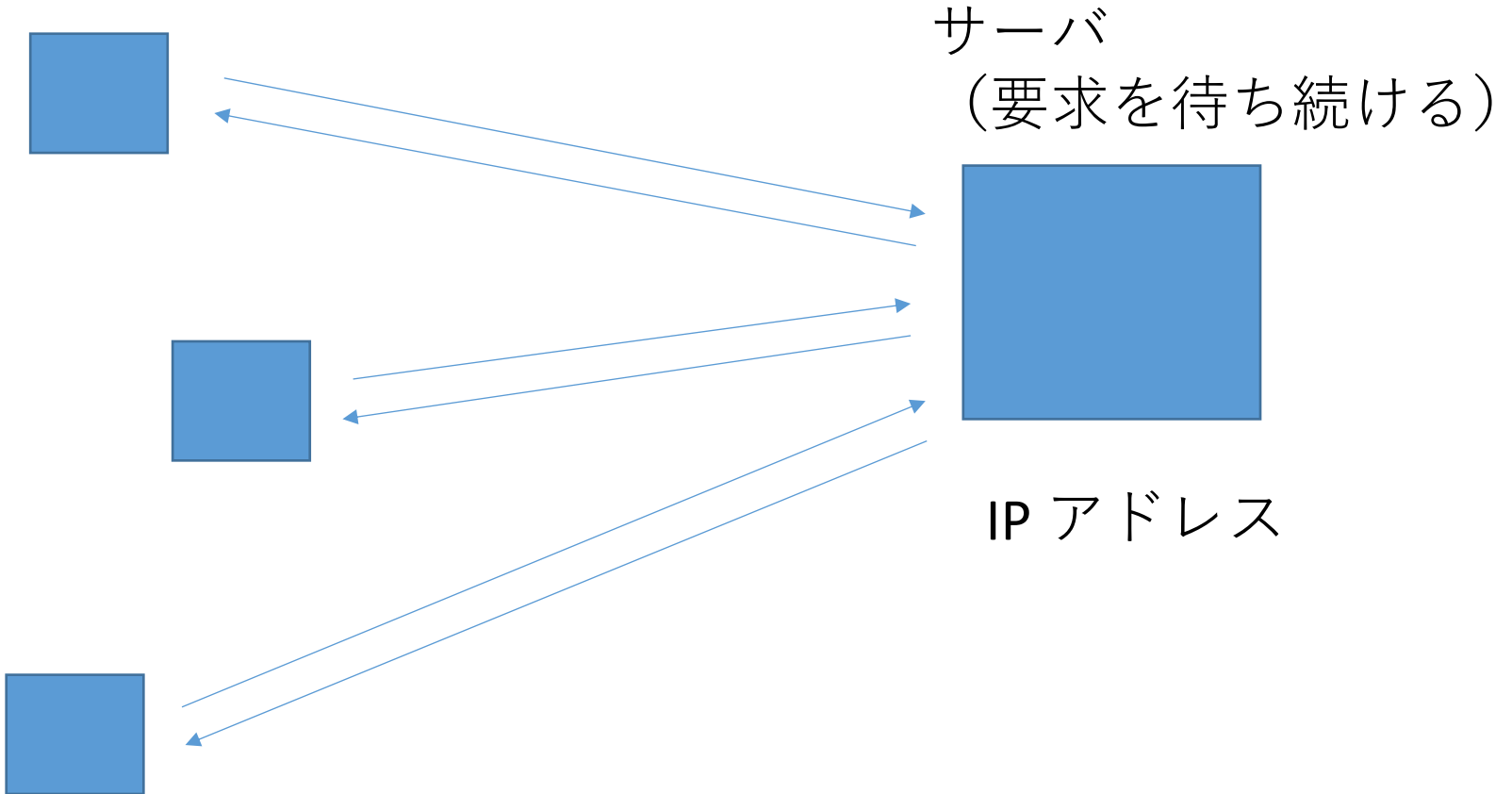


ソケットは、機器の間の通信や、プロセス間通信に使用されるインターフェース。

インターネットでの通信



さまざまなコンピュータ
(要求する)

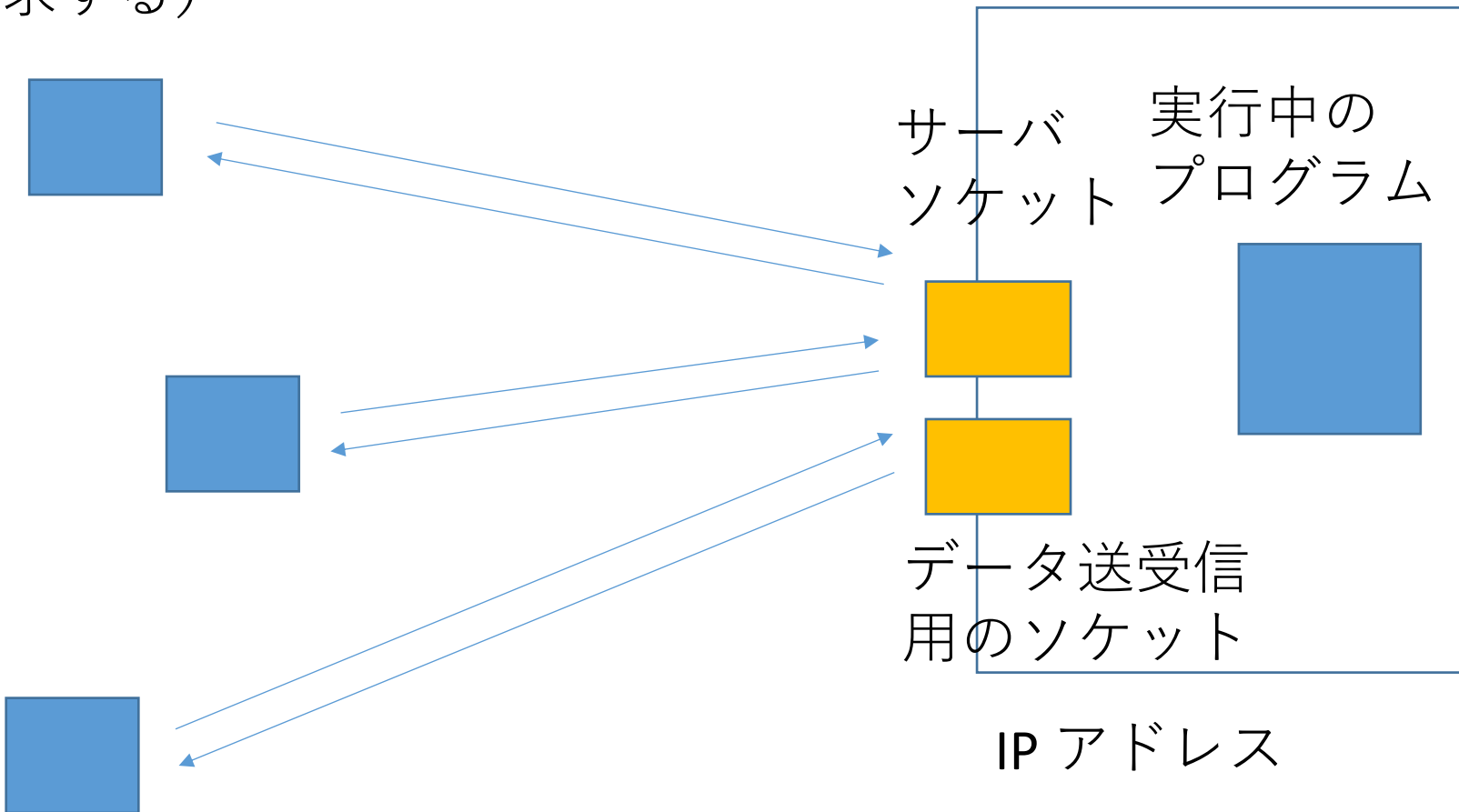


プログラムにバインドされたポート



さまざまなコンピュータ
(要求する)

サーバ
(要求を待ち続ける)



インターネットでの通信



- サーバ（要求を待ち続ける）

実行中のプログラムは、**サーバソケットにバインド**される

- さまざまなコンピュータ（要求する）

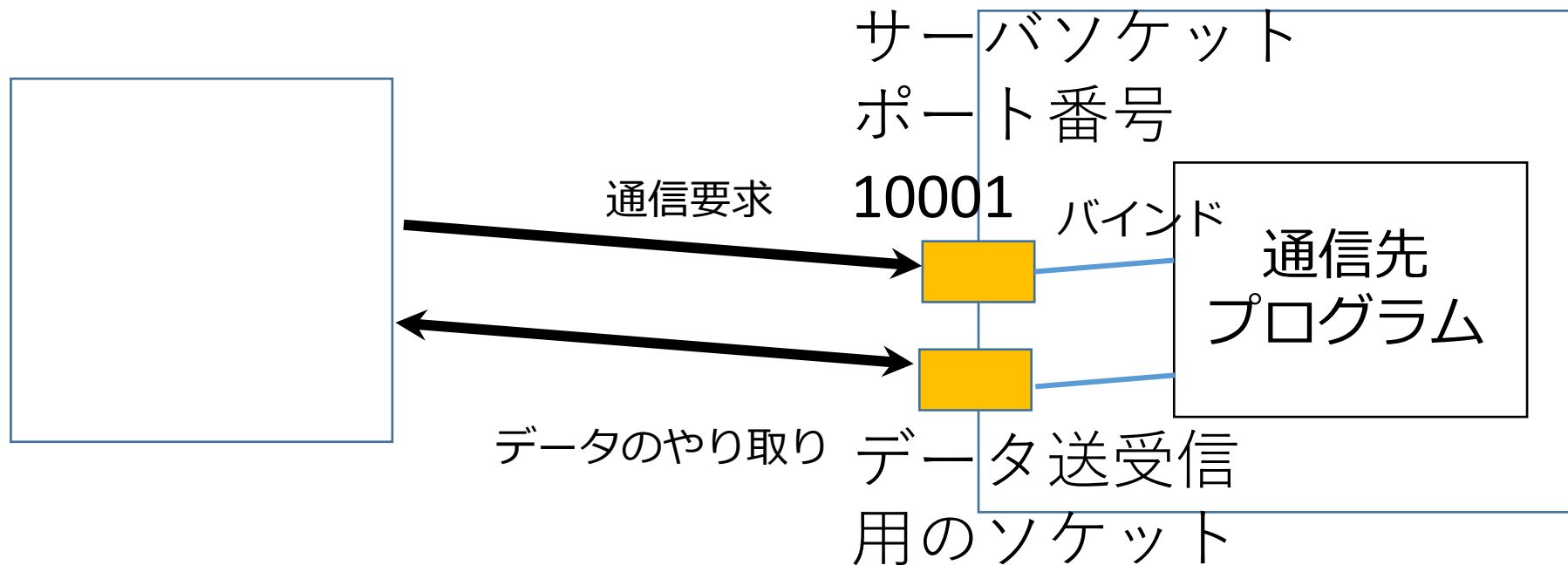
サーバの **IP アドレス（あるいはサイト名）** と **サーバソケットのポート番号**を指定して、通信を開始する

インターネットでの通信



通信元コンピュータ

通信先コンピュータ



14-5. Java でのソケット通信

さまざまなコンピュータ
(要求する)

サーバの **IPアドレス** (または
サイト名) とサーバソケット
の **ポート番号** を指定して
通信を開始



データの送受信

サーバ
(要求を待ち続ける)

プログラム を **サーバ**
ソケット に **バインド**



接続要求が来るのを待つ
(**accept** という)



データの送受信

サーバのプログラムの例



プログラムを、ポート番号 10001 にバインド。接続要求が来るのを待って、データの送受信を行う

```
ServerSocket sock;  
Socket s;  
BufferedReader reader;  
sock = new ServerSocket();  
sock.bind(new InetSocketAddress("<サイト名>", 10001));  
  
s = sock.accept();  
reader = new BufferedReader(new InputStreamReader  
                                (s.getInputStream()));  
String line = reader.readLine();
```

サーバソケットとデータ送受信のソケット



ServerSocketer sock;

Socket s;

BufferedReader reader;

sock = new ServerSocket();

sock.**bind**(new InetSocketAddress("<ホスト名>",
10001));

サーバソケット

データ送受信のソケット
※ データ送受信に使うソケット.
「サーバソケット」とは別のもの

s = sock.**accept**();

reader = new BufferedReader(new InputStreamReader
(s.getInputStream()));

String line = reader.**readLine**();

サーバとの通信のプログラムの例



サーバの IP アドレス（またはサイト名） と、ポート番号 10001 を指定して通信を開始。

```
Socket s = new Socket("<サーバのサイト名>", 10001);
```

```
BufferedWriter out = new BufferedWriter  
    (new OutputStreamWriter(s.getOutputStream()));  
out.write("GET /index.html HTTP/1.0¥r¥n");  
out.flush();
```

演習

資料：40 ～ 45

【トピックス】
・ ソケット通信

演習



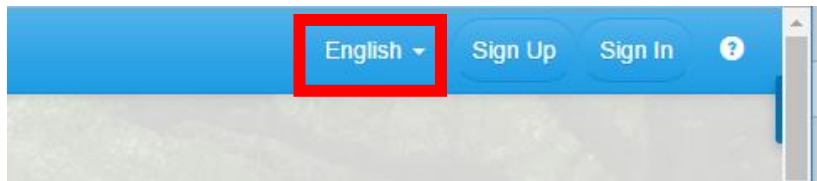
① ウェブブラウザを起動する

② 次の URL を開く

<https://paiza.io/>



③ もし、表示が英語になっていたら、**日本語**に切り替える



④ 「コード作成を試してみる」をクリック



⑤ 「Java」を選ぶ (左上のボタンをクリックするとメニューが出る)





Beta **paiza.io** 新規コード 一覧 ウェブ開発 プロク

日本語

Java ▾ Enter a title here

Main.java ✕ +

```
1 import java.util.*;
2
3 public class Main {
4     public static void main(String[] args) throws Exception {
5         // Your code here!
6
7         System.out.println("XXXXXXXXX");
8     }
9 }
10
```

▶ 実行 (Ctrl-Enter) ⓘ

入力

プログラムの
編集画面

プログラムを
書き換えること
ができる

実行ボタン

編集画面を確認する。

すでに、Java プログラムが入っている。

書き換えて使う。

```
1 import java.util.*;
2
3 public class Main {
4     public static void main(String[] args) throws Exception {
5         // Your code here!
6
7         System.out.println("XXXXXXXXX");
8     }
9 }
10
```

① 次のソースコードを入れる



```
1 import java.util.*;
2 import java.net.*;
3 import java.io.*;
4
5 public class Main {
6     public static void main(String[] args) throws Exception {
7         try {
8             Socket s = new Socket("www.kkaneko.jp", 80);
9             BufferedReader in = new BufferedReader
10                 (new InputStreamReader(s.getInputStream()));
11             BufferedWriter out = new BufferedWriter
12                 (new OutputStreamWriter(s.getOutputStream()));
13             out.write("GET /index.html HTTP/1.0\r\n");
14             out.write("Host: www.kkaneko.jp:80\r\n");
15             out.write("\r\n");
16             out.flush();
17             System.out.println(in.readLine());
18             System.out.println(in.readLine());
19             System.out.println(in.readLine());
20             System.out.println(in.readLine());
21         } catch (Exception e) {
22             e.printStackTrace();
23         }
24     }
25 }
```

② 実行をクリック．実行結果を確認



出力 入力 コメント 0

```
HTTP/1.1 301 Moved Permanently
Date: Wed, 15 Dec 2021 00:39:44 GMT
Server: Apache/2.4.41 (Ubuntu)
Location: https://www.kkaneko.jp/index.html
```

まとめ



- **ソケット**の機能により, **インターネットでの通信**ができる
- **サーバ側**では, **プログラム**を, **サーバソケット**を**バインド**する.
- **サーバとの通信**では, **サーバの IP アドレス** (またはサイト名) と、**ポート番号**を指定して通信を開始する.

関連ページ

- **Java プログラミング入門**

GDB online を使用

<https://www.kkaneko.jp/pro/ji/index.html>

- **Java の基本**

Java Tutor, GDB online を使用

<https://www.kkaneko.jp/pro/pi/index.html>

- **Java プログラム例**

<https://www.kkaneko.jp/pro/java/index.html>

14-2



```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        System.out.println("start");
        try {
            Thread.sleep(100000);
        } catch (InterruptedException e) {}
    }
}
```


14-2



```
import java.util.*;
```

```
class MyTask extends TimerTask {  
    public void run() {  
        System.out.println("hello");  
    }  
}  
  
public class Main  
{  
    public static void main(String[] args) {  
        Timer timer = new Timer();  
        timer.schedule(new MyTask(), 100, 500);  
        System.out.println("start");  
        try {  
            Thread.sleep(100000);  
        } catch (InterruptedException e) {}  
    }  
}
```

14-3



```
class Ball {
    double x;
    double y;
    public Ball(double x, double y) {
        this.x = x;
        this.y = y;
    }
    public void move() {
        this.x = this.x + 2;
        this.y = this.y + 3;
    }
    public void printout() {
        System.out.printf("%f %f\n", this.x, this.y);
    }
}

public class Main
{
    public static void main(String[] args) {
        Ball b = new Ball(0, 0);
        b.printout();
        b.move();
        b.printout();
    }
}
```

14-3



```
import java.util.*;

class Ball {
    double x;
    double y;
    public Ball(double x, double y) {
        this.x = x;
        this.y = y;
    }
    public void move() {
        this.x = this.x + 2;
        this.y = this.y + 3;
    }
    public void printout() {
        System.out.printf("%f %f\n", this.x, this.y);
    }
}

class MyTask extends TimerTask {
    Ball b;
    public MyTask() {
        super();
        this.b = new Ball(0, 0);
    }
    public void run() {
        b.move();
        b.printout();
    }
}

public class Main
{
    public static void main(String[] args) {
        Timer timer = new Timer();
        timer.schedule(new MyTask(), 100, 500);
        System.out.println("start");
        try {
            Thread.sleep(100000);
        } catch (InterruptedException e) {}
    }
}
```

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

public class Main {

    public static void main(String[] args) throws Exception {

        try {

            Socket s = new Socket("www.kkaneko.jp", 80);

            BufferedReader in = new BufferedReader
                (new InputStreamReader(s.getInputStream()));

            BufferedWriter out = new BufferedWriter
                (new OutputStreamWriter(s.getOutputStream()));

            out.write("GET /index.html HTTP/1.0\r\n");
            out.write("Host: www.kkaneko.jp:80\r\n");
            out.write("\r\n");

            out.flush();

            System.out.println(in.readLine());

            System.out.println(in.readLine());

            System.out.println(in.readLine());

            System.out.println(in.readLine());

        } catch(Exception e) {

            e.printStackTrace();

        }

    }

}
```