並列分散コンピューティング
(6)並行プログラミング
Java編

大瀧保広

今日の内容

- かる～く Javaの確認
- Javaによるスレッドの実装方法
  - Threadのサブクラス
  - Runnableインタフェース
  - インナークラス

- 排他制御と同期処理
- リクエストの処理
- 生産者消費者問題

Java：復習

- クラス：オブジェクトのひな形
  - C言語の構造体ぽいイメージ
  - メンバとして
    変数（フィールド）と
    関数（メソッド）がかける。
  - publicがつくと
    外から参照できる。
  - privateをつけると
    外からは参照できない。
    （そのクラスのメソッドのみが
    アクセスできる）

```
public class MyStack {
   private int stackPointer;
   private int[] stack;

   public MyStack() {
      stackPointer = 0;
      stack = new int[100];
   }

   public int push(int num) {
      return stack[stackPointer++] = num;
   }

   public int pop() {
      return stack[--stackPointer];
   }
}
```

Java：復習

```
public class Main {
    public static void main(String[] args) {
        MyStack sta = new MyStack();
        for (int i = 0; i < 10; i++) {
            sta.push( i );
        }
         for (int i = 0; i < 10; i++) {
              int x=sta.pop();
              System.out.println( x );
        }
    }
}
```

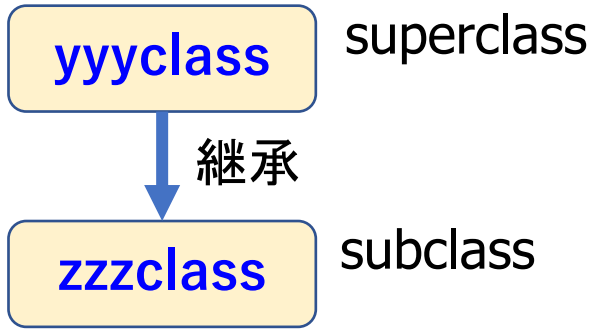
```
public class MyStack {
    private int stackPointer;
    private int[] stack;

    public MyStack() {
        stackPointer = 0;
        stack = new int[100];
    }

    public int push(int num) {
        return stack[stackPointer++] = num;
    }

    public int pop() {
        return stack[--stackPointer];
    }
}
```

- new でクラスのインスタンスが生成される。
- インスタンス作成時にはクラス名と同じ名前のメソッドである、「コンストラクタ」が実行される。

Java：復習

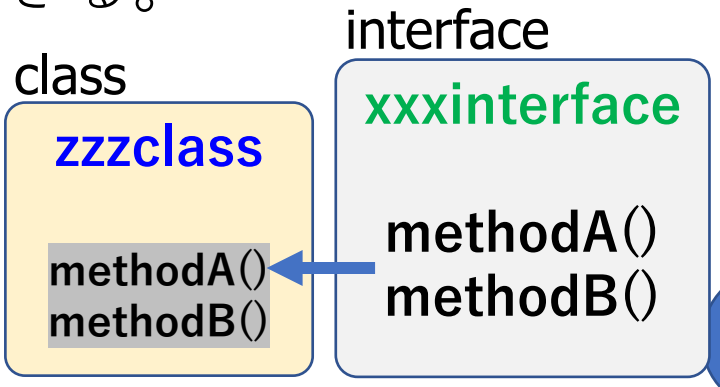
■クラスを定義するときに、他のクラスを継承して定義することができる。

```
public class zzzclass extends yyyclass {

}
```



■クラスを定義するときに、特定のメソッド群（インタフェース）を実装することができる。

```
public class zzzclass implements xxxinterface
{

}
```



Javaによるスレッドの起動方法

■Javaでは、スレッドの実行単位はクラス（オブジェクト）である。(C言語の関数に対応する メソッド ではない)

■Javaによるスレッドの実装方法は次の2つ。

1. java.lang.Threadクラスを継承したサブクラスを定義し、そのインスタンスをスレッドとして実行する。
2. Runnableインターフェスを実装したクラスを定義し、java.lang.Threadクラスのインスタンス上で実行する。

以下、順に説明します

(1) java.lang.Threadの継承

■Threadクラスを継承したサブクラスを定義し、runメソッドに動作を記述。

Thread クラスを継承したクラスを定義する。

```
class クラス名 extends Thread {
        public void run() {
                // スレッドで実行したい処理を記述
        }
}
```

(1) java.lang.Threadの継承

- スレッドを起動するコードでは、startメソッドを呼び出す。
- join()メソッドで終了を待つ。
  このとき、try-catch構文で例外処理を記述する必要がある。

```
Thread th = new クラス名();

th.start();

try {
  th.join();
} catch (InterruptedException e) {
  // 例外処理  e.printStackTrace(); とか。
}
```

th.start()メソッドを呼び出すと、
Javaの実行環境は新たにスレッドを
開始した上で、run()メソッドを実行する。

Threadオブジェクトのrun()メソッドを
直接呼び出すと、run()メソッドに書か
れた処理は、呼出し元のスレッドで
実行されてしまう。

サンプルプログラム(SimpleThread)

- PDC/Java-Thread/SimpleThread/
  - Main.java : ‘Ping! ’を100回出力
  - MyThread.java : ‘Pong.. ’を100回出力

- コマンドラインからのコンパイル（classファイルの生成）と実行

  javac Main.java MyThread.java

  java Main

- Makefileを利用したコンパイルと実行

  コンパイル: make

  実行：make run

スレッドのスリープ

- C言語のときと同様で、スレッドの実行時間が短すぎて
  並行に動いているか わかりにくい。
  sleepを入れて、動きを遅くしてみよう。

- JavaのsleepはThreadクラスのメソッドとして定義されている。
- 単位はミリ秒。ここも try-catch構文でエラー処理が必要。

```
try {
      Thread.sleep(100); // スレッドを100ミリ秒停止
} catch (InterruptedException e) {
}
```

サンプルプログラム(SimpleThread2)

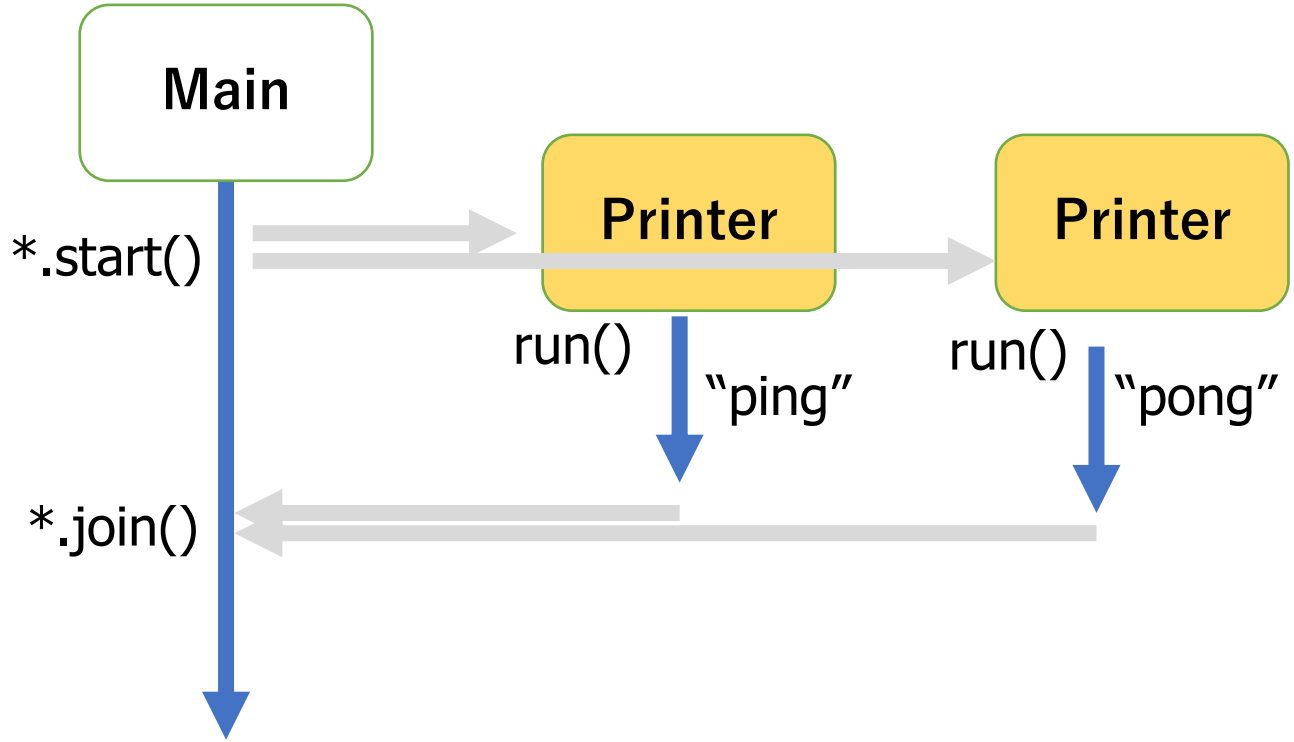
- Java-Thread/SimpleThread2/
  引数のあるサンプル

  - Main.java : Printerクラスのスレッドを2つ起動
  - Printer.java : 引数で渡された文字列を100回出力

- コンパイルと実行
  make
  make run

- SimpleThreadと同様、sleepを入れて実行せよ。

こんな感じ



(1')匿名インナークラスによるスレッド生成

■スレッドとして実行したいが、
　わざわざクラスを定義するほどのことでもないんだけど、、
という場合には、
匿名インナークラス（Anonymous Inner Class）
を使うことで、クラスを明示的には定義せずに
スレッドを生成することができる。

```
class AAAA extends Thread {
    public void run() {
        // スレッドで実行したい処理
    }
}
```

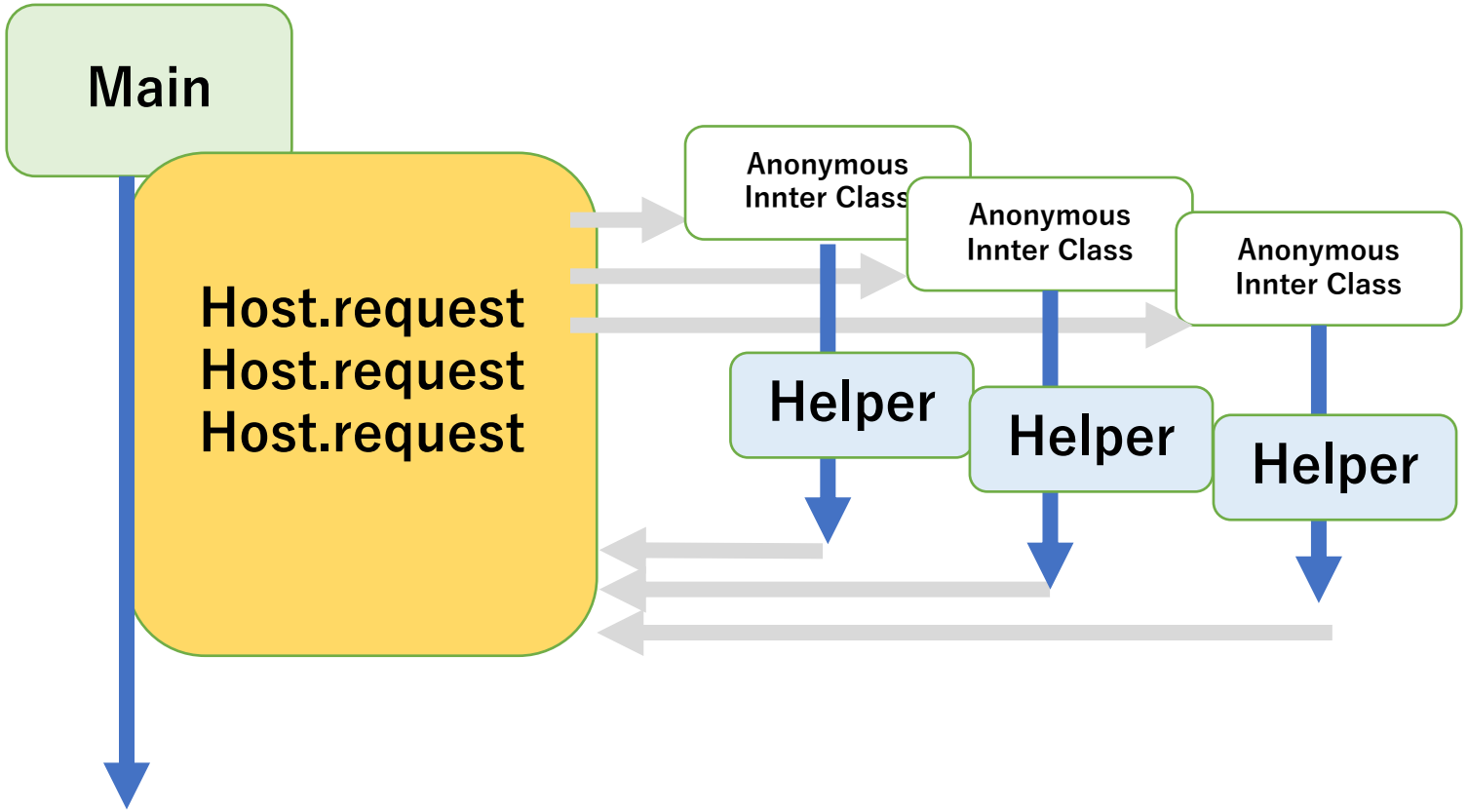
```
Thread th = new AAAA();
th.start();
try {
  th.join();
} catch( InterruptedException e){
}
```

```
Thread th=new Thread() {
    public void run() {
      //スレッドとして実行したい処理
    }
}
th.start();
try {
  th.join();
} catch( InterruptedException e){
}
```

(1')匿名インナークラス（Inner）

- **Java-Thread/Inner/**
  - 複数のスレッドを生成して、各スレッドに文字列を出力させるプログラム
- ファイル
  - **Main.java:**
    Hostクラスのインスタンスを生成後、
    そのrequestメソッドを引数を変えて3回呼び出す。
  - **Host.java:**
    requestメソッド内では、匿名インナークラスを利用してHelperクラスのメソッドを呼び出すスレッドを生成する。
    ここではjoinによる終了待ちをしないことに注意。
  - **Helper.java:**
    スレッド上で実行されるメソッドを持つクラス

こんな感じ


Main
Host.request
Host.request
Host.request
Anonymous
Innter Class
Anonymous
Innter Class
Anonymous
Innter Class
Helper
Helper
Helper

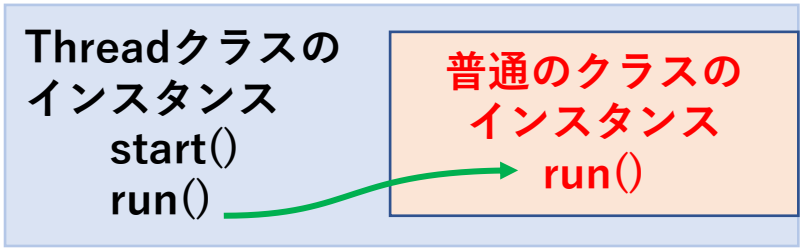
これで一応スレッド化はできるが…

- (1)の方法では、スレッドとして動かすクラスを Threadクラスのサブクラスとして作成している。
- Javaのクラスは、1つのクラスのサブクラスにしかなれない という制約がある。（↑多重継承の禁止。C++では可能。）
  - つまり既に他のクラスのサブクラスになっているクラスを、さらにThreadクラスのサブクラスにすることができない。

- これでは、並行に行いたい処理を すべてThreadクラスの サブクラスとして作るしかなくなる。 これは開発上 とても きつい制約である。

(2) Runnableインタフェースを実装する

- Threadクラスのサブクラスではないクラスを
  スレッドとして実行するには、次のようにする。
  - Threadクラスのインスタンスを1つ作成する。
    そのときに そのスレッド上で動かす普通のクラス
    (のインスタンス)を指定しておく。
  - Threadクラスのインスタンスをスレッドとして起動する
    と、その上で、指定していたインスタンスが実行される。



(2) Runnableインタフェースを実装する

/* 注意：このコードは正しくありません */

```
class クラスAA { //Threadクラスのサブクラスではない
        public void run(){
                //スレッドとして実行したい処理
        }
}
```

```
class MainThread{
        public static void main(String args[]){
                /* 別スレッドとして動作させたいクラスのインスタンス化 */
                クラス名 sub = new クラスAA ();

                /*実行したいインスタンスを渡してスレッドを作成 */
                Thread th = new Thread(sub);
                th.start();
        }
}
```

イメージとしてはこういう感じ。
しかし、Threadのコンストラクタの引数に指定できるクラスは、スレッドとして動作できるクラスでなければならないため、このままではエラーになる。

(2) Runnableインタフェースを実装する

- Runnableインタフェースは、Threadクラスのrun()と接続されるメソッドを提供する。
- スレッドとして実行したいクラスを定義するときに、Runnableインタフェースを実装し、run()に処理を記述する。

```
class クラス名 implements Runnable {
        public void run() {
                //スレッドとして実行したい処理
        }
}
```

- 定義したクラスのオブジェクトを引数に指定してスレッドクラスのスレッドを生成、起動する。

```
Runnable r1= new クラス名();
Thread th = new Thread(r1);
th.start();
```

サンプルプログラム(Runnable)

■Java-Thread/Runnable/
Runnableインタフェースを利用したスレッド実行

- ■Main.java : Printerクラスのスレッドを2つ起動
- ■Printer.java : 引数で渡された文字列を100回出力

■コンパイルと実行
make
make run

■SimpleThreadと同様、sleepを入れて実行せよ

JAVA-排他制御と同期

排他制御の方式

■(1) 処理ブロックの排他制御の仕方

```
synchronized (ロック用のインスタンス) {
    処理
}
```

■(2) 読み取りと書き込み専用のロック

```
ロックの確保
try {
    読み取り書き込み処理
} finally {
    ロックの解放
}
```

スレッドの排他制御

■排他的に処理したい処理を 一つのメソッドとして実装し、synchronizedメソッド（同期メソッド）として宣言する。

例　銀行の預金の引き下ろし

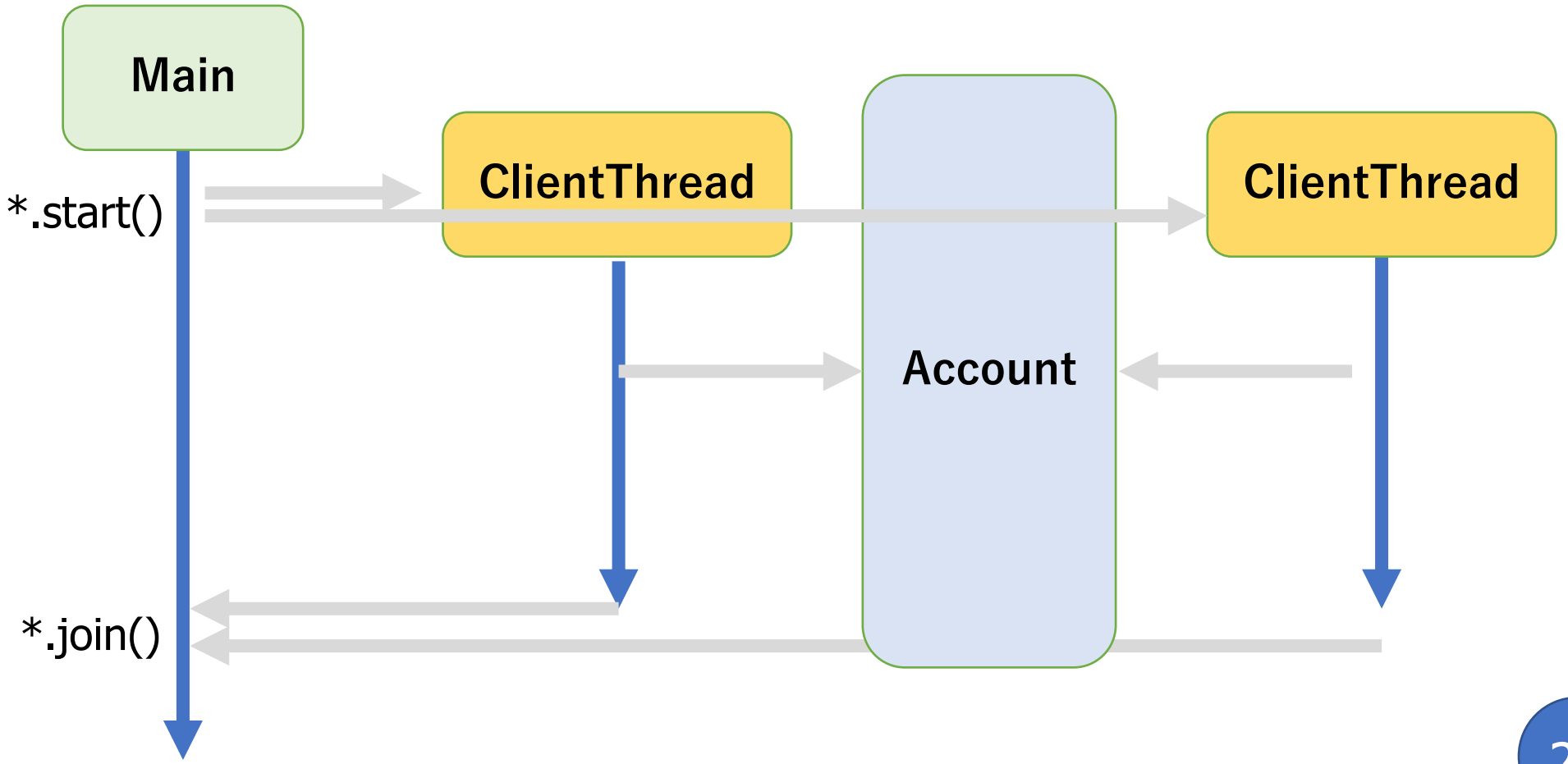
```
public synchronized boolean withdraw(int m) {
                                        // m: 引き下ろし額
        if (money >= m) {               // money: 預金残高
            money -= m;                 // m円引き下ろす
            return true;
        }
}
```

サンプルプログラム（Synchronized）

- Java-Thread/Synchronized/
  空っぽの口座に2つのスレッドが100万ずつを入れる。
- ファイル
  - Main.java: 口座と振り込みスレッドを2つ生成
  - Account.java ：口座クラス
  - ClientThread.java : 振り込みフレッド
    一円ずつ100万回 振り込む。

- 実行すると、残高が200万にならない。
  synchronized キーワードを追加して正しく動くようにせよ。

こんな感じ



サンプルプログラム（Reader）

- Java-Thread/Reader/
  ReaderWriter問題のプログラム

- ファイル
  - Main.java: 複数のReaderThread スレッドとWriterThread スレッドのインスタンスを生成
  - WriterThread.java: バッファに文字を書く処理
  - ReaderThread.java: バッファから文字を読み出して出力

  - Data.java: バッファ（文字を読む処理と書く処理）を定義
  - ReadWriteLock.java: 読み書きの処理のロックを定義

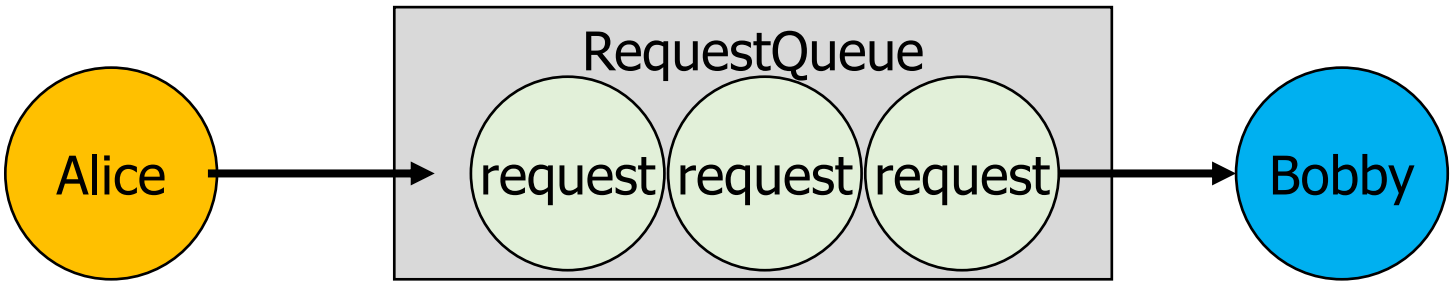
条件変数を利用した同期

- wait()：他のスレッドによって通知されるまで待機するメソッド
  (C言語での pthread_cond_wait()に相当)
- notify()：1つのスレッドを再開するメソッド
  (C言語での pthread_cond_signalに相当)
- notifyAll()：待機中のすべてのスレッドを再開するメソッド
  (C言語での pthread_cond_broadcastに相当)

```
while (論理条件 == null) {
    try {
        wait();
    } catch (InterruptedException e) {
    }
}
処理;
待っているスレッドに通知する;
```

サンプルプログラム（Request）

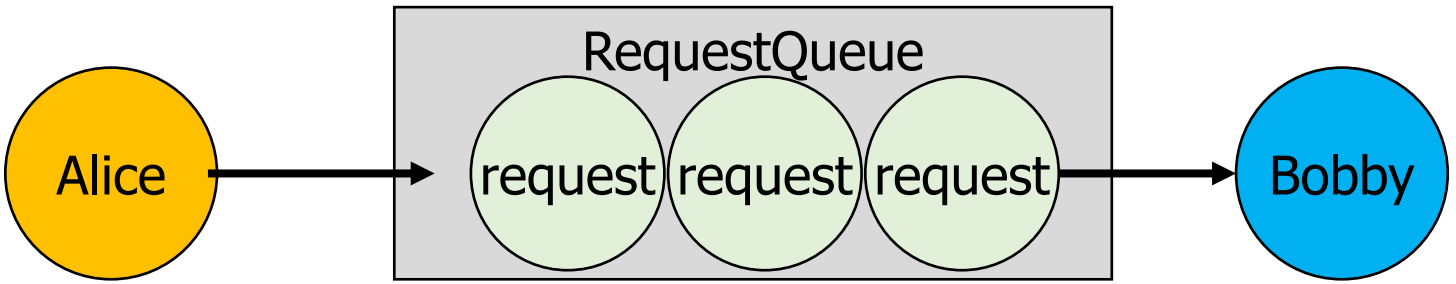
- Java-Thread/Request/
  - Main.java -- スレッドのインスタンスを作るプログラム
    - リクエストを出すスレッドのインスタンス： Alice
    - リクエストを処理するスレッドのインスタンス： Bobby
  - Request.java -- 「リクエスト」クラス
  - RequestQueue.java -- 「リクエストキュー」クラス
  - ClientThread.java -- リクエストを出すスレッド
  - ServerThread.java -- リクエストを処理するスレッド



サンプルプログラム（Request）

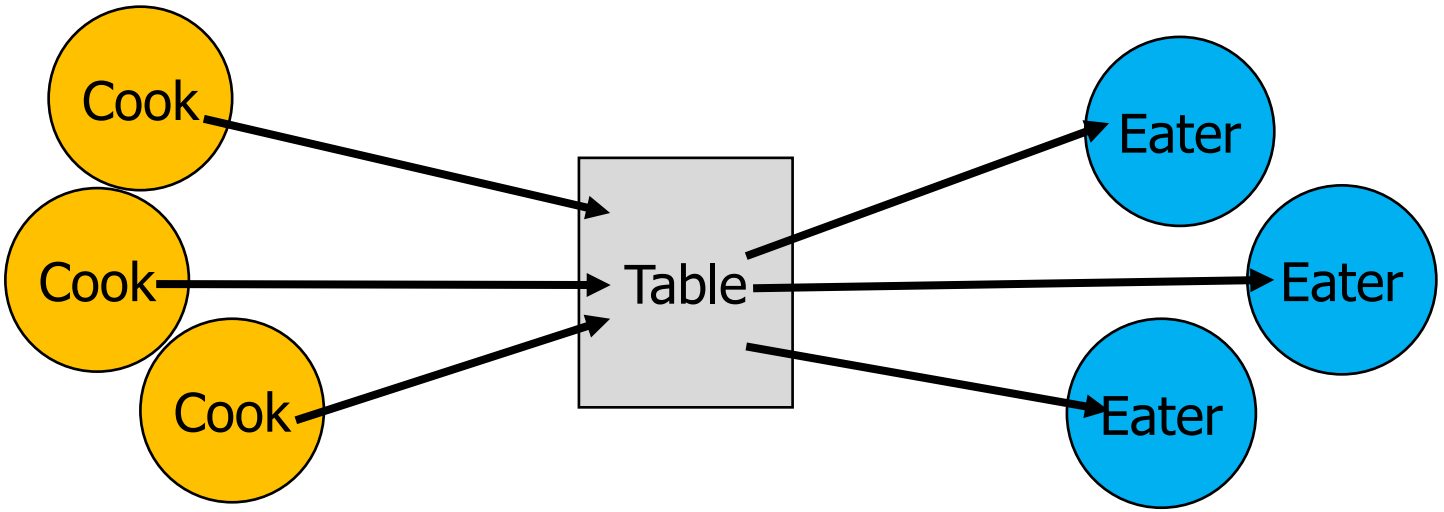
- コンパイルと実行

- ClientThread, ServerThreadのsleepの時間を変更して実行せよ。
  - サーバの処理時間に対して、クライアントからのリクエスト到着間隔が長い場合
  - クライアントからのリクエストの到着間隔にくらべて、サーバの処理時間が長い場合



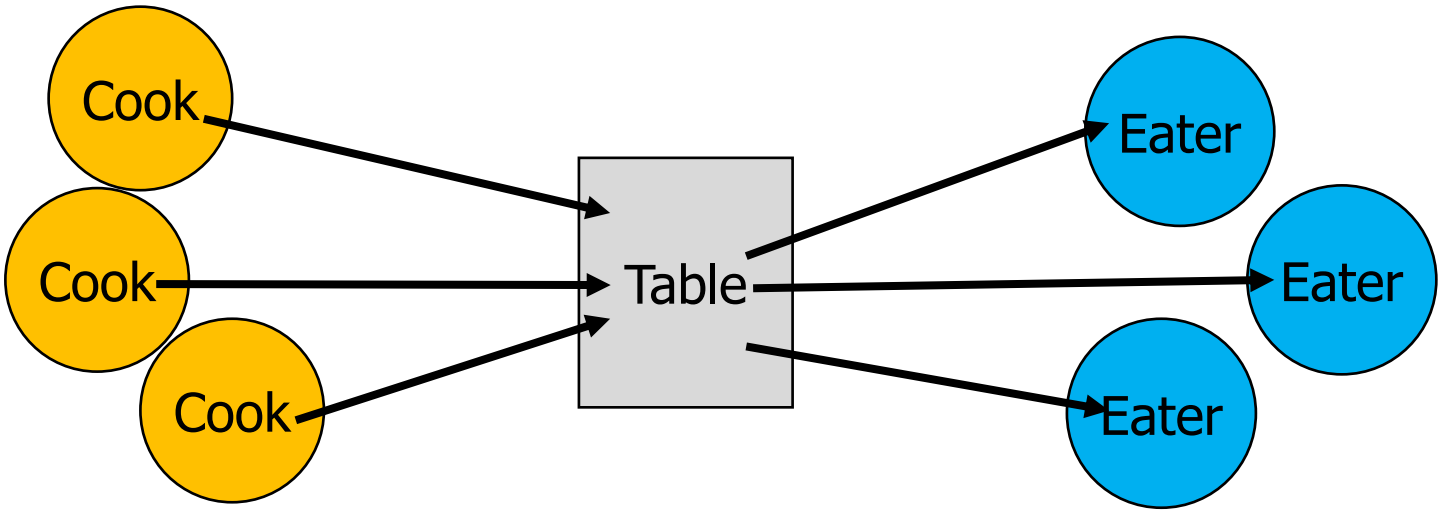
サンプルプログラム（Consumer）

- Java-Thread/Consumer/
- 生産者／消費者問題のプログラム
  - コック3名がケーキを作ってテーブルに置く。
  - 客3名がテーブルからケーキを取る。



サンプルプログラム（Consumer）

- ファイル
  - Main.java -- コック3名と客3名のスレッドを起動する。
  - MakerThread.java -- コック（ケーキをテーブルに置く）
  - EaterThread.java -- 客（テーブルのケーキを食べる）
  - Table.java -- ケーキを管理するテーブル



サンプルプログラム（Consumer）

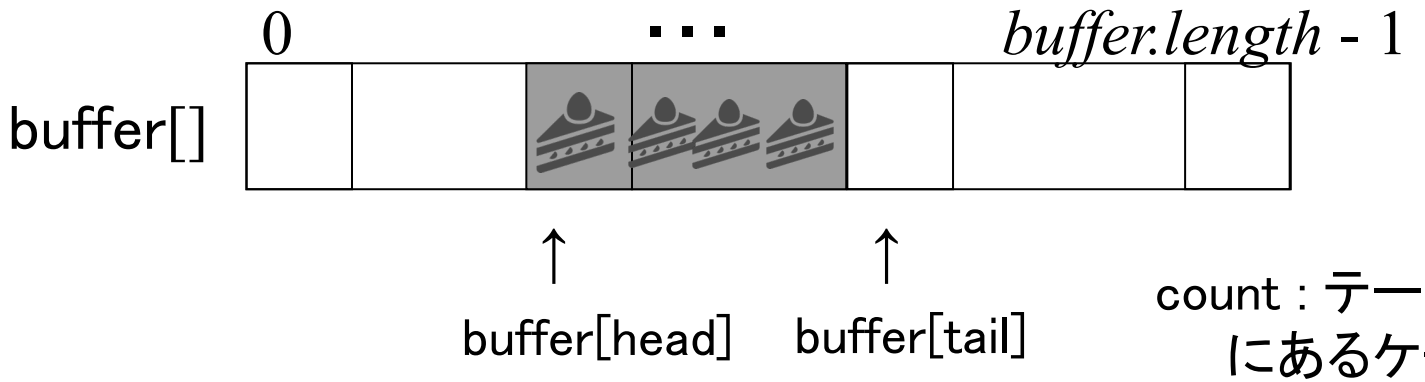
■テーブルの上のケーキの管理

コックがケーキを置く

```
while (count >= buffer.length) {
    wait();
}
buffer[tail] = cake;
tail = (tail + 1) % buffer.length;
count++;
notifyAll();
```

客がケーキを取る

```
while (count <= 0) {
    wait();
}
String cake = buffer[head];
head = (head + 1) % buffer.length;
count--;
notifyAll();
```



count：テーブル（バッファ）にあるケーキの個数

サンプルプログラム（Consumer）

- sleepなどを調整して以下の状況を作り出してみよう。
  - コックがケーキを作るスピードが早く、
    テーブルがケーキで埋め尽くされる状況
  - 客がケーキを食べる速度に、極端に差がある状況

今日のまとめ

- Java言語でのスレッド
  - Javaではクラスがスレッドの実行単位
  - スレッドを生成する方法で主なものは2つ
    - Threadクラスを継承したクラスを定義し、そのインスタンスを生成して実行。
    - Runnableインタフェースをもつクラスを定義し、そのインスタンスを、Threadクラスの実行時に引数として渡して実行する。
  - 排他制御は syncronizedメソッドを使用する。
    - 複雑な条件は wait(), notify(), notifyAllなどで同期を取る。