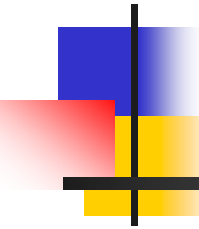


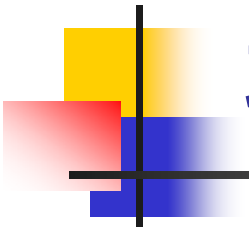
独習Java
第9章　9.3~9.5

9.3 同期

9.4 デッドロック

9.5 スレッドの通信

07/06/08　名雪　勲

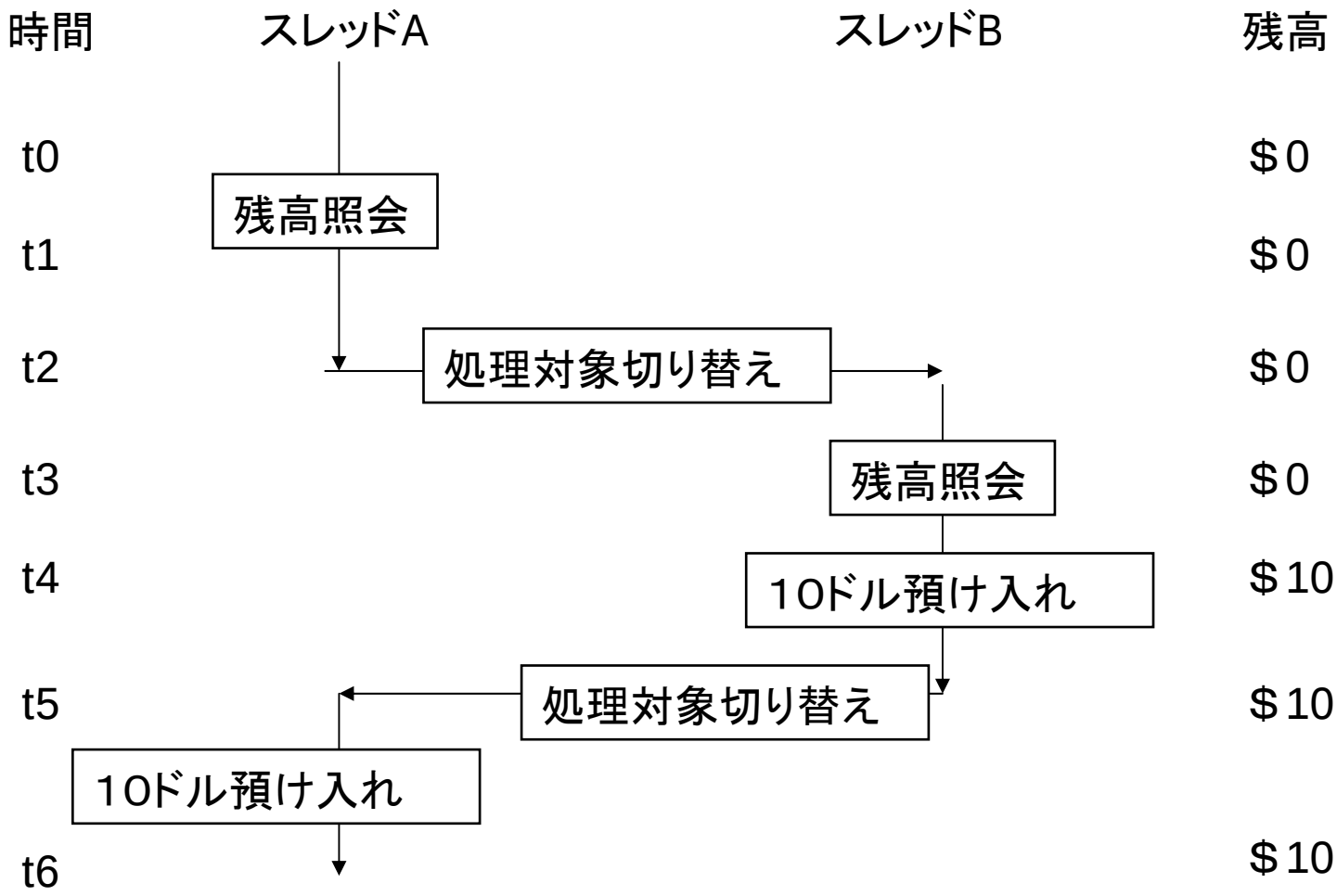


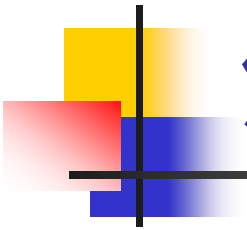
非同期の問題点

- 複数のスレッドから共有データを読み書きする場合は慎重に読み書きの順番を調節しないとデータの破損やシステムダウンの誘因となる。

- 次に共通の銀行口座への現金の振込みを例にとって説明する。

例）銀行口座


時間
スレッドA
スレッドB
残高
t0
$0
残高照会
t1
$0
t2
$0
処理対象切り替え
t3
$0
残高照会
t4
$10
10ドル預け入れ
t5
$10
処理対象切り替え
10ドル預け入れ
t6
$10



解決法

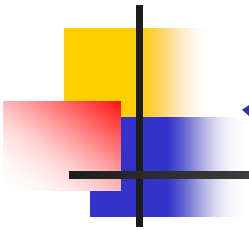
- この問題を解決するために共有データへの読み書きを同期させる必要がある。
- メソッド宣言にsynchronized修飾子を指定してメソッドを同期させる方法がある。
- synchronizedインスタンスメソッドではそのオブジェクトのロックが自動的に取得される。このロックはメソッドの処理が完了すると自動的に開放される。それまで2番目のスレッドを待たせておく。

デッドロックとは？

- マルチスレッドプログラムで発生するエラーの一種で複数のスレッドが互いにロックの解放を永久に待ち続けている状態のことです。

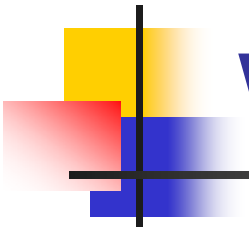
- 例）

スレッド1	スレッド2
オブジェクト1	オブジェクト2
オブジェクト2のロック解放を待機	オブジェクト1のロック解放を待機



スレッドの協調

- デッドロックを回避するためにはスレッドを協調させる必要があります。スレッドから一時ロックを解放し、他のスレッドにsynchronizedメソッドまたはsynchronizedブロックを実行させるチャンスを与えます。
- wait()メソッドはこれを可能とするメソッドのひとつで3タイプのwait()メソッドが存在します。



wait()メソッド

①void wait() throws InterruptedException

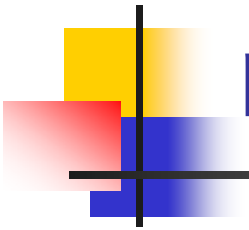
現在のスレッドは無期限に待機

②void wait(long msec) throws IngerruptedException

msecミリ秒間待機

③void wait(long msec, int nsec) throws InterruptedException

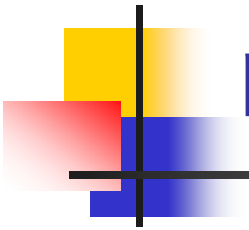
msecミリ秒とnsecナノ秒を足した時間の間待機



notify()メソッド

- notify()メソッドはsynchronizedメソッドまたはsynchronizedブロックを実行しているスレッドから、そのオブジェクトのロック解放を待機しているスレッドに対して通知を送る。
- 複数のスレッドが待機の場合は一つだけに通知する。

void notify()



notifyAll()メソッド

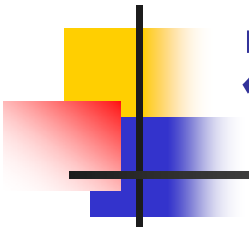
- notifyAll()メソッドはsynchronizedメソッドまたはsynchronizedブロックを実行しているスレッドからそのオブジェクトのロック解放を待機している全てのスレッドに対して通知を送る。

void notiryAll()

宿題(1/3)

- 次のプログラムでデッドロックが発生するか調べて説明しなさい。

```
class X {
synchronized void x1() {
   x2();
 }
synchronized void x2() {
  x1();
  }
 }
class ThreadX extends Thread {
 X x;
ThreadX(X x){
this.x = x;
}
public void run() {
for (int i = 0; i < 100000; i++)
```



宿題(2/3)

```
x.x1();
  }
}
class DeadlockQuestion {
  private final static int NUMTHREADS = 10;
  public static void main(String args[]){
//オブジェクトの作成
X x = new X();
//スレッドの作成
ThreadX threads[] = new ThreadX[NUMTHREADS];
for (int i = 0; i < NUMTHREADS; i++) {
threads[i] = new ThreadX(x);
threads[i].start();
}
```

宿題(3/3)

```
//スレッドの完了を待つ
for (int i = 0; i < NUMTHREADS; i++)
try {
        threads[i].join();
  }
  catch(Exception e) {
    e.printStackTrace();
}
//メッセージの表示
   System.out.println(“Done!”);
 }
}
```