# 独習Javaゼミ

９．１　スレッドの概要

９．２　スレッドの作成

07/06/08

鈴木 慧

# ９．１　スレッドの概要

- スレッドとは

プロセスの実行の流れ
スレッドの状態には、新規作成、使用可能、
実行中、待機中、破棄の5種類がある

スレッドには優先順位があり、優先順位の高いスレッド
がある限り、低いスレッドは実行されない
優先順位が同じ場合、複数のスレッドを並行して実行する

# スレッドの状態1

- 新規作成

スレッド作成直後の状態

- 使用可能

新規作成されたスレッドの起動後の状態

- 実行中

使用可能なスレッドの実行された状態

- 破棄

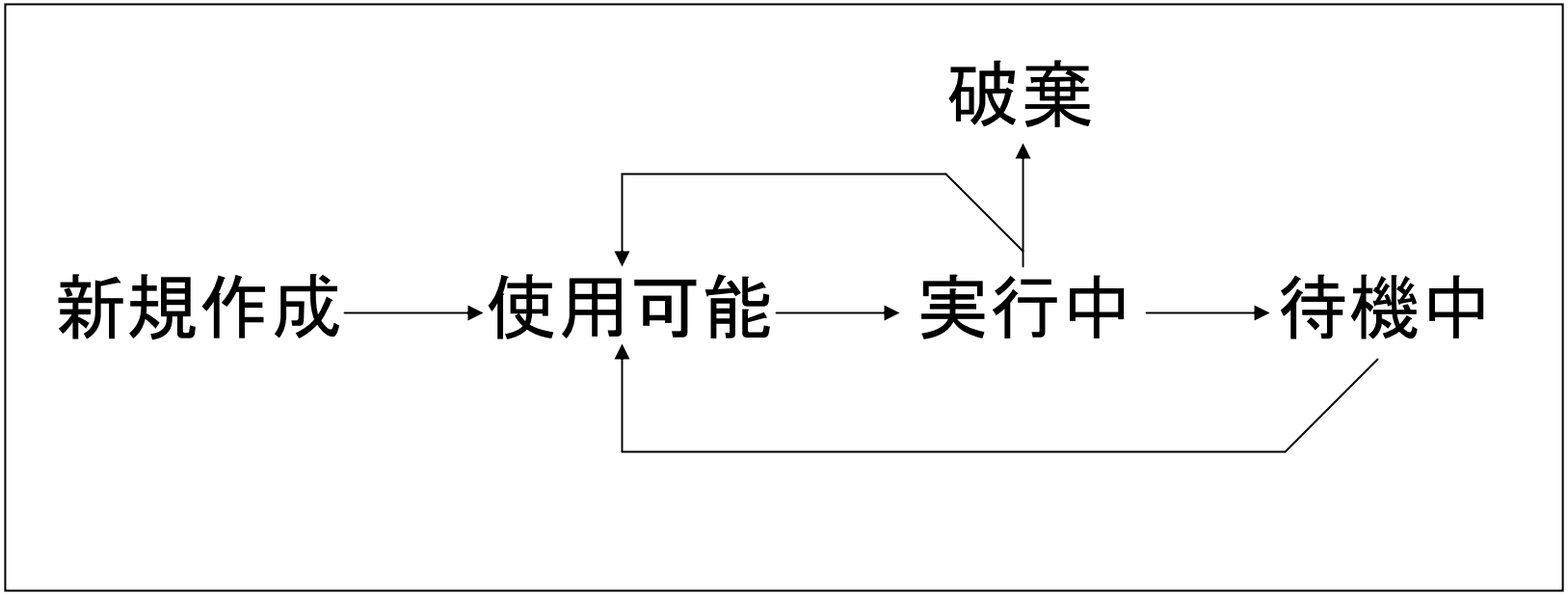
実行されたスレッドの処理が完了した状態

# スレッドの状態2

- 待機中

実行中のスレッドで休眠、待機、入出力操作のいずれかの実行を選択すると、この状態になる
休眠、待機、入出力操作の実行が完了すると使用可能状態に戻る

# スレッドのライフサイクル




スレッドのライフサイクル

# ９．２　スレッドの作成

スレッドの作成には Thread クラスを使用

```
class クラス名 extends Thread {
    public void run() {
        // スレッドの処理
    }
}
```

Thread クラスの拡張

```
クラス名 変数名 = new クラス名();
変数名.start();
```

スレッドの起動

# Thread クラスの静的メソッド

static Thread currentThread()
現在のスレッドへの参照を返す

static void sleep(long 変数) throws InterruptedException
現在のスレッドを変数ミリ秒間、待機させる

static void sleep(long 変数A, int 変数B) throws Interrupted Exception
現在のスレッドを変数Aミリ秒間＋変数Bナノ秒間、待機させる

static void yield()
現在のスレッドからほかのスレッドにCPU処理の制御を明け渡す

# Thread クラスのインスタンスメソッド

| | |
|---|---|
| String getName() | スレッドの名前を返す |
| int getPriority() | スレッドの優先順位を返す |
| void interrupt() | スレッドを中断する |
| void run | スレッドの処理を構成する |
| void setPriority(int **変数**) | スレッドの優先順位を**変数**にする |
| void start() | スレッドを開始する |
| void join() throws InterruptionException | スレッドが破棄されるまで、呼び出し元を待機させる |

Threadクラスに定義されている主なインスタンスメソッド
(p273 表9-2 参照)

# スレッドの作成の例

```
class A extends Thread {
    public void run() {
      try {
         while (true) {
            Thread.sleep(2000);
            System.out.println("Hello");
         }
      }
      catch (InterruptedException ex){
         ex.printStackTrace();
      }
    }
}
```

```
class ThreadDemo1 {
    public static void
     main(String args[]) {
        A tx = new A();
        tx.start();
    }
}
```

sleep() メソッドでは InterruptedException 例外を捕獲する必要がある

実行結果

Hello

Hello

Hello

…

# Runnable インターフェイス1

Runnable インターフェイスを実装したクラスを宣言
する事でもスレッドを作成する事が可能
このインターフェイスでは、メソッドは次のように一つだけ宣言する

```
public void run();
```

run() メソッド

```
class クラス名 implements Runnable {
    public void run() {
        // スレッドの処理
    }
}
```

Runnable インターフェイスクラスの宣言

# Runnable インターフェイス2

```
クラス名 変数名A = new クラス名();
Thread 変数名B = new Thread (変数名A)
変数名B.start();
```

スレッドの起動

- Thread()
- Thread(Runnable *r*)
- Thread(Runnable *r*, String *s*)
- Thread(String *s*)

Thread クラスの主なコンストラクタ

*r* はRunnableインターフェイスを実装したオブジェクトへの参照、
*s* はスレッドを表す文字列を意味する

# Runnable インターフェイスを実装してスレッドを作成する例(1/3)

```
class CountA implements Runnable {
  public void run() {
    try {
      for (int i=0; i <= 3; i++) {
        Thread.sleep(500);
        System.out.println("A: " + i);
      }
    }
    catch (InterruptedException ex){
      ex.printStackTrace();
    }
  }
}
```

```
class CountB implements Runnable
    {
  public void run() {
    try {
      for (int i=3; i >= 0; i--) {
        Thread.sleep(500);
        System.out.println("¥tB: " + i);
      }
    }
    catch (InterruptedException ex)
     {
      ex.printStackTrace();
    }
  }
}
```

# Runnable インターフェイスを実装してスレッドを作成する例(2/3)

```
class CountsTest {
    public static void main(String[] args) {
        CountA runA = new CountA(); //Runnableインスタンス化
        CountB runB = new CountB();
        Thread threadA = new Thread(runA);//スレッドインスタンス化
        Thread threadB = new Thread(runB);
        System.out.println("Thread 開始");
        threadA.start();  // スレッドの開始
        threadB.start();
        System.out.println("Thread の start() 終了");
```

# Runnable インターフェイスを実装してスレッドを作成する例(3/3)

```
        try {
            System.out.println("スレッドA、Bが終わるまで待機");
            threadA.join(); //threadAが終わるまで待機
            threadB.join(); //threadBが終わるまで待機
            System.out.println("両方のスレッドが終了しました");
        }
        catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

実行結果

```
Thread 開始
Thread の start() 終了
スレッドA、Bが終わるまで待機
A: 0
                B: 3
A: 1
                B: 2
A: 2
                B: 1
A: 3
                B: 0
両方のスレッドが終了しました
```

# 例の説明

実行結果が

A: 0
A: 1
A: 2
A: 3
　　B: 3
　　B: 2
　　B: 1
　　B: 0

とならない事から、threadAとthreadBは
並行に処理されている事がわかる

# 宿題

0から99までの整数を0.1秒間隔で表示するスレッドと、
1秒間隔で改行するスレッドを作成し並行に実行させよ。
スレッドは、Threadクラスのサブクラスを使っても、
Runnableインターフェイスを実装して作成してもよい。

実行結果の例

```
 0  1  2  3  4  5  6  7  8  9
10 11 12 13 14 15 16 17 18 19
20 21 22 23 24 25 26 27 28 29
30 31 32 33 34 35 …
```