

● J2Kad16D「スレッド①（Thread クラス）」

リスト 1 を参考に MyThread クラスと J2Kad16D クラスを作成し動作確認せよ。なお、MyThread クラスは Thread クラスを継承し、ファイル「J2Kad16D.java」に作成すること。

MyThread クラスの仕様（Thread クラスを継承、ファイル「J2Kad16D.java に作成」）

メンバ	仕様
public void run()	① 0 から 99 まで「run : 0」「run : 1」…「run : 99」と順次表示する。 ② 表示が終了したら「run : 終了しました!」と表示する。

main メソッドの仕様

- ① MyThread クラスをスレッドとして開始する。
- ② 0 から 99 まで「main : 0」「main : 1」…「main : 99」と順次表示する。
- ③ 表示が終了したら「main : 終了しました!」と表示する。

課題完成時の画面①

```
run : 0
run : 1
run : 2
run : 3
⋮
run : 25
main : 0
run : 26
main : 1
run : 27
main : 2
run : 28
run : 29
main : 3
run : 30
main : 4
run : 31
main : 5
⋮
main : 59
run : 98
run : 99
main : 60
run : 終了しました!
main : 61
⋮
main : 98
main : 99
main : 終了しました!
```

課題完成時の画面②

```
run : 0
main : 0
run : 1
main : 1
run : 2
main : 2
⋮
main : 67
run : 70
main : 68
run : 71
run : 72
run : 73
run : 74
run : 75
run : 76
main : 69
main : 70
⋮
main : 98
run : 94
main : 99
run : 95
main : 終了しました!
run : 96
run : 97
run : 98
run : 99
run : 終了しました!
```

run と main の実行タイミングは  
実行するごとに異なる

● J2Kad16C「スレッド② (Runnable インターフェイス)」

Sheep クラスが準備されている。Sheep クラスを継承して SheepRunner クラスを作成しスレッドとして実行せよ。また main メソッドにはスレッドの終了を確認する処理を作成せよ。

SheepRunner クラスの仕様 (Sheep クラスを継承、ファイル「J2Kad16C.java に作成」)

メソッド	仕様
public void run()	① 「xx が走ります！」と表示する (xx は名前)。 ② 「xx：残り 100 メートル」「xx：残り 99 メートル」…「xx：残り 1 メートル」と順次表示する。 ③ 「xx：ゴールしました！」と表示する。

main メソッドの仕様

- ① SheepRunner クラスをスレッドとして開始する。
- ② スレッドが終了するのを待つ。
- ③ 「おつかれさまでした！」と表示する。

課題完成時の画面

```
レントゲンが走ります！
レントゲン：残り 100 メートル！
レントゲン：残り 99 メートル！
レントゲン：残り 98 メートル！
レントゲン：残り 97 メートル！
レントゲン：残り 96 メートル！
レントゲン：残り 95 メートル！
レントゲン：残り 94 メートル！
レントゲン：残り 93 メートル！
レントゲン：残り 92 メートル！
レントゲン：残り 91 メートル！
レントゲン：残り 90 メートル！
レントゲン：残り 89 メートル！
レントゲン：残り 88 メートル！
レントゲン：残り 87 メートル！
レントゲン：残り 86 メートル！
レントゲン：残り 85 メートル！
レントゲン：残り 84 メートル！
レントゲン：残り 83 メートル！
レントゲン：残り 82 メートル！
    ⋮
レントゲン：残り 5 メートル！
レントゲン：残り 4 メートル！
レントゲン：残り 3 メートル！
レントゲン：残り 2 メートル！
レントゲン：残り 1 メートル！
レントゲン：ゴールしました！
おつかれさまでした！
```

「おつかれさまでした！」は  
main メソッドが表示する。

## ● J2Kad16B 「そうだ！銀行へ行こう！！（スレッド版）」

ECC 銀行が預金者を募集した！「あの ECC なら絶対大丈夫！」ということで定員いっぱいの 100 人が集まった。それぞれ 1 回あたり 1 円の預金を 1 万回繰り返し、計 1 万円預けた。ところが本来 100 万円 (1 万円×100 人) 集まっているはずの預金が少し足りない！このままでは ECC の信用にキズがつく。原因を究明し再発防止に努めよ！

- ① Bank クラス（銀行）、Customer クラス（預金者）を作成し、実際に預金額がおかしくなるのを確認せよ。
- ② 金額がおかしくならないようにプログラムを修正せよ。

## Bank クラスの仕様（ファイル「J2Kad16B.java に作成」）

メンバ	仕様
private static int money	預金額、初期値は 0。
public static int getMoney()	money の値を返す。
public static void addOneYen()	1 円預金する (money の値を 1 増やす)。

## Customer クラスの仕様（ファイル「J2Kad16B.java に作成」）

メンバ	仕様
コンストラクタ	「預金者がやってきた！「がんばってお金を預けるぞ！」と表示する。
public void run()	1 円預金を 1 万回繰り返す。

## main メソッドの仕様

- ① Customer クラスのインスタンスを 100 個生成する。
- ② 生成したインスタンスをスレッドとして開始する。
- ③ すべてのスレッドの処理が終了するのを待つ。
- ④ 預金額を表示する。

## 課題完成時の画面（仕様①まで）

お金を預けるのなら信用と信頼の ECC 銀行へ！  
預金者がやってきた！「がんばってお金を預けるぞ！」  
預金者がやってきた！「がんばってお金を預けるぞ！」  
⋮  
預金者がやってきた！「がんばってお金を預けるぞ！」  
預金額は 860902 円です！

仕様通りに作ったとき。預金額が足りない！

## 課題完成時の画面（仕様②まで）

お金を預けるのなら信用と信頼の ECC 銀行へ！  
預金者がやってきた！「がんばってお金を預けるぞ！」  
預金者がやってきた！「がんばってお金を預けるぞ！」  
⋮  
預金者がやってきた！「がんばってお金を預けるぞ！」  
預金額は 1000000 円です！

原因究明して対処すると正しい預金額になる。

● J2Kad16A「ウサギ vs カメ」

ウサギとカメが競争する処理を作成せよ。距離は 1000 メートル、ウサギはカメがスタートしてから 5 秒後にスタートするものとする。

Rabbit1 クラスの仕様（ファイル「J2Kad16A.java に作成」）

メンバ	仕様
コンストラクタ	「ウサギがやってきた！」と表示する。
public void run()	① 「ウサギが走ります！」と表示する。 ② 距離を 1000 から 0 までカウントダウンする。 このとき 10 メートルごとに「ウサギ：残り xx メートル」（xx は残りの距離）と表示する。 ③ 距離が 0 になったら「ウサギ：ゴールしました！」と表示する。

Turtle1 クラスの仕様（ファイル「J2Kad16A.java に作成」）

メンバ	仕様
コンストラクタ	「カメがやってきた！」と表示する。
public void run()	① 「カメが走ります！」と表示する。 ② 距離を 1000 から 0 までカウントダウンする。 <b>ただしカウントダウンする前に 10 ミリ秒のウェイトを入れる。</b> このとき 10 メートルごとに「カメ：残り xx メートル」（xx は残りの距離）と表示する。 ③ 距離が 0 になったら「カメ：ゴールしました！」と表示する。

main メソッドの仕様

- ① 「ウサギとカメが競争します！」と表示し、Rabbit1 と Turtle1 のインスタンスを生成する。
- ② カメのスレッドを開始する。
- ③ 5 秒（5000 ミリ秒）ウェイトしてからウサギのスレッドを開始する。

課題完成時の画面

ウサギとカメが競争します！  
ウサギがやってきた！  
カメがやってきた！  
カメが走ります！  
カメ：残り 1000 メートル！  
カメ：残り 990 メートル！  
⋮  
カメ：残り 540 メートル！  
ウサギが走ります！  
ウサギ：残り 1000 メートル！  
ウサギ：残り 990 メートル！  
⋮  
ウサギ：残り 10 メートル！  
ウサギ：ゴールしました！  
カメ：残り 530 メートル！  
⋮  
カメ：残り 10 メートル！  
カメ：ゴールしました！

← 5 秒後にウサギがスタート

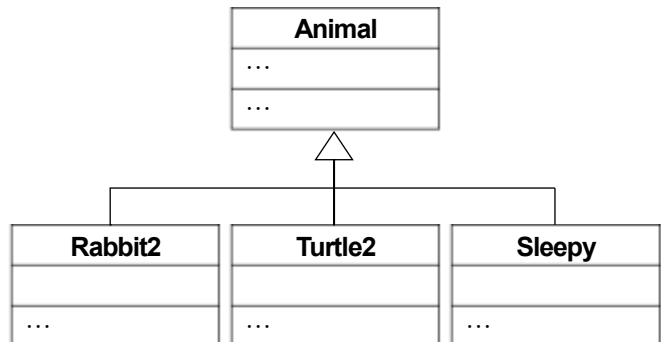
## ● J2Kad16S 「眠りウサギ参戦！」

J2Kad16A の Rabbit1 クラスと Turtle1 クラスは動物の名前（ウサギ、カメ）と距離をカウントダウンするときのウェイト（ウサギ：ウェイトなし、カメ：10 ミリ秒のウェイト）が異なるだけで、あとの処理は同じである。

- ① 共通部分を Animal クラスとして定義し、Animal クラスを継承してあらためて Rabbit2 クラス（Rabbit1 と同等の仕様）と Turtle2 クラス（Turtle1 クラスと同等の仕様）を作成せよ。
- ② さらに Sleepy クラス（眠りウサギ、ときどき昼寝する）を追加し、ウサギと同じく 5 秒後にスタートするようにせよ。なお、Sleepy の仕様は以下の通り。

## Sleepy クラスの仕様

- ・名前：「眠りウサギ」
- ・カウントダウンのとき、100 分の 1 の確率で昼寝する。  
「眠りウサギは昼寝した！！」と表示  
→500 ミリ秒ウェイト  
→「眠りウサギは目を覚ました！！」と表示



## 課題完成時の画面

ウサギとカメと眠りウサギが競争します！  
 ウサギがやってきた！  
 カメがやってきた！  
 眠りウサギがやってきた！  
 カメが走ります！  
 カメ：残り 1000 メートル！  
 カメ：残り 990 メートル！  
 ……  
 カメ：残り 540 メートル！  
 ウサギが走ります！  
 ウサギ：残り 1000 メートル！  
 ウサギ：残り 990 メートル！  
 眠りウサギが走ります！  
 ウサギ：残り 980 メートル！  
 ウサギ：残り 970 メートル！  
 ……  
 ウサギ：残り 820 メートル！  
 眠りウサギ：残り 1000 メートル！  
 ウサギ：残り 810 メートル！  
 眠りウサギ：残り 990 メートル！  
 ……  
 ウサギ：残り 720 メートル！  
 眠りウサギは昼寝した！！  
 ウサギ：残り 710 メートル！  
 ウサギ：残り 700 メートル！  
 ……

## (続き)

ウサギ：残り 20 メートル！  
 ウサギ：残り 10 メートル！  
 ウサギ：ゴールしました！  
 カメ：残り 530 メートル！  
 カメ：残り 520 メートル！  
 カメ：残り 510 メートル！  
 カメ：残り 500 メートル！  
 カメ：残り 490 メートル！  
 眠りウサギは目を覚ました！！  
 眠りウサギ：残り 910 メートル！  
 眠りウサギは昼寝した！！  
 カメ：残り 480 メートル！  
 ……  
 カメ：残り 20 メートル！  
 カメ：残り 10 メートル！  
 カメ：ゴールしました！  
 眠りウサギは目を覚ました！！  
 眠りウサギ：残り 30 メートル！  
 眠りウサギ：残り 20 メートル！  
 眠りウサギ：残り 10 メートル！  
 眠りウサギ：ゴールしました！

運が良ければカメが眠りウサギに勝つこともある。

## ● J2Kad16X 「右折できません！（デッドロック）」※実践編 P.69、P.113

車を表す Car クラスが準備されている。課題完成時の画面を参考に以下の仕様で交通渋滞のシミュレーションを行え。道路は片側 1 車線（対向車線と合わせて 2 車線）、それぞれ「レーン 1」「レーン 2」とする。

## レーン (Lane クラス) の仕様（スレッドとして並行処理する、クラス定義は各自で考えること）

- ・ 10 分の 1 の確率で車 (Car クラス) が並んでいく。
- ・ 5 分の 1 の確率で先頭の車が右折する。ただし対向車線に車が 5 台以上並んでいるときは右折できない。
- ・ 並んでいる車は Queue インターフェイスを介して LinkedList に格納する（使い方は実践編 P.113 「LinkedList クラスによるキュー」参照）。
- ・ スレッドの停止は running フラグを作って行う（実践編 P.69 「スレッドを止める」参照）。

## main メソッドの仕様

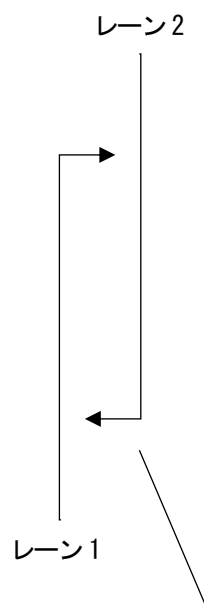
- ① Lane クラスのインスタンスを 2 つ生成する（レーン 1、レーン 2 とする）。
- ② それぞれもう片方を対向車線として設定する（レーン 1 にはレーン 2 を設定、レーン 2 にはレーン 1 を設定する）。
- ③ レーン 1、レーン 2 ともにスレッドを開始する。
- ④ 50 ミリ秒ウェイトする。
- ⑤ レーン 1、レーン 2 ともに停止させる。

## 課題完成時の画面（デッドロックが発生したパターン）

```

レーン 2：シエンタがやってきた！現在の行列：○
レーン 1：エクリプスがやってきた！現在の行列：○
レーン 2：シエンタは右折した！現在の行列：なし
レーン 1：エクリプスは右折した！現在の行列：なし
レーン 2：アルトがやってきた！現在の行列：○
レーン 1：クラウンがやってきた！現在の行列：○
レーン 2：アルトは右折した！現在の行列：なし
レーン 1：プリウスがやってきた！現在の行列：○○
    ⋮
レーン 1：フィットは右折した！現在の行列：○○○○○○○○○○
レーン 1：スイフトがやってきた！現在の行列：○○○○○○○○○○
レーン 2：アルトは右折できない！現在の行列：○○○
レーン 1：フェアレディ Z は右折した！現在の行列：○○○○○○○○○○
レーン 2：デリカがやってきた！現在の行列：○○○○
レーン 1：セレナは右折した！現在の行列：○○○○○○○○○○
レーン 2：カムリがやってきた！現在の行列：○○○○○
レーン 1：スカイラインは右折できない！現在の行列：○○○○○○○○○○
レーン 2：アルトは右折できない！現在の行列：○○○○○
レーン 1：ハリヤーがやってきた！現在の行列：○○○○○○○○○○
レーン 2：アコードがやってきた！現在の行列：○○○○○○○
レーン 1：スカイラインは右折できない！現在の行列：○○○○○○○○○○
レーン 2：インサイトがやってきた！現在の行列：○○○○○○○○○
レーン 1：ミラーージュがやってきた！現在の行列：○○○○○○○○○○○○
レーン 2：アルトは右折できない！現在の行列：○○○○○○○○
レーン 1：スカイラインは右折できない！現在の行列：○○○○○○○○○○○○
    ⋮
  
```

## デッドロックのイメージ



対向車線に 5 台以上並んでいると右折できない。

↓  
レーン 1・2 ともに右折できないとキューから車を取り出す処理ができなくなり、行列がどんどん長くなっていく（デッドロック）。