# **J2Kad15D「スレッド①（Threadクラス）」 （実践編P.31「複数のクラス宣言を持つプログラムコード」、P.61「方法1：Threadクラスを拡張する」）**

MyThreadクラスとJ2Kad15Dクラス（**リスト1**）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「J2Kad15D.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「J2Kad15D.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。 |

**リスト1：「スレッド①（Threadクラス）」（ファイル「J2Kad15D.java」）**

**MyThreadクラスを作成**

public class J2Kad15D {

public static void main(String[] args) {

**MyThreadクラスを生成しスレッド開始**

for (int i = 0; i < 100; i++) {

System.out.println("main：" + i);

}

System.out.println("main：終了しました！");

}

}

**課題完成時の画面① 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：95

main：99

run：96

main：終了しました！

run：97

run：98

run：99

run：終了しました！

run：0

run：1

run：2

…

run：25

main：0

run：26

main：1

run：27

main：2

run：28

…

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！

runとmainの実行タイミングは実行するごとに異なる

# **J2Kad15C「スレッド②（Runnableインターフェイス）」 （実践編P.63「方法2：Runnableインターフェイスを実装する」、P.67「スレッドの処理が終わるのを待つ」）**

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。

|  |
| --- |
| **Sheep** |
| # name : String |
| … |

**SheepRunnerクラスの仕様（Sheepクラスを継承、ファイル「J2Kad15C.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。 |

**リスト1：「スレッド②（Runnableインターフェイス）」（ファイル「J2Kad15C.java」）**

**SheepRunnerクラスを作成**

public class J2Kad15C {

public static void main(String[] args) {

**SheepRunnerクラスをスレッドとして開始**

**スレッドが終了するのを待つ**

System.out.println("おつかれさまでした！");

}

}

**課題完成時の画面**

ガリレオが走ります！

ガリレオ：残り100メートル！

ガリレオ：残り99メートル！

ガリレオ：残り98メートル！

ガリレオ：残り97メートル！

ガリレオ：残り96メートル！

ガリレオ：残り95メートル！

ガリレオ：残り94メートル！

…

ガリレオ：残り10メートル！

ガリレオ：残り9メートル！

ガリレオ：残り8メートル！

ガリレオ：残り7メートル！

ガリレオ：残り6メートル！

ガリレオ：残り5メートル！

ガリレオ：残り4メートル！

ガリレオ：残り3メートル！

ガリレオ：残り2メートル！

ガリレオ：残り1メートル！

ガリレオ：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。

# **J2Kad15B「そうだ！銀行へ行こう！！（スレッド版）」 （実践編P.71「マルチスレッドで問題が生じるケース」、P.74「スレッドの同期」）**

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「J2Kad15B.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（Threadクラスを継承、ファイル「J2Kad15B.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。

# **J2Kad15A「ウサギvsカメ」（実践編P.66「スレッドの処理を一定時間停止させる」）**

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「J2Kad15A.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「J2Kad15A.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。 **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。** 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

# **J2Kad15S「眠りウサギ参戦！」※Runnableインターフェイスを使うこと**

J2Kad15A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と同等の仕様）を作成せよ。なお、今回はThreadクラスではなく（Runnableインターフェイスの練習も兼ねて）Runnableインターフェイスを使うこと。
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。

|  |
| --- |
| **Animal** |
| … |
| … |

|  |
| --- |
| **Rabbit2** |
|  |
| … |

|  |
| --- |
| **Turtle2** |
|  |
| … |

|  |
| --- |
| **Sleepy** |
|  |
| … |

**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 （続き）**

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！

…

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。

# **J2Kad15X「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） デッドロックのイメージ**

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：ハリアーがやってきた！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：○○○○○○○○○○**

…

レーン2

レーン1

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）。