

第七回 演習問題（スレッド，例外処理）

諸注意

- 「Kadai.java」を Web から提出する。
- 課題 2-2 以降は，できた人から教員もしくは TA を呼び，その場でチェックを受ける。
- コピペ発覚時は見せた側も見せてもらった側も両方 0 点とする。
- 必ず**コンパイルエラーのない状態で提出すること**（自動採点したいのでコンパイルエラーがあると，全て 0 点になってしまう）。
- 課題の途中で提出することになった場合，**コンパイルエラーさえ出なければ，課題の途中の状態で提出してくれて構わない**。一部のメソッドだけが実現できていない場合，コンパイルエラー出ないならばそのままの状態で提出してくれてよい。
- 主にコンソール出力で評価しているため，デバッグに用いたようなコンソール出力が残っていないように気をつけること。**基本的にコンソール出力を指定しない限りは，課題内でコンソール出力はないものとする**。
- **Package は使わないこと**（デフォルトパッケージで実装する）。Package で実装すると，自動採点がうまくいきません。

課題 1

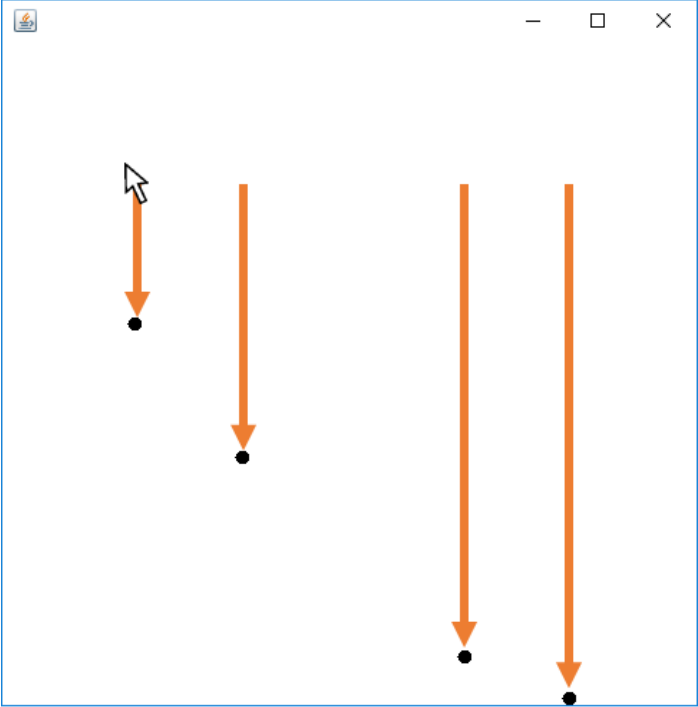
1	問題設定	<p>誰かが開発したメソッドを使って開発を行うこととなった。このメソッドは非常に便利なのだが、4 種類の例外が発生しうるチェック例外のメソッドである。次に示すメソッド <code>randomException()</code> がそのメソッドである。これを自分のファイルにコピペして使用し、例外対応を行ったメソッド <code>public void safe()</code> を <code>Kadai.java</code> に記述しこれを提出せよ。</p> <p><code>safe()</code> では <code>randomException()</code> を呼び出し、発生した例外の種類に応じて以下のコンソール出力を <code>println()</code> で行う。なお、<u>括弧も句読点も表示しないこと</u>。</p> <p>IOException : 「ファイルがないよ」 ClassCastException : 「キャスト間違えてるよ」 ArithmeticException : 「ゼロ除算でもしたのかな」 NullPointerException : 「ぬるぽ」</p> <p>また、IOException と ArithmeticException 発生時にはメソッドを再度呼び出しすることとする。（恐らく再度ファイルを選択したり、計算を変更したりさせることで例外を回避できるだろう、という仮定）</p>
	メソッド そのまま コピペせよ	<pre>public void randomException() throws IOException, ClassCastException, ArithmeticException, NullPointerException{ int rand = (int)(Math.random()*4); switch(rand){ case 0: throw new IOException(); case 1: throw new ClassCastException(); case 2: throw new ArithmeticException(); default: throw new NullPointerException(); } }</pre>
	テスト例	<p>(Main.javaのmain()メソッドにて)</p> <pre>Kadai k = new Kadai(); k.safe();</pre>
	テスト結果 の例	<p>(以下の出力例はランダムで変わる)</p> <pre>ゼロ除算でもしたのかな ファイルがないよ ぬるぽ</pre>
	ヒント	<p>try-catch の練習です。要するに、<code>randomException()</code> を呼び出して、例外を catch し、例外に応じて出力を変え、例外によっては <code>randomException()</code> を何度も実行せよということである。</p>

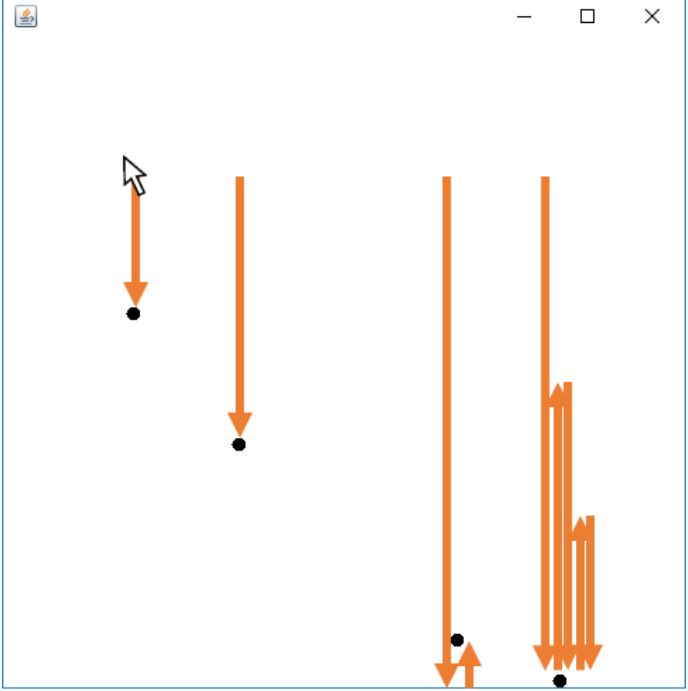
課題 2

2-1	問題設定	<p>モンテカルロシミュレーションとは、乱数を用いてシミュレーション等を行う手法である。今回、この手法を用いて、複数のサイコロの出目の和の期待値を計算することにした。ただし、サイコロの個数は可変とする。</p> <p><code>public double expectedValue(int n, long times)</code> を Kadai.java 上で開発してほしい。ここで引数の <code>n</code> はサイコロの個数（1～）であり、<code>times</code> は試行回数（1～）である。戻り値は算出した期待値である。</p> <p>今回のケースでは、サイコロ <code>n</code> 個の出目を乱数で決定し、和を <code>times</code> 回算出する、出目の結果から期待値を算出する。</p> <p>当然ではあるが、試行回数が増えるごとに待機時間が長くなる。計算機室環境で実施したところ、100,000,000 回の試行をサイコロ 3 個で実施した場合、10 秒弱計算に要した（実装方法による）。試行回数を少な目にしてデバッグすることを推奨しておく。</p>
	テスト例	<p>（Main.java の main() メソッドにて）</p> <pre>Kadai k = new Kadai(); double d = k.expectedValue(2, 1000L); System.out.println(d);</pre>
	テスト結果の例	<p>（試行回数を増やすと理論値に収束していく）</p> <p>7.018999999999999</p>
	ヒント	<p>当然だが理論値は、1 つの場合 3.5、2 つの場合は 7.0、3 つの場合は 10.5 である。</p> <p>期待値の算出手順は問わない。どうしても計算方法がわからない場合、以下に白字でヒントを示すので、コピーして読んでみると良い。</p>

2-2	問題設定	<p>課題 2-1 で試行回数を増やした際, 3 秒でも計算時間がかかっていると, 正常に動いているのか心配になってくる. そこで, 正常に動いていることをユーザに示すためにプログレスバーのようなものを実装しよう.</p> <p><code>public double expValProgress(int n, long times)</code> を開発してほしい. このメソッドでは, 課題 2-1 で開発したメソッドを Thread 上で実行する. その後結果が出るまでの間コンソール上にテスト例のような出力を表示する. (1 行目に「計算中」と表示後, 「■」を 100ms ごとに表示していく. 「■」は 10 個ごとに改行される.) Thread 側で計算完了後, プログレス表示を停止し, 算出結果を戻り値として返す.</p> <p>(本課題ができた学生は動作を教員に見せてチェックを受けること)</p>
	テスト例	<p>(Main.java の main() メソッドにて)</p> <pre>Kadai k = new Kadai(); double res = k.expValProgress(2, 100000000L); System.out.println(""); // ただ改行したいだけ System.out.println(res);</pre>
	テスト結果の例	<p>(■の個数は計算時間によって増減する)</p> <pre>計算中 ■■■■■■■■■■ ■■■■■■■■■■ ■■■■■■■■■■ ■■■■■■■■■■ 6.999997919999999</pre>
	ヒント	フィールドをうまく使ってくれてよい.

課題 3 (発展：多分誰も解けない)

3-1	問題設定	<p>図 1 のようにマウスを押した座標に黒い玉を描画し、その玉が自由落下運動（重力加速度 $g=9.8$）を行うアニメーションを作成せよ。ただし、アニメーション中もマウスイベントを取得し、複数の玉を表示できるようにすること。Panel のサイズは問わないが玉のサイズは 10px とせよ。玉が画面外に出る前に Thread を停止すること。</p> <p>(本課題ができた学生は動作を教員に見せてチェックを受けること)</p>
	動作画面例	 <p>図 1. 動作画面例</p>
	ヒント	<p>物理の公式を忘れた場合ググると良い 実装方法のヒントを先に読んでも面白くないので、下記に白字で示す。コピペしてどこかに貼り付けると読める。</p>
	もっとヒント	

3-2	問題設定	<p>課題 3-1 に反発係数を導入し, 図 2 のように反発係数に応じて地面 (画面の下端) で跳ね返るアニメーションを実装せよ. ちなみに反発係数を 0.0 にすると課題 3-1 と同じ動きとなる.</p> <p>(本課題ができた学生は動作を教員に見せてチェックを受けること)</p>
	動作画面例	 <p>図 2. 動作画面例</p>
	ヒント	課題3-1ができればできると思われる。