Programowanie w JAVA
Lab. 7 – Współbieżność / wątki

1. Wykorzystując metodę Monte Carlo obliczyć całkę z funkcji:

$$
f(x,y) = \left(\left(\frac{x}{7}\right)^2 \cdot \sqrt{\frac{\left||x|-3\right|}{|x|-3}} + \left(\frac{y}{3}\right)^2 \cdot \sqrt{\frac{\left|y+3\cdot\frac{\sqrt{33}}{7}\right|}{y+3\cdot\frac{\sqrt{33}}{7}}} - 1 \right) \cdot \left(\left|\frac{x}{2}\right| - \left(\frac{3\sqrt{33}-7)}{112}\right) \cdot x^2 - 3 + \sqrt{1-\left(\left|\left|x\right|-2\right|-1\right)^2} - y \right) \cdot
$$

$$
\left(9 \cdot \sqrt{\frac{\left|\left(|x|-1\right)\cdot\left(|x|-0.75\right)\right|}{\left(1-|x|\right)\cdot\left(|x|-0.75\right)}} - 8|x| - y \right) \cdot \left(3|x| + 0.75 \cdot \sqrt{\frac{\left(|x|-0.75\right)\cdot\left(|x|-0.5\right)}{\left(0.75-|x|\right)\cdot\left(|x|-0.5\right)}} - y \right) \cdot
$$

$$
\left(2.25 \cdot \sqrt{\frac{\left|\left(x-0.5\right)\cdot\left(x+0.5\right)\right|}{\left(0.5-x\right)\cdot\left(0.5+x\right)}} - y \right) \cdot \left(\frac{6\cdot\sqrt{10}}{7} + \left(1.5-0.5|x|\right) \cdot \sqrt{\frac{\left||x|-1\right|}{|x|-1}} - \left(\frac{6\cdot\sqrt{10}}{14}\right) \cdot \sqrt{4-\left(|x|-1\right)^2} - y \right)
$$

Dla następujących założeń:

$$
x, y \in \langle -8, 8 \rangle
$$

$$
f(x,y) \in \Re
$$

Wykres narysować na elemencie Canvas JavaFX w sposób nieblokujący (w osobnym wątku). Zbiór punktów należących do funkcji zaznaczyć kolorem żółtym (kolorując odpowiednie piksele), pozostałe punkty kolorem ciemnoniebieskim. **Dla ułatwienia udostępniam klasę Equation.java której metoda `boolean calc(double x, double y)` zwraca wartość PRAWDA/FAŁSZ w zależności od tego czy punkt <`x,y`> spełnia równanie `f(x,y)` czy nie**.

2. Podstawowa funkcjonalność:
   a. Możliwość prowadzenia obliczeń w sposób asynchroniczny z możliwością ich zastopowania w dowolnym momencie (patrz przykład 3.2). Wizualizacja wykresu na elemencie Canvas poprzez zaznaczanie pojedynczych pikseli na określony w zadaniu kolor
   b. Prezentowanie postępu obliczeń na komponencie `ProgressBar`
   c. Formatka z możliwością określenia ilości punktów do symulacji i wyświetleniem wyniku całkowanie oraz przyciskami START, STOP
   d. Obliczenie całki wyrażenie – wynik liczbowy wyświetlany w oknie
   e. Klasa realizująca metodę Monte Carlo musi być NIEZALEŻNA od GUI.

3. Wskazówki

3.1 Rysowanie w JavaFX może zostać zaimplementowane obiektowo (gdzie pojedyncze kształty to obiekty Java) lub rastrowo poprzez rysowanie na płótnie (Canvas). Na tych zajęciach skupimy się na rysowaniu rastrowym.

```
public class Controller {

    @FXML
    private Canvas canvas;

    @FXML
    private void handleRunBtnAction(){
        GraphicsContext gc =
        canvas.getGraphicsContext2D(); drawShapes(gc);
    }


    private void drawShapes(GraphicsContext gc)
        { gc.setFill(Color.BLUEVIOLET);
        gc.fillRect(gc.getCanvas().getLayoutX(),
                gc.getCanvas().getLayoutY(),
                gc.getCanvas().getWidth(),
                gc.getCanvas().getHeight());

    }
}
```

Gdzie canvas – zbindowany jest z kontrolką `Canvas` a `handleRunBtnAction` przypisany jest do metody `onAction` przycisku.

Pamiętaj, że w punkt `<0,0>` na płótnie zdefiniowany jest w lewym, górnym rogu co nie odpowiada typowemu układowi współrzędnych kartezjańskich gdzie punkt `<0,0>` zdefiniowany jest w lewym dolnym rogu - oznacza to, ze rysując wykres zdefiniowany w kartezjańskim układzie współrzędnych konieczna jest dodatkowa translacja punktu przed narysowaniem.

Przy rysowaniu dużej ilości pojedynczych pikseli lepiej jest rysować na obiekcie (BufferedImage) a następnie wyświetlać go na płótnie okresowo (tj. co n punktów).

```
BufferedImage bi= new BufferedImage(600, 400,
BufferedImage.TYPE_INT_RGB);
bi.setRGB(x, y, Color.YELLOW.getRGB());
gc.drawImage(SwingFXUtils.toFXImage(bi, null), 0,0
);
```

gdzie:

- x, y – położenie punktu
- width, height – wielkość obrazu

Uwaga!

Korzystanie z metody `drawImage` w pętli może być bardzo obciążające. Pamiętaj, że nie musisz odrysować obrazka na płótnie po każdym umieszczeniu piksela. Możesz to robić co jakiś czas, np.

```
if(i % 1000 == 0)
    gc.drawImage(SwingFXUtils.toFXImage(bi, null), 0,0 );
```

3.2 Asynchroniczność w JavaFX. Wątek to pewna klasa której metoda call uruchamiana jest w tle nie blokując tym samym wykonywania wątku (zadań) aplikacji. Zalecanym podejściem do wielowątkowości w JavaFX to użycie klasy Task

AsyncTask.java

```java
import javafx.concurrent.Task;


public class AsyncTask extends Task {
    @Override
    protected Object call() throws Exception {
        while(true){
            // code
            if(isCancelled()) break;
        }
        return null;
    }
}
```

Controller.java

```java
public class Controller {

    private DrawerTask task;

    @FXML
    private void handleRunBtnAction(){
        task = new DrawerTask();
        new Thread(task).start();

        // inne polecenia wykonywane w głównym wątku
    }

    @FXML
    private void handleStopBtnAction(){
        task.cancel();
    }

}
```

Aby przekazać do wątku dane można użyć konstruktora.
Zwróć uwagę, że Task zwraca obiekt typu Object – oznacza to w praktyce, że Task może zwracać dowolny typ obiektu (można go również sparametryzować, tak jak szablon).
Aby pobrać z wątku (Task) dane po jego zakończeniu należy dodać do niego zdarzenie `OnSucceeded`, np.

Controller.java

```java
@FXML
private Canvas canvas;

private DrawerTask task;

@FXML
private void handleRunBtnAction(){
    GraphicsContext gc = canvas.getGraphicsContext2D();
    task = new DrawerTask(gc);
```

```
    task.setOnSucceeded(new EventHandler<WorkerStateEvent>() {
        @Override
        public void handle(WorkerStateEvent event) {
            int var = (int) task.getValue();
        }
    });
    new Thread(task).start();

}
```

O szczegółach obsługi wątków można przeczytać w
http://docs.oracle.com/javafx/2/threads/jfxpub-threads.htm

3.3 Obsługa ProgressBar w wątku

Chcąc raportować postęp operacji w wątku (np. numer iteracji) do GUI, należy powiązać kontrolkę ProgressBar z wątkiem:

```
@FXML
private ProgressBar progressBar;


progressBar.progressProperty().bind(task.progressProperty());
```

gdzie task jest zainicjowanym obiektem klasy dziedziczącej po Task, a następnie uaktualnić postęp w obrębie metody `call()` wątku poprzez wywołanie:

```
updateProgress(i, max);
```

gdzie *i* – aktualna wartość postępu, *max* – maksymalna wartość jaką może przyjąć *i*

3.4 Losowa liczba rzeczywista z zakresu <MIN, MAX>:

```
Random random = new Random();
double x = MIN + (MAX - MIN) * random.nextDouble();
```

Aby Random działał poprawnie należy korzystać z tego samego obiektu do generowania kolejnych liczb – obiekt powinien być tworzony tylko raz!

3.5 Skalowanie wartości x z zakresu <A,B> do zakresu <C,D>

```
x' = ((D-C) * (x-A) / (B-A) + C)
```

3.6 Monte Carlo

Ideą metody Monte Carlo jest losowe próbkowanie przestrzeni w poszukiwaniu rozwiązania problemu. Metodę tę można utożsamiać ze strzałem na oślep i sprawdzeniem czy trafiło się w tarczę czy nie. W omawianym przypadku, jeśli wylosowana para liczb typu double *x,y* spełnia wymóg $f(x,y) \leq 0$ (tzn. wartość funkcji w punkcie x,y jest mniejsza od zera) oznacza to, że punkt leży w zakresie zdefiniowanym przez funkcję f.
Losując n (pewną dużą ilość) punktów istnieje pewna szansa (czyli prawdopodobieństwo), że część tych punktów (oznaczmy je przez *k*) trafi w obszar funkcji, zatem pole pod krzywą (czyli całka) wynosi:

$$P_f = P\frac{k}{n}$$

gdzie *P* jest powierzchnią przestrzeni rozwiązania (czyli np. polem kwadratu 16x16 w którym poszukiwane było rozwiązanie). W omawianym przykładzie n na poziomie 10 mln powinno być wystarczającą liczbą punktów aby policzyć całkę (czyli pole pod wykresem funkcji). Sprawdź wyniki dla różnej ilości punktów. Więcej informacji o metodzie wraz z przykładami: https://pl.wikipedia.org/wiki/Metoda_Monte_Carlo

https://www.codeproject.com/Articles/677591/Defining-Custom-Source-Event-Listener-in- Java

4. Teoria
   a. Metoda Monte Carlo
   b. Pojęcie wątku
   c. Sposoby tworzenia podstawowych wątków w Java – interfejs Runnable i klasa Thread.
   d. Przekazywanie i pobieranie danych z wątku.
   e. Synchronizacja kilku wątków.
   f. Specjalistyczne klasy do wątków GUI – na przykładzie JavaFx – klasa Task
   g. Thread Pools – czym są, kiedy i dlaczego ich używamy?

Po uzyskaniu zaliczenia na zajęciach, prześlij źródła w archiwum **zgodnie z konwencją nazewniczą** (patrz Lab0.pdf) do chmury na adres:
https://cloud.kisim.eu.org/s/5PJXWTywQX4cxMD

najpóźniej do następnych zajęć.