Callbacky – zpětná volání

V následující ukázce si vysvětlíme co jsou to tzv. callbacky a také jak s nimi pracovat. Využívají se jako méně náročná, a také přesnější možnost pro vyvolání zadané akce v určitý moment (například po té, co dojde cyklus ke konci). Práci s callbacky si ukážeme na projektu počítající prvočísla metodou Eratosthenových sít.

Co to je callback?

Jedná se o spustitelný kód, který je následně předán do jiného kódu, který má za úkol vykonat předem určenou činnost ve vhodný čas. V ukázce bude vhodným časem například určení dalšího prvočísla.

Tvorba projektu

Pro začátek si budeme muset vytvořit 3 třídy:

1. Callback – Do této třídy umístíme samotný základ callbacku. Musíme do něj však předat informace z třídy Eratosthen.
2. Eratosthen.java – Zde se bude nacházet jádro projektu. Budou zde vytvořeny callbacky jako takové a zároveň dojde k vytvoření kódu pro Eratosthenova síta za pomoci kontejnerů. Tato třída bude muset být spustitelná, proto bude implementovat Runnable.
3. MainWindow – Zde budeme mít okno aplikace, ve kterém nalezneme možnost zadání čísla, po které bude určování prvočísel probíhat a samozřejmě jejich samotný výpis. Bude se tedy jednat o jFrame, který bude implementovat třídu Callback, abychom mohli přeměnit MainWindow na callback.

Callback

Práce zde bude velice jednoduchá. Je třeba pouze vytvořit interface, který přebere informace z třídy Eratosthen.

Kód:

```
public interface Callback {
    public void reactToCall(Eratosthen er);
}
```

Eratosthen

V následující třídě budeme muset zaprvé vytvořit privátní proměnné Callback a následně jim vytvořit metody.

Kód:

```
  private Callback finalCall;
  private Callback newPrimeCall;
  private Callback changeCall;
  //Metodu addFinalCallback budeme využívat pro výpis finálních čísel.
  public void addFinalCallback(Callback finalCall) {
      this.finalCall=finalCall;
  }
  //Pomocí metody addNewPrimeCallback se dozvíme, že byl dosažen konec
programu.
  public void addNewPrimeCallback(Callback newPrimeCall) {
      this.newPrimeCall=newPrimeCall;
  }
  //Metoda addChangeCallback nás bude informovat o všech změnách v seznamu.
  public void addChangeCallback(Callback changeCall) {
      this.changeCall=changeCall;
  }
```

Momentálně sice máme už základ callbacků a víme, kdy by se měly aktivovat, avšak program toto zatím neví, proto musí být doplněn o samotné výpočetní jádro. Abychom jej mohli vytvořit, budeme potřebovat kontejnery tvořené celými čísly. První z nich naplníme čísly od 2 do max (kde max je hodnota zadaná uživatelem skrz input v MainWindow). Do druhého budeme postupně přidávat samotná prvočísla, respektive bude zatím prázdný.

Kód:

```
  private SortedSet<Integer> primeNumberList = new TreeSet();
  private SortedSet<Integer> numbersList = new TreeSet();
  public SortedSet<Integer> getPrimes() {
      return this.primeNumberList;
  }
  public SortedSet<Integer> getNumbers() {
      return this.numbersList;
  }
  private void fillNumbers() {
      for(int i = 2; i <= this.max; i++) {
          numbersList.add(i);
      }
  }
```

Následně se přesuneme k samotné výpočetní logice. Podle Eratosthenových sít budeme tedy postupně testovat všechna čísla v prvním seznamu na prvočísla pomocí násobení již známými (a do druhého kontejneru přesunutými) prvočísly. Důležité je při nalezení nového prvočísla / změně v listu / ukončení výpočtu informovat přiřazený callback pomocí zavolání metody reactToCall.

Kód:

```
  public void countPrimes() {
      this.fillNumbers();
      while(numbersList.size()>0) {
          Integer prime = numbersList.first();
          numbersList.remove(prime);
```

```
            primeNumberList.add(prime);
            if (newPrimeCall!=null) newPrimeCall.reactToCall(this);
            try {
                Thread.sleep(100);
            } catch (InterruptedException ex) {
            }
            Iterator<Integer> it = numbersList.iterator();
            while(it.hasNext()) {
                try {
                    Thread.sleep(100);
                } catch (InterruptedException ex) {
                }
                Integer number = it.next();
                if (number%prime==0) {
                    it.remove();
                    if (changeCall!=null) changeCall.reactToCall(this);
                }
            }
        }
        if (finalCall!=null) finalCall.reactToCall(this);
    }
```

Bystřejší si možná položili otázku, proč jsou v kódu pokusy o pozastavení vlákna. Odpověď se jednoduchá – k ničemu. Pozastavení vláken zde slouží pouze pro uživatele, aby měl dostatek času se podívat na práci programu a výpis prvočísel.

Celou třídu zakončíme vyvoláním metody run, která spustí výpočet prvočísel.

Kód:

```
    public void run() {
        countPrimes();
    }
```

MainWindow

Zde vytvoříme již zmíněné hlavní a viditelné okno programu. bude obsahovat v zásadě 4 objekty:

1. jButton – Při kliknutí na něj se celý výpočet spustí.
2. jTextField (zde pojmenován jako jMax) – Zde bude mít uživatel možnost zadat již zmíněný input, oznamující programu poslední kontrolované číslo.
3. jLabel1 – Bude vypisovat všechna dosud nalezená prvočísla.
4. jLabel2 – Zde bude naopak vypsán kontejner obsahující řadu kontrolovaných čísel. Jak se jeho obsah bude zmenšovat, budou ubývat vyřazená čísla i zde.

Nezapomeňme také, že je třeba, aby celý jFrame implementoval rozhraní Callback. Dále bude třeba vytvořit metodu reactToCall, která bude okopírována ze třídy Eratosthen a bude zapisovat data do zmíněních jLabelů.

Kód:

```
    public void reactToCall(Eratosthen er) {
    jLabel1.setText(er.getPrimes().toString());
    jLabel2.setText(er.getNumbers().toString());
}
```

Posledním krokem v celém projektu je vytvoření kódu, který se spustí při akci vykonané na jButton. Kód bude muset obsahovat okopírování kódu ze třídy Eratosthen a následné vytvoření callbacků na základě získaných dat z třídy Eratosthen. Aby bylo vše naprosto perfektní, vytvoříme i nové vlákno, které umožní výpisům do jLabelů paralelní běh.

Kód:

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    Eratosthen er = new Eratosthen(Integer.parseInt(jMax.getText()));
    er.addFinalCallback(this);
    er.addNewPrimeCallback(this);
    er.addChangeCallback(this);
    Thread th = new Thread(er);
    th.start();
}
```