Programowanie w języku Java - Adam Krechowicz

1 Wątki

Java posiada własny system obsługi wątków. Tworzenie wątku polega na stworzeniu klasy implementującej interfejs *Runnable* lub dziedziczącej po klasie *Thread*. Kod wykonywany w oddzielnym wątku umieszczany jest w metodzie *run()*. Wystartowanie wątku realizowane jest przez metodę *start()*.

1.1 Przykład użycia klasy Thread

```
package com.adeik.javatest.threading;

class Watek extends Thread{

  @Override
  public void run() {
    while (true){
      System.out.println("abc");
    }
  }

}

public class ThreadTest {

  public static void main(String[] args) {
    Watek w = new Watek();
    w.start();
    while (true){
      System.out.println("xyz");
    }
  }

}
```

Oczekiwanie na zakończenie innego wątku realizowane jest przez metodą *join()*.

1.2 Przykład użycia interfejsu Runnable i metody join

```
package com.adeik.javatest.threading;

class Watek1 implements Runnable{

  @Override
  public void run() {
      for (int i = 0; i < 10000; i++)
        System.out.println("Ale fajnie :]");
  }
```

```
}

public class RunnableTest {

  public static void main(String[] args) {
    Thread t = new Thread(new Watek1());
    t.start();
    try {
      t.join();
    } catch (InterruptedException e) {
      e.printStackTrace();
    }
  }

}
```

2 Czekanie określonego czasu

Aby uśpić wątek na określony czas można wykorzystać metodę *Thread.sleep()*. Metoda ta jako parametr przyjmuje liczbę milisekund do odczekania.

3 Synchronizacja

Aby zapewnić jednoczesny dostęp do zasobów przez wiele wątków konieczne jest zapewnienie metod synchronizacji.

3.1 Synchronizowana metoda

Dwie synchronizowane metody wywoływane na tym samym obiekcie nie mogą byc przez siebie przerwane.

```
package com.adeik.javatest.threading;

class Resource{

  public static synchronized void metoda(int watek){
    System.out.println(watek+"Poczatek");
    try {
      Thread.sleep(1000);
    } catch (InterruptedException e) {
      e.printStackTrace();
    }
    System.out.println(watek+"Koniec");
  }

}

public class SynchronizedTest {

  public static void main(String[] args) {
    Thread tt1 = new Thread(new Runnable() {
```

```
      @Override
      public void run() {
        while(true)
          Resource.metoda(1);
      }
    });

    tt1.start();
    Thread tt2 = new Thread(new Runnable() {

      @Override
      public void run() {
        while(true)
          Resource.metoda(2);
      }
    });
    tt2.start();
  }

}
```

3.2 Synchronizowany blok

Blok synchronizowany realizuje bezpieczną metodę dostępu do wspólnych zasobów.

```
package com.adeik.javatest.threading;

class Resource2{
  static Resource2 r = new Resource2();

  public static void metoda(int watek){
    synchronized (r) {
      System.out.println(watek+"Poczatek");
      try {
        Thread.sleep(1000);
      } catch (InterruptedException e) {
        e.printStackTrace();
      }
      System.out.println(watek+"Koniec");
    }
  }

}

public class SynchronizedBlock {

  public static void main(String[] args) {
    Thread tt1 = new Thread(new Runnable() {

      @Override
      public void run() {
        while(true)
          Resource2.metoda(1);

      }
    });
```

```
    tt1.start();
    Thread tt2 = new Thread(new Runnable() {

      @Override
      public void run() {
        while(true)
          Resource2.metoda(2);

      }
    });
    tt2.start();
  }

}
```

3.3 Monitor

Klasa *Object* zawiera zbiór metod pomagających realizować synchroniczny dostęp:

- *wait()* – Metoda próbująca uzyskać dostęp do obiektu
- *notify()* – Zakończenie używania obiektu i poinformowanie jednego z czekających wątków
- *notifyAll()* – Zakończenie używania obiektu i poinformowanie wszystkich czekających wątków

```
package com.adeik.javatest.threading;

class Monitor{
  public static Object o = new Object();
}

class Watek2 extends Thread{

  @Override
  public void run(){
    System.out.println("Czekam sobie...");
    synchronized(Monitor.o){
      try {
        Monitor.o.wait();
      } catch (InterruptedException e) {
        e.printStackTrace();
      }
    }
    System.out.println("Doczekalem sie :)");
  }
}

class Watek3 extends Thread{
```

```
  @Override
  public void run(){
    System.out.println("Jeszcze cos robie");
    System.out.println("Jeszcze cos robie");
    System.out.println("Jeszcze cos robie");
    synchronized(Monitor.o){
      Monitor.o.notify();
    }
  }
}

public class WaitTest {

  public static void main(String[] args) {
    Watek2 w2 = new Watek2();
    w2.start();
    Watek3 w3 = new Watek3();
    w3.start();
  }

}
```

4 Synchronizowane kolekcje

Dostęp do zwykłych kolekcji z kilku wątków może być niebezpieczny. Klasa *Collection* zawiera metody zwracające kolekcje bezpieczne do zastosowania w programach wielowątkowych.

- synchronizedList(List list)
- synchronizedMap(Map m)
- synchronizedSet(Set s)
- synchronizedSortedMap(SortedMap m)
- synchronizedSortedSet(SortedSet s)

5 Zadania do wykonania

1. Zapznać się z dokumentacją klasy Thread i interfejsu Runnable
2. Stworzyć wątki za pomocą klasy Thread i interfejsu Runnable
3. Zaczekać na zakończenie wszystkich wątków
4. Zrealizować dostęp za pomocą synchronizowanych metod
5. Zrealizować dostęp za pomocą synchronizowanych bloków

6. Porównać działanie programów z metodami i blokami synchronizowanymi i bez

7. Zrealizować dostęp za pomocą metod klasy Object

8. Zapznać się z dokumentacją metod zwracających synchronizowane kolekcje