1 Threads

Java have its own thread mechanisms. Creating thread means creating class that implements *Runnable* interface or derived from *Thread* class. Code that needs to be executed in other thread needs to be included in *run()* method. Starting the thread is done by executing *start()* method.

1.1 Example of using Thread class

```
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");
    }
  }

}
```

By invoking *join()* method we wait for thread to finish its execution.

1.2 Example of using Runnable interface and join method

```
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 Waiting specified time

To wait specified time method *Thread.sleep()* can be used. As input number of milliseconds are given.

3 Synchronization

Synchronization is used to perform safe access to resources from different threads.

3.1 Synchronized method

Two synchronized methods executed on the same objects can not be interrupt.

```
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 Synchronized block

Synchronized block is used to provide safe operations on resources.

```
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

Object class contains methods to perform low level synchronization

- *wait()* – Method that tries to access object
- *notify()* – Done working with object. Notify other thread that waits on this object
- *notifyAll()* – Done working with object. Notify all threads that waits for this object

```
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 Synchronized collections

Using typical collections in multithreaded programs is unsafe. *Collection* class can be used to create tread safe collections.

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

5 Tasks to complete

1. Make familiar with Thread and Runnable documentation
2. Create threads from Thread class and Runnable interface
3. Wait for thread to complete
4. Synchronize access by synchronized methods
5. Synchronize access by synchronized blocks
6. Compare program execution with synchronized elements and without them

7. Synchronize access using methods from Object class
8. Make familiar with methods that creates synchronized collections