Laboratorium 1

1. Watek a proces
2. Watki w javie
    - wbudowane w jezyk
    - dwa rodzaje implementacji
        - Przez dziedziczenie z klasy Thread

          Klasa:

          ```
          class MyThread extends Thread {
               . . .
               public void run() {
                    . . .
               }
          }
          ```

          Tworzenie obiektu i uruchamianie watku:

          ```
          MyThread t = new MyThread();
          t.start();
          ```

        - Przez implementacje interfejsu Runnable

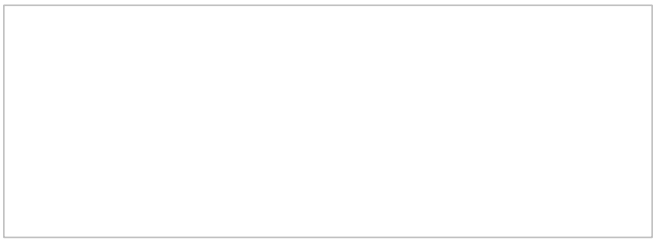
          Klasa:

          ```
          class MyThreadR implements Runnable {
               . . .
               public void run() {
                    . . .
               }
          }
          ```

          Tworzenie i uruchamianie:

          ```
          MyThreadR r = new MyThreadR();
          Thread t = new Thread(r);
          t.start();
          ```

    - metoda run()
    - klasa Thread -- konstruktory i metody
    - Cykl zycia watku, stany watku



3. Podstawowe mechanizmy synchronizacji wbudowane w jezyk monitory zwiazane z obiektem, slowo kluczowe `synchronized`, metody `wait` oraz `notify`

4. Wyscig
    - wiecej niz jeden watek korzysta jednoczenie z zasobu dzielonego, przy czym co najmniej jeden probuje go zmienic
    - przyczyna niedeterministycznego zachowania sie programu
    - moze prowadzic do trudnych do wykrycia bledow
    - pojecie thread-safety (bezpieczestwo dla watkow)
5. Animowane watki
6. Zadanie
    - Napisac program BEZ SYNCHRONIZACJI, w ktorym mamy obiekt klasy Counter przechowywujący pewną zmienną całkowitą oraz dwie metody inkrementującą i dekrementującą.

      Nastepnie jeden watek wywoluje na tym obiekcie metode inkrementująca 100000000 razy, drugi dekrementująca 100000000 razy.

      Czy wynik zawsze jest zero? Sprawdzić działanie na różnych systemach.

    - Wprowadzić synchronizację do programu wykorzystujac slowo kluczowe "synchronized"

    - Mamy klika procesów produkujacych wiadomosci (szkielet kodu) i kilka konsumujacych wiadomosci (szkielet kodu) do/z jednoelementowego bufora. Zadaniem jest napisanie klasy Buffer z metodami *put* i *take*, tak, aby dostep byl synchronizowany uzywajac monitora Javy dla obiektu klasy Buffer. Kazda wiadomosc jest produkowana przez jednego

producenta i konsumowana przez jednego, dowolnego konsumenta.

- wyjasnij, dlaczego przy sprawdzaniu warunku czy bufor jest pusty/pelny nalezy uzyc instrukcji *while* , a nie wystarczy instrukcja *if* .

Bibliografia

1. Z Weiss, T Gruźlewski "Programowanie współbieżne i rozproszone w przykładach i zadaniach"
2. Jacek Rumiki, Jezyk Java. Rozdzial o watkach
3. Bill Venners, Inside the Java Virtual Machine (rozdz. 5, *The Java Virtual Machine*), McGraw-Hill Companies; 2nd Bk&Cdr edition, 2000.

Katarzyna Rycerz, kzajac at agh.edu.pl

(takze na postawie opracowan dr B. Balisia)