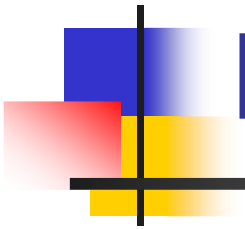
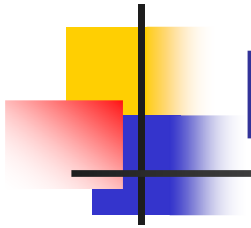


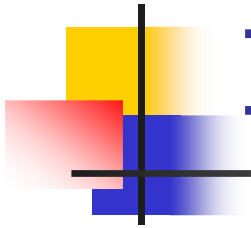
Programowanie w języku Java

Wykład 3: Programowanie współbieżne: techniki zaawansowane

Konstrukcje wysokiego poziomu

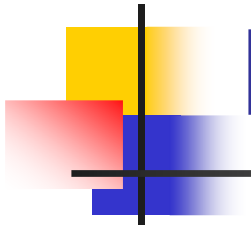


- JDK 5.0:
  - java.util.concurrent
  - java.util.concurrent.locks



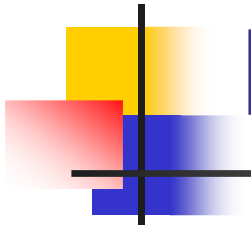
Intefejs Lock

- Implementacja: ReentrantLock
- lock() – początek sekcji krytycznej
- unLock() – koniec sekcji krytycznej
- boolean tryLock() – lock() wtedy gdy sekcja nie jest zablokowana (można podać timeout)
- boolean isLocked() – sprawdzenie czy sekcja jest zablokowana



Przykład tryLock()

```
Lock lock = ...;
if (lock.tryLock()) {
   try {
        // sekcja krytyczna
   }
   finally {
        lock.unlock();
   }
} else {
   // alternatywna akcja
}
```



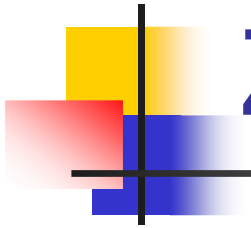
Mechanizm wait/notify

- Condition lock.newCondtition() – zwraca warunek związany z daną blokadą
- await()
- signal(), signalAll()

Przykład: producent/konsument

```
import java.util.concurrent.locks.*;
public class CubbyHole2 {
 private int contents;
 private boolean available = false;
 private Lock aLock = new ReentrantLock();
 private Condition condVar = aLock.newCondition();
 public int get(int who) {
  aLock.lock();
  try {
    while (available == false) {
        try { condVar.await(); }
        catch (InterruptedException e) { }
     }
    available = false;
    System.out.println("Consumer " + who + " got: " +
              contents);
    condVar.signalAll();
  }
 finally {  aLock.unlock(); return contents; }
}
```

```
public void put(int who, int value) {
  aLock.lock();
  try {
     while (available == true) {
        try {
           condVar.await();
         } catch (InterruptedException e) { }
      }
     contents = value;
     available = true;
   System.out.println("Producer " + who + „
                 put: " + contents);
    condVar.signalAll();
    } finally { aLock.unlock(); }
  }
}
```



Zarządzanie wątkami

- Executor e

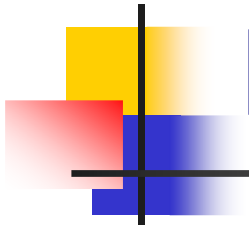
  (new Thread(r)).start(); == e.execute(r);

- ExecutorService

  submit() –zwraca obiekt Future zawierający status wątku oraz zwracana wartość (w przypadku wątków Callable

- ScheduleExecuterService:

  schedule() – szeregowanie wątków w czasie

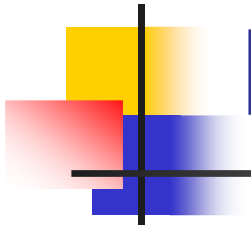


Przykład

```
import java.util.concurrent.*;
import java.util.*;

class TaskWithResult implements
Callable<String> {
  private int id;
  public TaskWithResult(int id) {
    this.id = id;
  }
  public String call() {
    return "result of TaskWithResult " + id;
  }
}
```

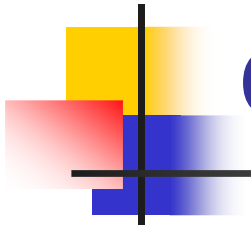
```
public class CallableDemo {
  public static void main(String[] args) {
    ExecutorService exec =
Executors.newCachedThreadPool();
    ArrayList<Future<String>> results =
      new ArrayList<Future<String>>();
    for(int i = 0; i < 10; i++)
      results.add(exec.submit(new TaskWithResult(i)));
    for(Future<String> fs : results)
      try {
        // get() blocks until completion:
        System.out.println(fs.get());
      } catch(InterruptedException e) {
        System.out.println(e);
        return;
      } catch(ExecutionException e) {
        System.out.println(e);
      } finally {
        exec.shutdown();
      }
  }
}
```



Pule wątków

- Klasa Executors:
  - newFixedThreadPool()
  - newScheduledThreadPool()
  - newSingleThreadExecutor ()
  - newCachedThreadPool()

Współbieżne kolekcje obiektów



- BlockingQueue
- ConcurrentMap, ConcurrentHashMap
- ConcurrentNavigableMap, ConcurrentSkipListMap

Operacje elementarne na zmiennych

java.util.concurrent.atomic

```
class Counter {
   private int c = 0;
   public void increment() {
      c++;
   }
   public void decrement() {
      c--;
   }
   public int value() {
      return c;
   }
}
```

```
class SynchronizedCounter {
   private int c = 0;
   public synchronized void increment() {
      c++;
   }
   public synchronized void decrement() {
      c--;
   }
   public synchronized int value() {
      return c;
   }
}
```

```
import java.util.concurrent.atomic.AtomicInteger;
 class AtomicCounter {
   private AtomicInteger c = new AtomicInteger(0);
   public void increment() {
      c.incrementAndGet();
   }
   public void decrement() {
      c.decrementAndGet();
   }
   public int value() {
      return c.get();
   }
}
```

Koniec