Java

AGH

Zagadnienia

- SRP
- Refaktoryzacja
- Wzorce projektowe
- Klasy anonimowe
- λ
- Strumienie
- Wątki

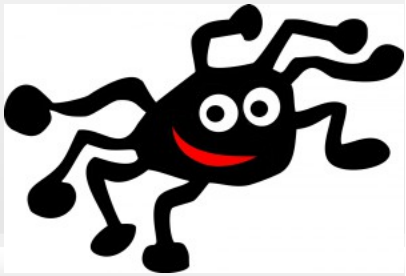
Java, Eclipse, jsoup

Crawler

Crawler sieci Web

- Sieć Web – rozproszony system serwerów udostępniających zasoby elektroniczne
- Dostępny dla użytkowników Internetu za pośrednictwem przeglądarek

- Crawler Web – **program** udający przeglądarkę
- Cele:
  - Indeksowanie
  - Wyszukiwanie w czasie rzeczywistym

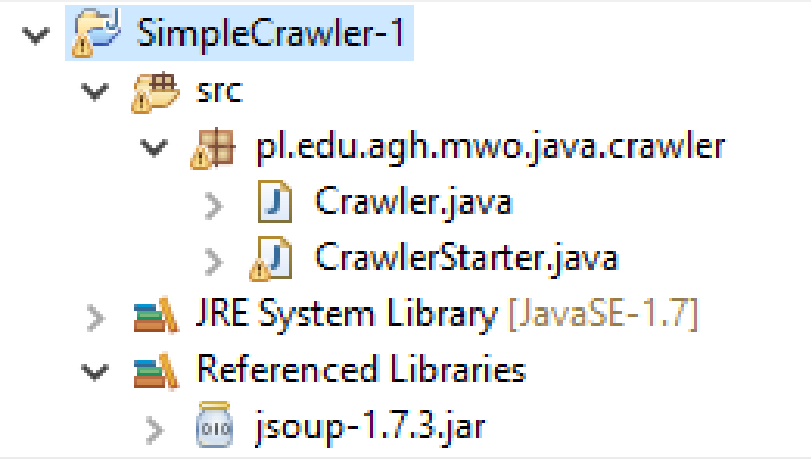


Crawler w Javie?

- Java jest dynamicznie rozwijana od 20 lat
- Typowe problemy zostały już rozwiązane
  → biblioteki
- Jsoup: https://jsoup.org/

Pierwszy Crawler

- Projekt *SimpleCrawler-1*
  - Zaimportujmy do Eclipse



Wyszukiwanie

- Dodaj funkcjonalność wyszukiwania tylko zdań zawierających określone słowo

```
String sentence = "Ala ma kota";
if (sentence.contains("kota"))
{
    Yes!
}
```

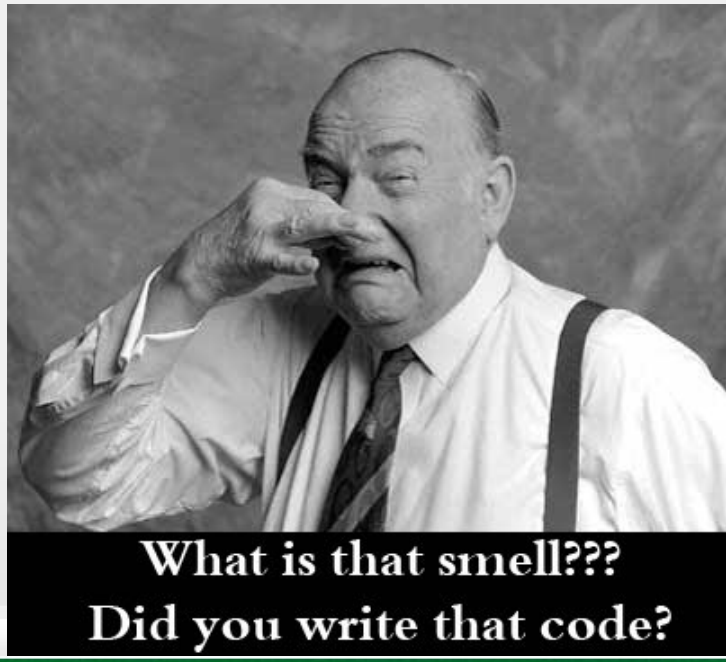
- Zmodyfikuj program by drukował poniżej zdania zawierające daną literę co najmniej *n* razy

```
String sentence = "Ala ma kota";
char charToFind = 'p';
for (int j = 0; j < sentence.length(); j++)
{
    if (sentence.charAt(j) == charToFind)
    {
        Yes++;
    }
}
```

- Parametry wyszukiwania Crawler powinien dostać „z zewnątrz”

Problem w kodzie?

- Nie... przecież działa!
- Jak będziemy dodawać kolejne funkcje?
- Jak sprawdzimy czy zmiany nie zepsuły kodu?
- Napiszmy testy!

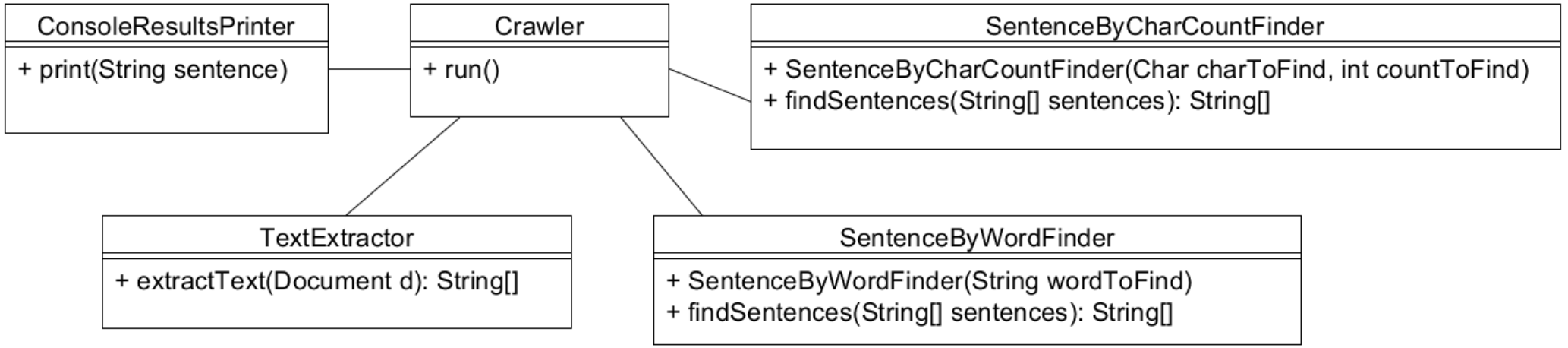


SRP

- Single Responsibility Principle
- Jakie odpowiedzialności ma obecnie klasa Crawler?
  1)
  2)
  3)
  4)

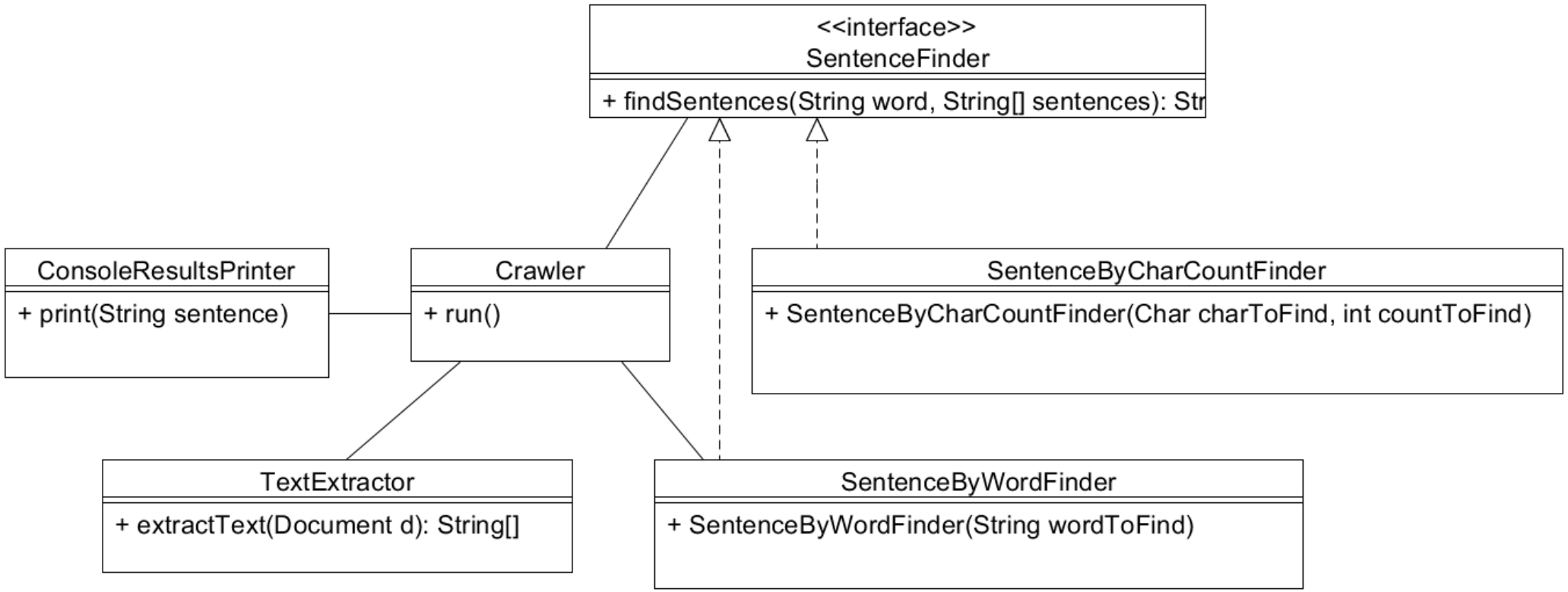
Refaktoryzacja

- Zmiana struktury kodu bez zmiany jego funkcjonalności



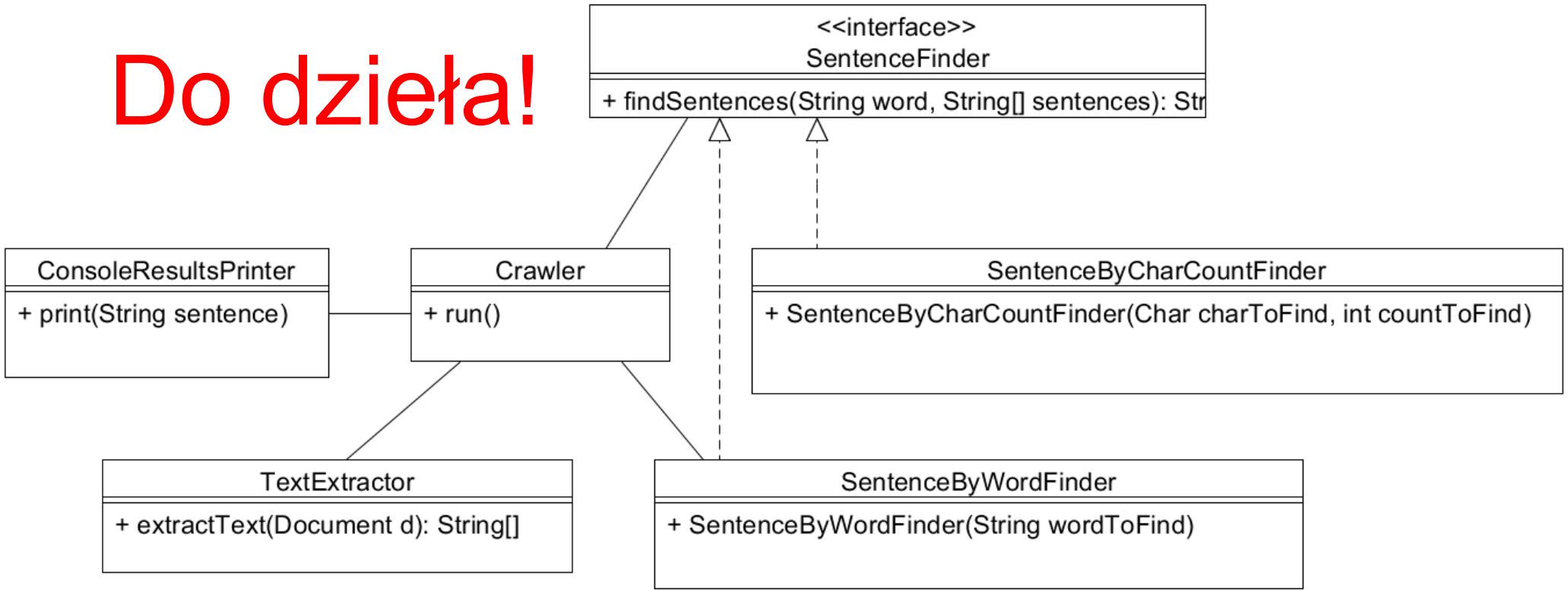
Refaktoryzacja

- Zawsze szukamy uogólnień
- Wzorzec projektowy: **strategia**



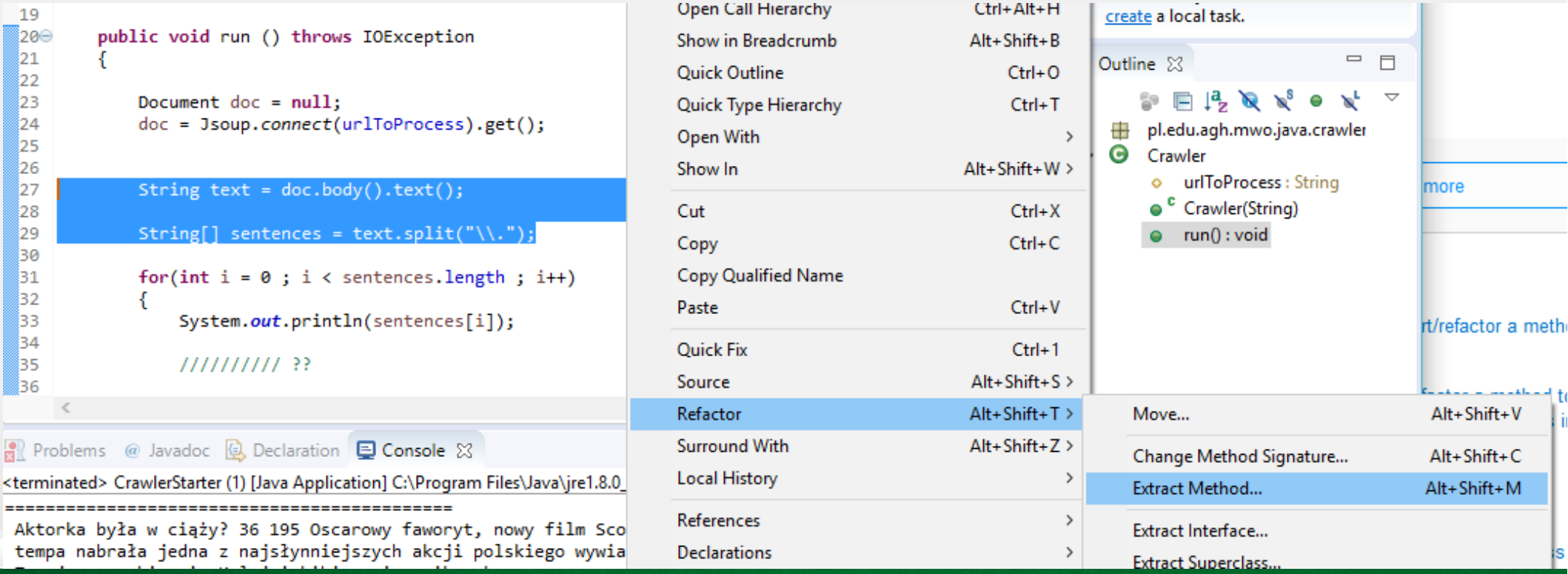
Refaktoryzacja

Do dzieła!



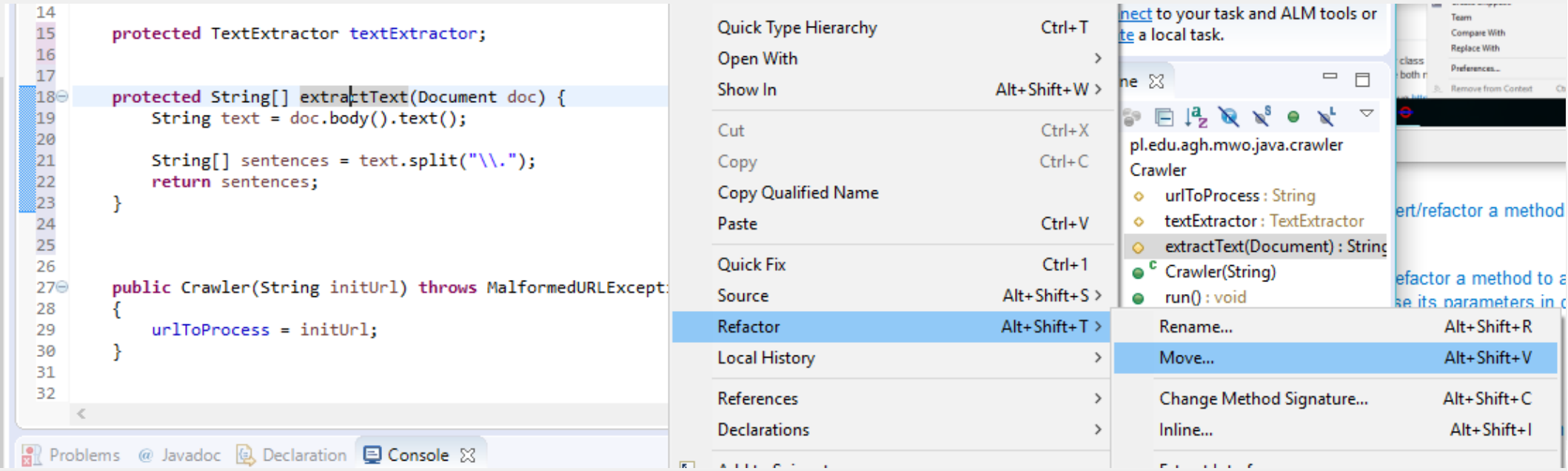
Refaktoryzacja

- Eclipse pomaga:
  - Refactor → Extract Method
  - Nowe pole w Crawler: `protected TextExtractor textExtractor;`



Refaktoryzacja

- Eclipse pomaga:
  - Quickfix [Ctrl+1] → create class 'TextExtractor'
  - Metoda extractText → Refactor → Move
- Sprawdźmy czy program działa...



Refaktoryzacja

- Kolejne klasy:
  - interface SentenceFinder
  - SentenceByWordFinder
  - SentenceByCharCountFinder
  - ConsoleResultsPrinter
- Po każdej zmianie sprawdźmy czy program działa...

Zależności

- Gdzie powstają instancje klas
  - SentenceByWordFinder
  - SentenceByCharCountFinder
  - TextExtractor
  - ConsoleResultsPrinter ?
- Crawler od nich zależy – nie powinien ich tworzyć

Wzorzec DI

- Wstrzykiwanie zależności:
  - Przez konstruktor
  - Przez setter / adder

```
protected String urlToProcess;
protected TextExtractor textExtractor;
protected ConsoleResultsPrinter consoleResultsPrinter;
protected List<SentenceFinder> sentenceFinders;

public Crawler(String urlToProcess, TextExtractor textExtractor, ConsoleResultsPrinter consoleResultsPrinter)
{
    this.urlToProcess = urlToProcess;
    this.textExtractor = textExtractor;
    this.consoleResultsPrinter = consoleResultsPrinter;
    sentenceFinders = new ArrayList<>();
}

public void addSentenceFinder(SentenceFinder sentenceFinder)
{
    sentenceFinders.add(sentenceFinder);
}
```

Wzorzec Obserwator

- Lista obiektów potencjalnie przetwarzających zdarzenia
- Crawler „nie wie” ile i jakie Findery działają

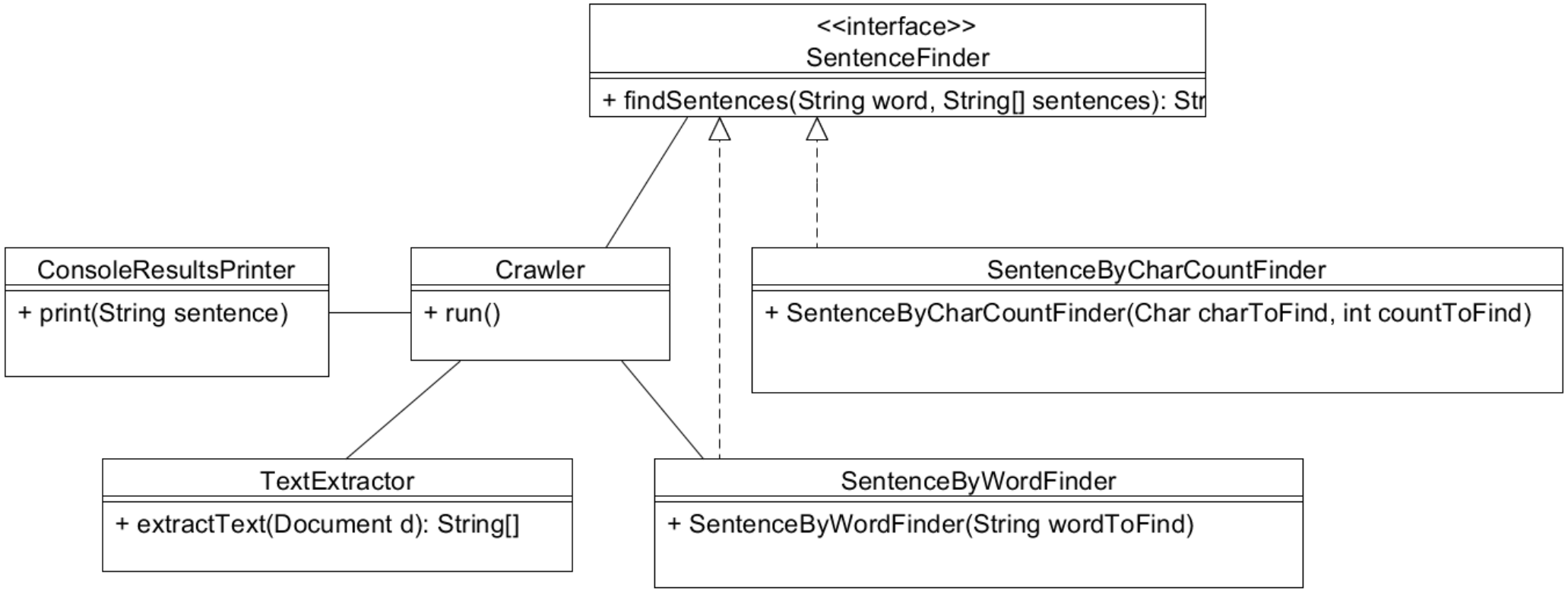
```
public void addSentenceFinder(SentenceFinder sentenceFinder)
{
    sentenceFinders.add(sentenceFinder);
}

public void run() throws IOException
{
    Document doc = null;
    doc = Jsoup.connect(urlToProcess).get();

    String[] sentences = textExtractor.extractText(doc);

    for(SentenceFinder sentenceFinder : sentenceFinders)
        consoleResultsPrinter.print(sentenceFinder.findSentences(sentences));
}
```

Testy



- Napiszmy testy!
  - Które elementy da się testować automatycznie?

Testy

- Napiszmy testy!
  - New → Source folder → name: test
  - New → JUnit test case → TextExtractorTests
- Jsoup Document w testach:

```
String html = "<html><body>Pierwsze zdanie. Drugie
zdanie. Trzecie, ostatnie zdanie.</body></html>";
Document doc = Jsoup.parse(html);

TextExtractor textExtractor = new TextExtractor();
List<String> sentences =
    textExtractor.extractText(doc);
```

TDD

- Import projektu *SimpleCrawler-2*
- Czy testy przechodzą?

  → Poprawiamy implementację!

TDD

- Test-Driven Development
  - Najpierw piszemy testy, które specyfikują działanie klasy
  - Później implementujemy klasę, aż jest zielono...

- Dodajmy do projektu funkcjonalność szukania zdań o długości większej niż *m* znaków
  - Tworzymy klasę SentenceByLengtFinder implementującą SentenceFinder
  - Tworzymy i implementujemy testy
  - Implementujemy findSentences by przeszły testy

Klasy anonimowe

- Dodawanie nowych sposobów wyszukiwania wymaga dodania nowej klasy
  - Nowy plik, dużo kodu opakowania, mało logiki
  - Klasa jest użyta tylko raz
- Rozwiązanie: klasa anonimowa:

```
public static void main(String[] args) throws IOException
{

    Crawler c = new Crawler("http://www.onet.pl", new TextExtractor(), new ConsoleResultsPrinter());

    c.addSentenceFinder(new SentenceByWordFinder("szpital"));
    c.addSentenceFinder(new SentenceByCharCountFinder(10, 'd'));

    c.addSentenceFinder(new SentenceFinder(){
        @Override
        public List<String> findSentences(List<String> sentences) {
            List<String> result = new ArrayList<>();
            for (String sentence : sentences)
                if (sentence.length() > 300)
                    result.add(sentence);
            return result;
        }
    });
```

Klasy anonimowe

- Dodajmy tą metodą kolejny SentenceFinder, który będzie szukał zdań zaczynających się na literkę J

```
public static void main(String[] args) throws IOException
{

    Crawler c = new Crawler("http://www.onet.pl", new TextExtractor(), new ConsoleResultsPrinter());

    c.addSentenceFinder(new SentenceByWordFinder("szpital"));
    c.addSentenceFinder(new SentenceByCharCountFinder(10, 'd'));

    c.addSentenceFinder(new SentenceFinder(){
        @Override
        public List<String> findSentences(List<String> sentences) {
            List<String> result = new ArrayList<>();
            for (String sentence : sentences)
                if (sentence.length() > 300)
                    result.add(sentence);
            return result;
        }
    });
```

- Można jeszcze krócej – wyrażenia Lambda
- Podejście funkcyjne
  - Funkcja jako „first class citizen”
  - Funkcje wyższego rzędu
  - Operacje map, reduce, filter, …
  - Pure-functional - efekty uboczne
  - Współbieżność
  - Języki funkcyjne

- Można jeszcze krócej – wyrażenia Lambda
- Składnia:

  (parametry, funkcji) -> {ciało; funkcji}

```
c.addSentenceFinder( (sentences) -> {
        List<String> result = new ArrayList<>();
        for (String sentence : sentences)
            if (sentence.charAt(0) == 'J')
                result.add(sentence);
        return result;
    }
);
```

- λ są implementacjami interfejsów, które mają tylko jedną metodę – Java wymyśli którego...

- Dodajmy tą metodą kolejny SentenceFinder, który będzie szukał zdań o liczbie **słów** większej niż 30

```
if (sentence.split(" ").length ....
```

```
c.addSentenceFinder( (sentences) -> {
        List<String> result = new ArrayList<>();
        for (String sentence : sentences)
            if (sentence.charAt(0) == 'J')
                result.add(sentence);
        return result;
    }
);
```

Klasy anonimowe, λ, ...

- Napiszmy testy...

Strumienie

- Mechanizmy przetwarzania elementów kolekcji

```
c.addSentenceFinder( (sentences) -> {
    List<String> result = new ArrayList<>();
    for (String sentence : sentences)
        if (sentence.split(" ").length > 70)
            result.add(sentence);
    return result;
    }
);
```

```
c.addSentenceFinder(sentences ->
    sentences.stream()
        .filter(sentence -> sentence.split(" ").length > 70)
        .collect(Collectors.toList()));
```

Strumienie

- Dodajmy tą metodą kolejny SentenceFinder, który będzie szukał zdań krótszych niż 10 znaków

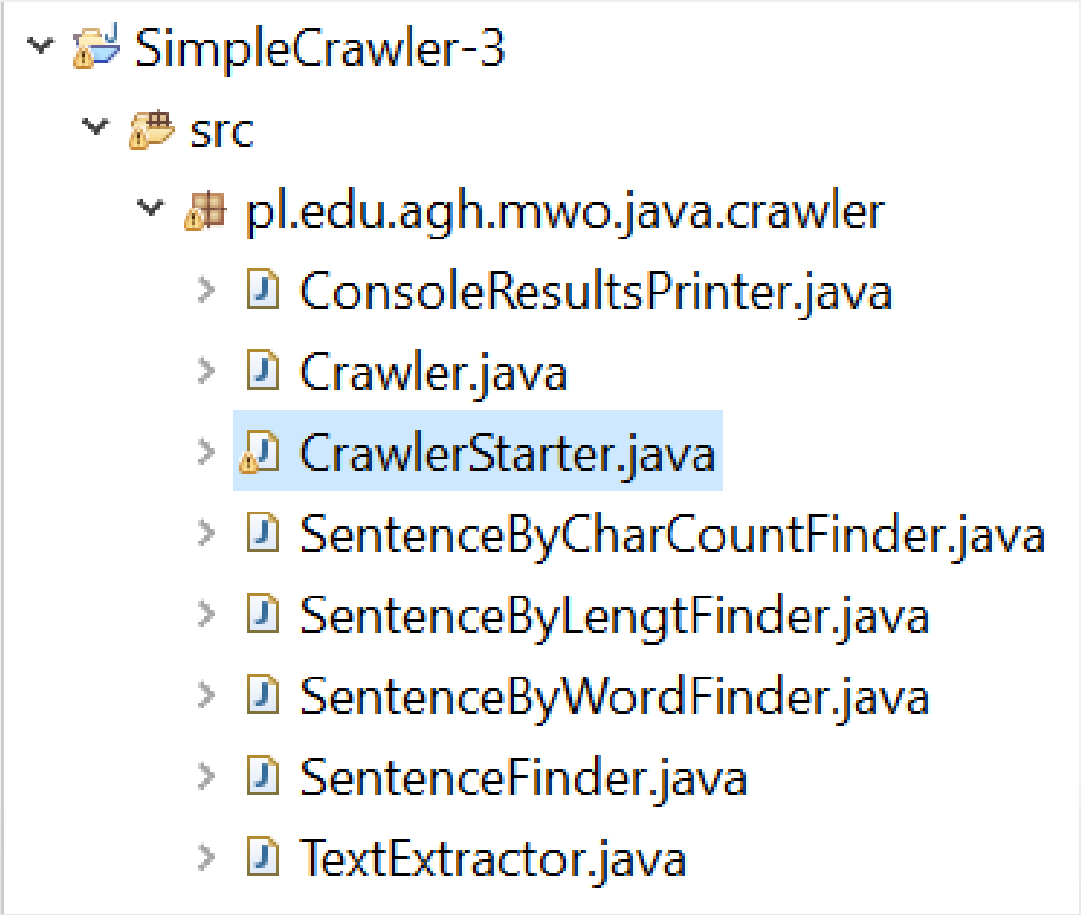
```
c.addSentenceFinder(sentences ->
    sentences.stream()
        .filter(sentence -> sentence.split(" ").length > 70)
        .collect(Collectors.toList()));
```

Strumienie - operacje

- .map(λ)
- .reduce(initVal, λ)
- .sum()
- .forEach()
- .parallelStream()

Przetwarzanie wielu stron

- Projekt SimpleCrawler-3



Przetwarzanie wielu stron

- Dodajmy przetwarzanie wszystkich stron z zadanej listy:
  - W pętli: `for (String url : urlsToProcess)`
  - albo używając: `urlsToProcess.stream().forEach(`

Ile to trwało?

- Mierzymy czas wykonania zadania:
  - Przed:

```
long startMillis = System.currentTimeMillis();
```

  - Po:

```
System.out.println("\n took: "+(currentMillis-startMillis)+"ms");
```

Ile to trwało?

- Mierzymy czas wykonania zadania:
  - Przed:

```
long startMillis = System.currentTimeMillis();
```

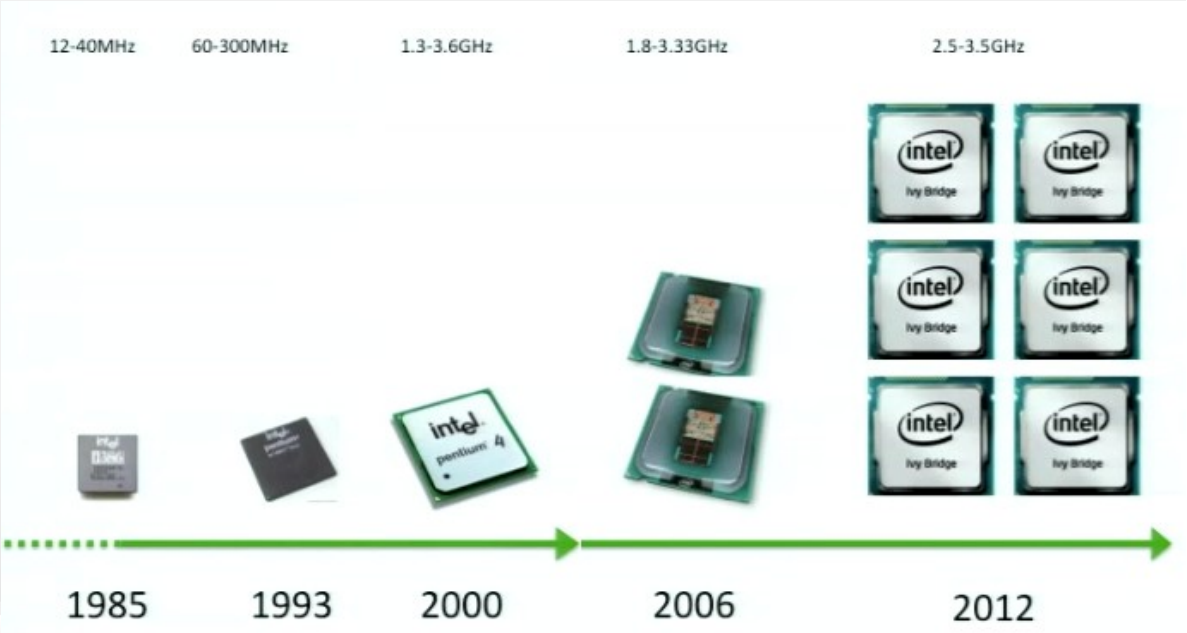
  - Po:

```
System.out.println("\n took: "+(currentMillis-startMillis)+"ms");
```

Czy da się szybciej?
Czy komputer był w tym czasie zajęty?

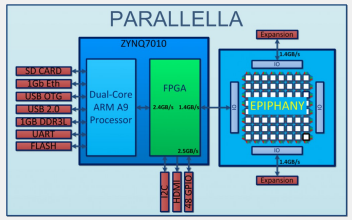
Wątki

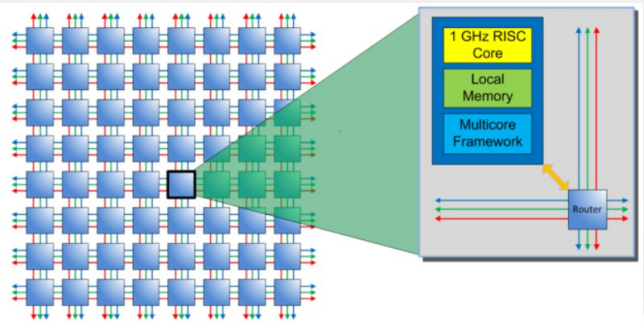
- Przełączanie kontekstu (1995)
  - Współbieżność w ramach jednego komputera
- Wiele rdzeni (2005)
  - Równoległość w ramach jednego komputera

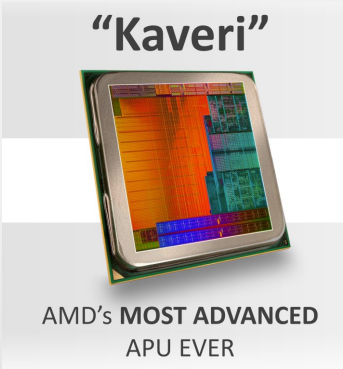


Architektury procesorów

- Przyspieszanie zegara nie jest aktualnie możliwe i nie jest opłacalne energetycznie
- Dodawanie kolejnych rdzeni jest tanie i pozwala na uzyskiwanie teoretycznego zwiększania mocy procesorów







Wątki w Javie

- Implementacja interfejsu `Runnable`
- Rozszerzenie klasy `Thread`
- Klasa musi posiadać metodę run(), która będzie uruchomiona w nowym wątku
- Wątki mogą mieć dostęp do wspólnych zmiennych → shared memory concurrency model
- Współdzielone zmienne są główną metodą komunikacji między wątkami
- Wszystkie wątki mają dostęp do usług OS

Wydajność crawlowania...

- Sprawdźmy czy procesor jest zajęty...
  - Większość czasu to IO – pobieranie stron
- Strony można przetwarzać równolegle, w wątkach
  - Klasa Crawler może dziedziczyć z Thread
  - Trzeba utworzyć kilka wątków, kilka instancji klasy Crawler
  - Każdy trzeba uruchomi:
    - Zmieniamy `c.run();` na `c.start();`

Pomiar czasu wykonania

- Jaki jest czas wykonania zadania?
- Wątek zaczyna się na żądanie
  - I kończy się bez hałasu gdy wychodzi z run()...
- Oczekiwanie na wątek:

```
ArrayList <Crawler> crawlers = new ArrayList<Crawler>();
for (String url : urlsToProcess)
{
    Crawler c = new Crawler(...);
    crawlers.add(c);
    c.run();
}

for (Crawler c : crawlers)
    c.join();
```

Wątki w praktyce

- Bardzo rzadko zdarza się by problemów do rozwiązania było kilka czy kilkaset
- Co jeśli adresów do przetworzenia będzie 100.000 czy 1.000.000 czy 256.000.000 ?
  - Tworzenie wątków jest kosztowne

→ **Pula wątków**

  - Wątek to „worker”
  - Bierze kolejne zadanie gdy skończy poprzednie

Zarządzanie adresami

- Nowa klasa … UrlListsManager
  - Zbiór adresów do przetworzenia
  - Metoda do dodawania nowych adresów
  - Metoda do pobierania kolejnego adresu do przetworzenia
- Przenieśmy tu też warunek stopu całego programu
  - Jeśli pobranie kolejnego adresu zwróci null, wątek Crawlera się kończy.
- UrlListsManager jest kolejną zależnością Crawlera

Race conditions

1

```
if (urlsToProcess.size() > 0)
{
    String urlToProcess = urlsToProcess.iterator().next();
    urlsToProcess.remove(urlToProcess);
    return urlToProcess;
}
```

3

```
if (urlsToProcess.size() > 0)
{
    String urlToProcess = urlsToProcess.iterator().next();
    urlsToProcess.remove(urlToProcess);
    return urlToProcess;
}
```

2

Race conditions

1

```
if (urlsToProcess.size() > 0)
{
    String urlToProcess = urlsToProcess.iterator().next();
    urlsToProcess.remove(urlToProcess);
    return urlToProcess;
}
```

3

4: BUM!

```
if (urlsToProcess.size() > 0)
{
    String urlToProcess = urlsToProcess.iterator().next();
    urlsToProcess.remove(urlToProcess);
    return urlToProcess;
}
```

2

Synchronizacja

- Metoda synchronizowana akceptuje tylko jeden wątek wykonujący jej zawartość

```
public synchronized String getUrlToProcess()
{ .... }
```

- Fragment metody również może być tak zabezpieczany

```
public String getUrlToProcess() {
    synchronized(this) { .... } ...
}
```

Tadam!

- Wielowątkowy Crawler
- Z opcjami przeszukiwania według:
  -
  -
- Z możliwością drukowania wyników
- Z możliwością pomiaru czasu
- Z testami

Całkiem nieźle :)