Borland Developer Days 2004
2-3 czerwca 2004

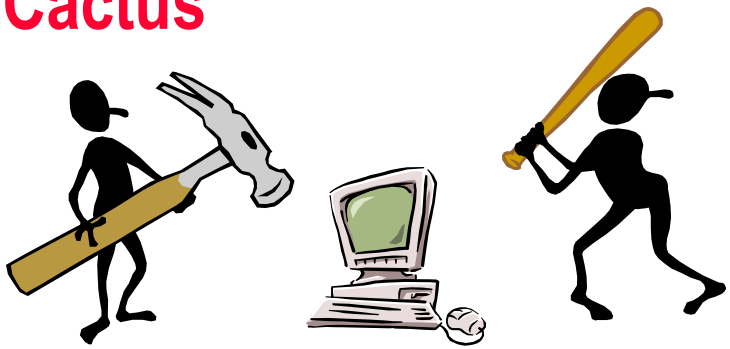
Testowanie aplikacji Java Servlets

Bartosz Walter

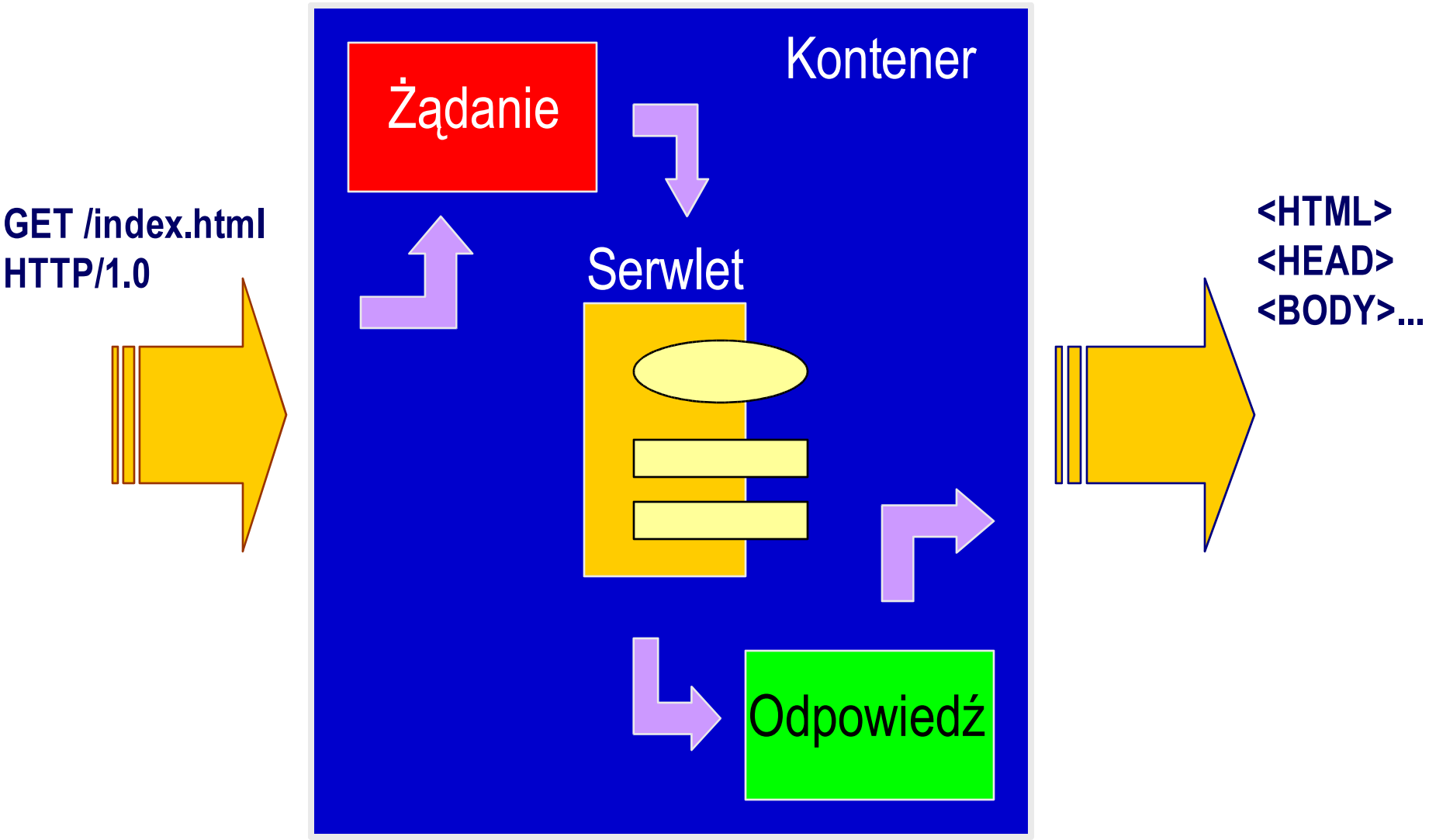
mailto: Bartek.Walter@man.poznan.pl

Agenda

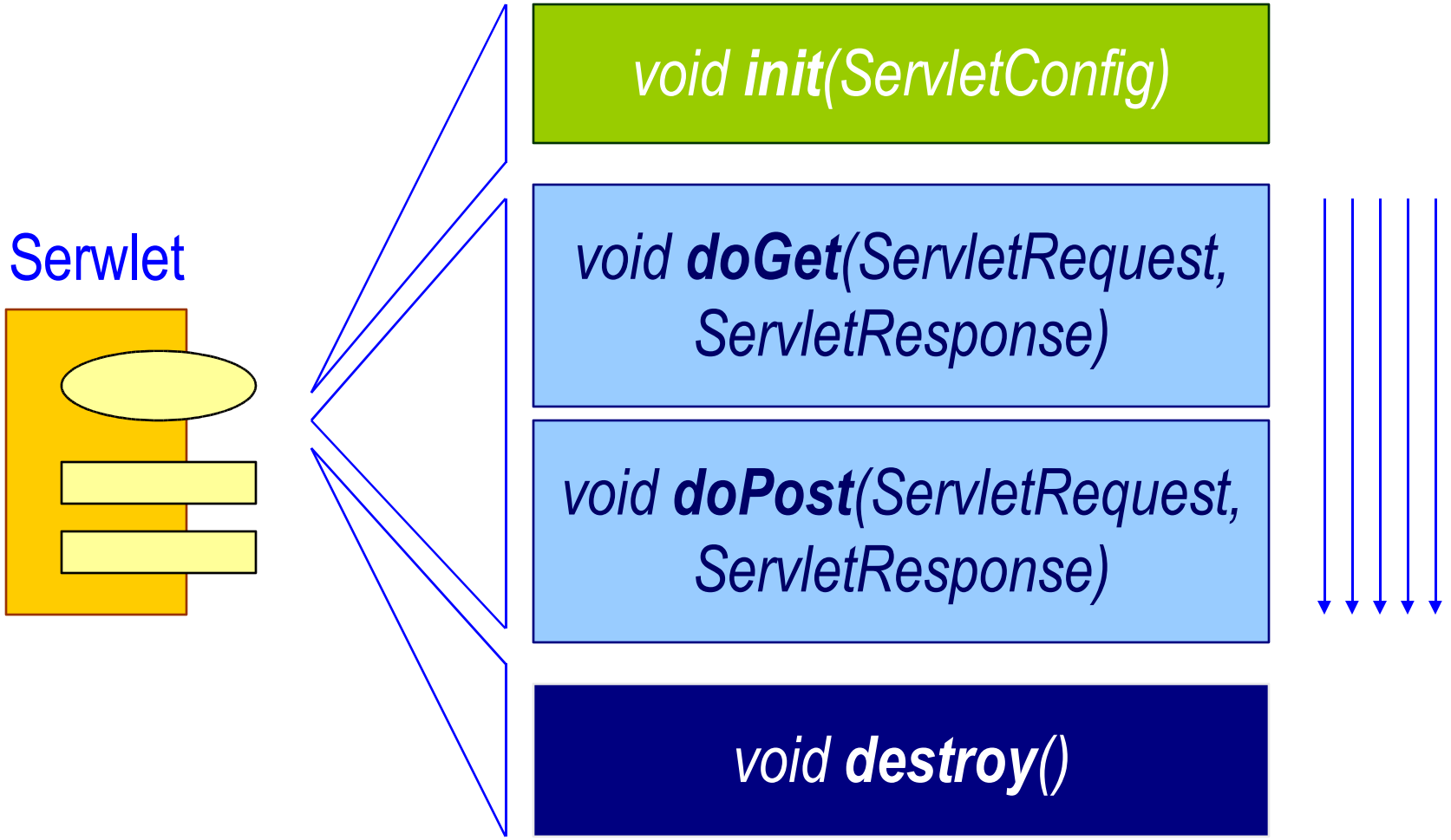
- Aplikacje Java Servlets™
- Jak testować aplikacje internetowe?
  - W małej skali, czyli testy jednostkowe
  - Całymi funkcjami, czyli testy funkcjonalne
  - Jednak API, czyli obiekty zastępcze
  - Czy można mieć wszystko, czyli Cactus



Serwlet i jego środowisko



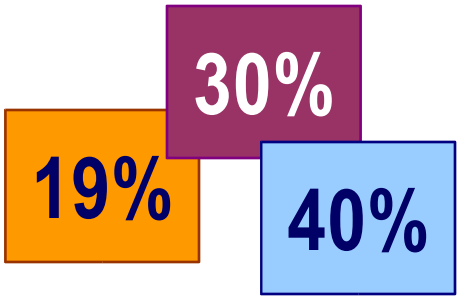
Życie serwleta

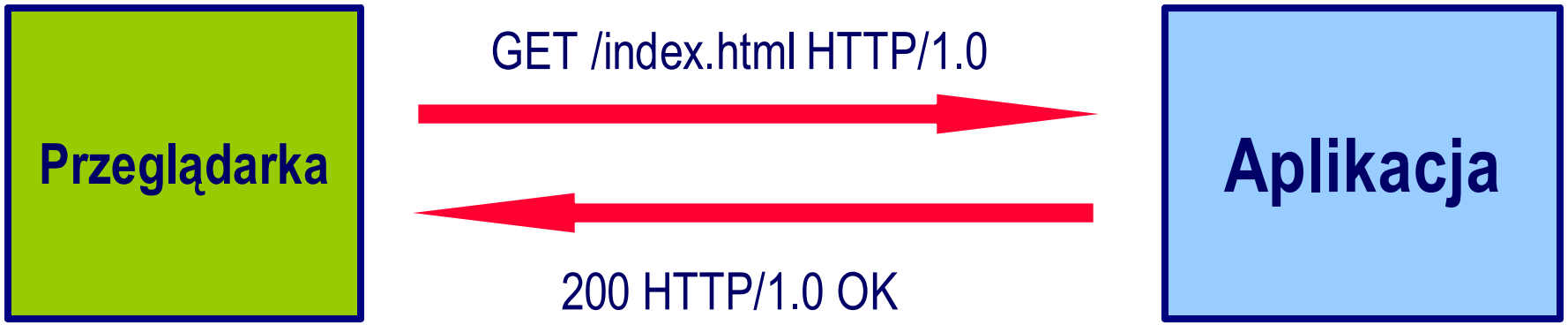


`javax.servlet.http.HttpServlet`

Serwlet podatkowy

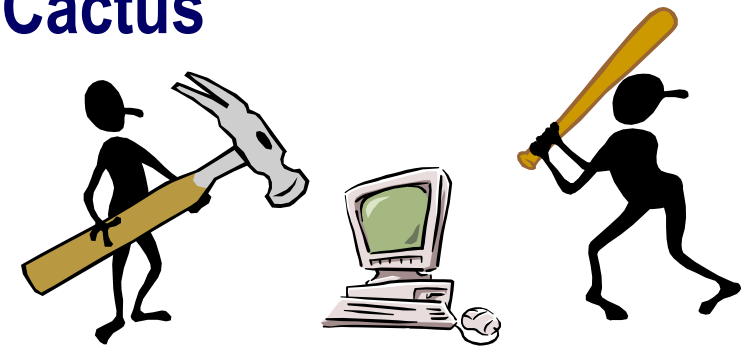
dochód: 100	podatek: 0
dochód : 1000	podatek: 89,28
dochód : -100	podatek: 0
dochód : xxx	Niepoprawna liczba





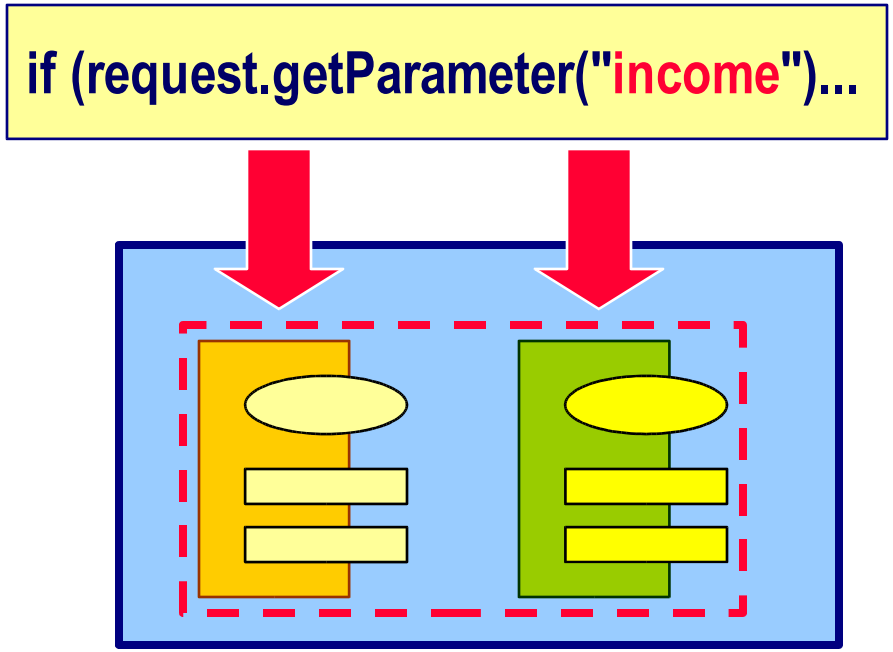
Agenda

- Aplikacje Java Servlets™
- Jak testować aplikacje internetowe?
  - W małej skali, czyli testy jednostkowe
  - Całymi funkcjami, czyli testy funkcjonalne
  - Jednak API, czyli obiekty zastępcze
  - Czy można mieć wszystko, czyli Cactus

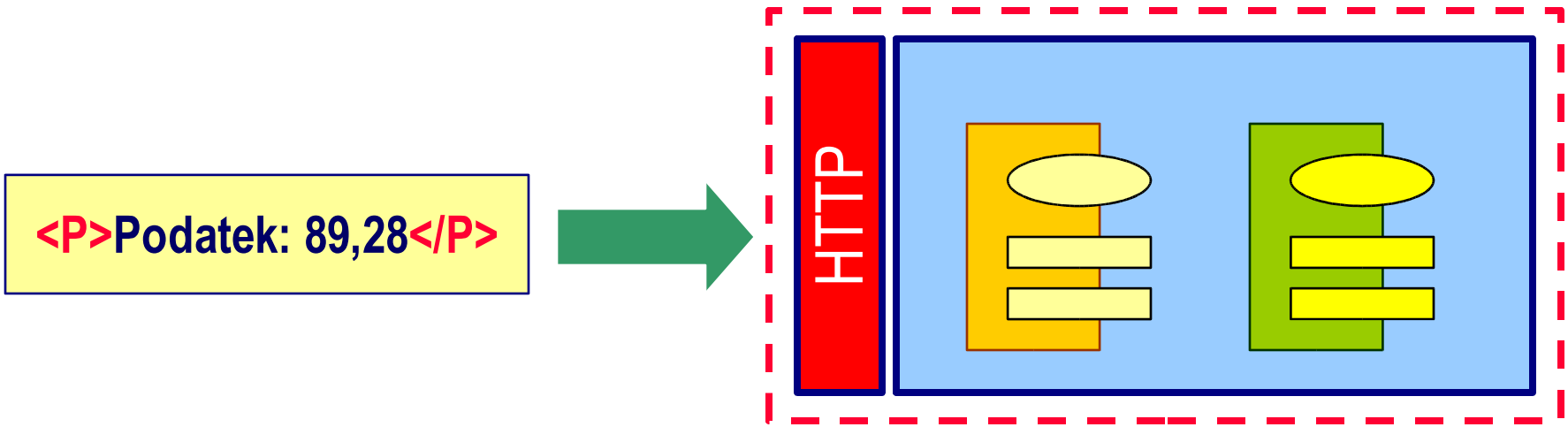


Testowanie aplikacji internetowych

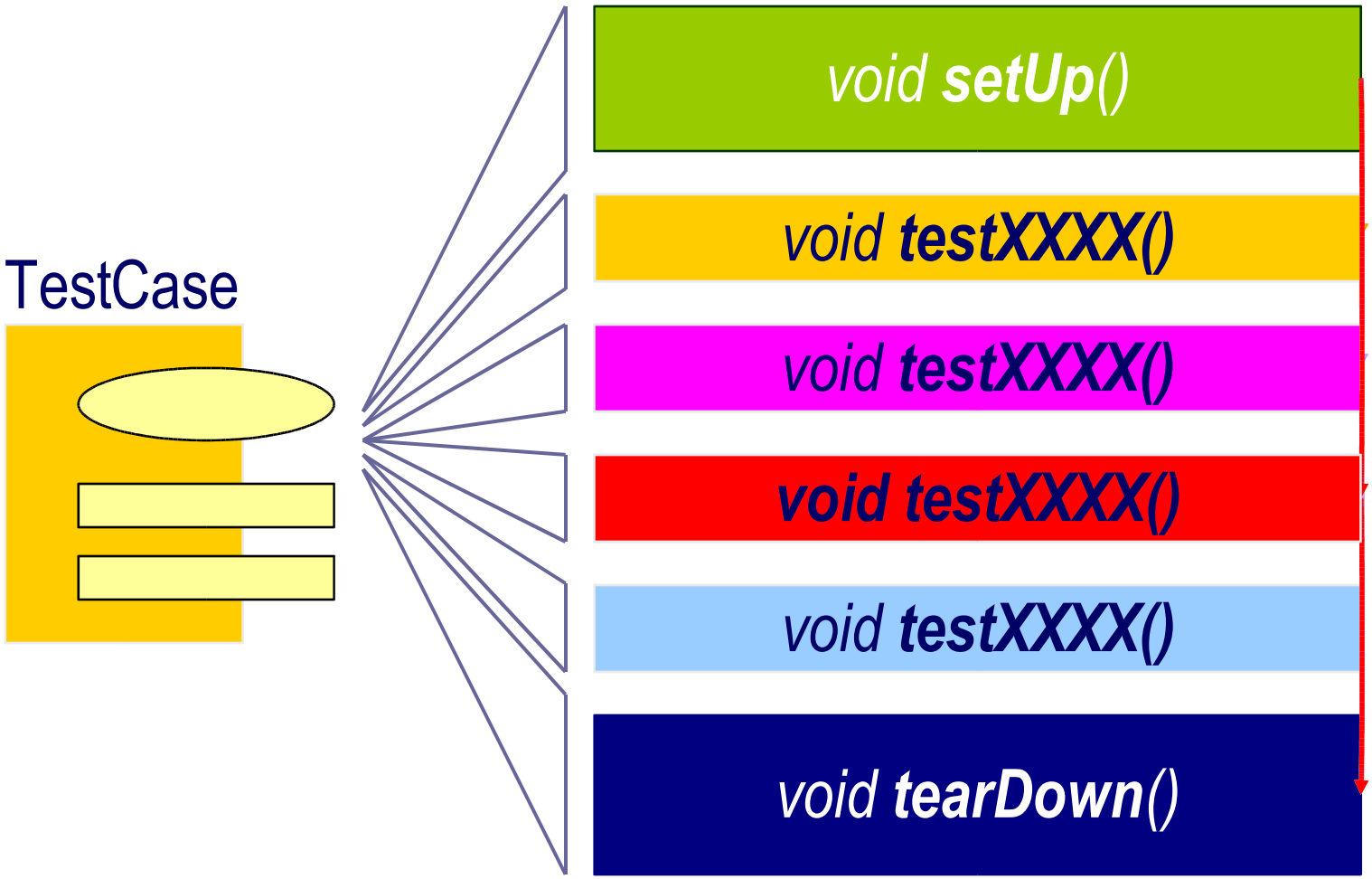
- **Co jest obiektem testowania?**
  - **Pojedyncze obiekty aplikacji**



- **Co jest obiektem testowania?**
  - **Pojedyncze obiekty aplikacji**
  - **Aplikacja jako całość**

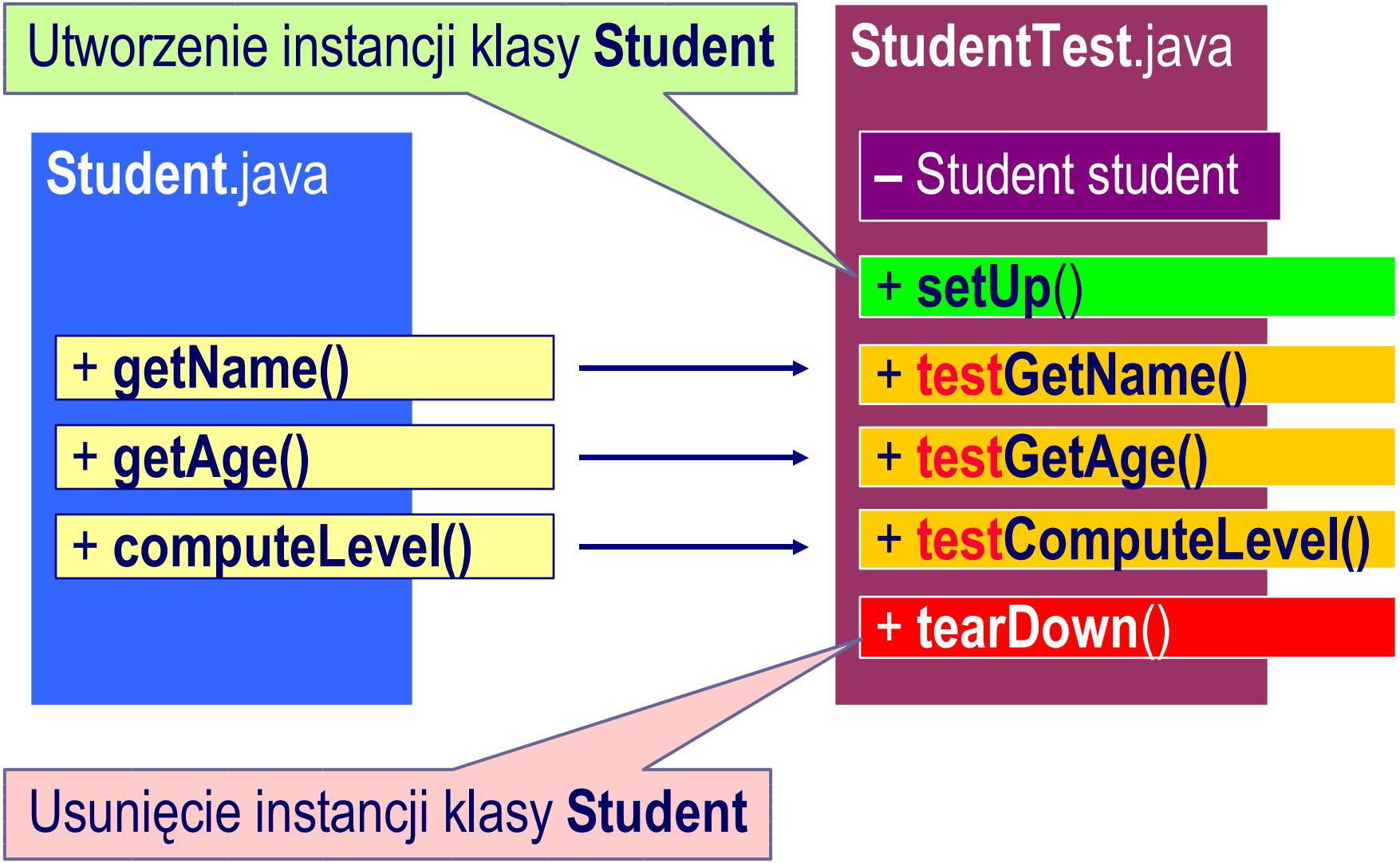


Przypadek testowy xUnit



`junit.framework.TestCase`

Klasa i jej przypadek testowy



Testowanie jednostkowe metody

```
Student student = null;
```

void ***setUp()***

```
public void setUp() {
   student = new Student();
   student.setAge(20);
   student.setName(„Janek”);
}
```

void ***testGetAge()***

```
public void testWiek()
throws Exception {
   int age = student.getAge();

   assertEquals(age, 20);
}
```

void ***tearDown()***

```
public void tearDown() {
   student = null;
}
```

Testowanie jednostkowe aplikacji internetowych

void ***testDoGet()***

```
testDoGet(request, response){

   ....???......

}
```

void ***testDoPost()***

```
testDoPost(request, response){

   ....???......

}
```

void ***testComputeTax()***

```
testComputeTax(int income) {

   int tax = computeTax(100);
   assertEquals(0, tax);
}
```

Testowanie jednostkowe aplikacji internetowych

void ***testDoGet()***

brak żądania i odpowiedzi

void ***testDoPost()***

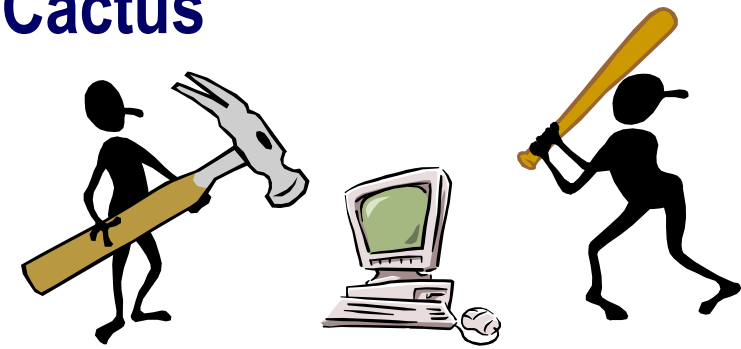
brak żądania i odpowiedzi

void ***testComputeTax()***

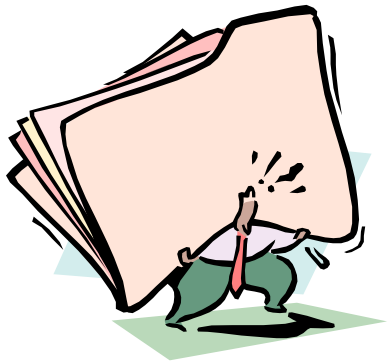
```
testComputeTax(int income) {

  int tax = computeTax(100);
  assertEquals(0, tax);
}
```

Agenda

- Aplikacje Java Servlets™
- Jak testować aplikacje internetowe?
  - W małej skali, czyli testy jednostkowe
  - Całymi funkcjami, czyli testy funkcjonalne
  - Jednak API, czyli obiekty zastępcze
  - Czy można mieć wszystko, czyli Cactus



Perspektywa użytkownika: testowanie funkcjonalne



perl

ASP

PHP

Przeglądarka

GET /index.html HTTP/1.0

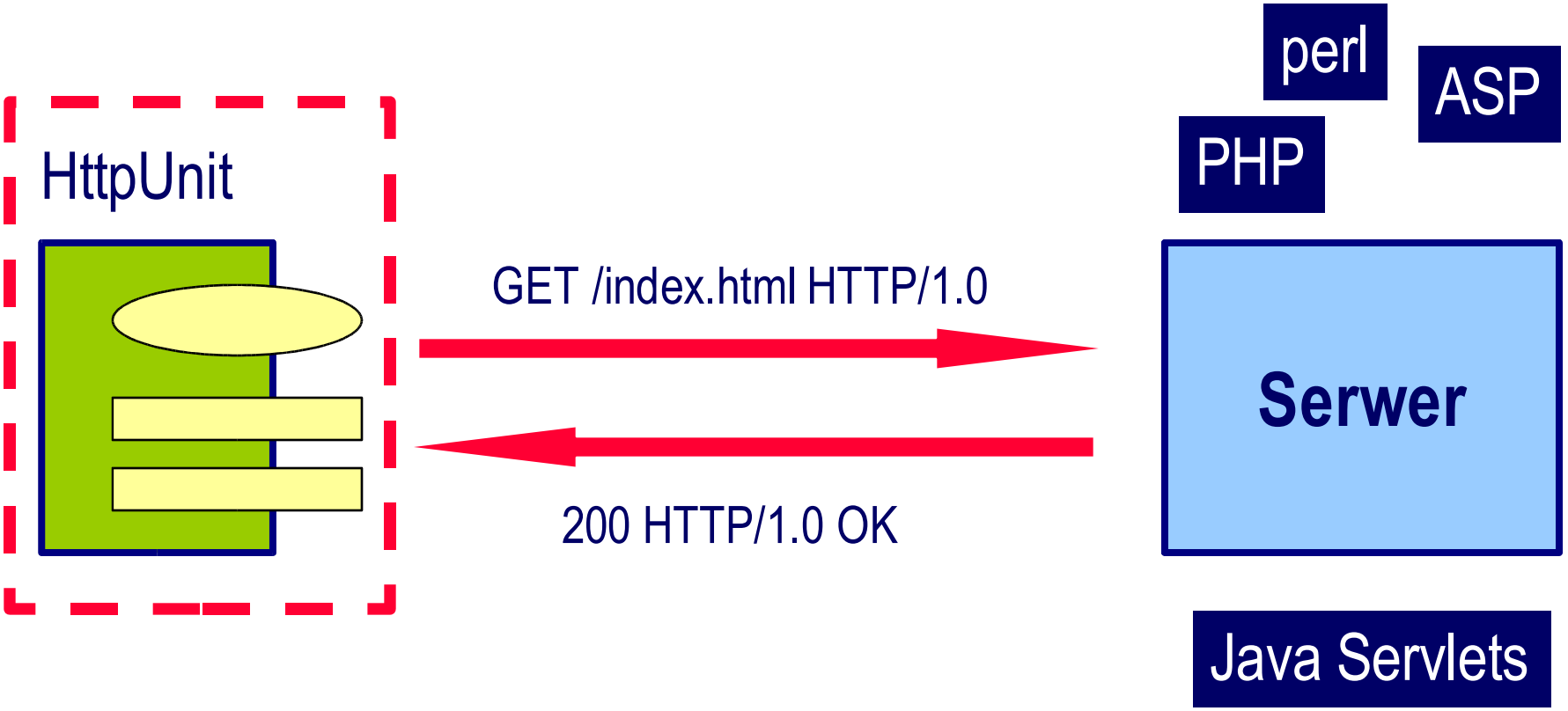
200 HTTP/1.0 OK

Serwer

Java Servlets

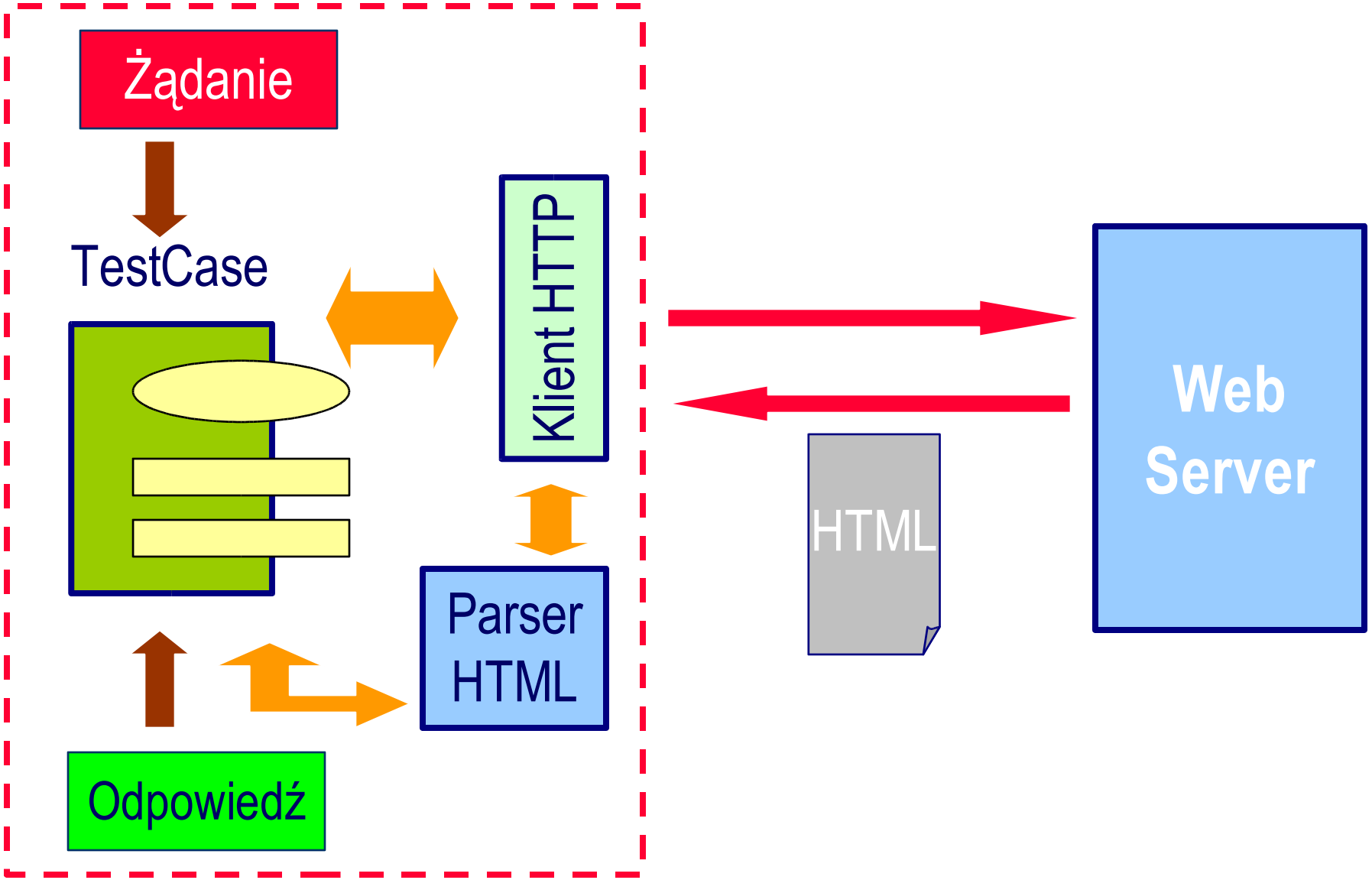
Przeglądarka ma dostęp jedynie do protokołu HTTP

Perspektywa użytkownika: testowanie funkcjonalne



Zamiast testera z przeglądarką użyjmy automatu...

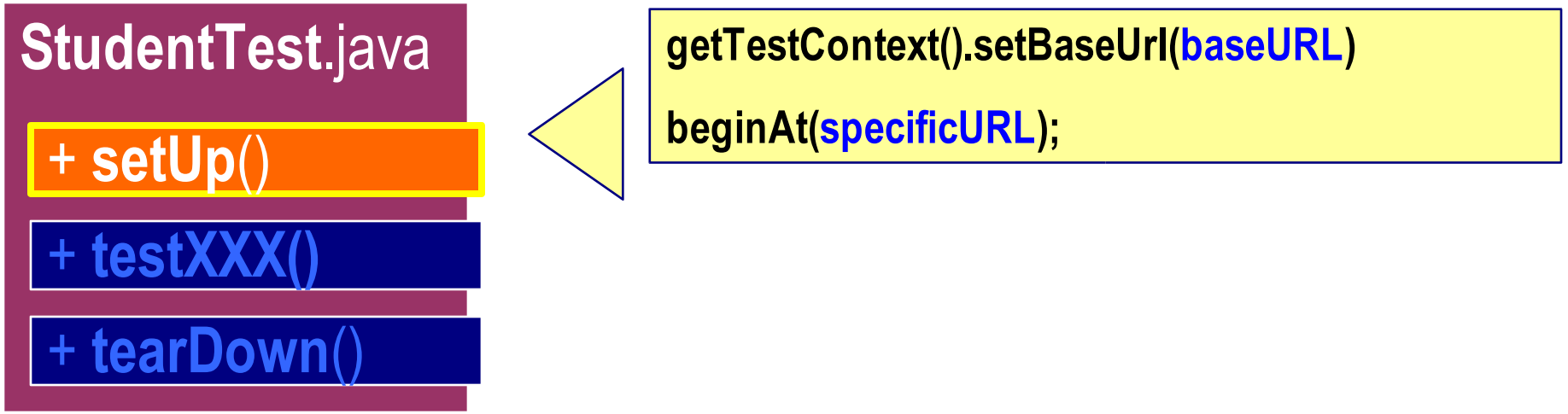
Architektura HttpUnit i jWebUnit



Przypadek testowy jWebUnit

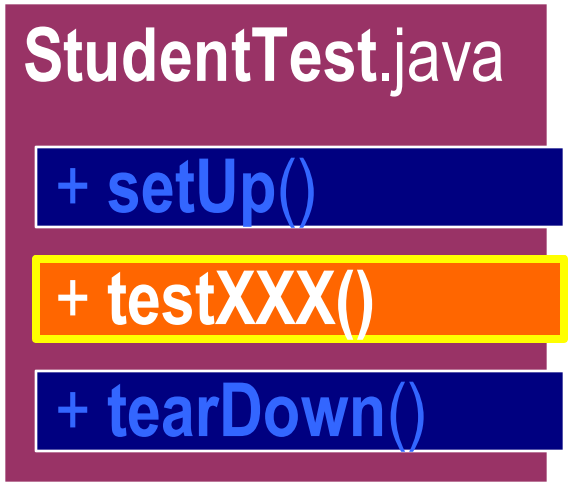
Klient

Wskazanie adresu aplikacji



Przypadek testowy jWebUnit

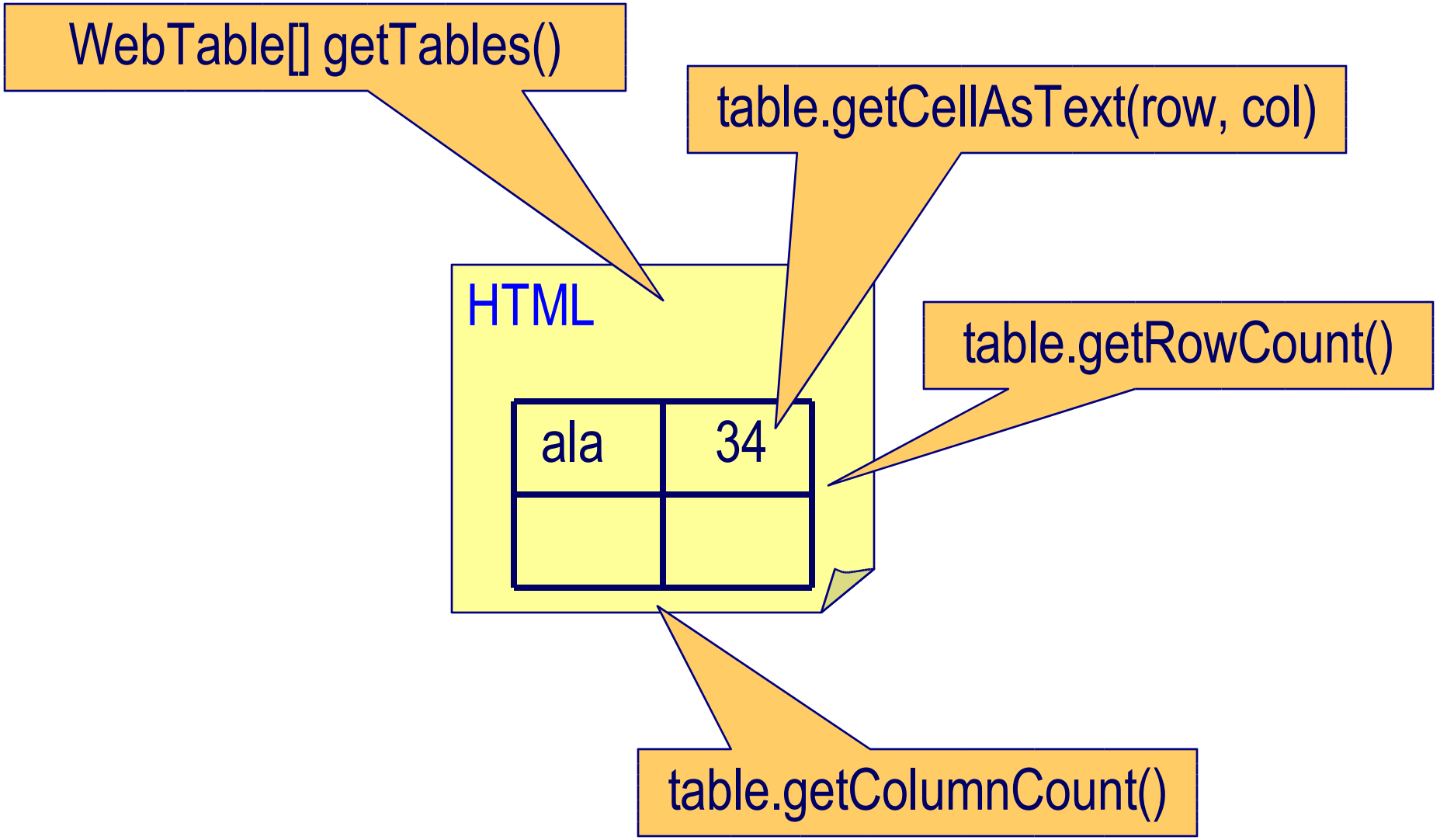
Klient



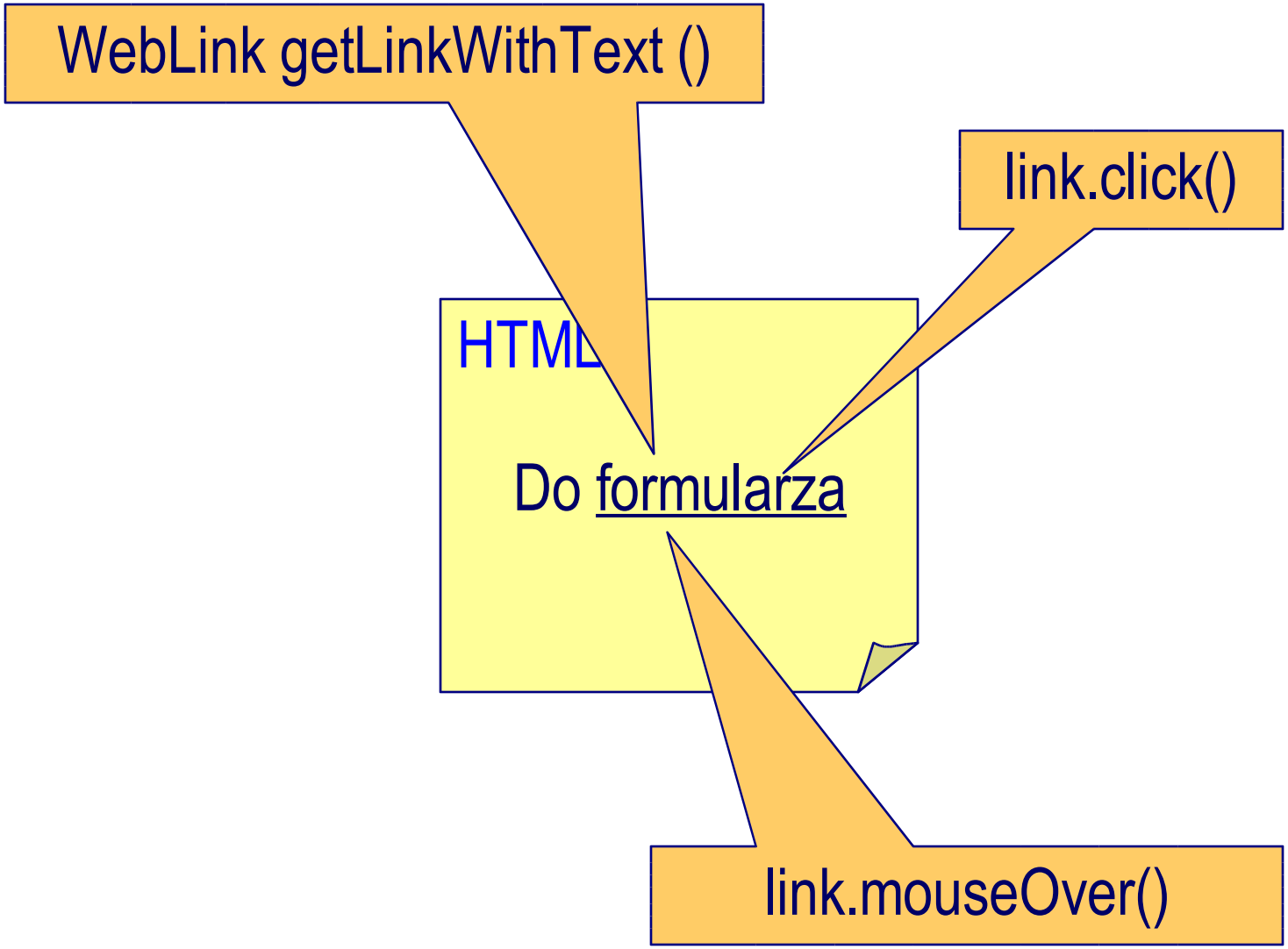
Nawigacja i asercje

```
clickLinkWithText("Formularz");
setFormElement("id", "12345");
click();

assertTextPresent("Wiek: 20 lat");
assertLinkWithTextPresent("Powrót");
```

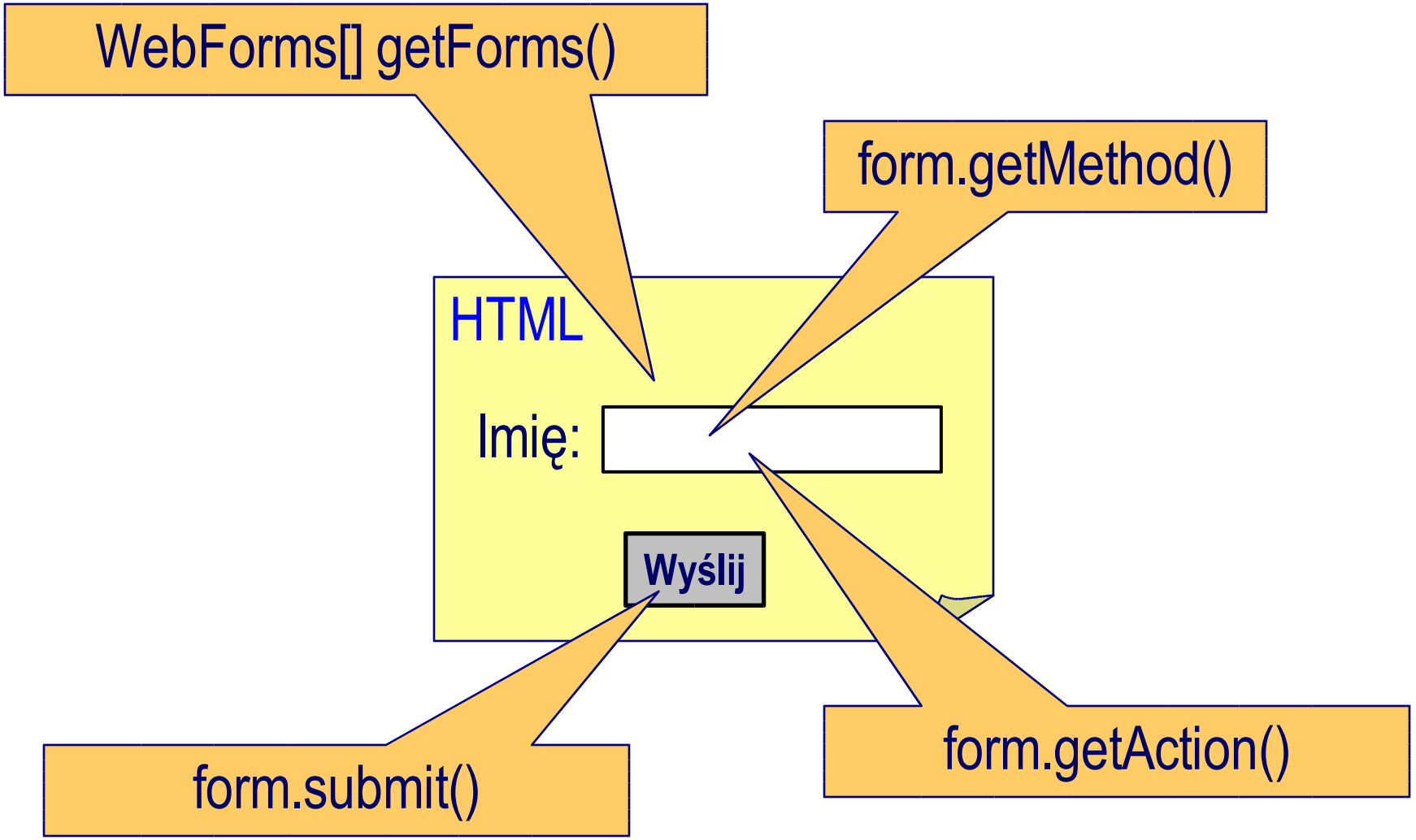
Nawigacja: Tabele



Nawigacja: Linki



Nawigacja: Formularze



Asercje jWebUnit

OGÓLNE

- assertFramePresent (ID)
- assertWindowPresent (ID)
- assertTitleEquals(oczekiwany)
- assertTextPresent(tekst)
- assertLinkPresent(ID)
- xt(ID)
- age(ID)

FORMULARZE

- assertFormPresent (ID)
- assertFormElementPresent (ID)
- assertFormElementPres
- assertFormElementEqua
- assertCheckboxSelected
- assertButtonPresent(ID)
- assertRadioOptionPrese
- assertOptionsEqual(nazwa, wartości[])

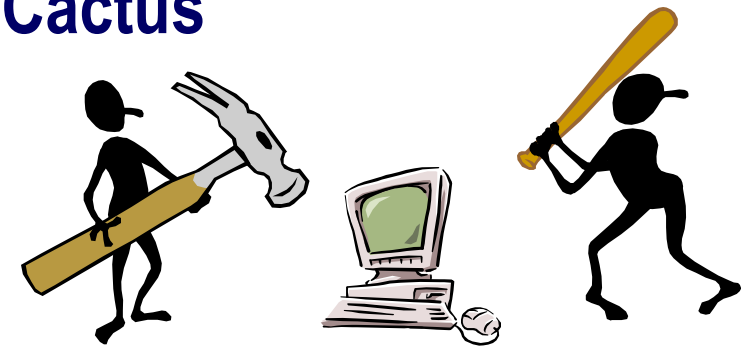
TABELE

- assertTablePresent (ID)
- assertTableEquals(ID, tabela[][])
- assertTextInTable (ID, tekst)
- assertTableRowsEqual(ID, wierszPoczątkowy, oczekiwana)

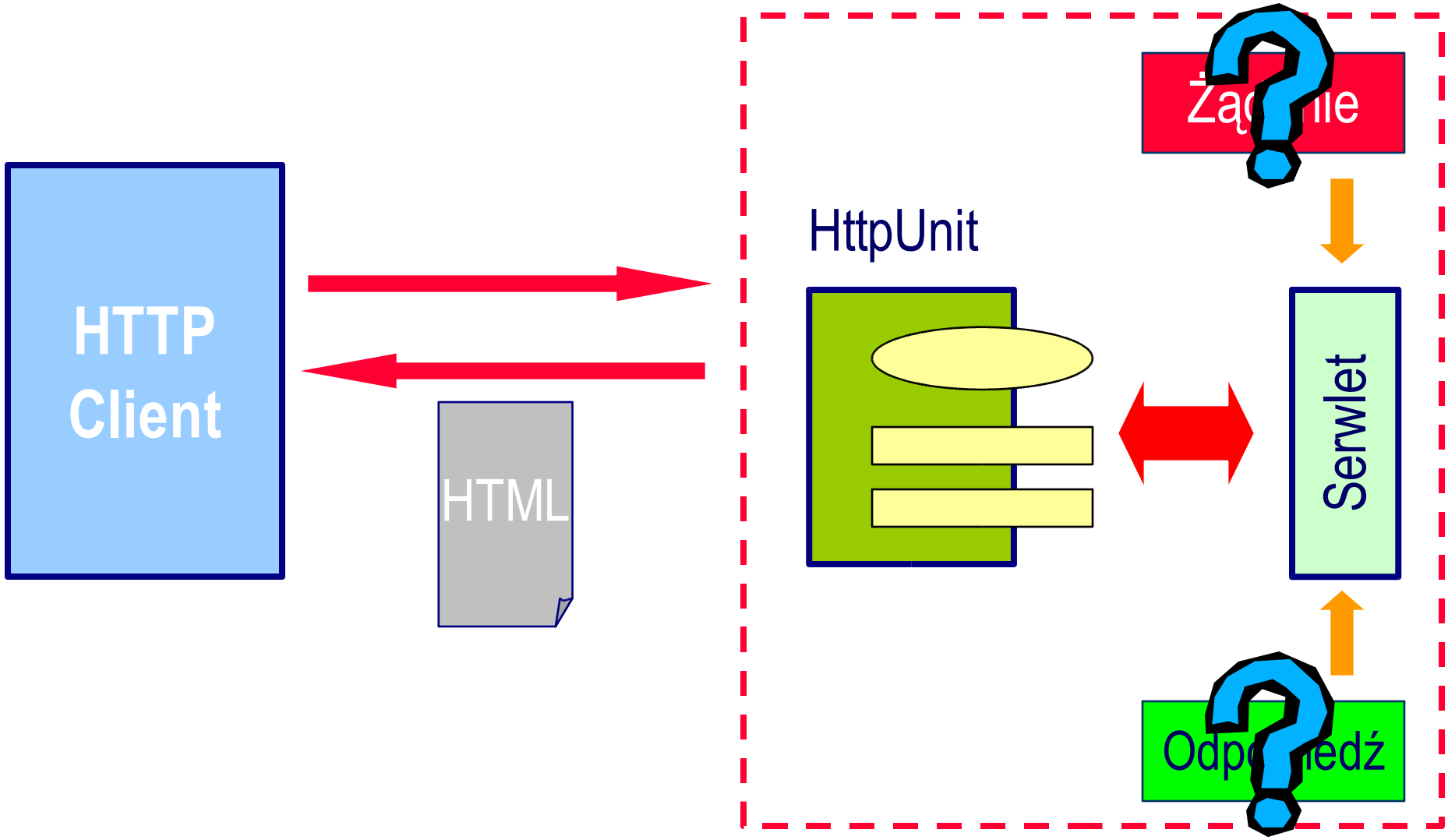
Przykład: serwlet podatkowy

Agenda

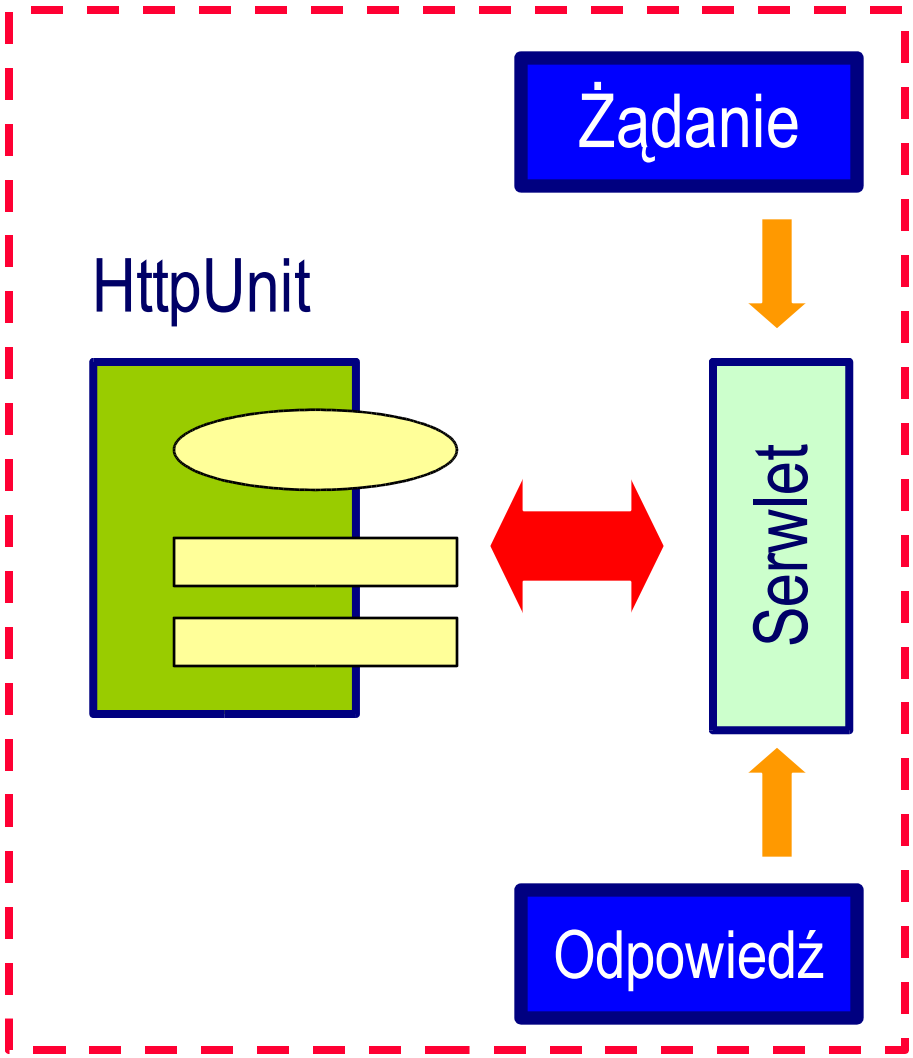
- Aplikacje Java Servlets™
- Jak testować aplikacje internetowe?
  - W małej skali, czyli testy jednostkowe
  - Całymi funkcjami, czyli testy funkcjonalne
  - Jednak API, czyli obiekty zastępcze
  - Czy można mieć wszystko, czyli Cactus



Architektura testów z wykorzystaniem Mock Objects



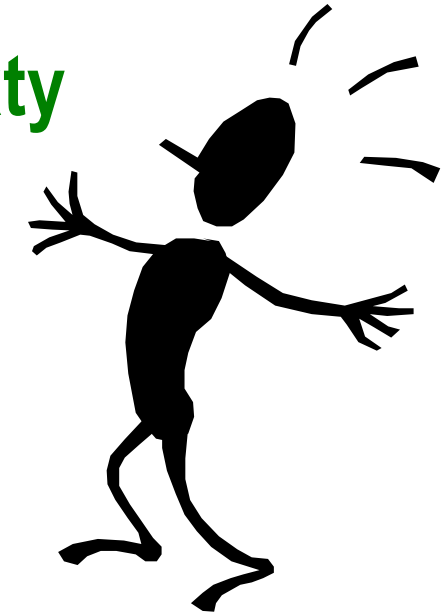
Architektura testów z wykorzystaniem Mock Objects



Obiekty zastępcze

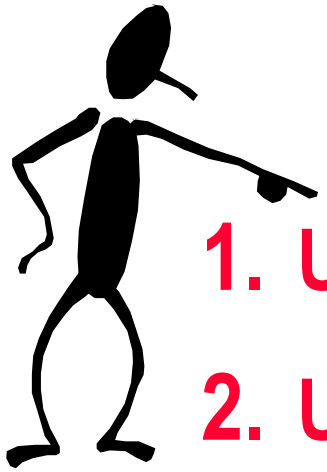
Obiekt zastępczy (*mock object*)

- naśladuje obiekt, który zastępuje
- posiada minimalną funkcjonalność
- obserwuje w jaki sposób inne obiekty wywołują jego metody
- porównuje faktyczne zachowanie z oczekiwanym



source: www.mockobjects.com

Wykorzystanie obiektu zastępczego



1. Utwórz instancje obiektów zastępczych
2. Ustaw wartości parametrów tych obiektów
3. Zdefiniuj oczekiwane zachowanie obiektów na nich operujących
4. Wywołaj testowany kod przekazując obiekty zastępcze jako parametry
5. Zweryfikuj spójność obiektów zastępczych

Obiekty zastępcze: inicjalizacja

```
MockHttpServletRequest mockRequest = null;

MockHttpServletResponse mockResponse = null;

MockServletConfig mockConfig = null;

MyServlet servlet = null;


public void setUp() {

    mockRequest = new MockHttpServletRequest();

    mockResponse = new MockHttpServletResponse();

    mockConfig = new MockServletConfig();

    servlet = new MyServlet();

}
```

Obiekty zastępcze: test

```
public void testXXX() {
   mockRequest.setupAddParameter("imie", "Ala");
   mockRequest.setupAddParameter("wiek", "34");
   mockRequest.setExpectedAttribute("loggedIn", "true");
   mockResponse.setExpectedOutput(
      "<html><head/><body>A GET request</body></html>");

   servlet.init(mockConfig);
   servlet.doGet(mockRequest, mockResponse);

   assertEquals("Atrybut nie został ustawiony", "true",
      mockRequest.getAttribute("loggedIn"));
   mockRequest.verify();
   mockResponse.verify();
}
```

Weryfikacja wyników

Asercje

sprawdzają, czy testowane obiekty dają spodziewane wyniki

Weryfikacje

sprawdzają, czy testowane obiekty poprawnie porozumiewają się z obiektami zastępczymi

- liczba wywołań metod
- obecność parametru, atrybutu, sesji etc.

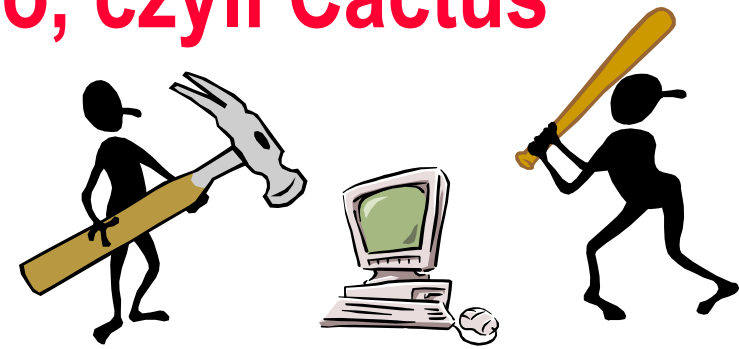
Obiekty zastępcze: test

```
public void testXXX() {
   mockRequest.setupAddParameter("imie", "Ala");
   mockRequest.setupAddParameter("wiek", "34");
   mockRequest.setExpectedAttribute("loggedIn", "true");
   mockResponse.setExpectedOutput(
      "<html><head/><body>A GET request</body></html>");

   servlet.init(mockConfig);
   servlet.doGet(mockRequest, mockResponse);

   assertEquals("Atrybut nie został ustawiony", "true",
      mockRequest.getAttribute("loggedIn"));
   mockRequest.verify();
   mockResponse.verify();
}
```

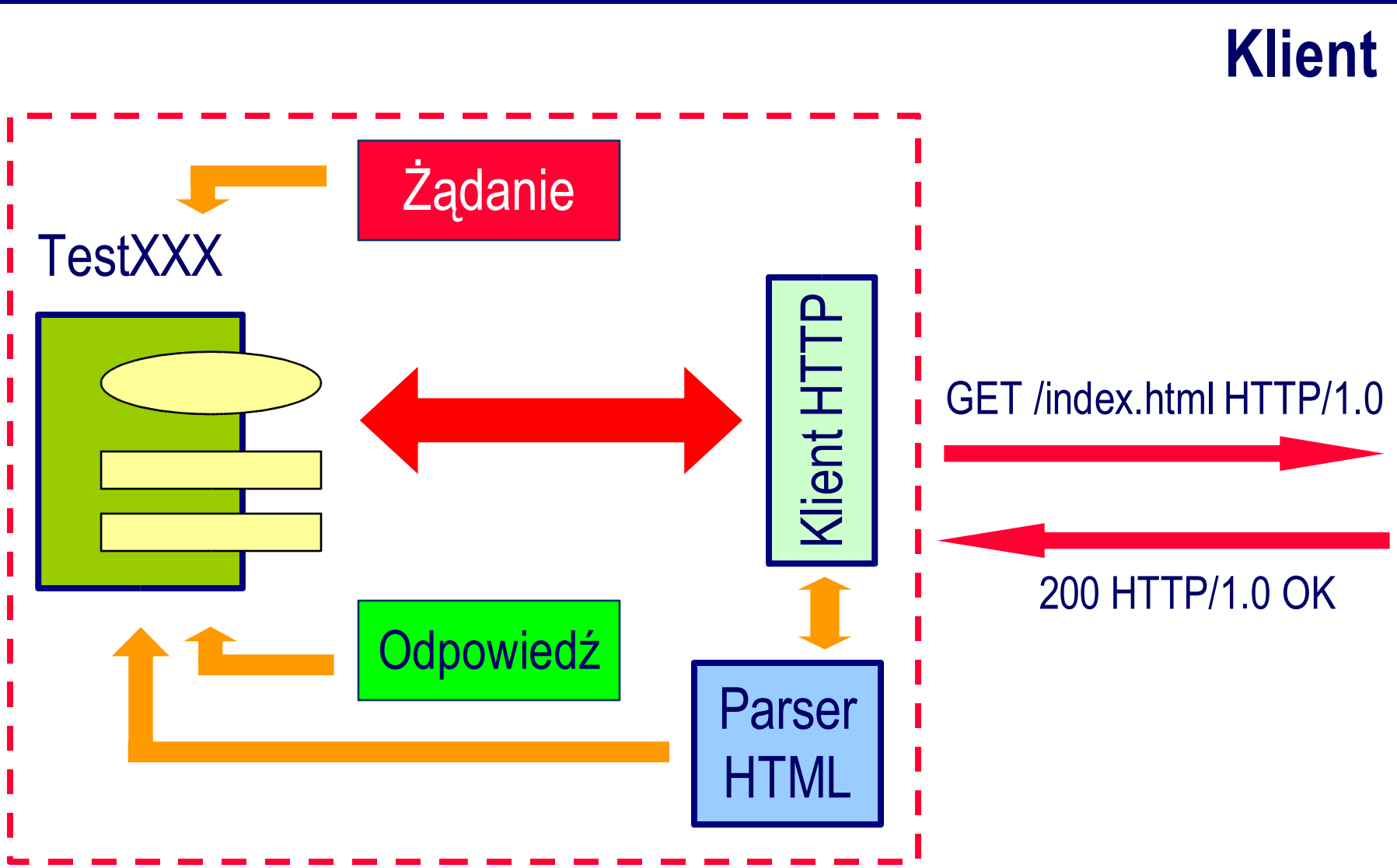
Przykład: serwlet podatkowy

Agenda

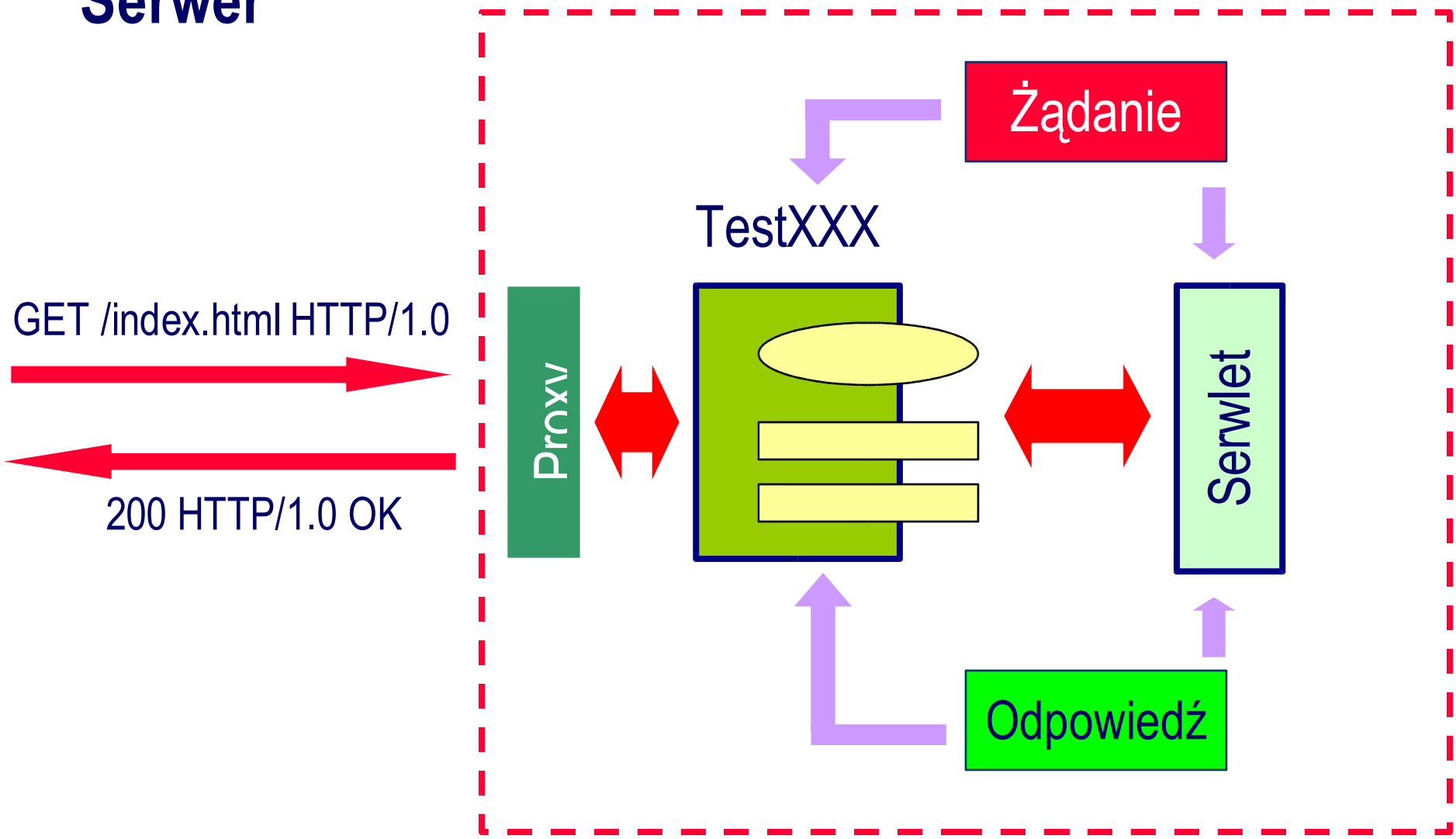
- Aplikacje Java Servlets™
- Jak testować aplikacje internetowe?
  - W małej skali, czyli testy jednostkowe
  - Całymi funkcjami, czyli testy funkcjonalne
  - Jednak API, czyli obiekty zastępcze
  - Czy można mieć wszystko, czyli Cactus



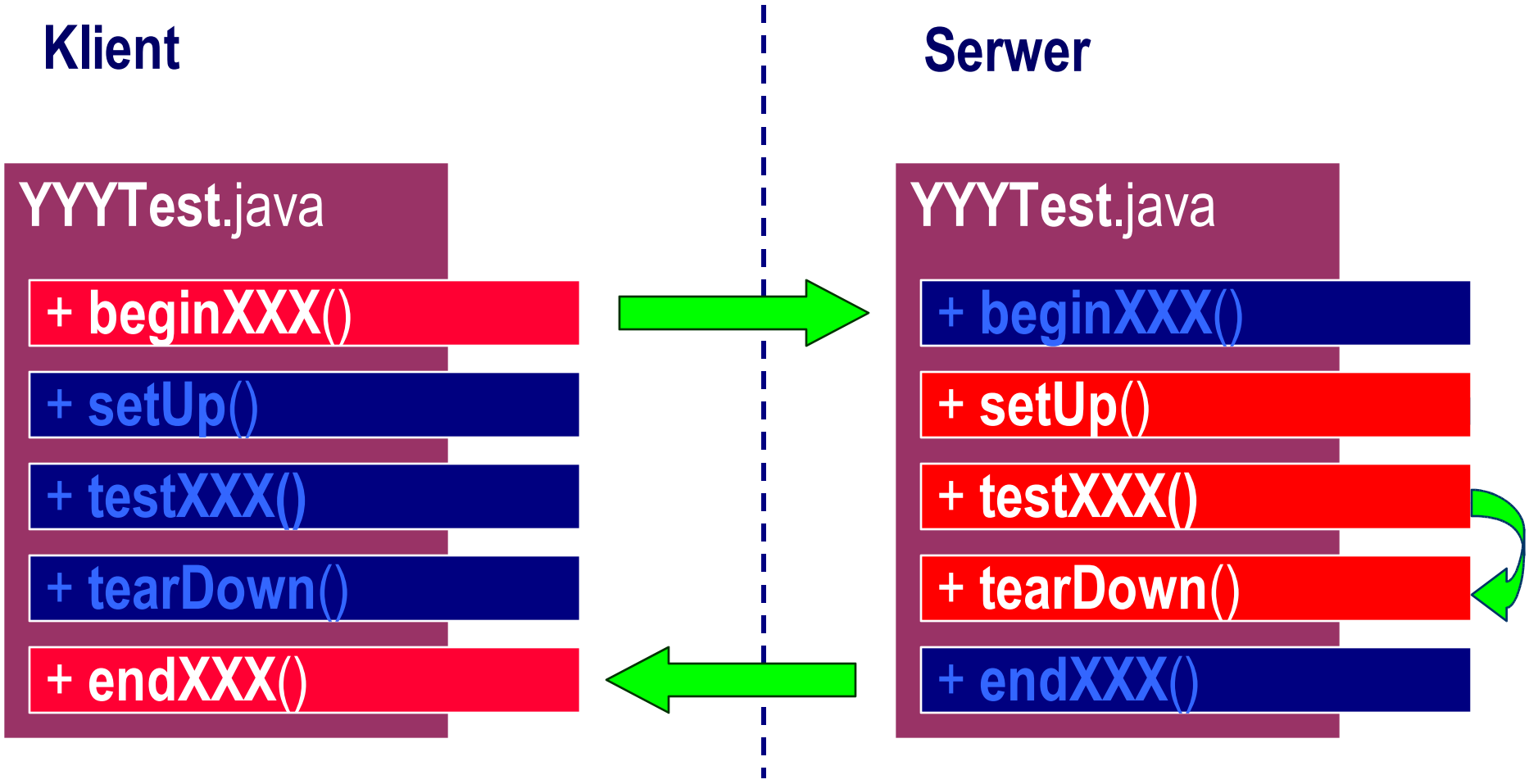
Architektura testów Cactusa



Architektura testów Cactusa



Przypadek testowy Cactusa



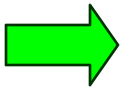
Przypadek testowy Cactusa

Klient

- konfiguracja połączenia HTTP
- definicje parametrów, nagłówków, cookies, sesji
- obsługa autentykacji HTTP

YYYTest.java

+ beginXXX()
+ setUp()
+ testXXX()
+ tearDown()
+ endXXX()



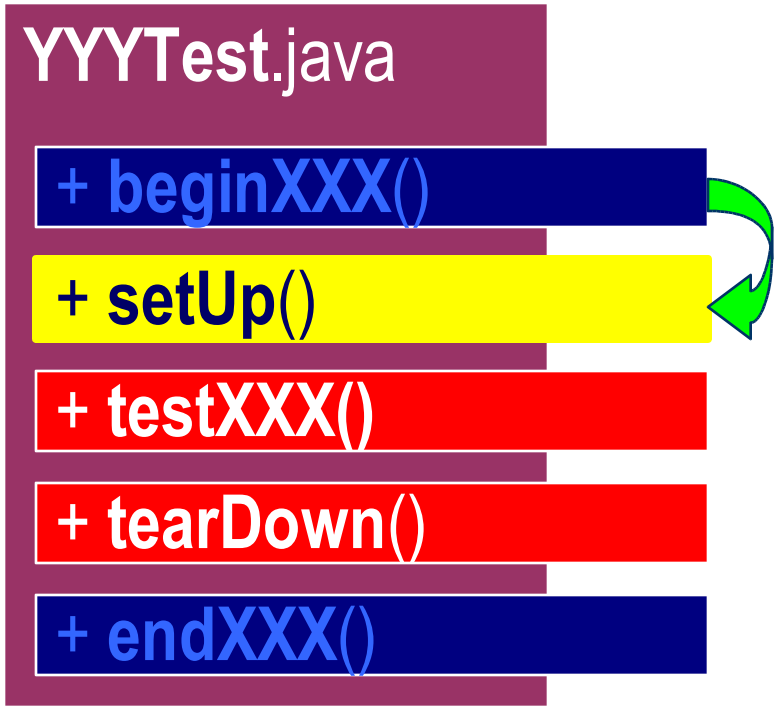
```
void beginXXX(WebRequest request) {
  request.addParameter("income", "100");
  request.addHeader("naglowek", "brak");
  request.addCookie("id", 1234);
}
```

Przypadek testowy Cactusa

- inicjacja środowiska testowego
- utworzenie obiektów wymaganych przez test
- dostęp do wybranych obiektów: request, response, config, session

```
ServletContext context = null;

void setUp() {

  context = config.getServletContext()

  session.setAttribute("loggedIn", "true");

}
```

Serwer

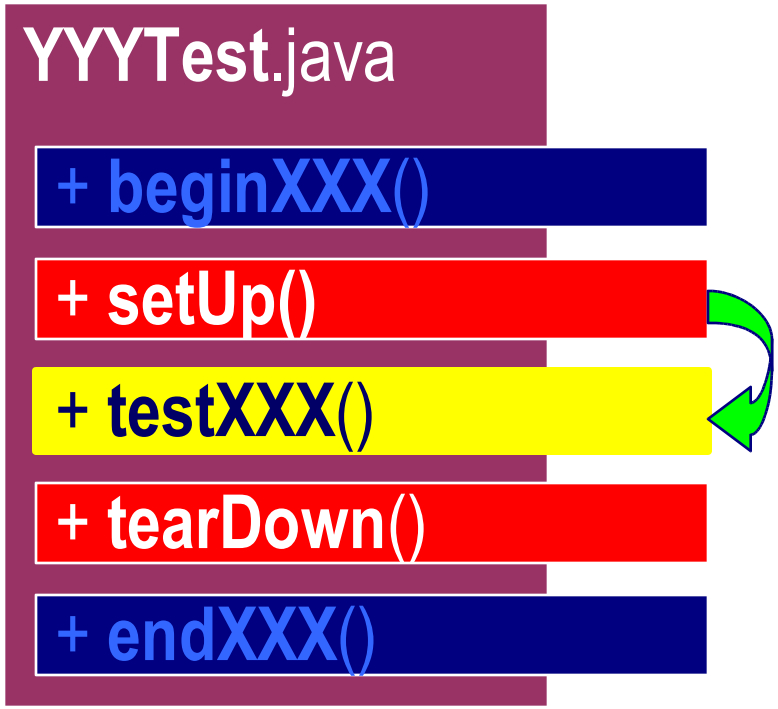


Przypadek testowy Cactusa

- utworzenie testowanego obiektu
- wykonanie metody testowanego obiektu
- weryfikacja wyników
- dostęp do wybranych obiektów: request, response, config, session

```
void testXXX() {

  MojSerwlet srv = new MojServlet();

  srv.doPost(request, response);

  assertEquals("true", session.
    getAttribute("loggedIn"));

}
```

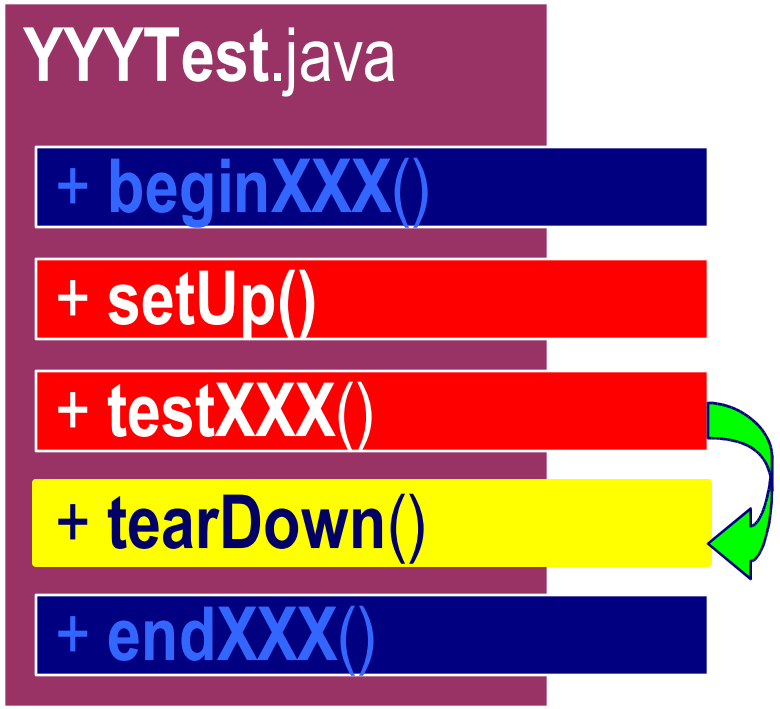
Serwer



Przypadek testowy Cactusa

- usunięcie środowiska testowego
- dostęp do wybranych obiektów: request, response, config, session

```
ServletContext context = null;

void tearDown() {

  session.removeAttribute("loggedIn");

  context = null;

}
```



Przypadek testowy Cactusa

Klient

org.apac | com.meterware.httpunit.WebResponse

- weryfika… …ków… …e klienta
- zakończenie … …al…

YYYTest.java

+ beginXXX()
+ setUp()
+ testXXX()
+ tearDown()
+ endXXX()

```
void endXXX(WebResponse response) {

  assertEquals("brak", response.
    getConnection().getHeaderField(
    "naglowek"));

  assertEquals("1234",
    response.getCookie ("id"));

}
```

Przykład: serwlet podatkowy

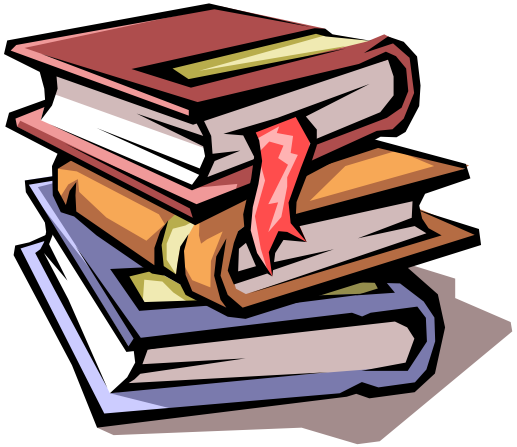
Porównanie metod testowania

- **Testy funkcjonalne**
  - niezależne od technologii wykonania aplikacji
  - powolne w wykonaniu
  - gruboziarniste
- **Testy z wykorzystaniem obiektów zastępczych**
  - bardzo szybkie w wykonaniu, uciążliwe w implementacji
  - drobnoziarniste
- **Testy hybrydowe**
  - powolne w wykonaniu
  - ziarnistość zależy od implementacji
  - kompletne

Podsumowanie

- **Aplikacje internetowe można testować na różne sposoby**
- **Testy uzupełniają się, a nie wykluczają**
- **Różne testy – różny nakład pracy**

Readings



1. ***JUnit***, http://www.junit.org/
2. ***JWebUnit***, http://jwebunit.sf.net/
3. ***HttpUnit***, http://httpunit.sf.net/
4. ***Endo-Testing. Unit Testing with Mock Objects***, http://www.mockobjects.com/wiki/MocksObjectsPaper?action=AttachFile&do=get&target=mockobjects.pdf
5. ***MockObjects***, http://mockobjects.sf.net/
6. ***Jakarta-Cactus***, http://jakarta.apache.org/catus/

Q & A