Design Pattern Density Defined

(Definicja gęstości wzorców projektowych)

Marcin Mieteń

Na podstawie pracy pana Dirka Riehle

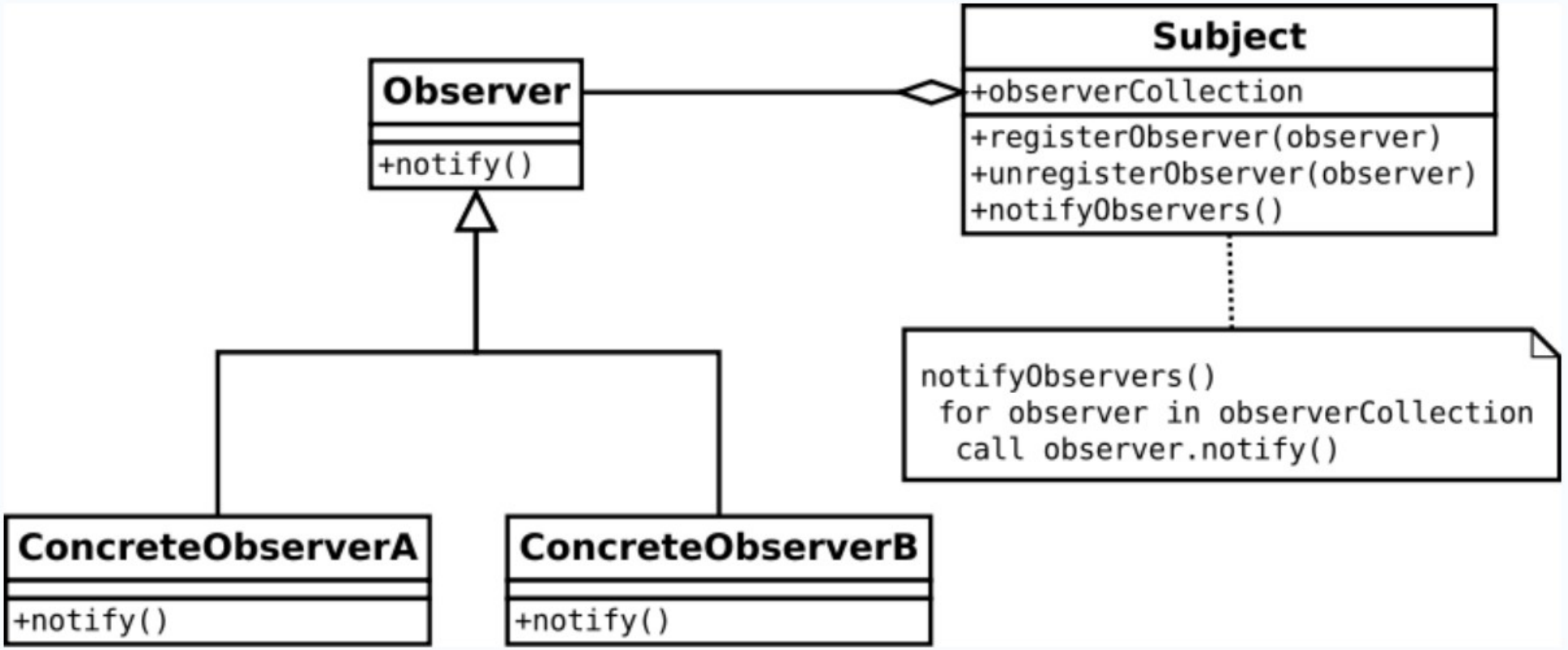
Wzorce Projektowe (przypomnienie)

Przypomnienie idei wzorców, z których będziemy korzystać w dalszej części prezentacji:

- wzorzec Obserwator
- wzorzec Polecenie
- wzorzec (obiektowy) Adapter

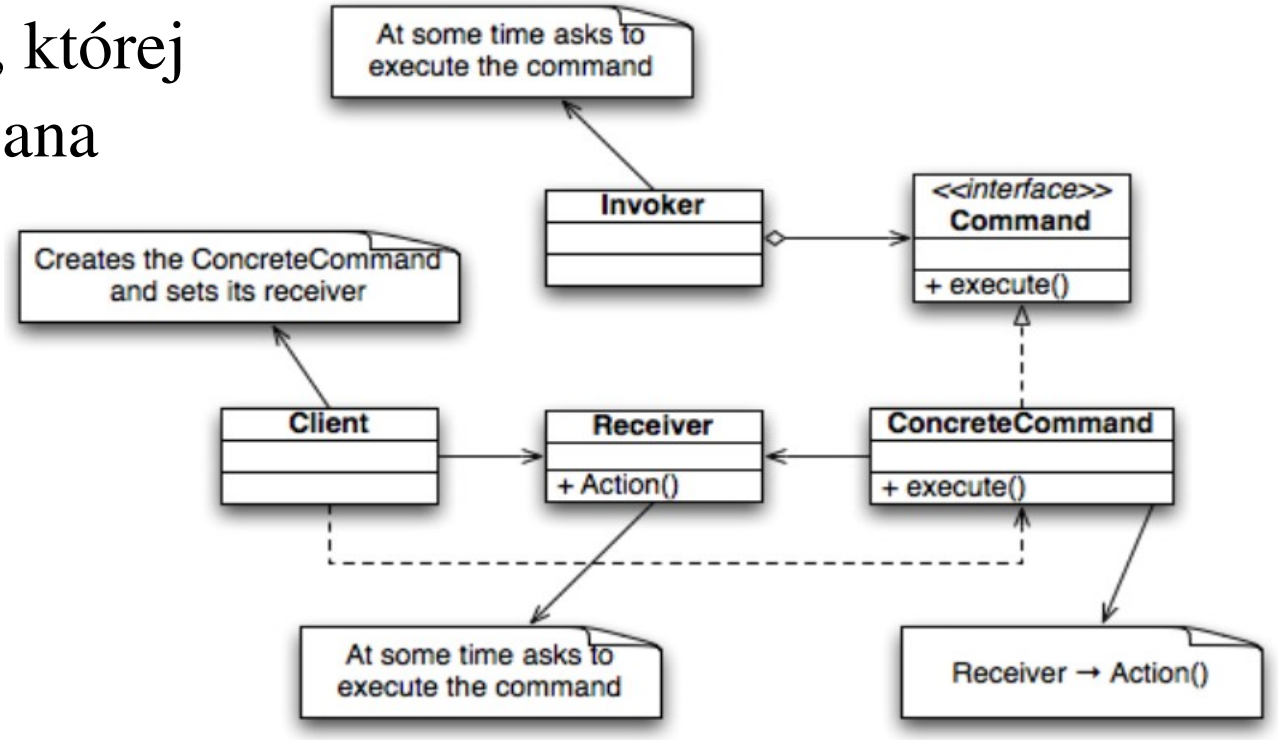
Wzorzec Obserwator

- **Subject** utrzymuje listę **obserwatorów** i informuje ich o zmianie swojego stanu, zazwyczaj przez wywołanie ich metod.
- Rozwiązanie stosowane do implementacji zdarzeń



Wzorzec Polecenie

- **Command** – klasa reprezentująca wszystkie informacje potrzebne do wykonania danej metody (zna konkretny obiekt, jego metodę i dane potrzebne do uruchomienia metody)
- **Client** – tworzy konkretny obiekt klasy Command
- **Invoker –** decyduje, kiedy komenda ma być wykonana.
- **Receiver** – obiekt klasy, której metoda jest wywoływana

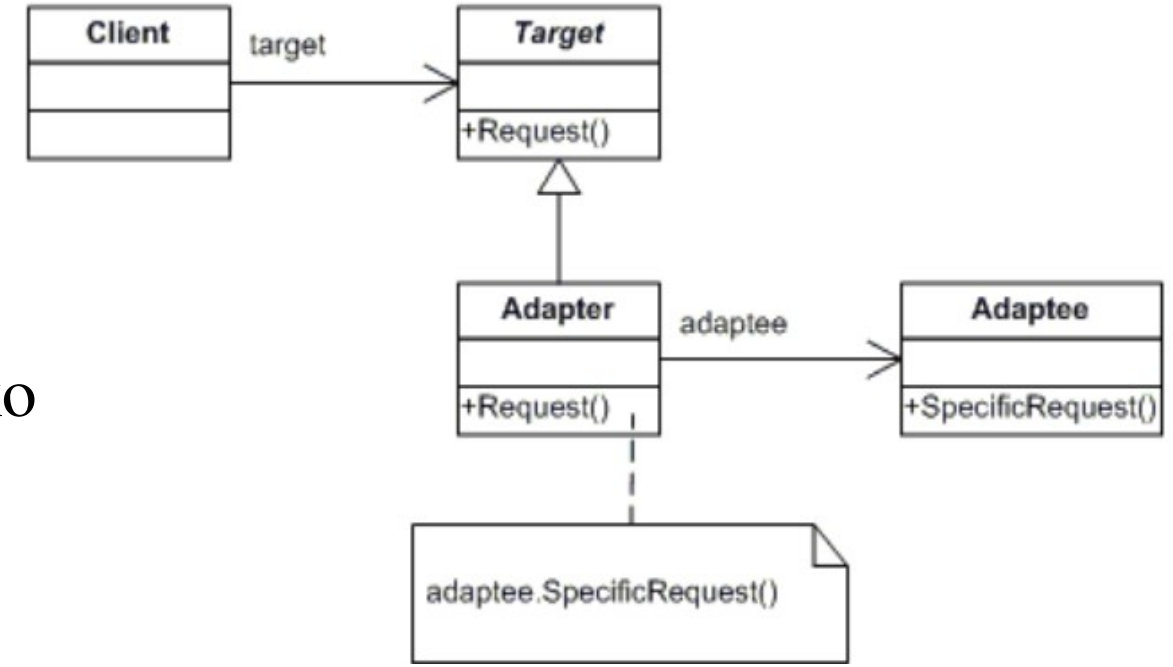


Wzorzec Adapter (także kojarzone z wrapper)

- Używany w sytuacji gdy chcemy ponownie użyć danej klasy, ale interfejs aplikacji nie pasuje do interfejsu naszej klasy.
- Adapter zazwyczaj w ramach wywołania danej metody uruchamia inną metodę pochodzącą od klasy o innym interfejsie.
- Adapter jest też odpowiedzialny za transformowania danych do dobrego formatu, np. gdy wartości boolowskie są trzymane jako 0 i 1, a chcemy mieć true i false.

(Object) Adapter Pattern

- **Target** – definiuje interfejs, którego używa Client
- **Adapter** - adaptuje interfejs **Adaptee** do do interfejsu wymaganego przez obiekt klasy **Target**
- **Adaptee** – definiuje istniejący interfejs, który musi być zaadaptowany
- **Client** – współpracuje tylko z obiektem o interfejsie zgodnym z Target



Cel

- Precyzyjne zdefiniowanie pojęcia gęstości wzorców projektowych (**dalej nazywane GWP lub gęstość WP**)
- Dostarczenie instrumentu do szacowania GWP
- Ilustracja użyteczności GWP
- Przedstawienie hipotez związanych z GWP
  - hipotezy te nie będą udowadniane
  - będą wyznaczały zakres przyszłych prac na temat GWP
- Skupimy się głównie na **GWP zrębów**

Motywacja

- Eksperci głęboko wierzą, że duża gęstość wzorców projektowych wiąże ze sobą dużą dojrzałość danego rozwiązania.
- (jeśli wzorce projektowe powodują, że łatwiej uczy się zrębów to GWP jest miarą jak łatwo nauczyć się danego zrębu)

Definicja (pierwsze podejście)

- Definicja gęstości wzorców projektowych
  - Metryka która mierzy ile procentowo funkcjonalności w danym projekcie może być rozumianych i reprezentowanych jako instancje wzorców projektowych.
  - Potrzebujemy ziarnistej miary funkcjonalności

Współpraca

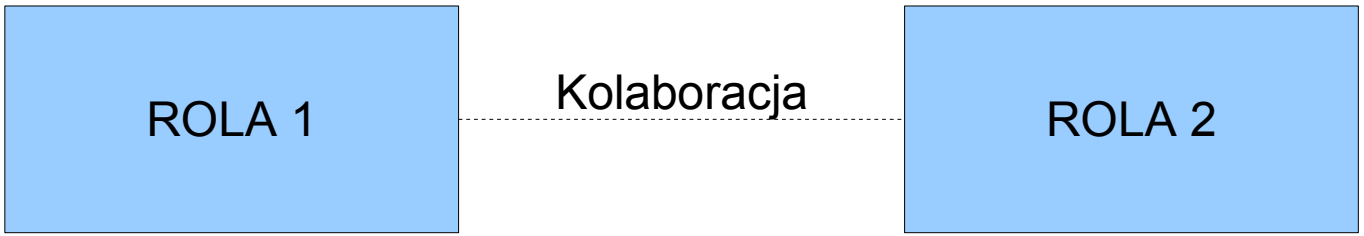
- Zdefiniujmy pojęcie **obiektowych współprac** jako **atomową jednostkę funkcjonalności**. Pozwoli nam to na policzenie liczby wzorców projektowych w dany zrębie.
- Współprace pokazują jak rozproszone są odpowiedzialności pomiędzy klasami, w celu osiągnięcia przez ich instancje odpowiednich celów.

Współpraca

- Pojęcie uczestnika w współpracy jest nazwane **rolą**.
- Projektowanie oparte na współpracach narodziło się jakoś **modelowanie ról** i jest mocno powiązane z CRC (Class Responsibility Collaboration)
- W podejściu kolaboracyjnym istotne jest zwrócenie uwagi na zależności między obiektami, a nie na strukturę klasy.

projektowanie oparte na współpracy

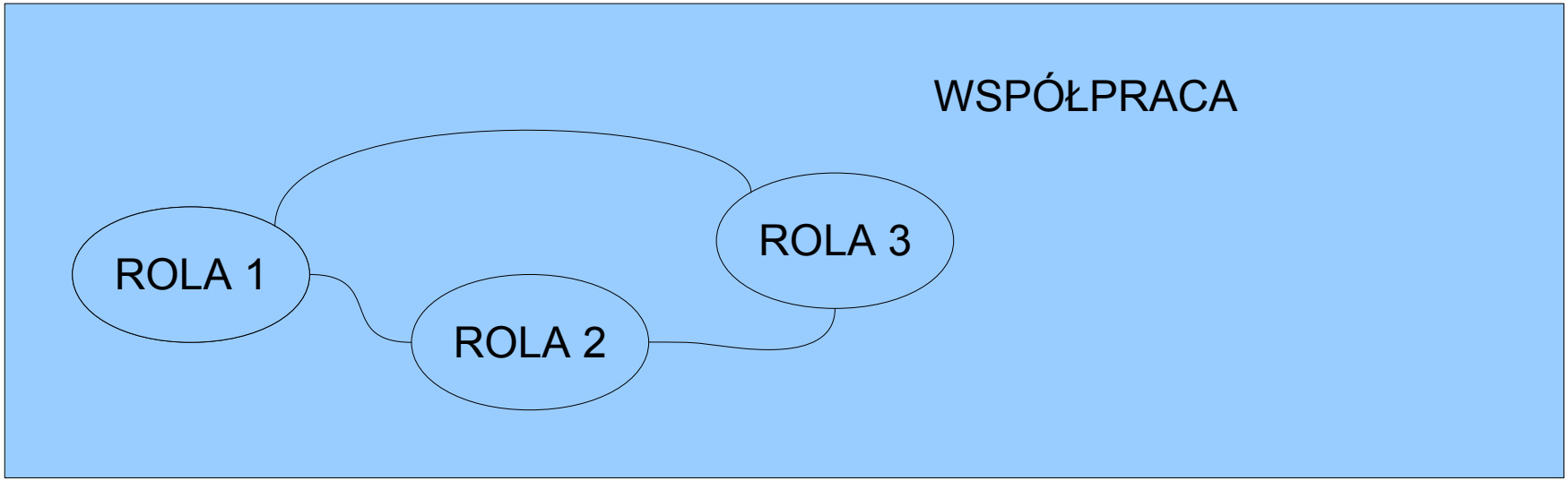
- Obiekty posiadają **role,** które definiują jak obiekty współpracują, aby osiągnąć zdefiniowany cel.
- **Rola** odnosi się tylko do danej **współpracy** (collaboration) i nie wychodzi poza jej granice



Role a współprace

ROLA definiuje zachowanie obiektu w danej **współpracy.**

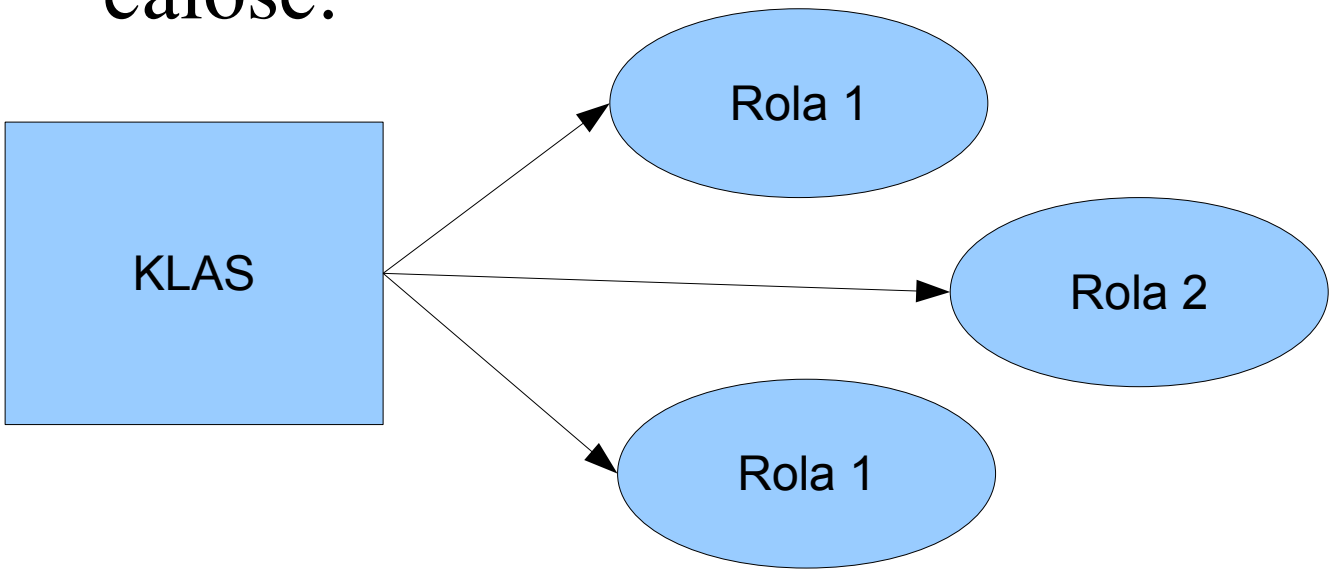
WSPÓŁPRACA grupuje role i definiuje zakres interakcji między **rolami**.



Klasy a Role

ROLE definiują zachowanie instancji danej **klasy** w szczególnej **współpracy**

KLASY definiują jak ich instancje realizują oczekiwane od nich zachowania (powiązane z pełnieniem wielu ról) jako jedną zintegrowaną całość.



Definicja (podejście drugie)

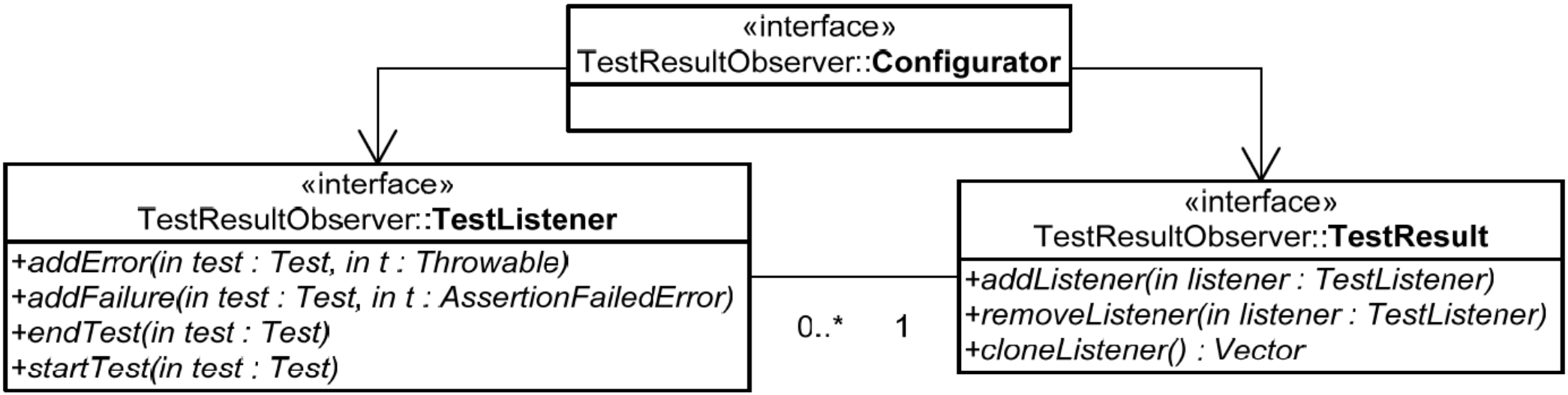
Gęstość wzorców projektowych zrębów to procent współprac, które są **instancjami wzorców projektowych**

Role View

- Role View to specjalny sposób oznaczania współprac i ról w UML
- Jest uzupełnieniem **class view**
  - **Role view** => zachowanie w konkretnym kontekście
  - **Class view** => zachowanie i integracja wielu współprac
- Oznaczenie na diagramie:

<nazwa współpracy>::<nazwa roli>

Kolaboracja TestResultObserver w Junit 3.8



- Przykład wzorca **Observer**
  - TestResult **role** reprezentuje uczestnika **Subject**
  - TestListener **role** reprezentuje uczestnika **Observer**
  - Dodatkowa rola **Configurator** zajmuje się rejestracją TestListenerów w TestResult

Role instancji klasy TestResult (z Junit 3.8)

Metody klasy TestResult są sumą metod zdefiniowanych przez role, które pełni instancja w poszczególnych współpracach

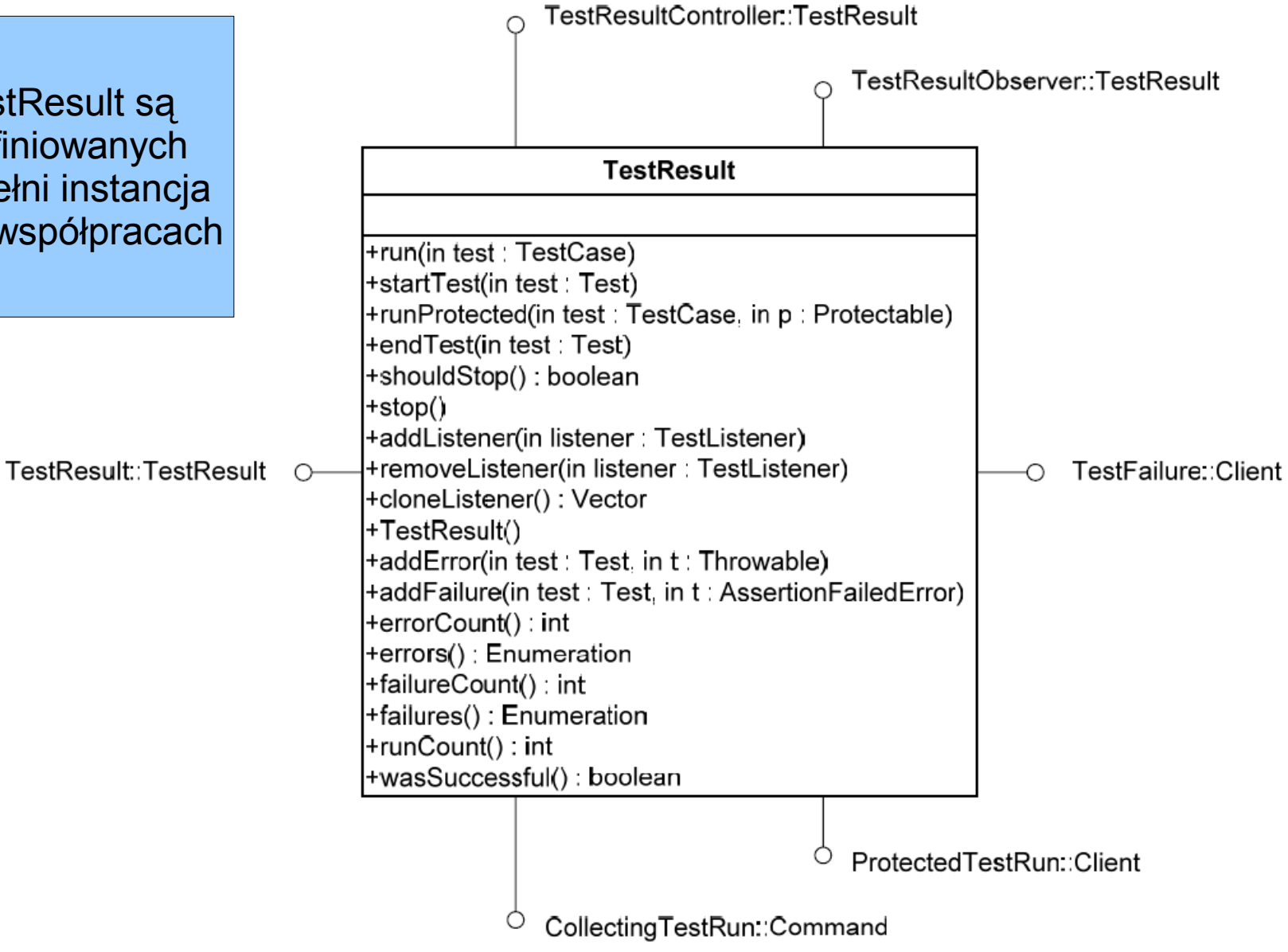
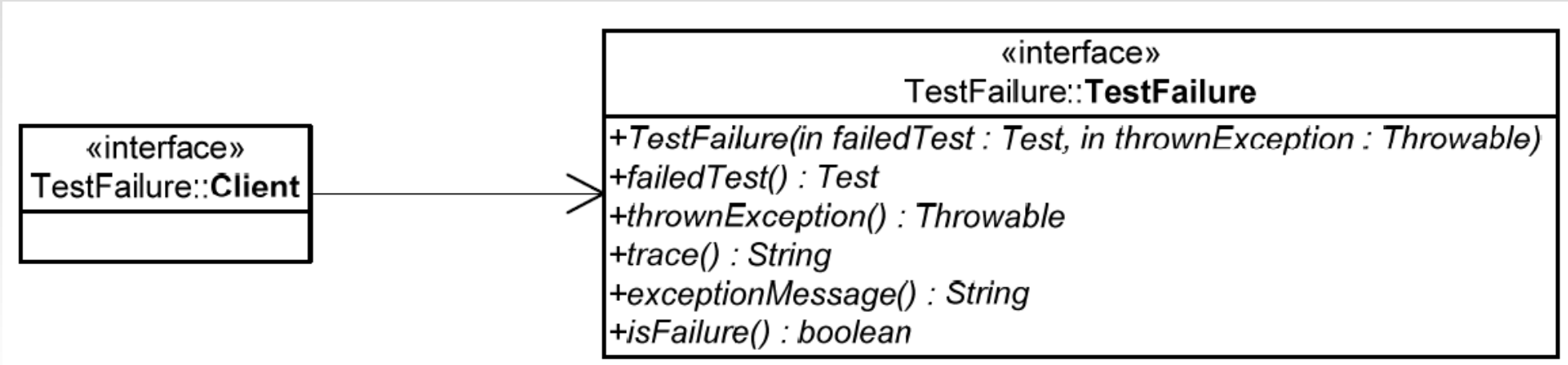


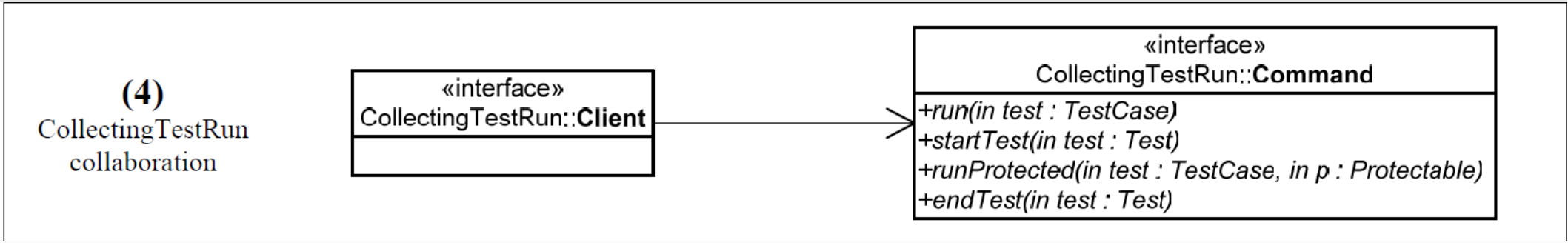
Figure 2: The TestResult class and the different roles its instances play.

client/server collaboration



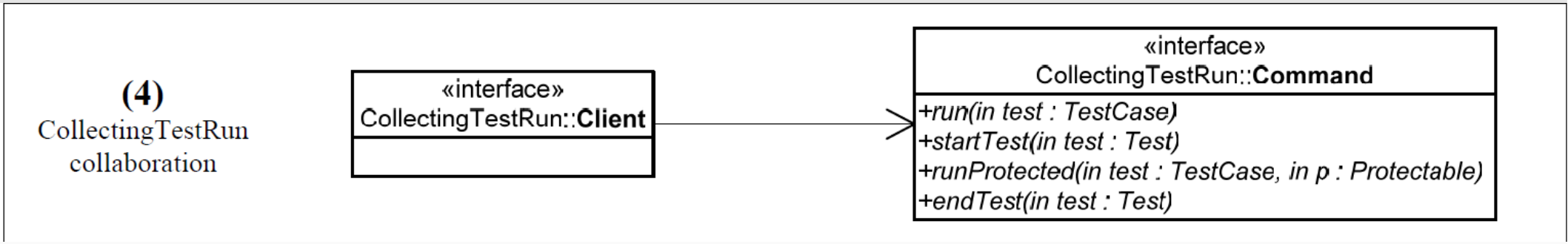
- Często nazwa roli w danej współpracy jest taka sama jak nazwa samej klasy.
- **client/server collaboration** - zazwyczaj w większości zrębów mamy kilka współprac, które są zwyczajnym wywołaniem przez jedną klasę metody innej klasy
- client/server collaboration **nie** jest wzorcem projektowym

CollectingTestRun collaboration



- Rola **Client** → klasa TestCase
- Rola **Command** → klasa TestResult
- Do TestResult zapisują się **obserwatorzy**. W momencie wywoływania startTest i endTest wywoływane są też odpowiednie metody obserwatorów.
- Czy jest to wzorzec projektowy? Jeśli tak to jaki?

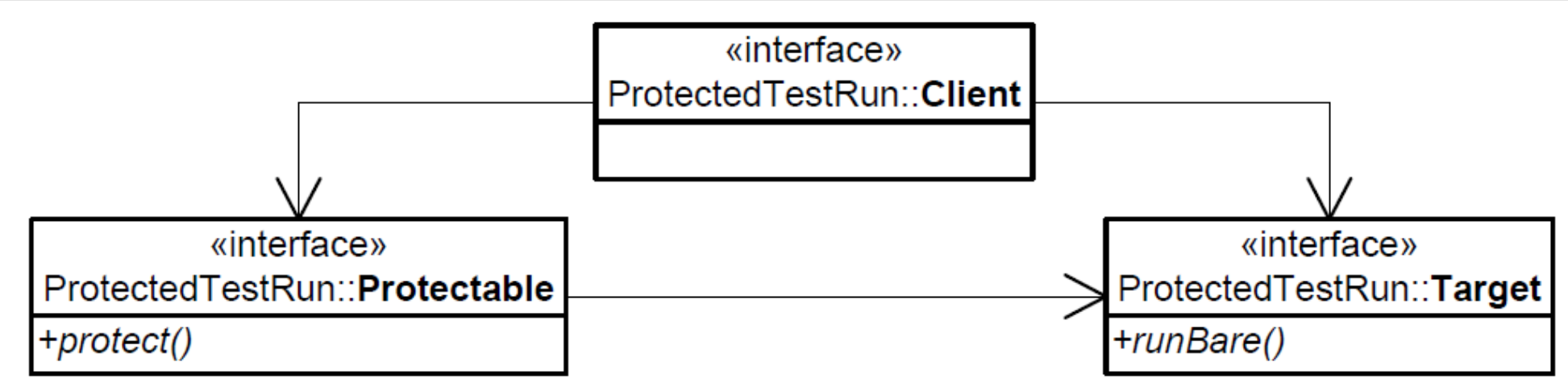
CollectingTestRun collaboration



Command Pattern

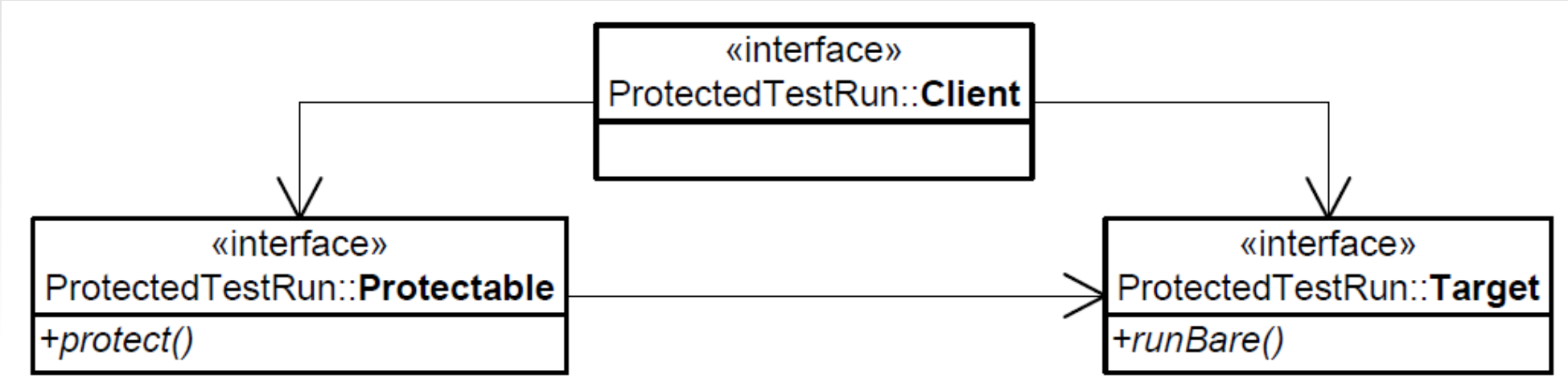
- role Client i Command odpowiadają nazwami rolom w definicji wzorca Command Pattern
- Obserwator TestResulta (rola Command) odpowiada obiektowi **Receiver** (klasa której metoda jest wywoływana przez obiekt pełniący rolę Command)

ProtectedTestRun collaboration



- Klasy odpowiadające **Protectable** i **Target** implementują interfejs TestCase (posiadają metodę run). Client potrafi wywoływać TastCasy.
- **Protectable** opakowuje jedynie **Traget** w ten sposób, że wywołuje jego metodę run i ewentualnie przechwytuje i obsługuje wyjątki.

ProtectedTestRun collaboration



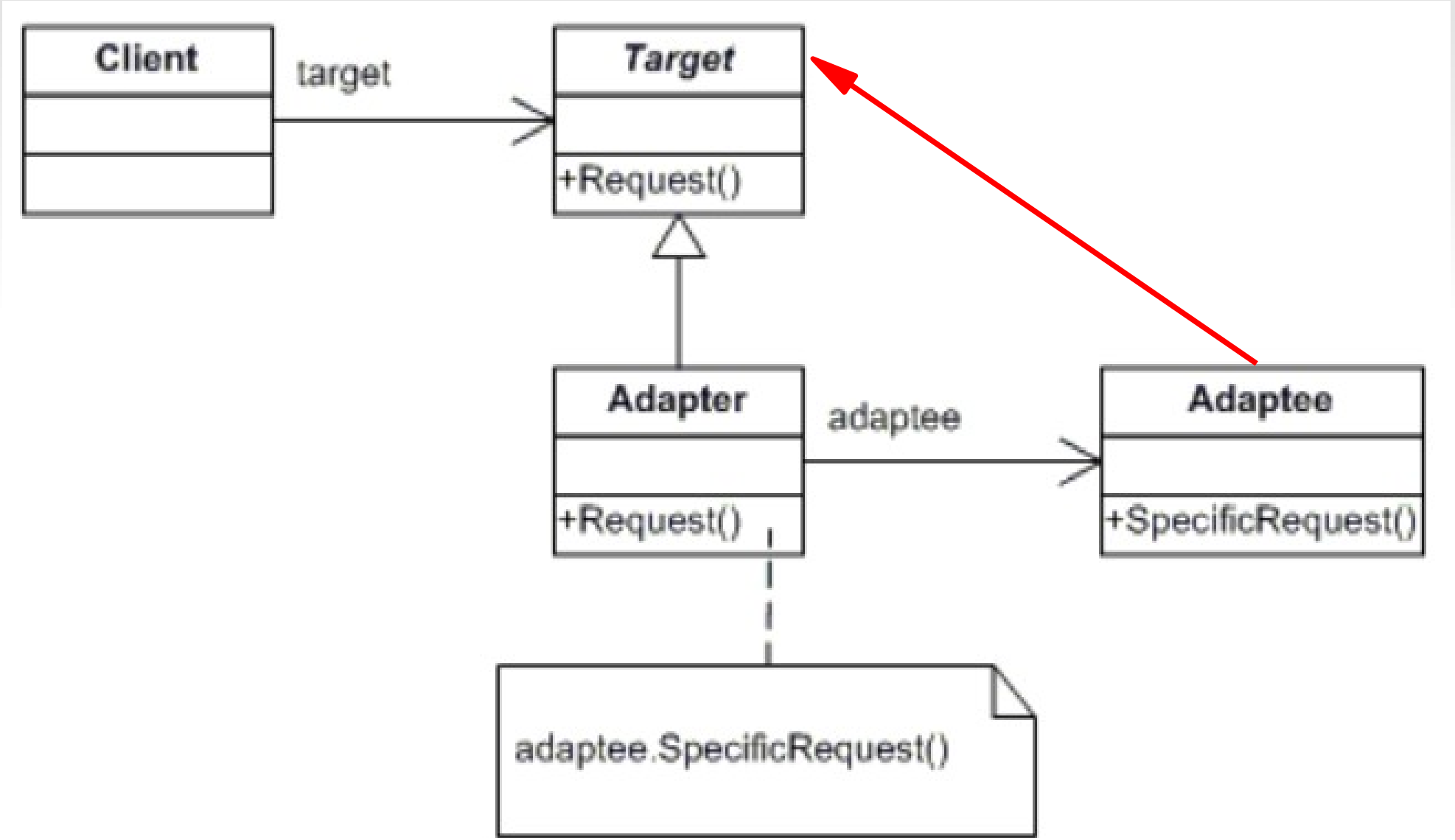
Wzorzec adapter

Adapter → Protectable

Adaptee → Target (nietypowy bo też implementuje interfejs Target z wzorca Adapter, może dla tego tak nazwali tę rolę)

Adaptacja polega na usunięciu możliwości rzucenia wyjątku (bo np. klient tego nie przewiduje)

Nasz szczególny przypadek



Analiza JUnit 3.8 framework

Collaboration name	Number of Collaborations	Number of Roles in Collaboration	Number of Methods in Collaboration	Is it a pattern? If so which?
TestCase	1	2	4	-
TestSuite	1	2	4	-
TestSuiteTestCreation	1	2	4	-
TestRun	1	2	1	Command
TestCaseTestRun	1	2	2	-
TestSuiteTestRun	1	2	1	-
TestHierarchy	1	3	13	Composite
TestResult	1	3	9	Collecting Parameter
TestResultController	1	2	2	-
TestResultObserver	1	3	7	Observer
CollectingTestRun	1	2	4	Command
ProtectedTestRun	1	3	2	Adapter
TestRunMethod	1	2	4	Template Method
Assertions	1	2	38	-
TestFailure	1	2	6	-
AssertionFailedError	1	2	2	-
ComparisonFailure	1	2	3	-
ComparisonCompactor	1	2	2	Strategy
CompactMethod	1	2	6	Composed Method
Total	**19**	**42**	**114**	**9**

Gęstość Wzorców Projektowych

Gęstość wzorców projektowych to procentowa zawartość współprac, które są instancjami wzorców projektowych

(#(współprace będące WP) / #(współprace)) *100%

Analiza Junit 3.8

JUnit 3.8 Case Study	
Number of classes/interfaces	11
Number of collaborations	19
Number of pattern instances	9
Number of roles in total	42
Ratio roles per class/interface	3.8
Design pattern density	**47%**

$$(9/19) * 100\% = 47{,}37\ \%$$

Dwa rodzaje gęstości WP

- Gęstość WP w interfejsie architektury
  - Procent kolaboracji, które są WP w architekturze samego interfejsu danego zrębu
  - Może doszukiwać się zależności między łatwością nauki zrębu a tą gęstością.
- Gęstość WP w całym projekcie
  - Procent kolaboracji, które są WP w kompletnym projekcie.
  - Ta miara może być przydatna, np. gdy ktoś chce rozbudować zręb.

Analiza trzech frameworków

1. „**The Geo system**, a metalevel-architecture-based implementation of a distributed object system”
2. „**The KMU Desktop framework** used to build tools in a financial risk assessment application”
3. „**The JHotDraw framework** for building graphical editors„

Table 3: Summary data from three case studies.

Case study	**[1]**	**[2]**	**[3]**
Number of interfaces and interface classes	17	13	20
Number of collaborations	34	20	28
Number of pattern instances	20	12	20
Number of roles assigned to classes	75	44	66
Ratio roles per class/inter-face	4.4	3.4	3.3
Design pattern density (interface architecture)	**59%**	**60%**	**71%**

[1] - The Geo framework

[2] - The KMU Desktop framework

[3] - The JHotDraw core framework

Statystyki

Table 4: The maturity level, pattern density, roles per collaboration data of the case studies.

Case study	Maturity Level (1-3)	Design Pattern Density	Number of roles per collaboration
Assessed on the interface architecture level			
Geo System	2	59%	2.21
KMU Desktop	2	60%	2.20
JHotDraw	3	71%	2.36
Assessed on the complete design level			
JUnit	3	47%	2.21

- Poziom dojrzałości:
  1. nowy
  2. ulepszony
  3. dojrzały

HIPOTEZY

- Nowa miara jest prosta, precyzyjna, kompletna i mierzalna.
- Ale czy jest użyteczna??
- Przeprowadzone analizy prezentowane w pracy są niewystarczające, aby dostarczyć znaczących konkluzji.
- Możemy tylko użyć nowej miary i metod jej mierzenia do przedyskutowania pewnych hipotez

HIPOTEZA 1

Wraz ze wzrostem dojrzałości zrębów rośnie gęstość WP.

(wyjściowa hipoteza)

HIPOTEZA 2

- Wraz z kolejnymi wersjami zrębu zmienia się jego gęstość WP.
- GWP <= x, gdzie x < 100%
- **Punkt stały –** GWP perfekcyjnego projektu dla danego zrębu (może być inny dla różnych zrębów)
- Można przypuszczać, że GWP w kolejnych rewizjach dąży do **punktu stałego** od dołu.

HIPOTEZA:

Wraz ze wzrostem dojrzałości wartość GWP zbliża się do punktu stałego

HIPOTEZY 3,4,5

- Wartości punktów stałych dla wszystkich możliwych zrębów tworzą odpowiednią zmienną losową i na tej podstawie odpowiedni rozkład prawdopodobieństwa
- (Średnia wartość rozkładu prawdopodobieństwa GWP interfejsu architektury) > (średnia wartość rozkładu prawdopodobieństwa GWP całego projektu)
- **GWP interfejsu** mocno powiązane z **GWP całości**
  - Dzięki temu nie musimy mówić, o które GWP nam chodzi w różnego rodzaju rozważaniach

HIPOTEZA 6

- Finalne dopracowanie pierwszej hipotezy:

Różnica między konkretną wersją a punktem stałym danego zrębu jest miarą dojrzałości zrębu.

HIPOTEZA 7

Czym mniejsza różnica między GWP a punktem stałym tym łatwiej ***nauczyć się i używać danego zrębu****.*

Problemy

- Nie wszystkie wzorce można rozpoznać dzięki kolaboracjom
- **Rozpoznanie wzorca projektowego** (często sama dokumentacja jest dwuznaczna)
- Brak dokładnej definicji **co stanowi wzorzec projektowy**
- Mało deweloperów korzysta z projektowania opartego o współprace
- Przydałyby się narzędzia do automatycznego wykrywania wzorców projektowych

Prace do wykonania

- Zbadanie zależności GWP między kolejnymi wersjami zrębów
  - Badania powinny dać odpowiedź, czy punkt stały istnieje i jak metryka zbiega do tego punktu stałego.
- Zbadanie jak rozkład prawdopodobieństwa punktów stałych ma się do punktu stałego rozwijanych zrębów.

Podsumowanie

- Przedstawiono nową miarę, która w przyszłości może być dobrym odzwierciedleniem użyteczność obiektowych zrębów i może pomóc w wyborze, który z nich wybrać.
- Przedstawiono dodatkowe możliwości, jakie daje projektowanie oparte na współpracach, dzięki którym możemy obliczyć wartość naszej metryki w danym zrębie.

KONIEC