Zaawansowane programowanie w C#

Paweł Biesiada
https://www.linkedin.com/in/pawelbiesiada/

- Architekt, software developer, ex. Scrum Master, IT Trainer
- .NET developer since 11.2011
- IT Trainer since 06.2018
- *pawel.piotr.biesiada@gmail.com*

Delegaty

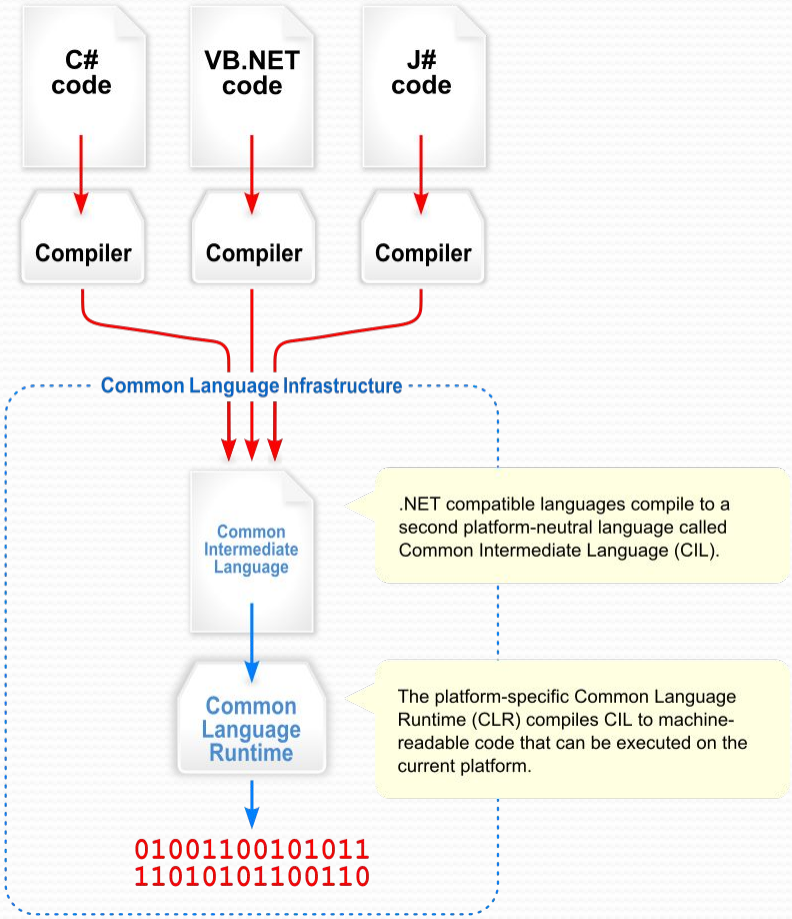
- Wskaźniki na funkcje
- delegate vs event
- Dostęp do zmiennych
- Delegaty zdefiniowane
  - Action, Action<>
  - Func<>
- Lambdy

Metody rozszerzeń

- Zamiast dziedziczenia
  - Nie zmienia definicji samej klasy
  - Można stosować do klasy do której nie mamy dostępu
- this
  - Uwaga na referencje do zmiennej this

.NET Framework

- CLI – Common Language Infrastructure
- CIL – Common Intermediate Language
- CLR – Common Language Runtime



Refleksja

- Dostęp do pakietów (assembly)
- Dostęp do klas i właściwości
- Late Binding

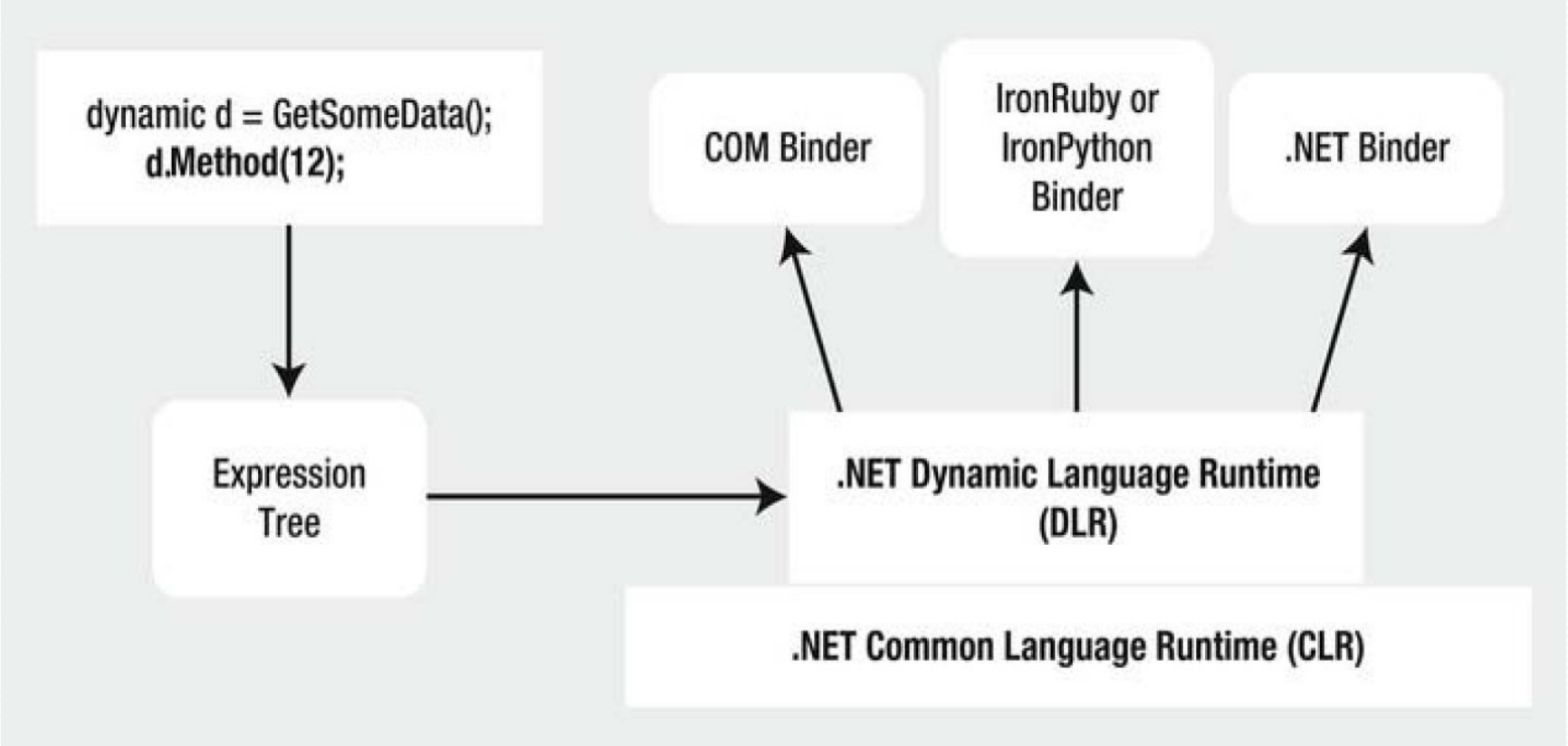
Atrybuty

- Klasa Attribute
- Atrybuty jako metadane klasy
- Wykorzystanie własnych atrybutów

Drzewa wyrażeń - *dynamic*

- Wykorzystuje DLR (dynamic language runtime)
- Usprawnia pracę z obiektami COM, refleksją
- Słowo kluczowe dynamic zakres stosowania:
  - pole, właściwość, typ zwracany, zmienna lokalna
- Słowa kluczowego dynamic nie można używać z lambdami

Drzewa wyrażeń - *dynamic*



Pliki wykonywalne

- Łączenie wielu plików wykonywalnych w jeden
  - ILMerge
- Dekompilacja kodu
  - ILSpy, ILDasm, dotPeek
- Zabezpieczenia przed dekompilacją
  - Obfuskacja
  - Ukrywanie stringów

Debugowanie kodu

- Klasa Trace
- Klasa Debug
- Klasa StackTrace
- Klasa Stopwatch
- Visual Studio – okna debugowania

Komponenty webowe

- WebResponse/WebRequest
  - Bardziej ogólne
  - daje większą kontrolę nad requestami
- WebClient
  - Wspiera metody synchroniczne i asynchroniczne
- HttpClient
  - Wspiera metody asynchroniczne
  - Implementuje metody wspierające REST

Architektura klient-serwer

- Serwer (thick):
  - Procesuje przychodzące zadania
  - Odsyła wyniki do klienta
- Klient (thin):
  - Wysyła zapytania do serwera
  - Odbiera wyniki od serwera

- Warstwa komunikacji:
  - musi być ustalona publicznie i jednolita dla obu stron
  - http (REST/SOAP), TCP, pipe

Windows Communication Foundation (SOAP)

- ABC Concept
  - Address (Where)
  - Binding (How)
  - Contract (What)
    - ServiceContract
    - OperationContract
    - DataContract
    - DataMemberContract
    - FaultContract

Representational State Transfer (REST)

- Komunikacja przez HTTP (JSON)
  - GET
  - POST
  - PUT / PATCH
  - DELETE

Unity (DI)

- Inversion of Control
- Kontenery
- Rejestrowanie typów
- Rozwiązywanie zależności

Remote Procedure Call (gRPC)

- Komunikacja przez HTTP2 (binarna)
- Protocol Buffers (proto3)
- Dedykowane typy danych

- Specyfikacja: https://developers.google.com/protocol-buffers/docs/proto3

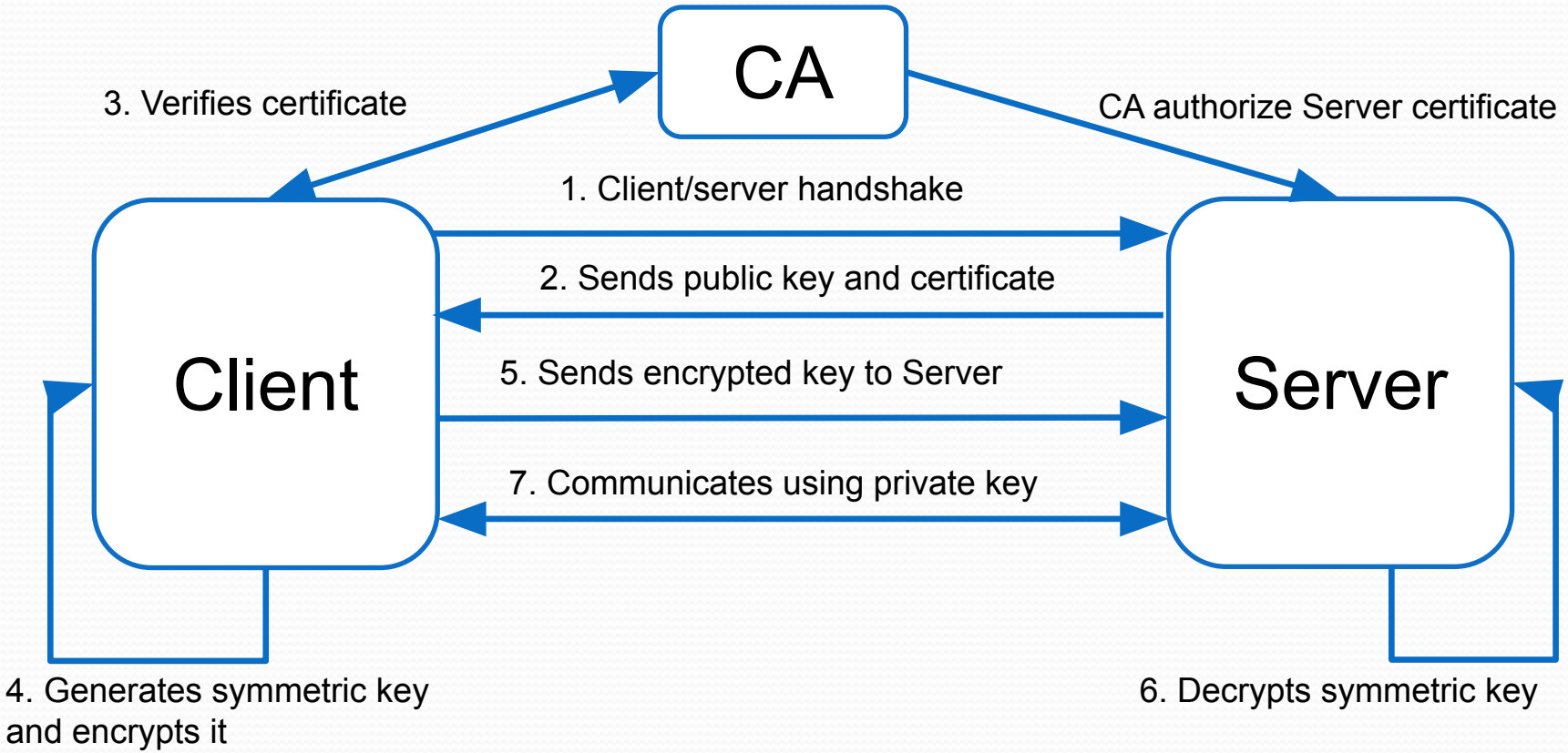
Bezpieczeństwo aplikacji webowych

- Uwierzytelnianie (authentication)
  - Host side
  - OAuth i OpenID
- Autoryzacja (authorization)
- Ataki na aplikacje webowe:
  - DDoS (distributed denial of service)
  - XSS (cross site scripting)
  - CSRF (cross site request forgery)

Szyfrowanie i hashowanie danych

- Szyfrowanie:
  - Symetryczne
    - AES, DES, RijndaelManaged
  - Asymetryczne
    - RSA, DSA
- Hashowanie
  - MD5, SHA

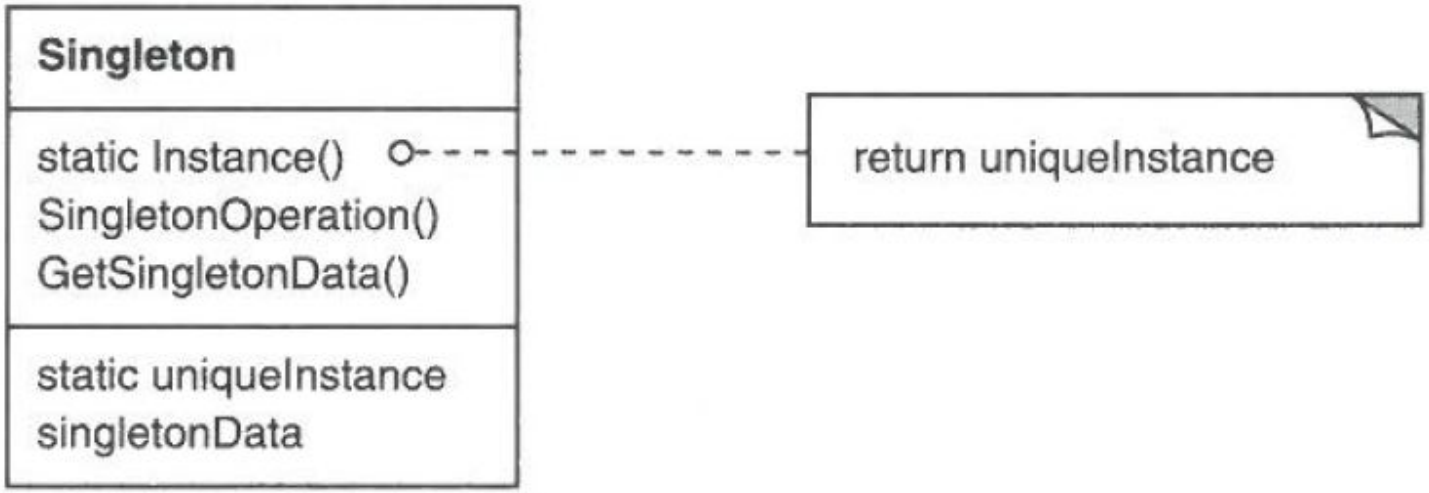
TLS/SSL


CA
3. Verifies certificate
CA authorize Server certificate
1. Client/server handshake
2. Sends public key and certificate
Client
5. Sends encrypted key to Server
Server
7. Communicates using private key
4. Generates symmetric key
and encrypts it
6. Decrypts symmetric key

Wzorce projektowe

- Kreacyjne
  - Singleton, Fabryka Abstrakcyjna, Fabryka Metod
- Strukturalne
  - Fasada
- Czynnościowe
  - Obserwator, Stan, Strategia, Polecenie, Iterator

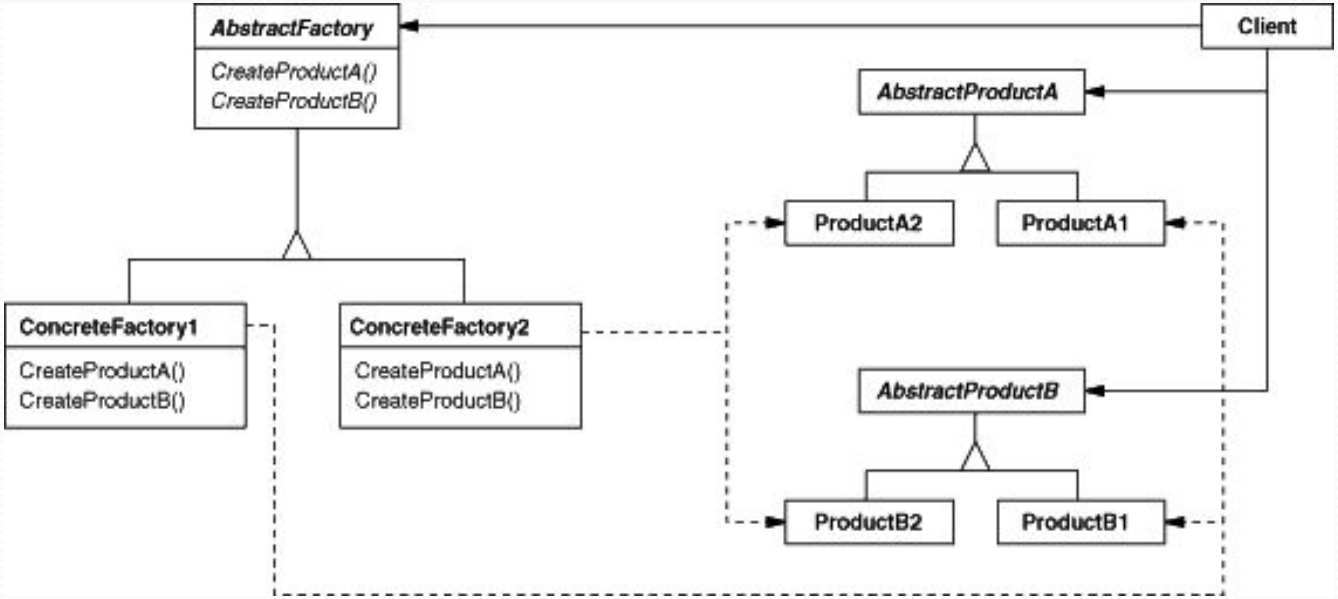
Singleton (Kreacyjny)



- Istnieje tylko jedna instancja klasy

Źródło: “Design Patterns, Elements of Reusable Object Oriented Software” E.Gamma, R. Helm

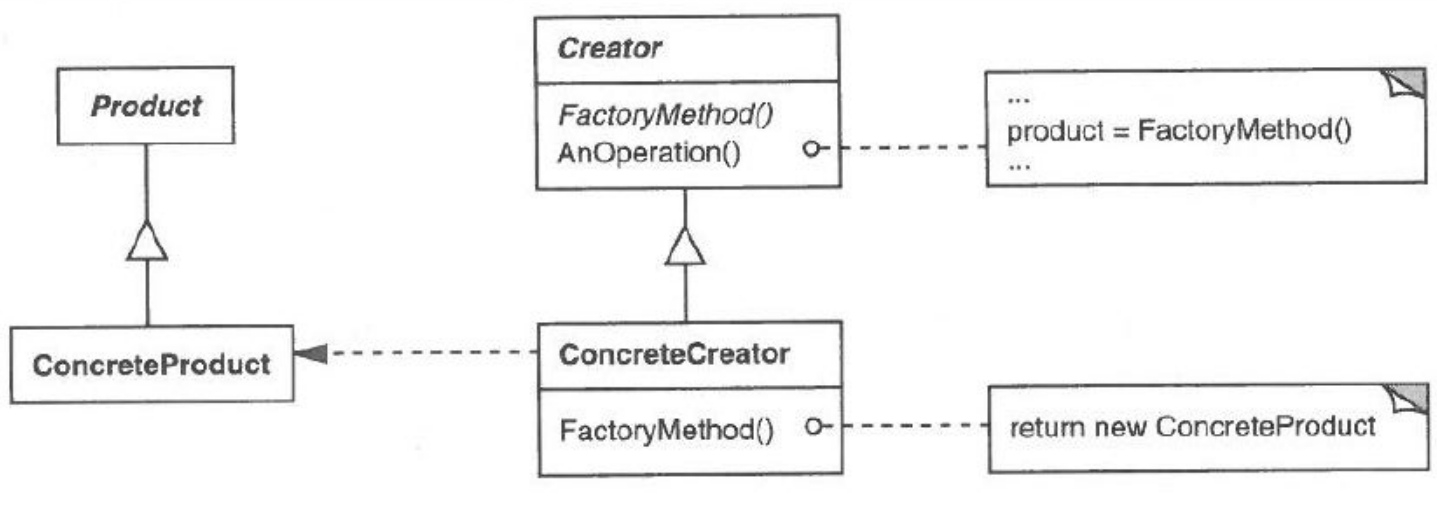
Fabryka abstrakcyjna (Kreacyjny)



- Udostępnia interfejs do tworzenia pewnego podzbioru podobnych obiektów
- To konkretna implementacja Fabryki dostarcza konkretne implementacje obiektów

Źródło: “Design Patterns, Elements of Reusable Object Oriented Software” E.Gamma, R. Helm

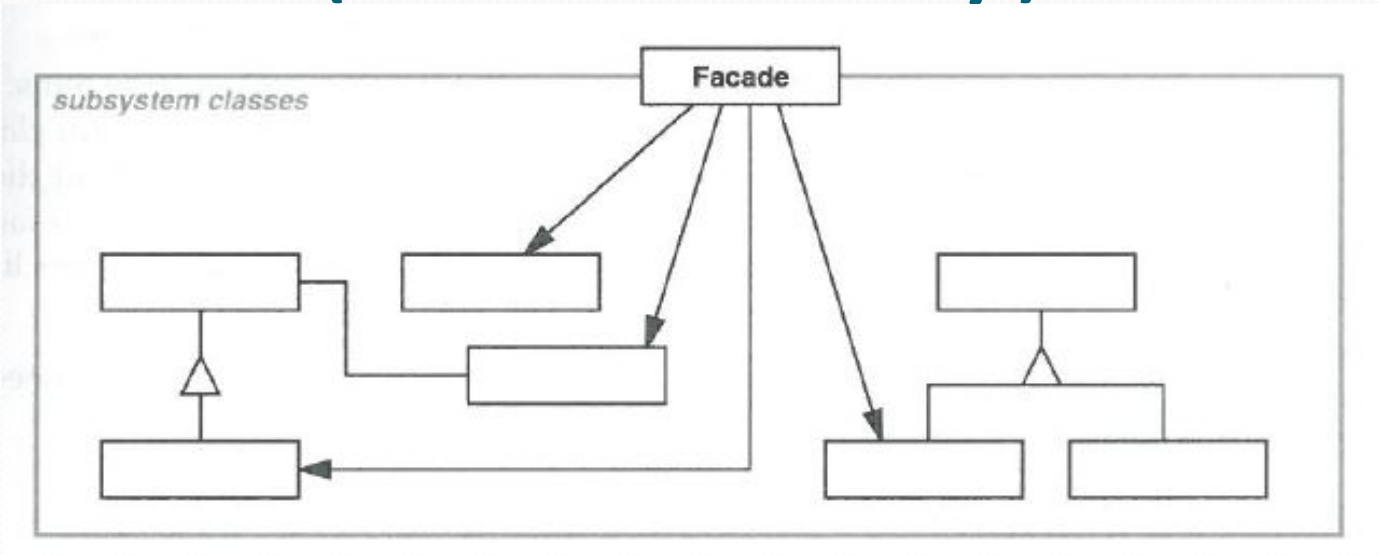
Fabryka metod (Kreacyjny)



- Określa interfejs, który zwracany jest przez metodę
- To metoda decyduje, którą implementację interfejsu zwrócić

Źródło: “Design Patterns, Elements of Reusable Object Oriented Software” E.Gamma, R. Helm

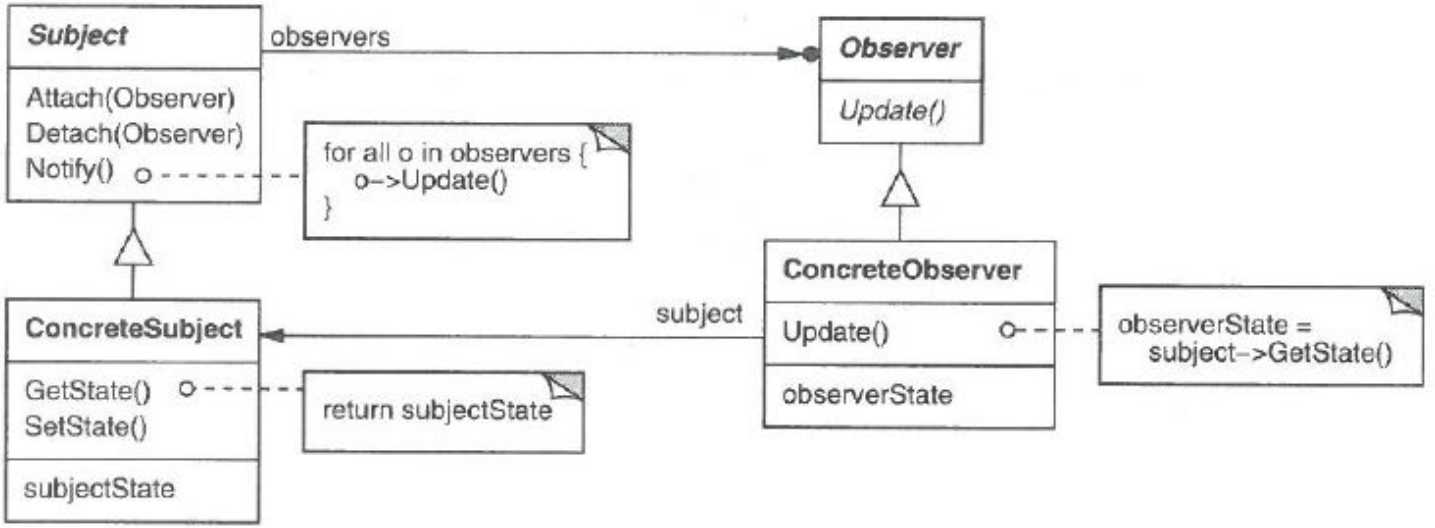
Fasada (Strukturalny)



- Fasada tworzy jeden interfejs do komunikacji ze środowiskiem zewnętrznym
- Interfejs składa się połączenia kilku klas i ich metod
- Zmniejsza skomplikowanie systemu dla odbiorcy

Źródło: “Design Patterns, Elements of Reusable Object Oriented Software” E.Gamma, R. Helm

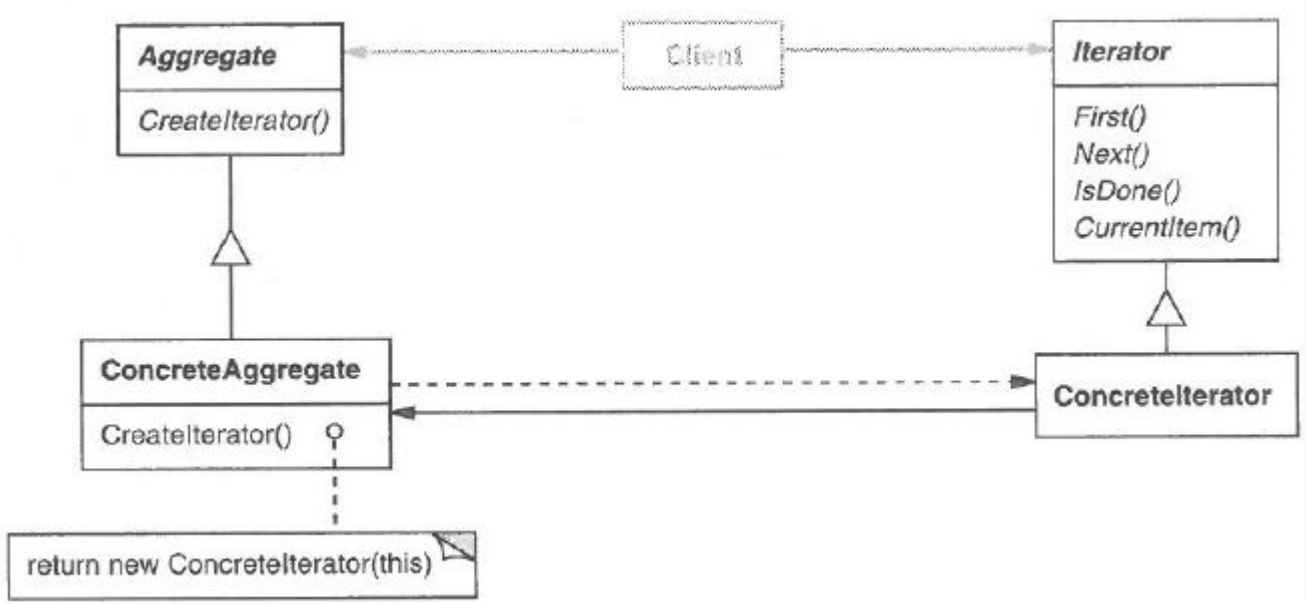
Obserwator (Czynnościowy)



- Subject wysyła do wszystkich obserwujących obiektów informację o swojej zmianie stanu
- Zmniejsza zależność klas od siebie
- Obserwator nie musi znać implementacji klas

Źródło: “Design Patterns, Elements of Reusable Object Oriented Software” E.Gamma, R. Helm

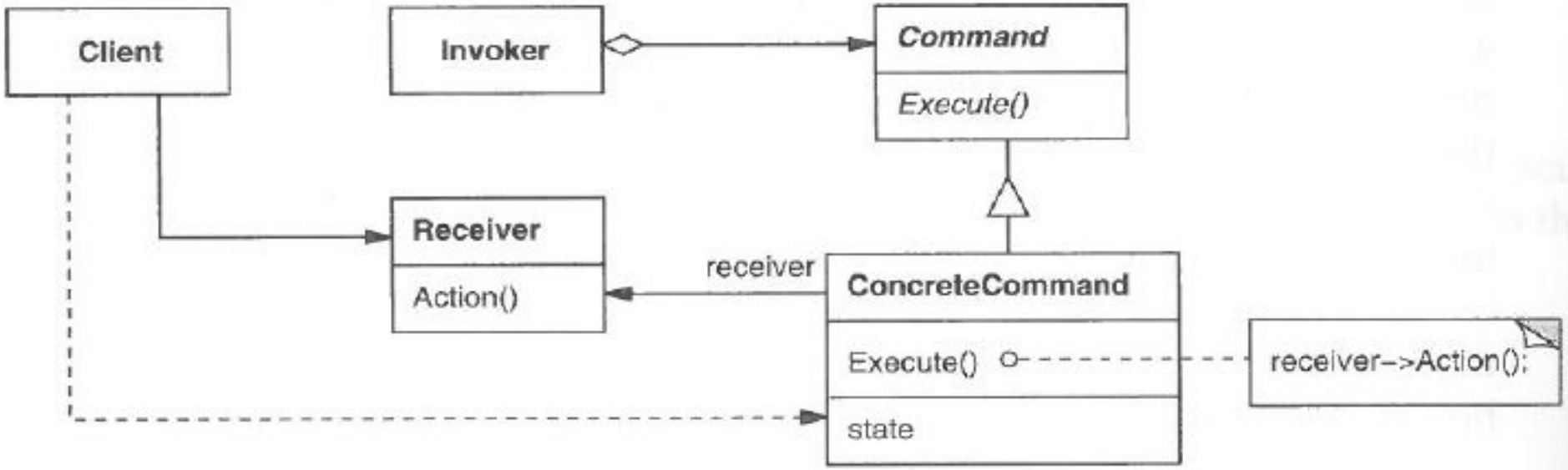
Iterator (Czynościowy)



- Zwraca obiekty iterując po kolekcji
- Zapamiętuje obecną pozycję w iterowanej kolekcji
- Umożliwia różne implementacje kolejności iteracji

Źródło: “Design Patterns, Elements of Reusable Object Oriented Software” E.Gamma, R. Helm

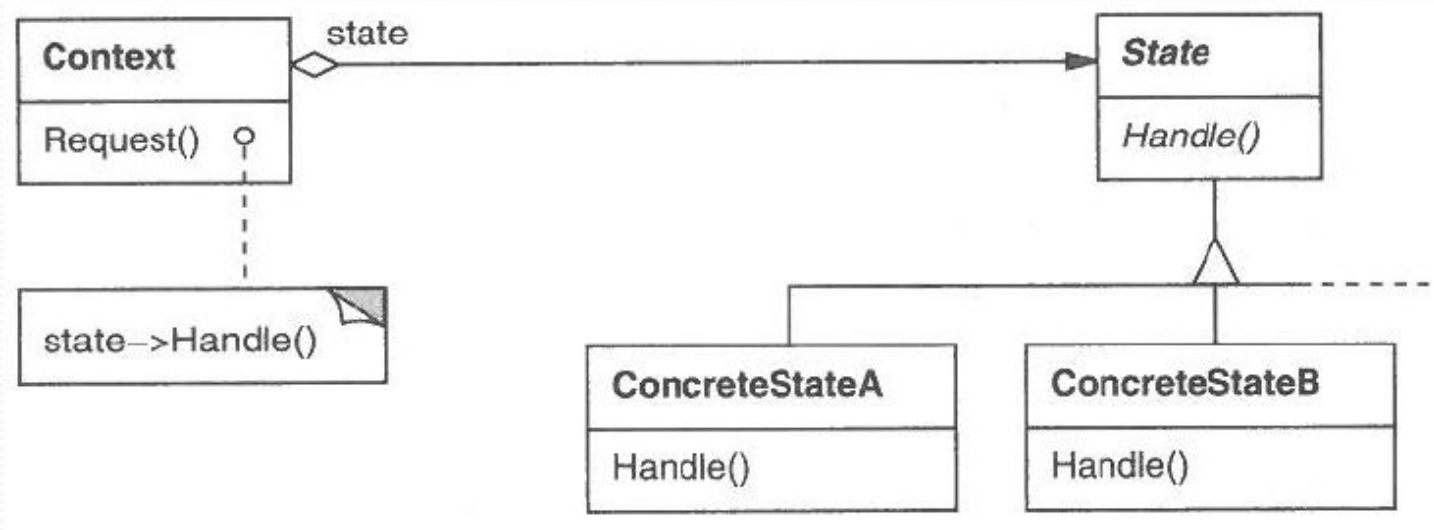
Polecenie (Czynnościowy)



- Akcja jako obiekt
- Oddziela obiekt wywołujący akcję od tego, który ją obsługuje
- Łatwo dodać nowe polecenie

Źródło: “Design Patterns, Elements of Reusable Object Oriented Software” E.Gamma, R. Helm

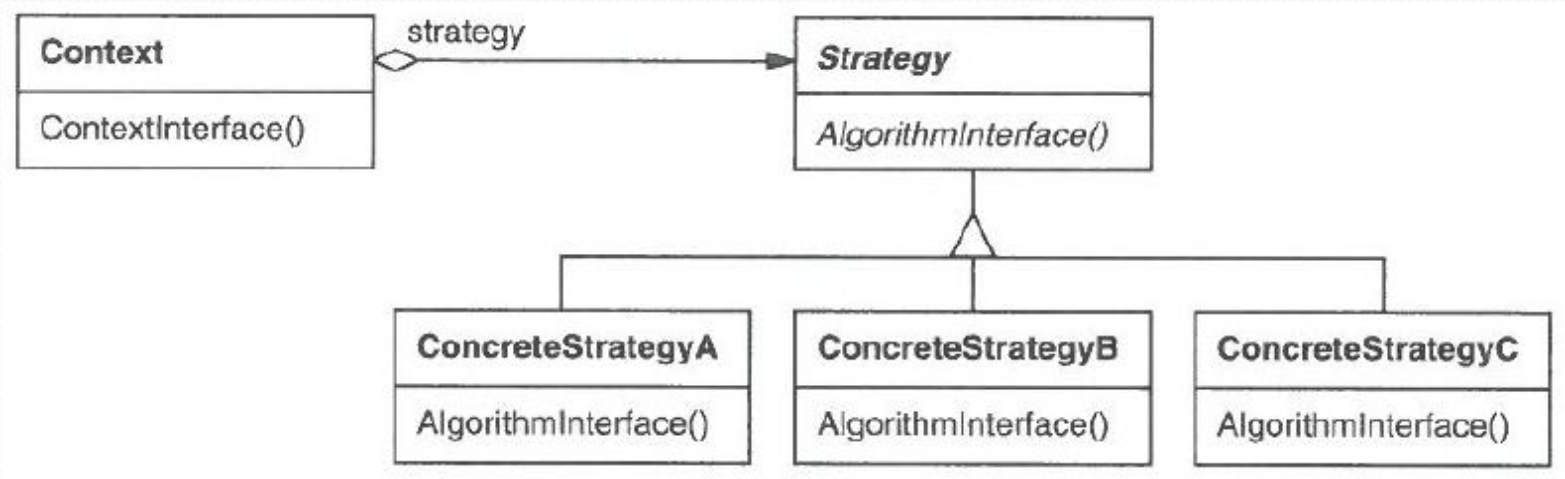
Stan (Czynnościowy)



- Zachowanie obiektu jest zmieniane w zależności od jego stanu

Źródło: “Design Patterns, Elements of Reusable Object Oriented Software” E.Gamma, R. Helm

Strategia (Czynościowy)

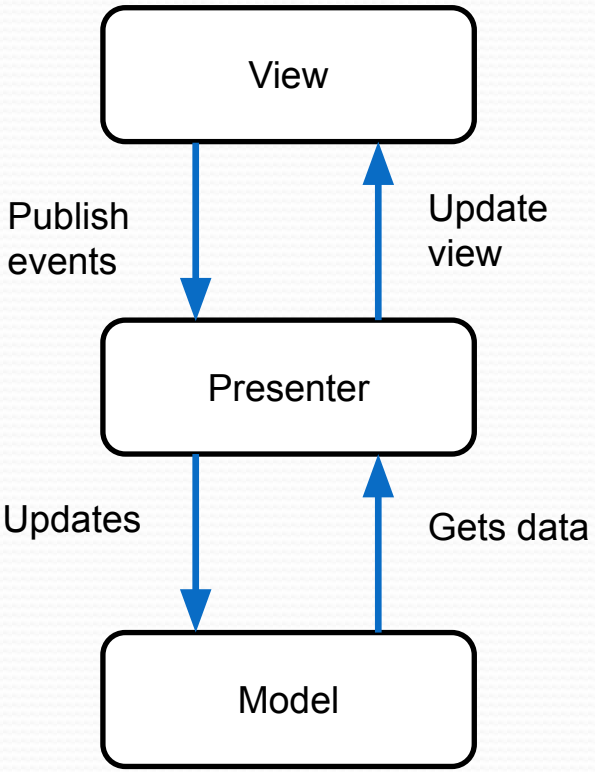


- Istnieje jeden interfejs wspólny dla wszystkich klas
- Jedna klasa (kontekst) używa metod tego interfejsu w swojej implementacji
- W zależności od potrzeb kontekst jest tworzony z wybraną implementacją interfejsu

Źródło: “Design Patterns, Elements of Reusable Object Oriented Software” E.Gamma, R. Helm

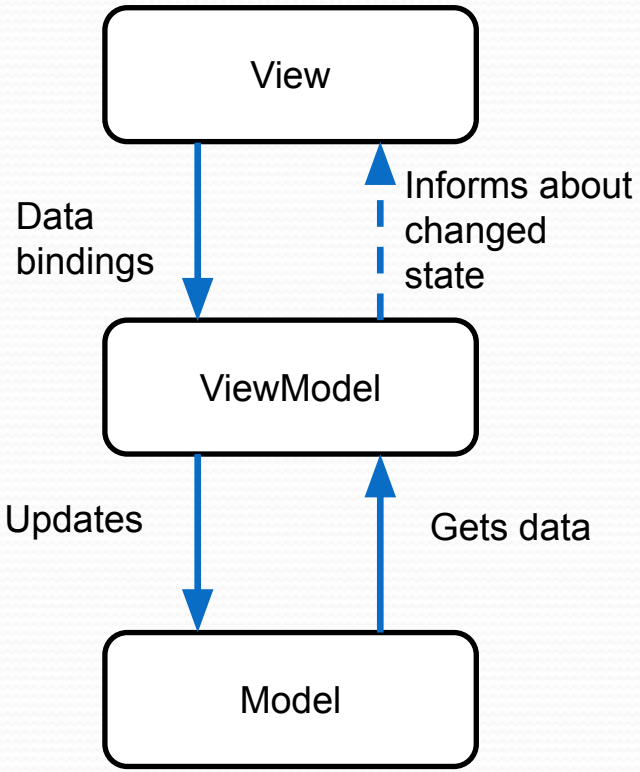
Model-View-Presenter

- Widok
  - Deleguje zdarzenia do Presentera
  - Odświeża wygląd po zmianie Presentera
- Model
  - Odzwierciedla strukturę danych
- Presenter
  - Wykonuje akcje wywołane na widoku
  - Modyfikuje model
  - Zmienia widok



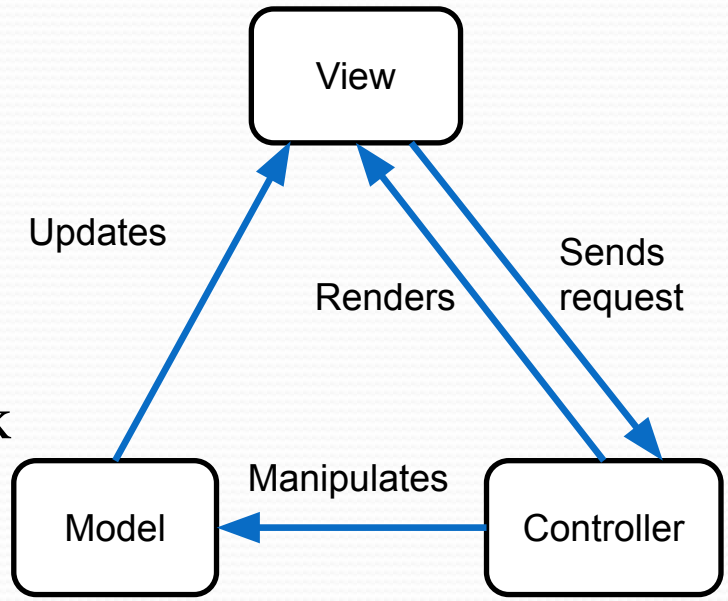
Model-View-ViewModel

- Widok
  - Odczytuje interakcje użytkownika i wysyła zdarzenia do ViewModela za pomocą mechanizmu data binding
  - Odświeża wygląd po zmianie
- Model
  - Odzwierciedla strukturę danych
- ViewModel
  - Przy pomocy bindera odczytuje gesty użytkownika na widoku i wykonuje akcje
  - Modyfikuje stan modelu
  - Za pomocą bindera do publicznych właściwości oraz komend modyfikuje widok



Model-View-Controller

- Widok
  - Wywołuje akcje (requests) na Controlerze
  - Wyświetla Dane zawarte w modelu
- Model
  - Odzwierciedla strukturę danych
  - Przekazuje dane do widoku
- Kontroler
  - Wykonuje akcje żądane przez Widok
  - Procesuje całą logikę i zmienia Model
  - Podpina model do wybranego Widoku i przesyła do użytkownika



Wyrażenia Regularne

- Biblioteka Regex
- Przetwarzanie danych z użyciem wyrażeń regularnych
- Wykorzystanie grup
- Zastosowanie wyrażeń regularnych

Wielowątkowość i asynchroniczność

- Programowanie równoległe
  - Rozdzielenie zadań *(robienie kilku rzeczy naraz na różnych wątkach)*
  - Wielowątkowość *(forma współbieżności)*
- Programowanie asynchroniczne
  - Obiekty typu *future (obietnica wykonania zadania w przyszłości – Task : z void, Task<> z wartością zwrotną)*
- *Programowanie reaktywne*
  - Reactive Extensions (Rx) *(programowanie na zdarzeniach)*
- Wady i zalety
- Kiedy NIE stosować

Wzorce programowania asynchronicznego

- APM - .NET 1.1
  - BeginInvoke/EndInvoke
- EAP - .NET 2.0
  - Events
  - BackgroundWorker
- *TAP - .NET 4.0 (async 4.5)*
  - async/await
  - Task, Task<>
  - CancellationToken

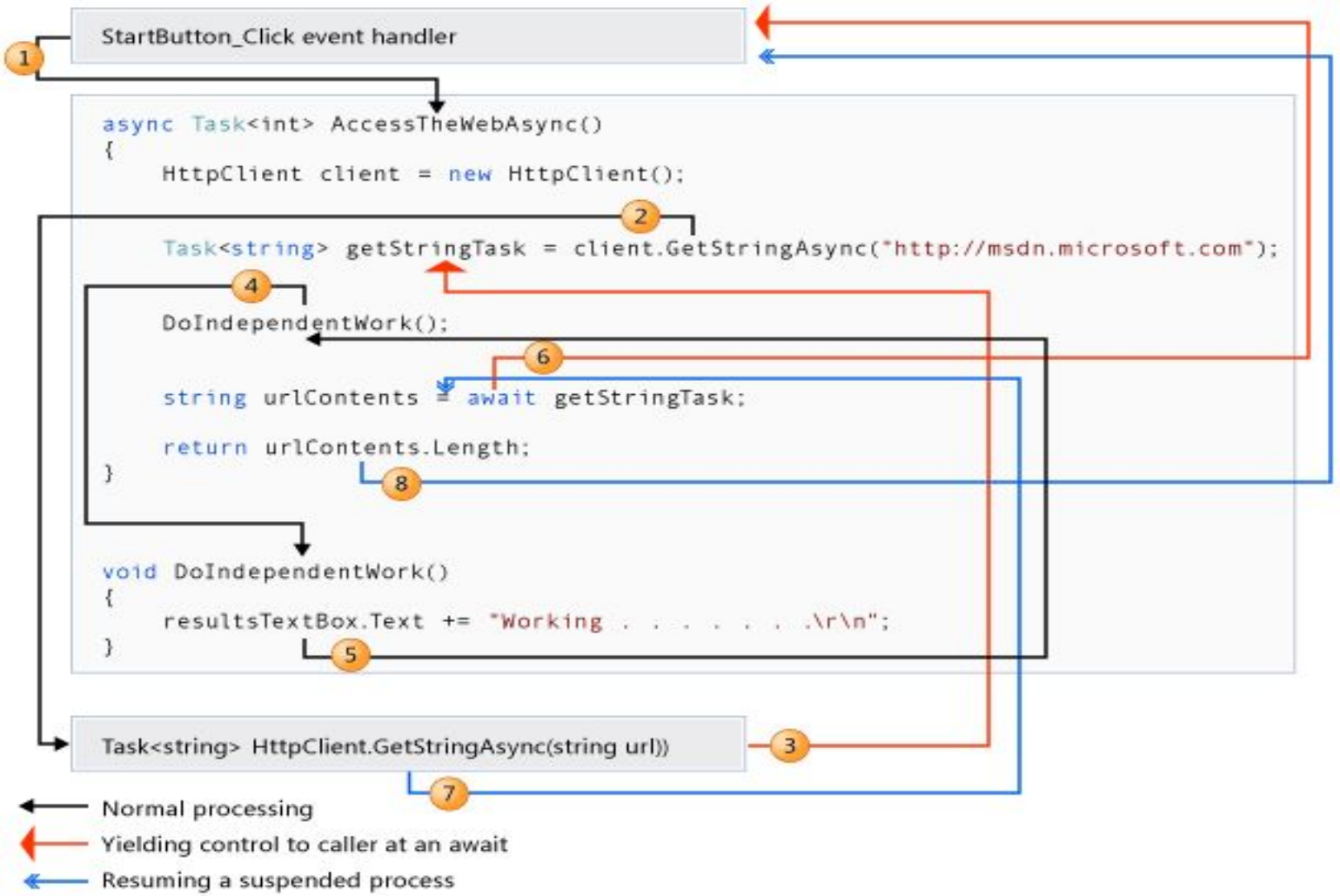
Wątki

- *Czym są wątki (wątek vs proces)*
- *Klasa System.Threading.Thread*
  - *Utworzenie, Uśpienie, Przerwanie*
- *Sekcje krytyczne*
- *Synchronizacja*
  - *Join/Abort/Sleep*
  - *ThreadPool*
  - *Interlocked*
  - *Monitor (lock)*
  - *Semafor*
  - *Bariera*

Zadania

- Wątek vs Zadanie
- Klasy *Task* i *Task<>*
  - *Wykorzystanie*
  - *Synchronizacja*
  - *async await*
- Fabryka zadań (*TaskFactory*)
- Stan zadań, przerywanie działania

async/await - przetwarzanie



Źródło: https://docs.microsoft.com/pl-pl/dotnet/csharp/programming-guide/concepts/async/

- Delegaty
  - BeginInvoke/EndInvoke
    - IAsyncResults
    - AsyncCallBack
- Parallel
  - *For*
  - *Foreach*
  - *Invoke*
- PLINQ
  - Czym jest Parallel LINQ
  - *AsParallel*
  - *ForAll*

Dobre praktyki

- zagnieżdżone blokady
- volatile
- async Task/Task<>
- ConfigureAwait(false)
- Ustawianie timeout'u

Optymalizacja

- lock vs SpinLock vs no locks
- klasy slim (SemaphoreSlim, ReaderWriterLockSlim)
- Task.Run vs Task.Factory.StartNew vs new Task
- PLINQ - AsParallel
- Task vs ValueTask - .NET Core & .NET only

Podsumowanie