



# PASSION *for* TECHNOLOGY

Zostań developerem C# / .NET

## Paweł Biesiada

- Software developer, Scrum Master, IT trainer, Sii Poznań, CC Digital,
- .NET Developer since 11.2011
- Working for Sii since 04.2016,
- pbiesiada@sii.pl



# Zostań developerem C# / .NET



CLI

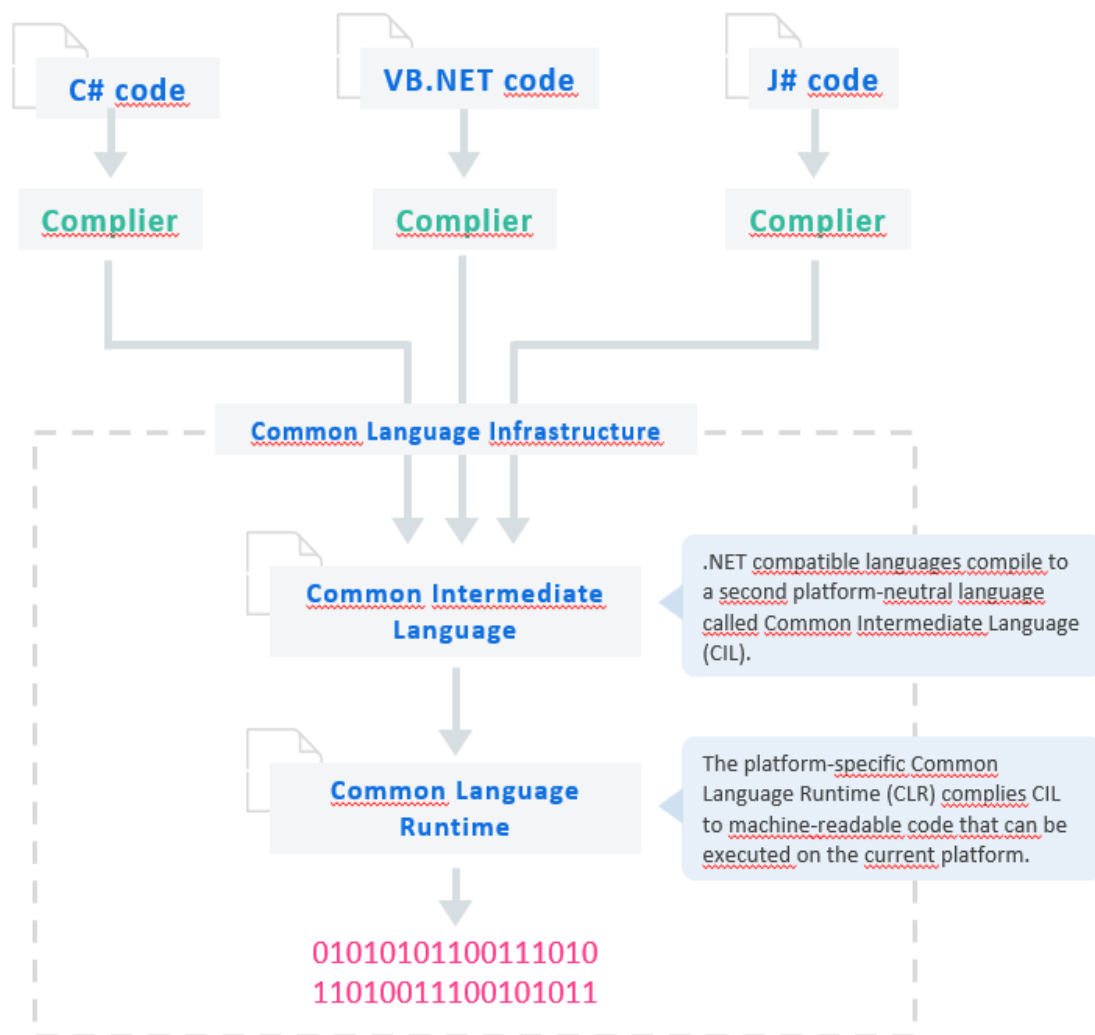
Common Language Infrastructure

CIL

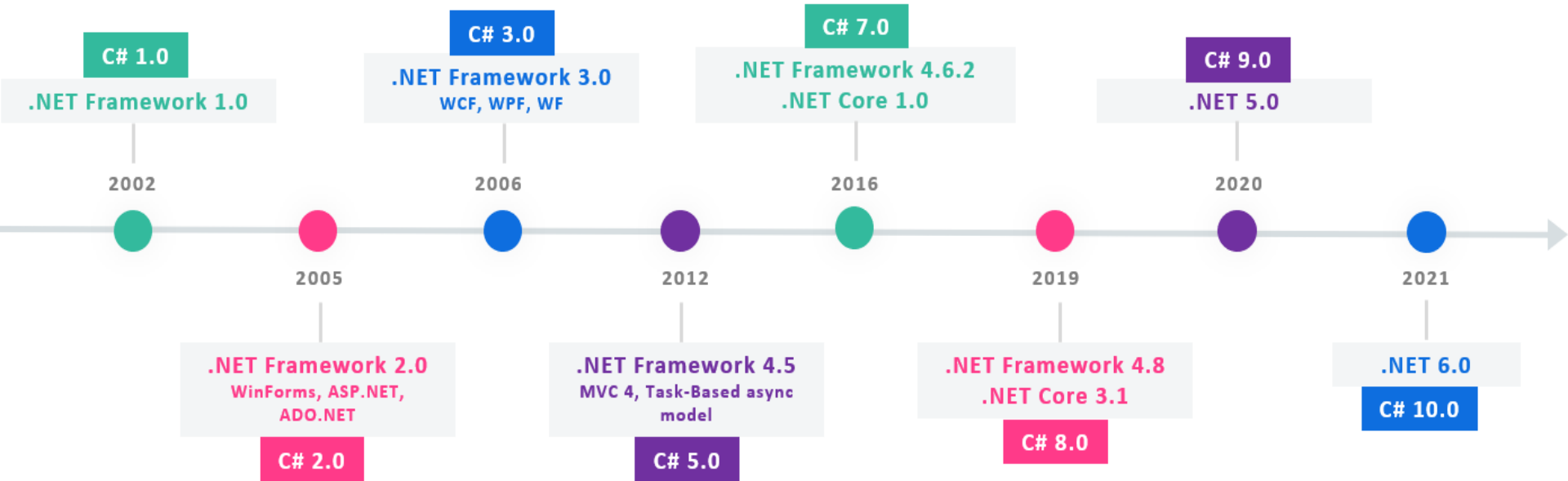
Common Intermediate Language

CLR

Common Language Runtime

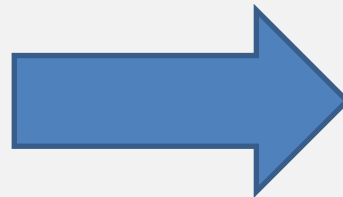


# Zostań developerem C# / .NET



## Składowe klasy

- **Pole**
- **Metoda**
- **Właściwość**
- Indeksator
- **Konstruktor**
- Destruktor
- Zdarzenie
- Operator



**field**  
**method**  
**property**  
indexer  
**constructor**  
finalizer  
event  
operator



## Typy danych

### Typy wartościowe

(mają wartość domyślną)

- Proste
  - int, short, long, sbyte, byte, ushort, uint, ulong, char, float, double, decimal, bool
- Wyliczeniowy – enum
- Struktura - struct
- Nullable
- Tuple

### Typy referencyjne

(mogą nie mieć wartości - *null*)

- Klasa - *class*
  - object, string
- Interfejs - *interface*
- Kolekcje
  - array, list, dictionary
- Delegaty - *delegate*

## Typy danych

Type	Size in Bytes	Description	Minimum	Maximum	Example
bool	1	Named literal	false	true	
sbyte	1	Signed byte	-128	127	
byte	1	Unsigned byte	0	255	
short	2	Signed short integer	-32768	32767	
ushort	2	Unsigned short	0	65535	
int	4	Signed integer	-2147483648	2147483647	
uint	4	Unsigned integer	0	4294967295	
long	8	Signed long int	-9.2233E+18	9.2233E+18	
ulong	8	Unsigned long int	0	18446E+19	
char	2	Unicode character, contained within single quotes.	0	128	a,b,4
float	4	floating point	-3.402823E+38	3.402823E+38	3.14159
double	8	Floating point	-1.7976931E+308	1.7976931E+308	3.14159
decimal	16	Floating point, accurate to the 28th decimal place.	-7.9228E+24	7.9228E+24	
object	8+	Base type for all other types	n/a	n/a	n/a
string	20+	Immutable character array	n/a	n/a	"Hello World"
DateTime	8	Represents an instant in time, typically expressed as a date and time of day.	00:00:00 01/01/0001	23:59:59 31/12/9999	14:289:35 08/05/2010



## Instrukcje

- Deklaracje – zmiennych, const
- Warunkowe – if, switch
- Pętle – for, foreach, while, do/while
- Skoku – return, yield, throw, continue, break
- Wrappery na instrukcje – using, lock



## Modyfikatory dostępu (hermetyzacja)

- public – dostęp nieograniczony
- private – dostęp tylko wewnątrz klasy
- protected – dostęp wewnątrz klasy i klasach dziedziczących
- internal – dostęp w obrębie jednego assembly
- protected internal – dostęp wewnątrz klasy, klasach dziedziczących, lub w obrębie danego assembly
- private protected (C# 7.2) – dostęp wewnątrz klasy, klasach dziedziczących w obrębie danego assembly

## Klasa vs struktura

### class

- Typ referencyjny
- Wspiera dziedziczenie i polimorfizm
- Posiada destruktora
- Może mieć zdefiniowany konstruktor bezparametrowy
- Przechowywany na stercie
- Do metody przekazujemy tylko wskaźnik na obiekt w pamięci

### struct

- Typ wartościowy
- Nie można dziedziczyć ze struktury
- Nie posiada destruktora
- Zdefiniowany konstruktor musi posiadać parametry
- Przechowywany na stosie
- Do metody przekazujemy cały obiekt
- Szybki – ale dla “małych” obiektów



## Elementy statyczne

- Słowo kluczowe *static*
- Zmienna statyczna
- Metoda statyczna
- Klasa statyczna



# Paradygmaty programowania obiektowego

- Abstrakcja
- Hermetyzacja (Enkapsulacja)
- Dziedziczenie
- Polimorfizm

## Interfejs vs klasa abstrakcyjna

### Interface

- Wspiera wielokrotne dziedziczenie
- Nie posiada zmiennych klasy (field)
- Zawiera tylko sygnatury składowych klasy
- Nie posiada modyfikatorów dostępu (wszystkie domyślnie są jako public)
- Nie można zdefiniować statycznej składowej klasy (C# 8.0)

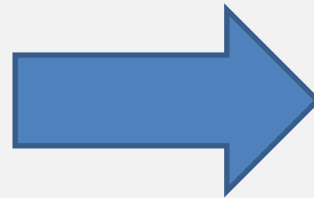
### Abstract class

- Można dziedziczyć tylko z jednej klasy
- Może posiada wszystkie elementy klasy
- Może zawierać całe implementacje lub tylko sygnatury składowych klasy
- Można deklarować różne modyfikatory dostępu
- Tylko w pełni zadeklarowana składowa klasy może być statyczna

## LINQ

### Language integrated query

- Typy generyczne
- Delegaty
- Metody rozszerzające
- Wyrażenia lambda
- Metody anonimowe
- Typy anonimowe



Generics  
Delegates  
Extension methods  
Lambda expressions  
Anonymous methods  
Anonymous types



## Metody rozszerzające

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



## Delegaty

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





## SOLID

- **S** – Single responsibility principle
- **O** – Open/Closed principle
- **L** – Liskov substitution principle
- **I** – Interface segregation principle
- **D** – dependency inversion principle

# Aplikacje desktopowe - WPF

- Technologia
  - Model MVVM
- User interface (UI)
  - Język xaml
- Kontrolki
  - Zdarzenia
  - binding

# Structured Language Query (SQL)

- Relacyjna baza danych
- Składowe bazy danych
  - Tabele i indeksy
  - Widoki
  - Procedury składowane i funkcje
  - Wyzwalacze
- T-SQL
  - Transakcje



## Entity Framework

- Object-relational Mapping (ORM)
- Tryby pracy i tworzenia połączeń
  - Database first
  - Model first (.NET Framework)
  - Code first
- Klasy *ContextDb*, *SetDb*
- Ładowanie danych
  - Lazy loading
  - Eager loading



## Praca z plikami

- System.IO (File, Path, Directory)
- Pliki tekstowe i binarne
- Strumienie danych (Stream)
  - FileStream, MemoryStream
- Odczytywanie z pliku
  - StreamReader
- Zapis do pliku
  - StreamWriter
- Klauzula *using*



## Architektura klient-serwer

- Serwer (thick)
  - Procesuje przychodzące zadania
  - Odsyła wyniki do klienta
- Klient (thin)
  - Wysyła zapytania do serwera
  - Odbiera wyniki od serwera
  - Renderuje wygląd aplikacji (w przypadku aplikacji z interfejsem graficznym)
- Warstwa komunikacji
  - Musi być ustalona publicznie i jednolita dla obu stron
  - http (REST/SOAP), TCP, pipe

# Windows Communication Foundation (WCF)

- SOAP (Simple Object Access Protocol)
- Warstwa komunikacji
  - xml
- Koncept ABC
  - Adress (gdzie?)
  - Binding (jak?)
  - Contract (Co?)
- WSDL



## Representational State Transfer (REST)

- Warstwa komunikacji
  - JSON
- Bezstanowość protokołu HTTP
- Metody
  - GET
  - POST
  - PUT / PATCH
  - DELETE
- Swagger





## Aplikacje webowe MVC

- Technologia
  - ASP.NET
  - Model MVC
- User interface (UI)
  - Język html, css
- Zdarzenia



# Refleksja

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



# Atrybuty

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

## Testy jednostkowe

- Testy powinny testować pojedynczą funkcjonalność
  - Jedna funkcjonalność może mieć wiele różnych testów
- Testy powinny być wykonywane w izolacji, bez zewnętrznej zależności
  - Zależności powinny być wstawiane za pomocą mocków lub stubów
- Testy powinny być powtarzalne i niezawodne
- Testy powinny być szybkie
- Testy powinny być łatwe, czytelne

# Wielowątkowość i asynchroniczność

- Programowanie równoległe
  - Rozdzielenie zadań (przetwarzanie 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 zwrótną
- Programowanie reaktywne
  - Reactive Extensions (Rx) (programowanie na zdarzeniach)
- Wady i zalety
- Kiedy NIE stosować



## Wątki

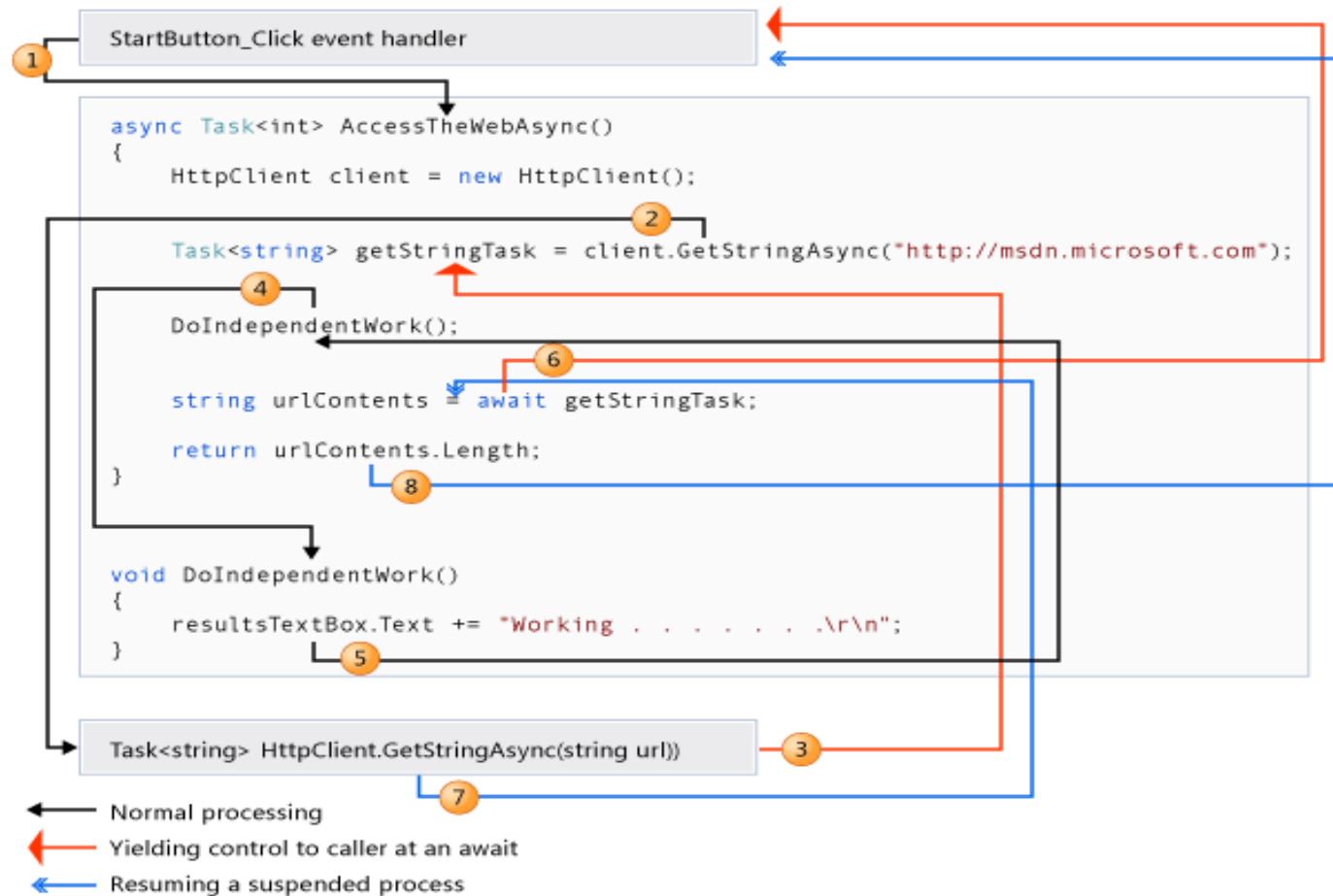
- Czym są wątki (wątek vs proces)
- Klasa *System.Threading.Thread*
  - Utworzenie, uśpienie, przerwanie
- Sekcje krytyczne
  - lock
- Synchronizacja
  - Join, Abort, Sleep
  - ThreadPool
  - Interlocked
  - Monitor
  - Semafor



## Zadania

- Wątek vs zadanie
- Klasy *Task* i *Task<>*
  - Wykorzystanie
  - Synchronizacja
  - `async / await`
- Fabryka zadań
  - `TaskFactory`
- Stan zadań
- Przerywanie działania

## async / await





# Structured Language Query (SQL)

- Relacyjna baza danych
- Składowe bazy danych
  - Tabele i indeksy
  - Widoki
  - Procedury składowane i funkcje
  - Wyzwalacze
- T-SQL
  - Transakcje

## Typy danych – MsSQL

- Typy numeryczne
  - BIT, TINYINT, SMALLINT, INT, BIGINT, FLOAT, REAL, DECIMAL, MONEY
- Typy tekstowe
  - CHAR, VARCHAR, NCHAR, NVARCHAR, BINARY, VARBINARY, TEXT
- Daty
  - DATE, TIME, DATETIME, DATETIME2, TIMESTAMP
- Pozostałe
  - UNIQUEIDENTIFIER, XML, CURSOR

## Operacje SQL

- DQL (Data Query Language)
  - SELECT
- DML (Data Manipulation Language)
  - INSERT, UPDATE, DELETE
- DDL (Data Definition Language)
  - CREATE, ALTER, DROP, TRUNCATE
- DCL (Data Control Language)
  - GRANT, DENY, REVOKE
- TCL (Transaction Control Language)
  - COMMIT, ROLLBACK, SET TRANSACTION



## SELECT

- Pobieranie danych z tabeli
  - Używanie indeksów
  - WITH
- Łączenie tabel
  - INNER, OUTER JOIN
- Filtrowanie danych
  - WHERE
- Grupowanie
  - GROUP BY oraz HAVING
- Sortowanie
  - ORDER BY



## T-SQL

- Tabele
- Widoki
- Tabele tymczasowe
- Procedury składowane
- Funkcje
- Wyzwalacze

# Procedura vs Funkcja

## Procedura składowana

- Może modyfikować dane
- Może zwracać wartość, ale nie musi
- Plan wykonania jest zapisywany po stronie serwera
- Nie można jej użyć w zapytaniu

## Funkcja

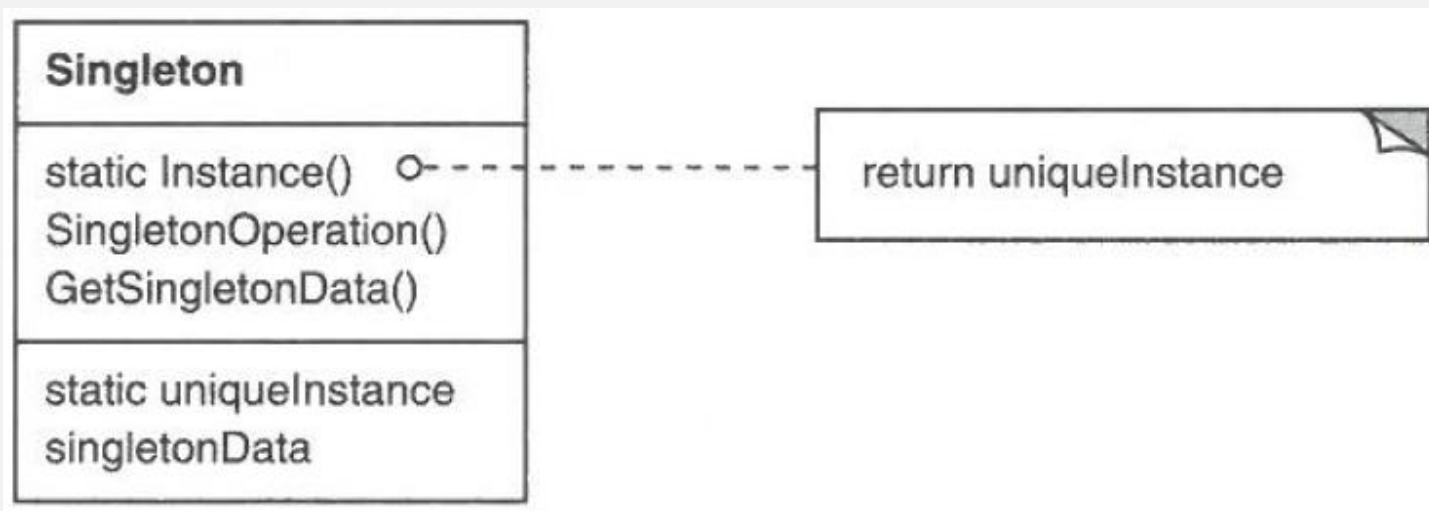
- Nie może modyfikować danych
- Zawsze zwraca wartość
- Nie posiada destruktora
- Plan wykonania może być cachowany po stronie serwera
- Może być użyta jako część kwerendy



## Wzorce projektowe

- **Kreacyjne**
  - Singleton, Fabryka abstrakcyjna, Fabryka metod
- **Strukturalne**
  - Fasada
- **Czynnościowe**
  - Obserwator, Stan, Strategia, Polecenie, Iterator

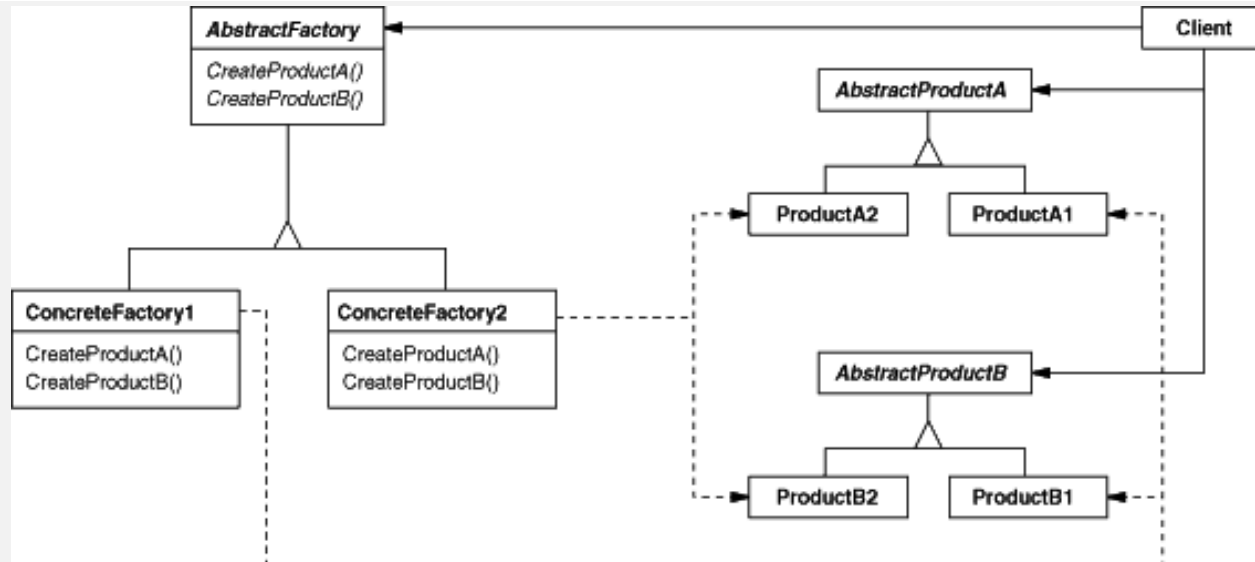
## Singleton (Kreacyjny)



- Istnieje tylko jedna instancja klasy



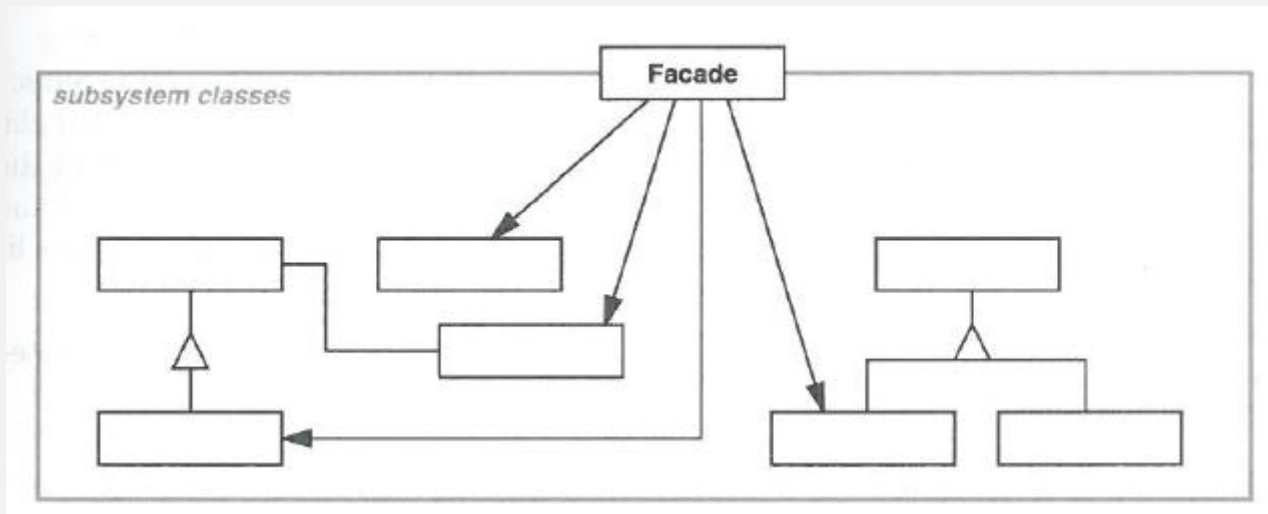
## Fabryka abstrakcyjna (Kreacyjny)



- 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

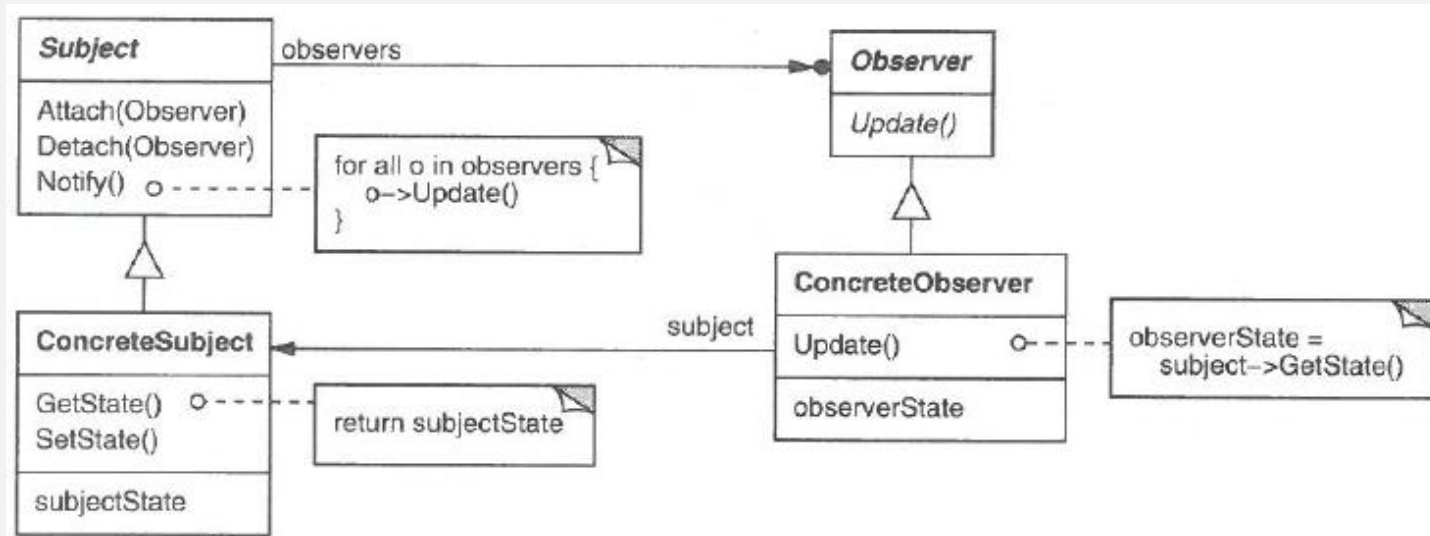
## Fasada (Strukturalny)



- Fasada tworzy jeden interfejs do komunikacji na zewnątrz
- 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

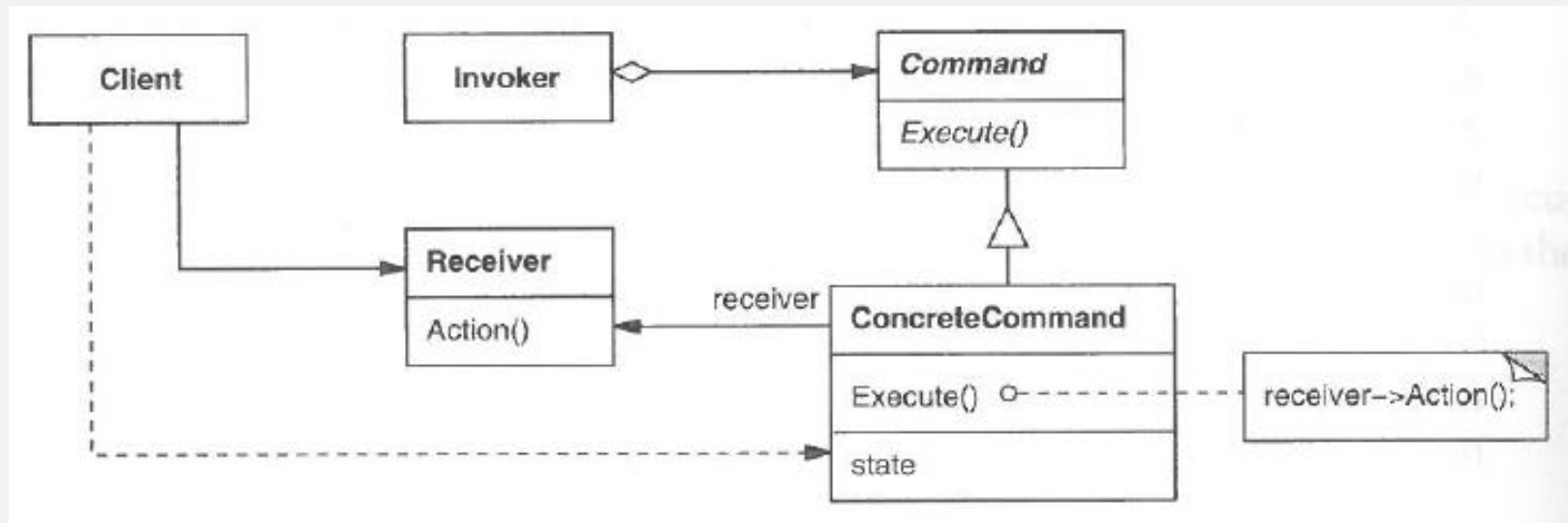
## Obserwator (Czynnościowy)



- Obserwator wysyła do wszystkich obserwowanych obiektów informację o zmianie stanu obiektu
- Zmniejsza zależność klas od siebie
- Obserwator nie musi znać implementacji klas

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

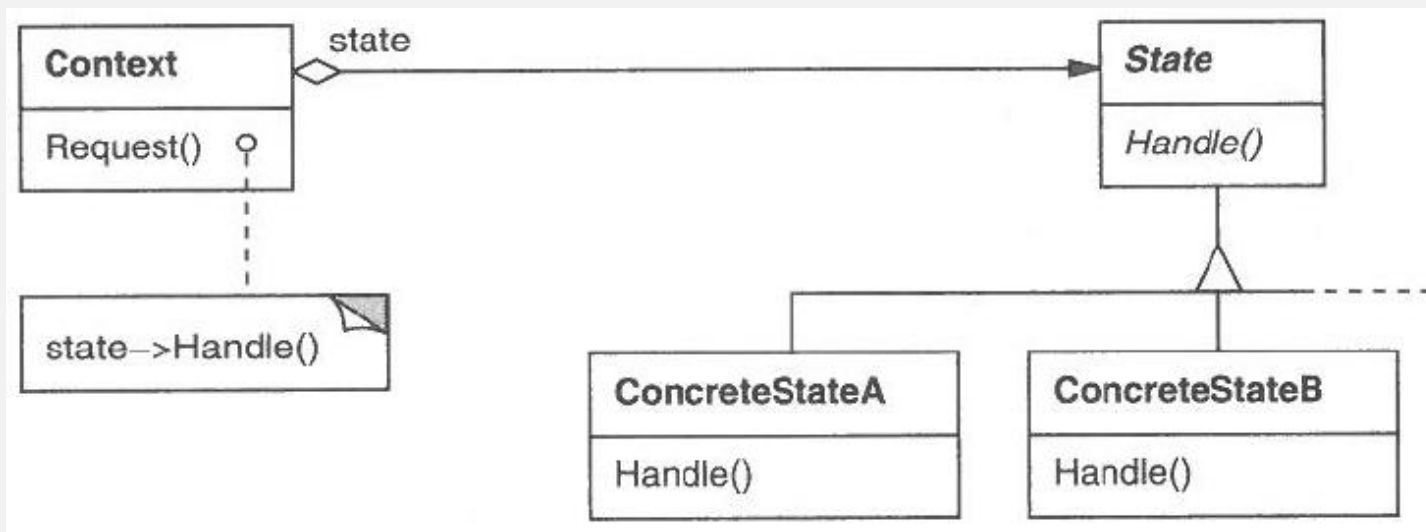
## Polecenie (Czynnościowy)



- 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

# Wstrzykiwanie zależności

- Dependency Injection (DI)
- Implementacja zasady D – Dependency inversion
- Typy DI:
  - przez konstruktor
  - przez metodę
  - przez właściwość
- Kontenery: Unity, Autofac, Ninject

## GIT

- Repozytorium kodu źródłowego
- Baza rozproszona
- Komendy:
  - pull
  - push
  - fetch
  - commit
- Tworzenie branchy, mergowanie
- Rozwiązywanie konfliktów
- Sprawdzanie kodu z wykorzystaniem Pull Request



# Wyrażenia regularne

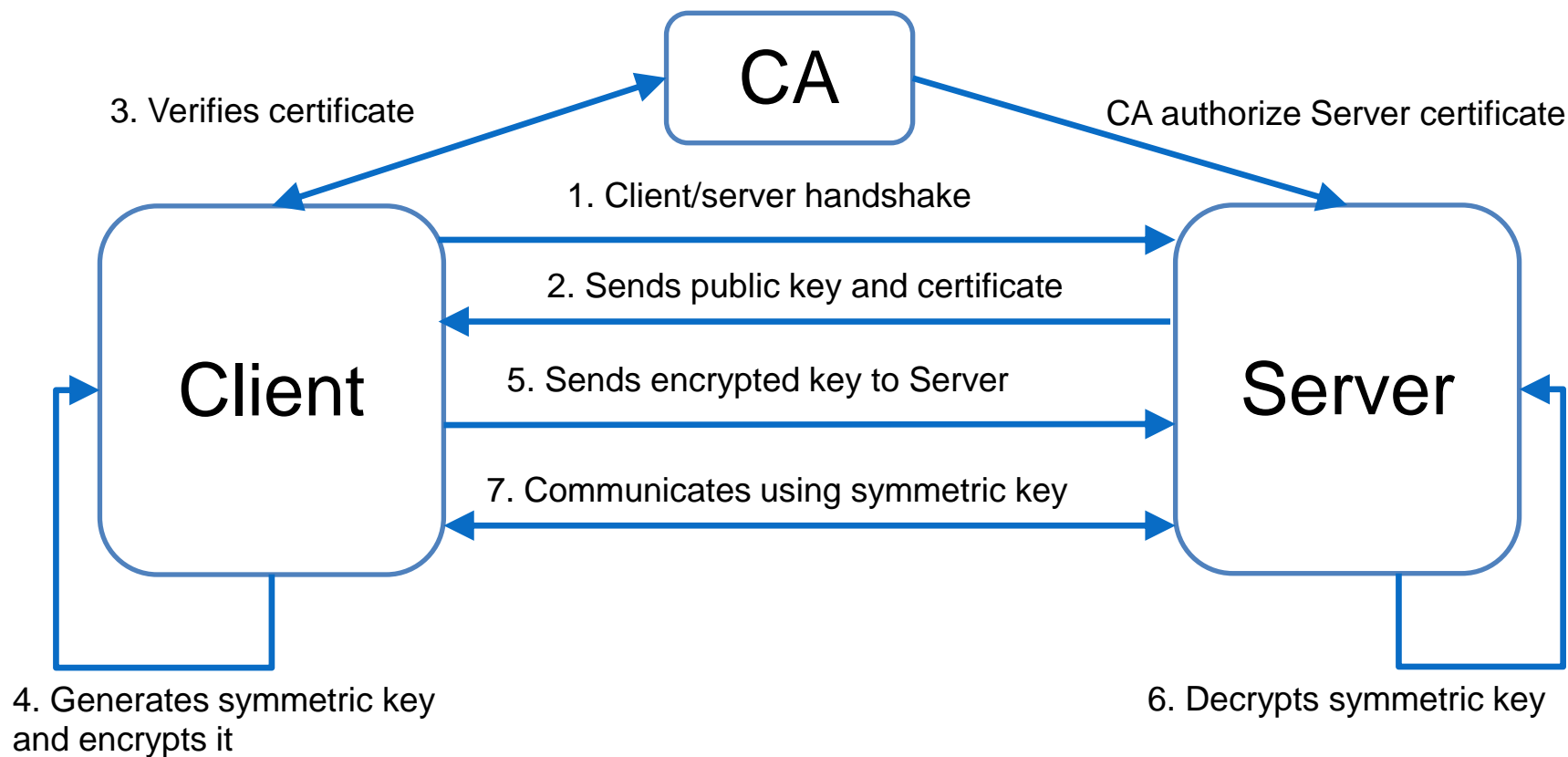
- Biblioteka Regex
- Przetwarzanie danych z użyciem wyrażen regularnych
- Wykorzystanie grup
- Zastosowanie wyrażen regularnych



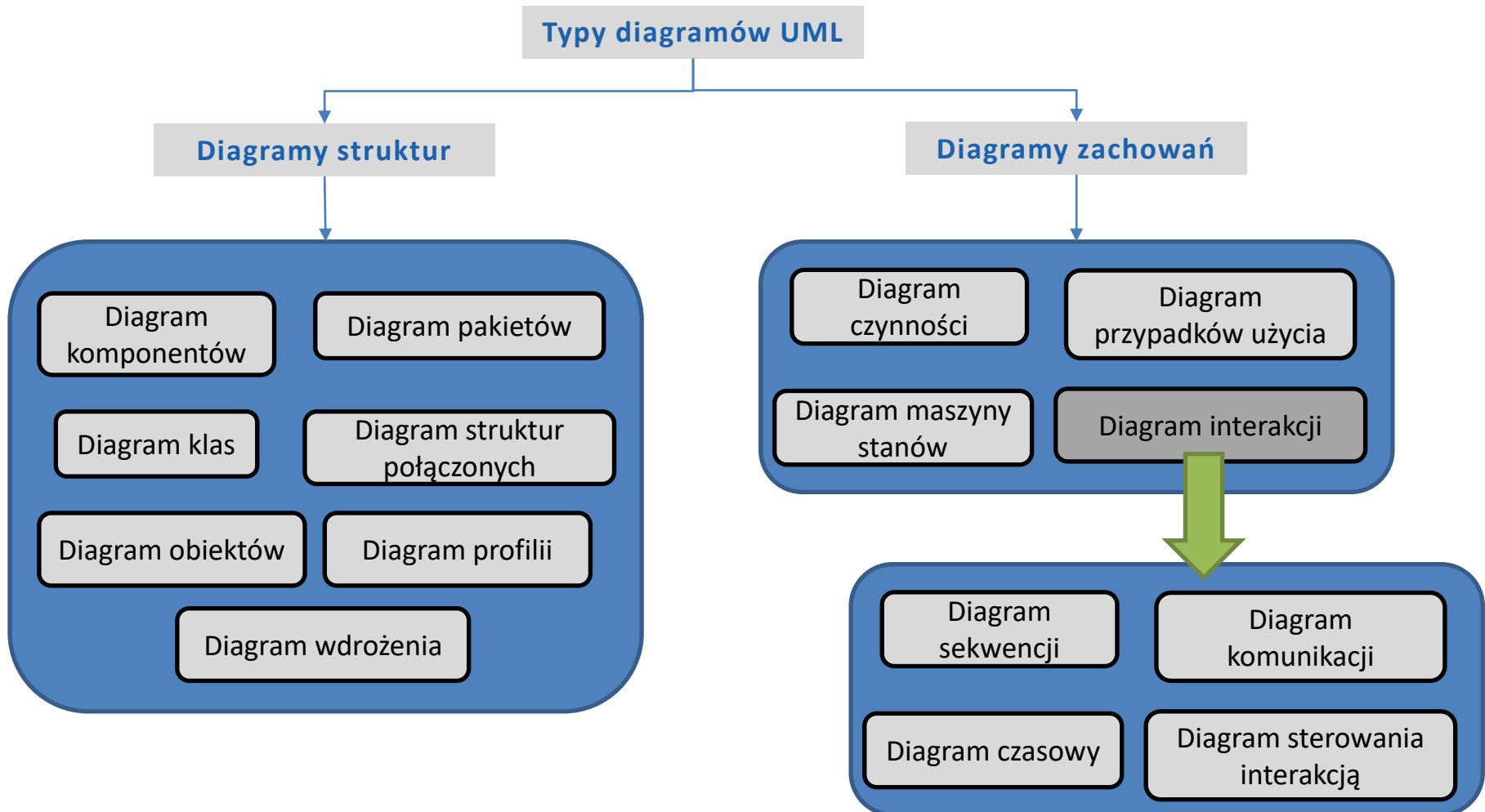
# Szyfrowanie i hashowanie danych

- Szyfrowanie:
  - Synchroniczne
    - AES, DES, RijndaelManaged
  - Asynchroniczne
    - RSA, DSA
- Hashowanie
  - MD5, SHA

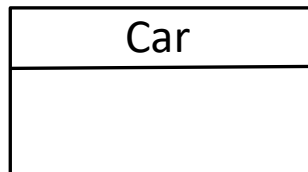
## TLS / SSL



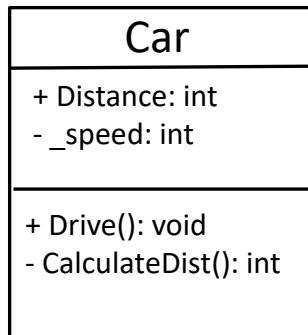
## Diagramy UML







## Diagram klas

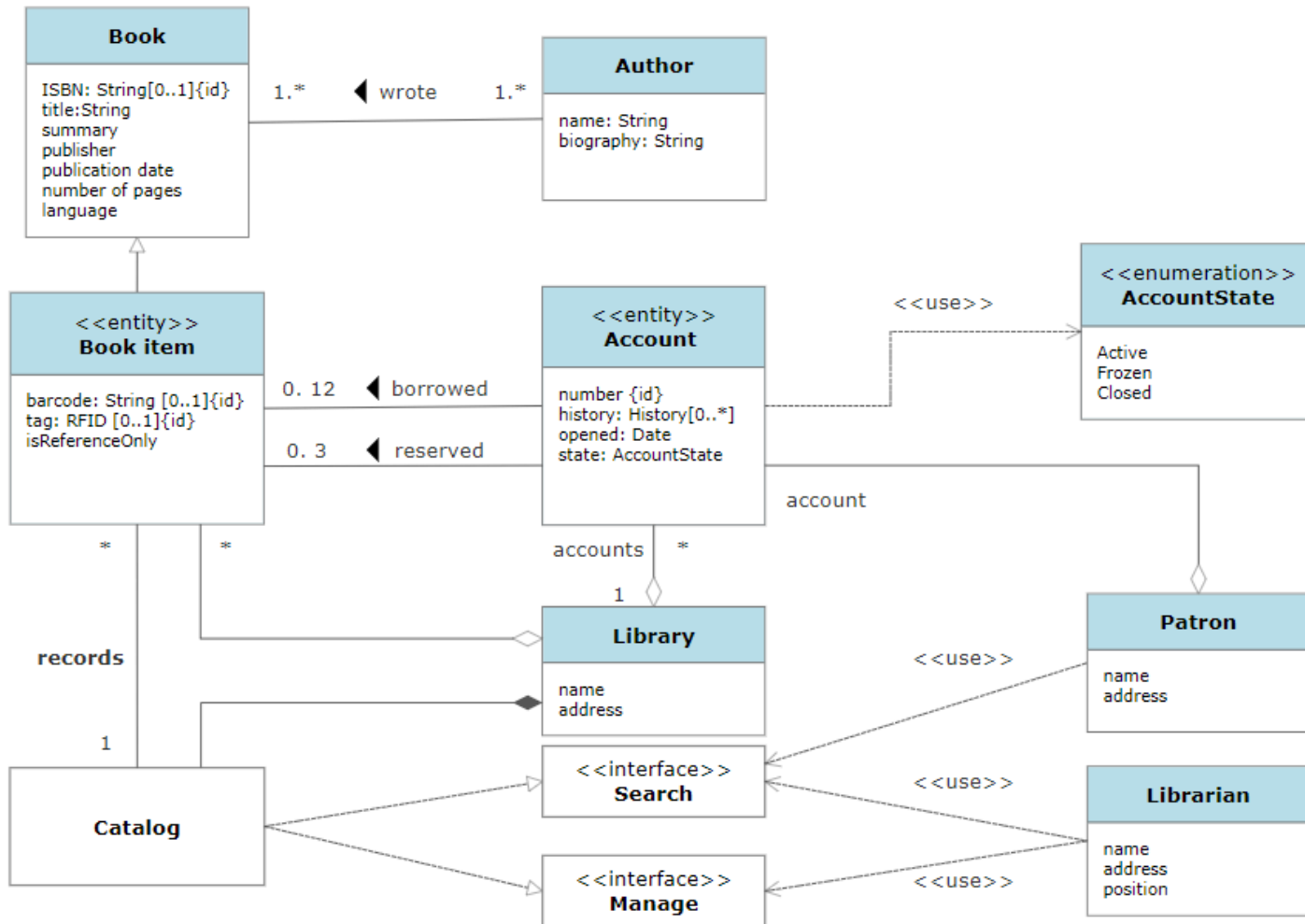


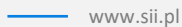
Postać prosta



Postać szczegółowa

Oznaczenie	Opis
 Zależność	Gdy jedna klasa chwilowo wykorzystuje drugą, lub wie o jej istnieniu
 Asocjacja	Gdy jedna klasa wykorzystuje drugą, ale nie są niezależne
 Agregacja	Gdy klasa zawiera drugą klasę, ale współdzielili odwołanie do niej z inną
 Kompozycja	Gdy klasa zawiera drugą klasę i są od siebie zależne
 Dziedziczenie	Gdy jedna klasa jest rozszerzeniem drugiej

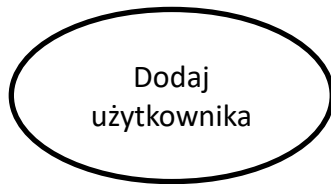








## Diagram przypadków użycia

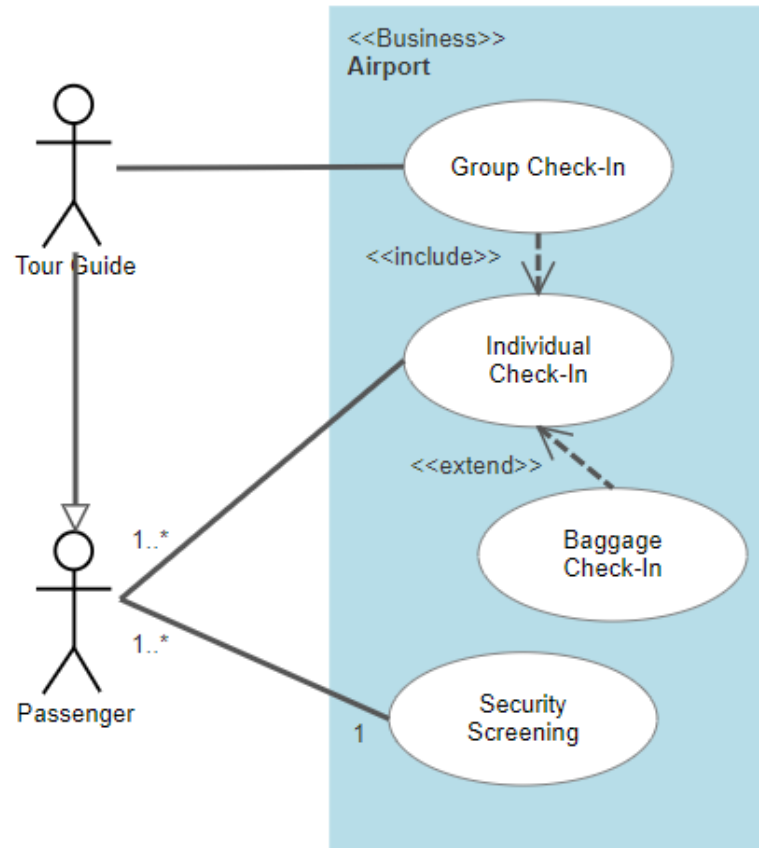


Aktor



Przypadek użycia

Oznaczenie	Opis
Asocjacja 	Połączenie aktora z przypisanymi mu przypadkami użycia
Asocjacja skierowana 	Połączenie aktora z przypadkami użycia wskazujące kto inicjuje zdarzenie
Związek zawierania -- <<include>> -- 	Sytuacja, gdy jeden przypadek użycia rozszerza funkcjonalność bazowego przypadku użycia o nową funkcjonalność
Związek rozszerzania ← -- <<extends>> -- 	Sytuacja, gdy jeden przypadek użycia rozszerza daną funkcjonalność bazowego przypadku

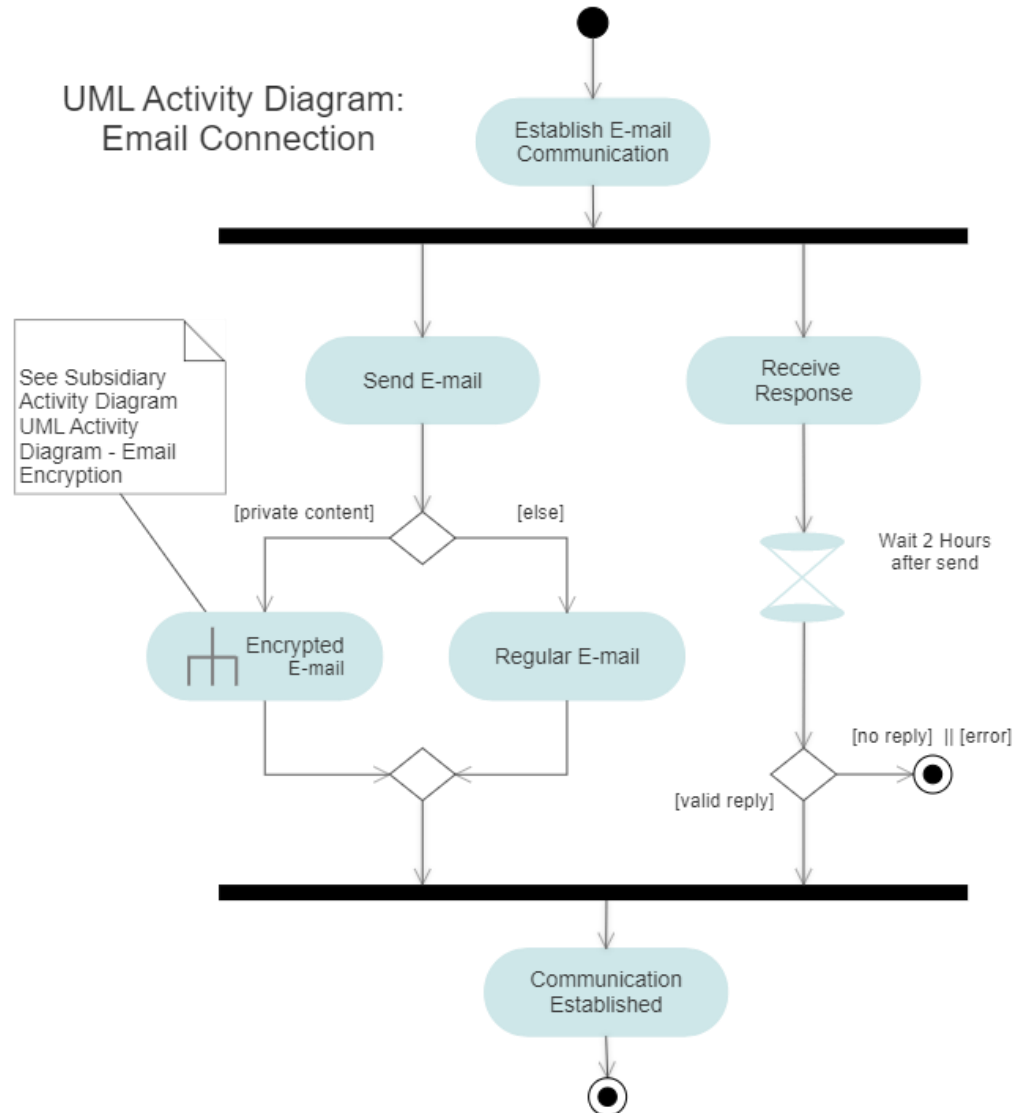




## Diagram aktywności

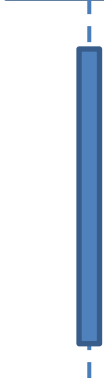
Oznaczenie	Znaczenie	Opis
	Czynność	Węzeł aktywności prezentuje process modelowanego systemu
	Przepływ sterujący	Połączenie aktora z przypadkami użycia wskazujące kto inicjuje zdarzenie
	Decyzja	Węzeł decyzyjny umożliwia dokonanie wyboru pomiędzy kilkoma możliwościami
	Start	Węzeł początkowy określa miejsce rozpoczęcia aktywności – może być tylko jeden
	Koniec	Węzeł końcowy określa miejsce zakończenia aktywności – może być ich kilka

UML Activity Diagram:  
Email Connection



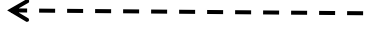



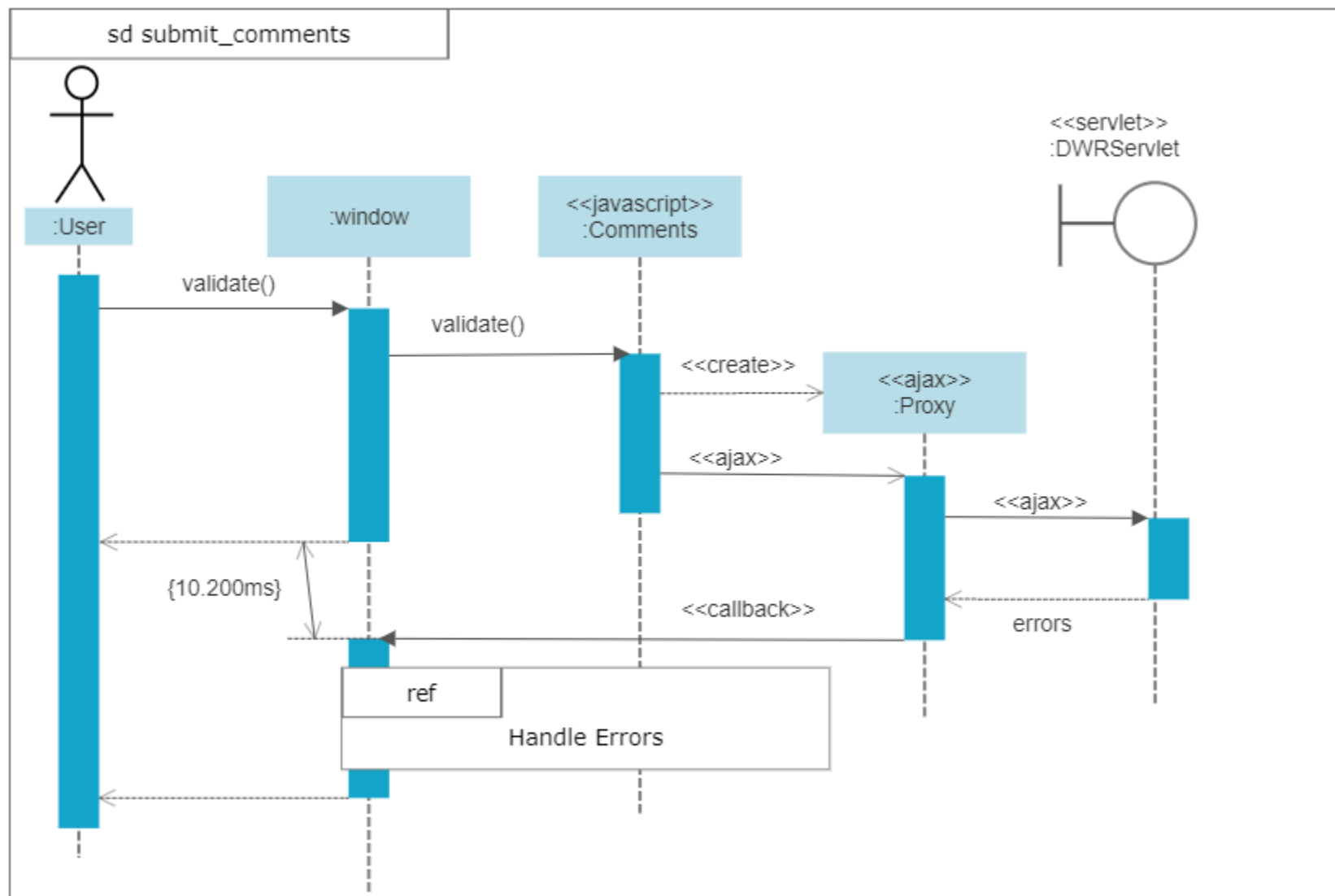
## Diagram sekwencji

Moduł



Linia życia

Oznaczenie	Opis
	Wiadomość synchroniczna
	Wiadomość asynchroniczna
	Odpowiedź
	Wiadomość do samego siebie



A woman with blonde hair and glasses is sitting at a desk in an office, smiling at the camera. She is wearing a blue long-sleeved shirt. In front of her is a laptop and a large monitor displaying a dashboard with charts. The entire image is covered with a semi-transparent blue filter. The text 'Dziękujemy!' is written in white, bold, sans-serif font across the center of the image.

**Dziękujemy!**