

# Paradygmaty Programowania

Marcin Bączyk

Wykład 9

10 maja 2021

- programowanie zdarzeniowe
  - zdarzenie
  - dyspozytor
  - pętla zdarzeń
- systemy przesyłania wiadomości i ich topologie
  - broker
  - mediator
- programowanie graficznych interfejsów użytkownika
- wzorzec projektowy obserwowator

## Programowanie Zdarzeniowe – Event-driven programming

Metodyka wykorzystująca założenie, że w danym systemie istnieje ograniczona liczba zdarzeń, które mogą wystąpić i które mogą być obsłużone. Do programisty należy wybór zdarzeń, które należy obsłużyć.

W programowaniu zdarzeniowym występuje niedeterministyczny przepływ

## Zdarzenie – Event

Obiekt utworzony przez źródło zdarzenia. Obiekt zdarzenia zawiera wszystkie informacje niezbędne do jego obsłużenia.

Celem utworzenia obiektu zdarzenia jest enkapsulacja jego parametrów i umożliwienie przekazania ich do obiektu lub funkcji, w którym dane zdarzenie jest obsługiwane.

Najczęściej spotykane zdarzenia związane są z obsługą graficznego interfejsu użytkownika

- wciśnięcie klawisza,
- kliknięcie lewym lub prawym przyciskiem myszy,
- ruch myszą w określony sposób.

Zdarzenia mogą być związane z obsługą wszelkiego rodzaju komunikacji:

- otrzymanie komunikatu od innych programów,
- otrzymanie wyniku pomiaru.

Programista może też ustawić zdarzenia, które będą wygenerowane wyznaczonej chwili czasowej.

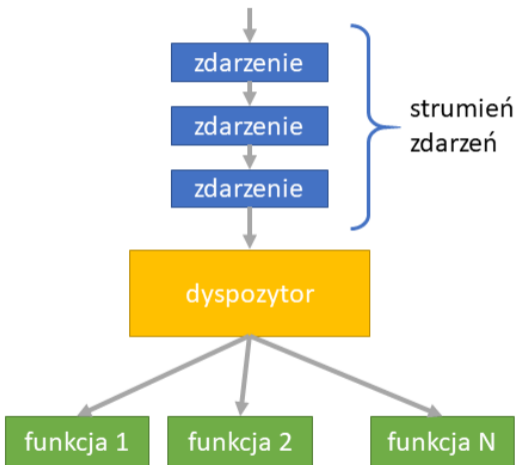
Programowanie zdarzeniowe dotyczy również obsługi przerwania sprzętowych (oraz niekiedy programowych) programów uruchamianych bezpośrednio na procesorze lub mikrokontrolerze (bez pośredniczenia systemu operacyjnego). Zdarzenia w takim systemie nazywane są przerwaniem i mogą być zgłoszone na przykład przez:

- kontroler pamięci,
- kontroler magistrali,
- licznik zegara.

W takim systemie do konkretnych przerwania (zdarzeń) podpięte są procedury ich obsługi.

System obsługi zdarzeń składa się z:

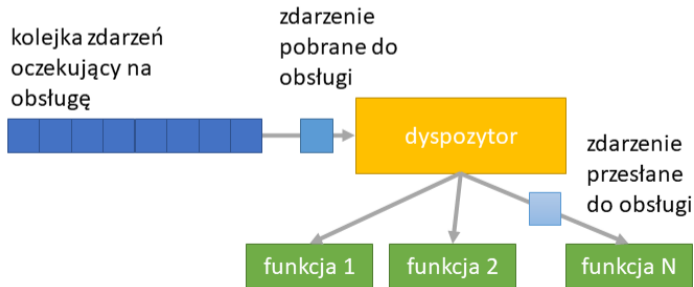
- strumienia napływających zdarzeń,
- dyspozytora zdarzeń,
- zestawu funkcji obsługujących zdarzenia.



## Dyspozytor

Zadaniem dyspozytora jest dostarczanie poszczególnych zdarzeń do konkretnej funkcji obsługującej w zależności od typu zdarzenia.

Dyspozytor to nieskończona pętla, w której przetwarzany jest wejściowy strumień zdarzeń. Zdarzenia kolejno pobierane są z pętli zdarzeń, dekodowane i wysyłanie do odpowiednich funkcji obsługujących je.



## Pętla (kolejka) zdarzeń

W niektórych przypadkach dyspozytor oraz procedury obsługi zdarzeń mogą nie być w stanie obsłużyć zdarzeń dostatecznie szybko. W takich przypadkach rozwiązaniem jest buforowanie strumienia zdarzeń wejściowych w postaci kolejki.

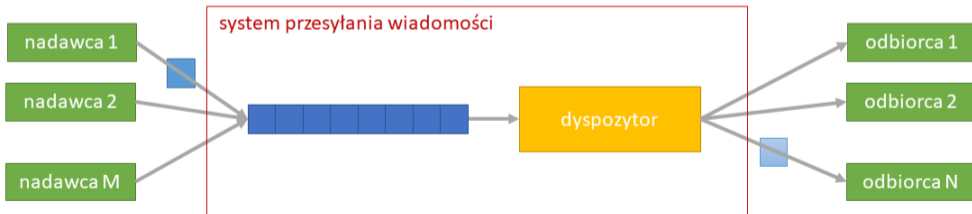
Wydarzenia są dodawane na koniec kolejki tak szybko, jak przybywają, natomiast dyspozytor pobiera je z początku kolejki, przekazując poszczególnych funkcjom obsługującym.

Aplikacje GUI zazwyczaj zawierają kolejkę zdarzeń. Niektóre istotne zdarzenia, takie jak kliknięcia myszą, mogą wymagać złożonej obsługi. Podczas ich obsługi inne zdarzenia, takie jak ruch myszy, mogą gromadzić się w buforze. Następnie od dyspozytora zależy, czy zdarzenia te zostaną obsłużone czy pominięte.



## Systemy rozsyłania wiadomości

Systemy przesyłania wiadomości reprezentują skrajną wersję wzorca obsługi zdarzeń. Celem systemu jest odbieranie zdarzeń (komunikatów) od generatorów zdarzeń (nadawców) i przesyłanie ich do funkcji obsługujących (odbiorców) w sytuacjach, gdy nadawcy i odbiorcy znajdują się w różnych lokalizacjach fizycznych lub działają na różnych platformach.



System rozsyłania wiadomości może być zorganizowany w dwóch topologiach:

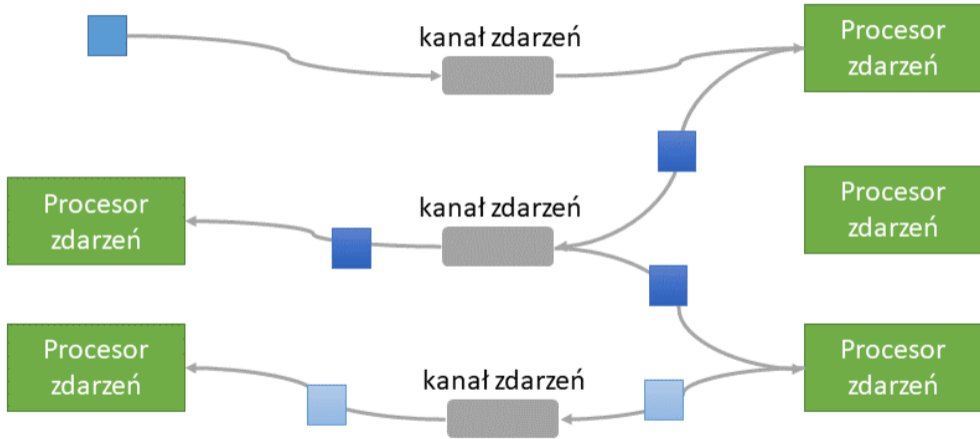
- brokera

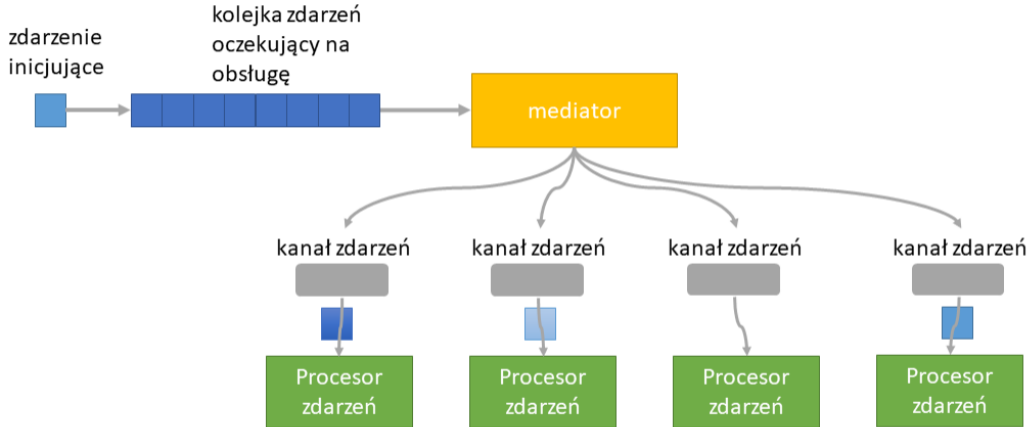
Wykorzystywana, gdy niezbędny jest wysoki stopień responsywności i dynamiczna kontrola nad przetwarzaniem zdarzenia.

- mediatora

Wykorzystywana, gdy niezbędna jest kontrola nad przebiegiem procesu przetwarzania zdarzenia

zdarzenie inicjujące

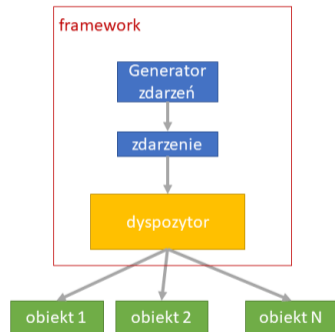




## Programowanie zdarzeniowe a programowanie obiektowe

Dzięki programowaniu zorientowanemu obiektowo stosunkowo łatwo jest opracować uogólnione klasy wielokrotnego użytku. Jest to jedna z zalet tej technologii. Uogólnione klasy wielokrotnego użytku stanowią podstawę do tworzenia własnych implementacji, które następnie włączane są (plug-in) w większą strukturę. Użytkownik, tworząc własną klasę, implementuje jedynie mały fragment kodu obsługujący konkretne zdarzenie.

System obsługi zdarzeń używa używa obiektów klas zdefiniowanych przez programistę tak jakby były one obiektami klas zdefiniowanych przez twórców danego systemu. Tworzone przez programistę obiekty wstawiane są w miejsce, w którym powinny się znaleźć obiekty obsługujące dane zdarzenie.

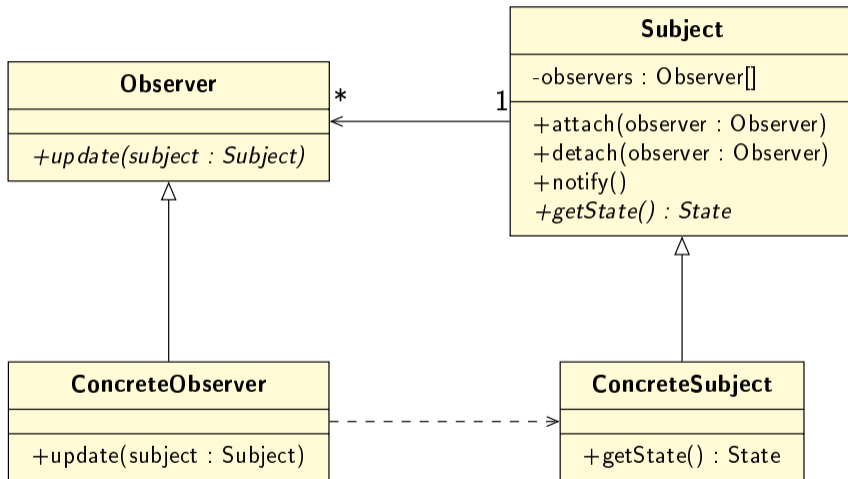


## Tworzenie graficznych interfejsów użytkownika

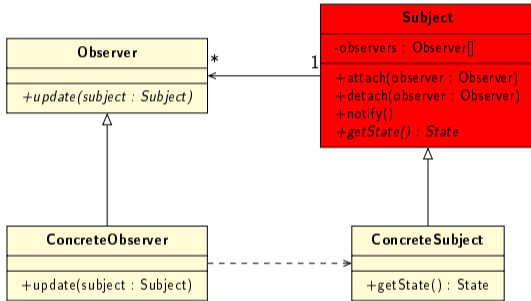
- W programowaniu GUI najważniejsze jest zdefiniowanie **obiektu** obsługującego dane zdarzenie.
- Należy określić wygląd każdego aspektu projektowanego interfejsu.
- Istnieje wiele rodzajów zdarzeń, które muszą być obsługiwane przez GUI.
- Istnieje wiele modyfikatorów zachowania standardowych akcji (wciśnięcie klawisza Ctrl, Alt lub Shift).
- Narzędzia do tworzenia graficznych interfejsów użytkownika dostarczane są jako frameworki, które z jednej strony zdejmują dużą część pracy z programisty, a z drugiej strony trzeba się ich nauczyć.
- Często wykorzystywany jest wzorzec obserwatora, który trzeba zrozumieć i wiedzieć, jakie są jego ograniczenia.

### Przykład

Obiekt reprezentujący zegar informuje inne obiekty o upływającym czasie w ustalonych interwałach. Informowane obiekty mogą wyświetlać czas lub przetwarzać i przekazywać informację dalej. Na przykład obiekt reprezentujący alarm może informować od osiągnięciu danej chwili lub o upłynięciu ustalonego czasu.





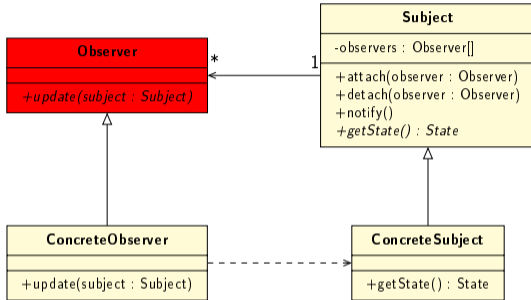


## Obserwowany

- Zna swoich **Obserwatorów** i może go obserwować dowolna ich liczba.
- Zapewnia interfejs dołączania i odłączania **Obserwatorów**.

**Obserwowany** może zapewniać osobną metodę pozwalającą na uzyskanie informacji o stanie lub przekazać stan w wywołaniu metody powiadającej **Obserwujących**.

# wzorzec projektowy «obserwator»

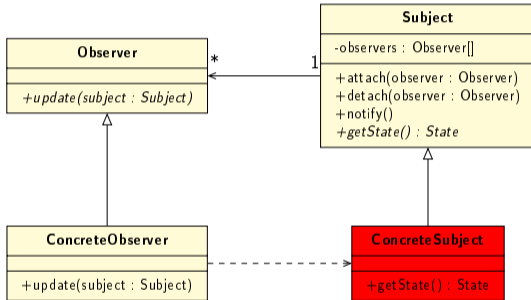


## Obserwator

- Definiuje interfejs uaktualniania dla obiektów, które powinny być powiadomione o zmianach.

Interfejs powiadamiania może, przyjmować komplet informacji niezbędnych do aktualizacji **Obserwatora** lub referencję do obiektu, którego obserwuje, w celu pozyskania dalszych informacji. Metoda powiadamiania, może też nie przekazywać żadnych informacji a być jedynie oznaką tego, że coś zostało zmienione. W takim przypadku **Obserwator** niezbędne informacje do aktualizacji swojego stanu musi pozyskać w inny sposób.

# wzorzec projektowy «obserwator»

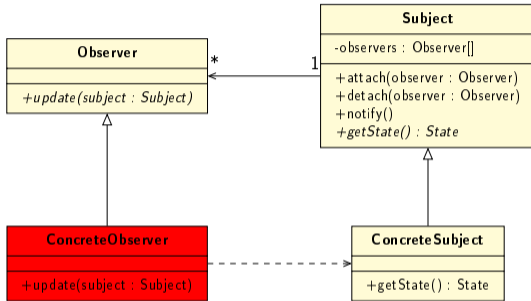


## Konkretny Obserwowany

- Przechowuje stan.
- W razie zmiany stanu wysyła powiadomienia do swoich **Obserwatorów**.

**Konkretny Obserwowany** w razie potrzeby może implementować interfejsy wielu **Obserwowanych**.

# wzorec projektowy «obserwator»



## KonkretnyObserwator

- Przechowuje stan zgodny ze stanem obserwowanego obiektu.
- Implementuje interfejs **Obserwatora** w celu zachowania spójności swojego stanu ze stanem **Obserwatora**.

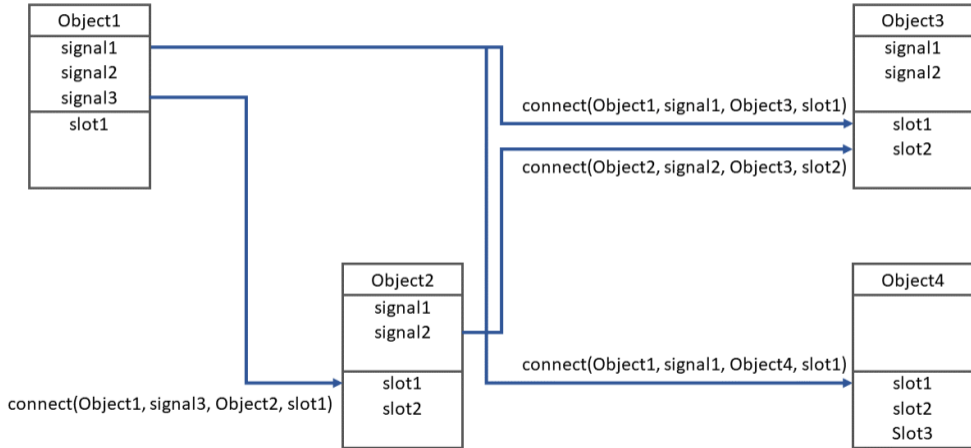
**KonkretnyObserwator** może nie mieć świadomości, który konkretnie obiekt obserwuje. **KonkretnyObserwator** w razie potrzeby implementuje mechanizm komunikacji z **Obserwowanym** w celu uzyskania niezbędnych informacji.

**KonkretnyObserwowany** poprzez **Obserwowanego** zawsze powiadamia swoich **Obserwatorów**, gdy wystąpi zmiana jego stanu. Po otrzymaniu powiadomienia **KonkretnyObserwator** aktualizuje swój stan. W zależności od interfejsu powiadamiania, konieczne może być dodatkowa komunikacja w celu uzupełnienia informacji.

## Obszar zastosowań

Stosuj wzorzec obserwatora, gdy

- jakaś abstrakcja ma dwa aspekty, jeden zależy od drugiego, a kapsułkowanie tych aspektów w oddzielnych obiektach umożliwia niezależne zmienianie i używanie tych obiektów,
- zmiana jednego obiektu wymaga zmiany innych i nie wiadomo ile obiektów trzeba zmienić,
- obiekt powinien być w stanie powiadamiać inne obiekty, nie przyjmując żadnych założeń co do tego co te obiekty reprezentują.



## Oprogramowanie oparte na zdarzeniach – podsumowanie

- Metodyka szeroko wykorzystywana do tworzenia aplikacji z interfejsem graficznym.
- Często w systemie implementowana jest pętla obsługi zdarzeń, do której trafiają konkretne zdarzenia i która informuje inne moduły o zaistniałym zdarzeniu.
- Mechanizm sterujący pętlą komunikatów jest zazwyczaj niedostępny dla programisty. Sposób obsługi pętli jest ukryty przed programistą.
- Programista decyduje, które zdarzenia należy obsłużyć, w jaki sposób oraz jaki ma być efekt obsługi danego typu zdarzenia.
- Programista zazwyczaj nie decyduje o tym, jaki mechanizm jest wykorzystany do powiązania danego zdarzenia z procedurą jego obsługi – jest narzucony przez środowisko – pętla zdarzeń lub wzorzec obserwatora.



## Zalety

- ograniczenie kodu do opisu tylko obsługiwanych zdarzeń,
- łatwość opisanie interakcji z otoczeniem,
- posługiwanie się inną metodologią podczas samego opisu funkcji obsługi zdarzeń,
- przetwarzanie równoległe – zwiększenie wydajności poprzez wykorzystanie wielu procesorów.
  
- w przypadku graficznego interfejsu użytkownika istotna jest prostota komunikacji z użytkownikiem

## Wady

- wykorzystanie ukrytej, lecz de facto istniejącej części programu odpowiadającej za obsługę pętli komunikatów – dodatkowe sprzężenia,
- konieczność znajomości możliwych zdarzeń i sposobu dostarczenia ich parametrów – mozolne przeglądanie dokumentacji,
- skomplikowana budowa systemu – programowania w danym środowisku trzeba się nauczyć jak programowania w danym języku,
- przetwarzanie równoległe – znacząco utrudnione uruchamianie i wykrywanie błędów ze względu na asynchroniczny charakter zdarzeń.