



# Systemy rozproszone

Informatyka, sem. 6  
część II

Aynchroniczne  
wywołanie metod



# Delegates oraz events w .NET

- delegate
  - implementacja koncepcji wskaźnika funkcji
  - różnica: type-safe
- Główne zastosowania
  - startowanie wątków
  - generic class libraries - wykonanie fragmentu kodu specyficznego, np. sortowanie -> funkcja porównująca
  - zdarzenia



# Delegaty w C#

- Utworzenie delegata analogicznie jak klasy: definicja, instantiate (utworzenie obiektu)
- Definicja
  - `delegate void IntMethodInvoker (int x);`
    - np. każda instancja przyjmuje jeden parametr
  - `delegate string GetAString();`
  - (Microsoft tip) delegat jest czymś, co nadaje nazwę sygnaturze metody wraz z parametrami



# Delegaty w C# cd

- Utworzenie "obiektu"
  - `int x = 40; ...`
  - `GetString firstStringMethod = new GetString(x.ToString);`
  - `Console.WriteLine("Bleble " + firstStringMethod());`
- Faktyczne utworzenie nowej klasy i obiektu
  - delegaty są implementowane jako podklasy `System.MulticastDelegate` (`System.Delegate`)
  - składnia dziedziczenia jest ukryta przed programistą



# Zdarzenia (event) w .NET

- Windows-based applications are message-based
- Aplikacje obiektowe - zdarzenia (events)
  - mogą opakowywać Windows messages
- Events
  - zaimplementowane na bazie mechanizmu delegate



# Perspektywa odbiorcy

- np. aplikacja WindowsForms
  - `buttonOne.Click += new EventHandler (Button_Click);`
  - EventHandler - delegat, który przypisuje `Button_Click` do zdarzenia `Click`
    - operator `+=` - skrócone przypisanie metody do delegata
  - `private void Button_Click ...`
- Parametry przetwarzane w `Button_Click`



# Generowanie zdarzeń (events)

- Utworzenie event oraz odpowiedniego delegate
  - public delegate void CustomActionHandler(object sender);
    - utworzenie "swojego" delegata
  - public static event CustomActionHandler Action;
    - zdefiniowanie event, wymaga podania delegata, który będzie go obsługiwał
- Wywołanie (raise) event
  - Action("event raised");
  - ale ...



# Generowanie zdarzeń (events)



- Wywołanie Action("event raised");
  - spowoduje błąd, jeżeli nie zdefiniowano funkcji obsługującej (event handler)
    - Action będzie null
  - dopiero przypisanie event handler zainicjalizuje obiekt event
    - Action += new  
Form1.CustomButtonHandler(HandleAction);
- Wywołanie Action po sprawdzeniu null
  - if (Action!=null) Action (...) ...



# Asynchroniczne wywołanie operacji .NET



- Dostępne wzorce w środowisku
  - z użyciem obiektów `IAsyncResult`
  - z użyciem zdarzeń (events)
- Zastosowanie
  - IO (pliki, sockety, strumienie)
  - XML Web services
  - Web forms
- <http://msdn.microsoft.com/en-us/library/2e08f6yc.aspx>



# Wywołanie asynchroniczne metod synchronicznych



- Zdefiniowanie delegata odpowiadającego sygnaturze metody
- Wykorzystanie BeginInvoke oraz EndInvoke
  - BeginInvoke zwraca IAsyncResult, który może być wykorzystany do monitorowania wykonania
  - EndInvoke zablokuje się, jeżeli operacja jeszcze się nie zakończyła



# Operacje po BeginInvoke Demo



- Wykonanie innych operacji i wywołanie EndInvoke
- Wykorzystanie WaitHandle na oczekiwanie zakończenia
- Polling IAsyncResult
- Wywołanie funkcji po zakończeniu
  - Przekazanie delegata funkcji do BeginInvoke
  - wykorzystuje wewnętrznie ThreadPool



# Wzorzec Command

- Cele
  - wywołanie operacji bez wiedzy na temat faktycznych zadań
  - np. Menu -> Menu item -> Command
    - konkretna komenda OpenDocument, CloseDocument
    - aplikacja konfiguruje komendę, system wywołuje konkretną akcję
    - komenda implementuje interface umożliwiający wywołanie operacji

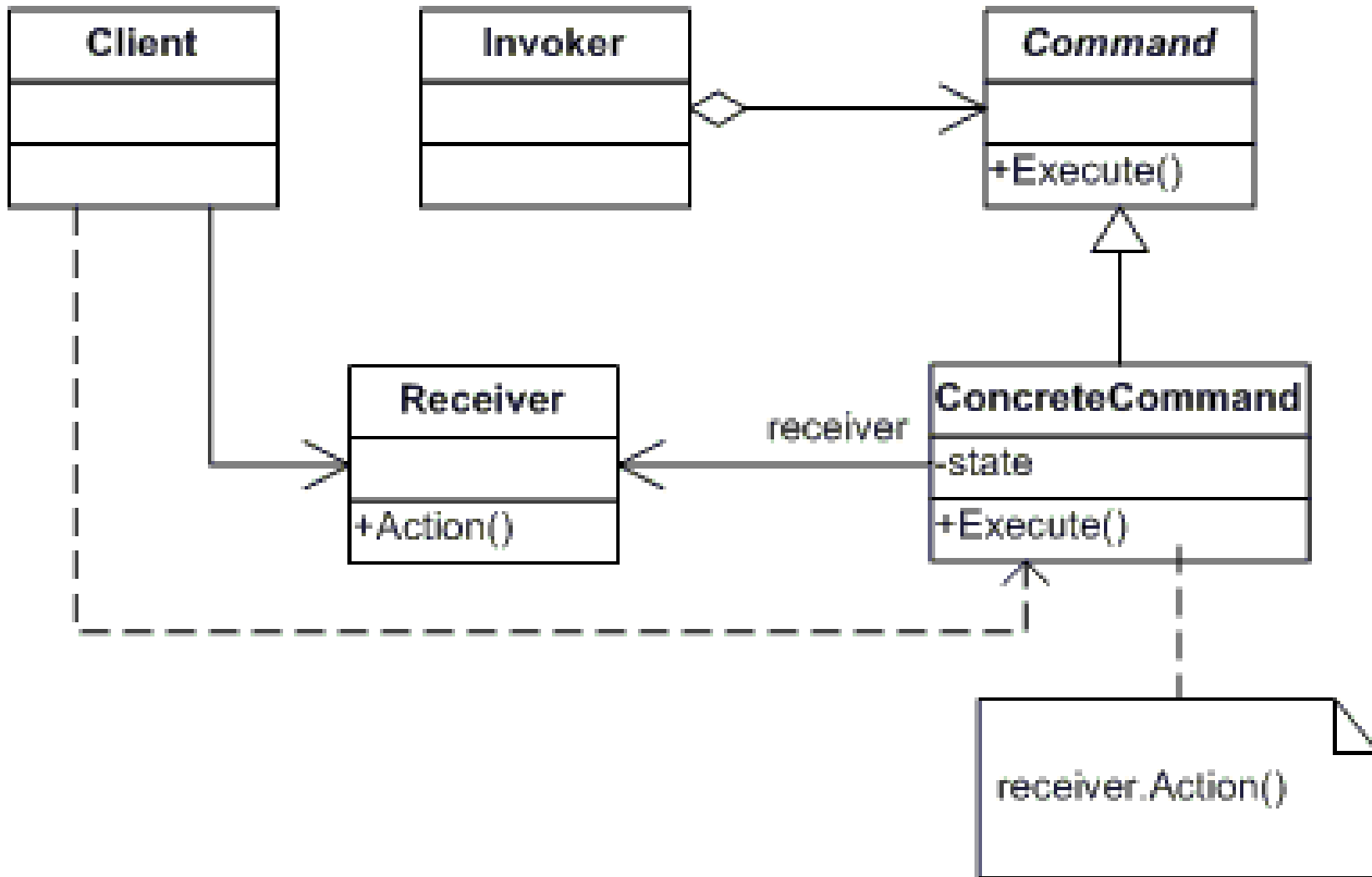


# Zastosowanie

- Mechanizm callback z języków proceduralnych
  - zastąpienie rejestracji funkcji przez Command
- Kolejowanie i wykonywanie zadań w różnym czasie
  - np. przesłanie zadania do innego procesu, innej przestrzeni adresowej
- Wsparcie "undo"
  - operacja unexecute
- Wsparcie dla niezawodności: logowanie, transakcje



# Struktura





# Uczestnicy

- Command
  - deklaruje interface dla wywoływanych operacji
- ConcreteCommand
  - wiąże Receiver oraz akcję
  - implementuje Execute oraz wywołuje operacje na Receiver
- Client (application)
  - tworzy ConcreteCommand oraz określa Receiver
- Invoker (MenuItem)
  - żąda wykonania operacji
- Receiver
  - wie, w jaki sposób wykonać operację



# Współpraca

- Client tworzy ConcreteCommand oraz określa Receiver
- Invoker przechowuje ConcreteCommand
- Invoker wywołuje Execute
- ConcreteCommand wywołuje operację na Receiver



# Command Demo





# Asynchroniczne wywołanie w Command Pattern



- Demo
- Główne różnice
  - SetCommand tworzy delegata
  - ExecuteCommand wywołuje delegata asynchronicznie



# Wzorzec Observer

- Cele
  - zapewnienie spójności danych między luźno powiązаныmi obiektami
- np. Dane są wykorzystywane przez okna wyświetlające tabelę oraz graf
  - okna powinny być powiadomione, gdy nastąpi zmiana danych
  - subject, observer - observer jest powiadamiany o zmiana subject
  - znany również jako publish-subscribe

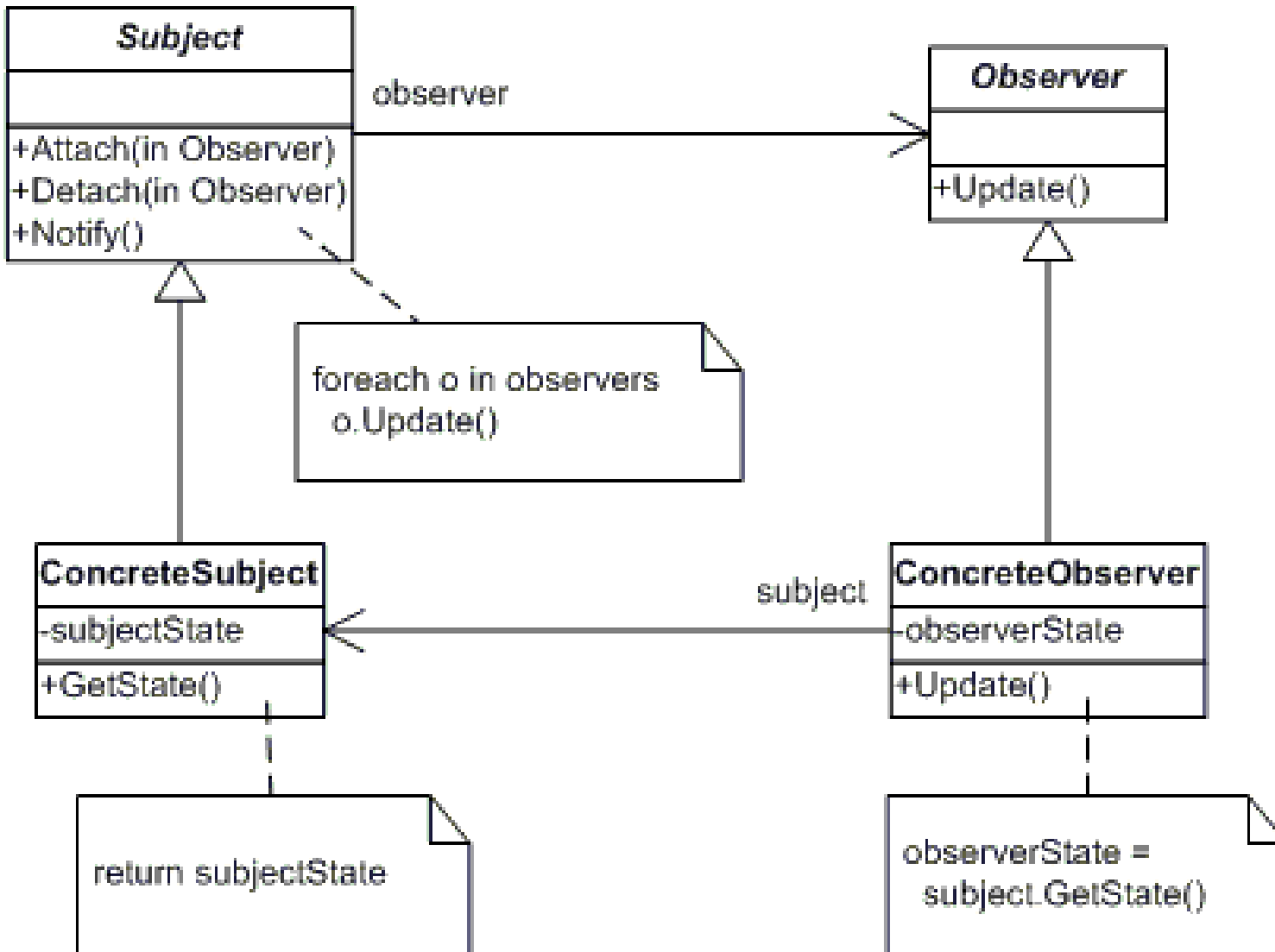


# Zastosowanie

- Dwa różne aspekty problemu, jeden zależny od drugiego, rozłączenie aspektów pozwala na lepsze reużycie modułów
- Zmiana jednego obiektu wymaga zmiany innego
- Obiekt powinien posiadać możliwość powiadomienia innych obiektów nie znając tożsamości tych obiektów



# Struktura





# Uczestnicy

- Subject
  - zna obserwatorów
  - udostępnia interfejsy do podłączenia, odłączenia obserwatorów
- Observer
  - definiuje interfejsy umożliwiające powiadamianie o zmianach
- ConcreteSubject
  - przechowuje stan, wysyła powiadomienia (notifications)
- ConcreteObserver
  - zawiera referencję do ConcreteSubject
  - zawiera stan, który jest uaktualniany
  - implementuje interfejsy umożliwiające uaktualnienie



# Współpraca

- ConcreteSubject powiadamia obserwatorów, gdy następują zmiany
- Po otrzymaniu powiadomienie, obserwator może odpytać Subject o szczegóły zmian
- Subject jest obiektem, który inicjalizuje modyfikacje stanu