Definicje

- ***Abstrakcja***
  - uproszczony model rzeczywistości
  - Poziom złożonosci modelu ze świata rzeczywistego
- ***Konstruktor***
  - metoda, która zawsze jest inicjalizowana przy wywołaniu klasy
  - ma tą samą nazwę co klasa
- ***Interfejs***
  - definiowanie zadań jakie może wykonać obiekt lub aktor
  - lista metod publicznych
  - zawiera jedynie specyfikacje metod, bez implementacji
- ***Wersjonowanie semantyczne***
  - MajorVersion.MinorVersion.Patch
    - MajorVersion– Dodanie nowej funkcjonalności, API traci wsteczną kompatybilność
    - MinorVersion– Dodanie nowej funkcjonalności, API jest wstecznie kompatybilne
    - Patch– Drobna poprawa błędów, API się nie zmienia
- ***Agregacja***
  - modelowanie relacji część-całość występującej pomiędzy bytami świata rzeczywistego
  - np. związek pomiędzy silnikiem a samochodem
- ***Kompozycja***
  - silna forma agregacji
  - związek nierozerwalny w tym znaczeniu, że nie ma sensu samodzielne istnienie bytu podrzędnego
  - cykl życiowy składowej zawiera się w cyklu życiowym całości
    - byt podrzędny nie może istnieć, gdy nie istnieje byt nadrzędny
    - byt podrzędny musi zostać usunięty, gdy usuwany jest byt nadrzędny
  - kompozycja – zawieranie, klasa nadrzędna zawiera podrzędne (Trajdos)
- ***Dziedziczenie***
  - Mechanizm umożliwiający wywodzenie nowych klas z klas już istniejących, wraz z przejmowaniem ich metod
  - obiekt w trakcie swojego istnienia zmienia klasę, bez zmieniania swojej tożsamości (dziedziczenie dynamiczne)
  - dziecko jest szczególnym przypadkiem rodzica (Trajdos)
- ***Polimorfizm***
  - "wiele form" (postaci) jednego bytu
  - polimorfizm metod
    - jedna operacja (byt wirtualny) może posiadać wiele form (metod implementujących)
  - polimorfizm typów
    - istnienie funkcji, które mogą zarówno przyjmować wartości różnych typów jako swoje argumenty, jak też i zwracać wartości różnych typów
  - polimorfizm parametryczny
    - typ bytu programistycznego może być parametryzowany innym typem, np. klasa *Wektor*(*int*) czy *Wektor*(*char*)
- ***Hermetyzacja (enkapsulacja)***
  - oddzielenie implementacji od obiektu (Trajdos)
  - udostępnianie tylko pewnego zbioru metod, które są widoczne z zewnątrz i mogą być wywoływane

- ***Aktor***
  - coś co używa stworzonego kodu/klass
  - czynnik poza systemowy, kótry może wywołać akcje (Trajdos)
- ***Klasa abstrakcji***
  - oznaczona slowem kluczowym abstract
  - klasa, która nie ma i nie może posiadać wystąpień bezpośrednich
  - nadklasa dla innych klas, stanowiąc swego rodzaju wspólną część definicji grupy klas o podobnej semantyce
  - moze zawierac metody abstrakcyjne
- ***Delegacja***
  - operacje, które można wykonać na danym obiekcie, są własnością innego obiektu
  - alternatywa dla klasycznego mechanizmu dziedziczenia
- ***Parsowanie***
  - Parsowanie danych polega na przetwarzaniu informacji, ich porządkowaniu i dostarczaniu gotowych danych
- ***garbage collector***
  - komponet JVM ktorego glownym zadaniem jest usowanie z pamieci nieuzywanych obiektow
- ***Klasa spójna***
  - klasa implementuje określony wąski zakres funkcjonalności
- ***Finalize***
  - Called by the garbage collector on an object when garbage collection determines that there are no more references to the object (Javadoc)
    - Wywoływana przez GC kiedy GC wykryje brak referencji do obiektu
- ***klasa finalna***
  - declaration to indicate that the method cannot be overridden by subclasses (Javadoc)
    - deklaracja wskazująca, że metoda nie może być nadpisana przez podklasę
  - A final class cannot be subclassed (Wikipedia)
    - nie można tworzyć podklas klasy finalnej
- ***metoda statyczna***
  - create fields and methods that belong to the class, rather than to an instance of the class (Javadoc)
    - tworzenie pól I metod, które należą do klasy, a nie do instancji klasy

Pytania

- ***Jakie są różnice pomiędzy podejściem proceduralnym a obiektowym?***
  - W programowaniu strukturalnym mamy podprogramy (funkcje), które odpowiednio ułożone i wywołane tworzą program.
  - Natomiast w p. obiektowym funkcjonalności są przypisane obiektom, zatem trudniej tutaj o pomyłkę np. taką jak przekazanie złej zmiennej do funkcji.
- ***Jaka jest różnica pomiędzy klasą a obiektem?***
  - Klasa – schemat ogólny
  - Obiekt – instancja klasy, pola sa zainicjalizowane
- ***Jakie modyfikatory dostępu do składowych klasy są dostępne w języku Java?***

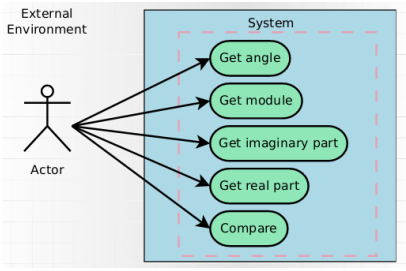
Modyfikator	Uml	Class	Package	Subclass	World
public	+	Y	Y	Y	Y
protected	#	Y	Y	Y	N
	~	Y	Y	N	N
private	-	Y	N	N	N

- ***Z jakiego mechanizmu języka Java skorzystasz aby hermetyzować zachowania?***
  - Modyfikatorów dostępu
- ***Jaka jest różnica między interfejsem języka Java a klasą abstrakcyjną?***
  - Klasa abstrakcyjna zawiera wszystkie metody I zmienne, które są wykorzystywane
  - interfejs posiada tylko deklaracje metod publicznych
  - interfejs są to wszystkie metody I obiekty, które może wykorzystać aktor
    - można używać modyfikatorów dostępu do ukrywania niektórych własności klas
- ***Czy w języku Java jest możliwe dziedziczenie wielobazowe?***
  - nie
- ***Jak wiele interfejsów może implementować klasa w języku Java?***
  - W teorii nieskończenie wiele (w praktyce 2^16-1)

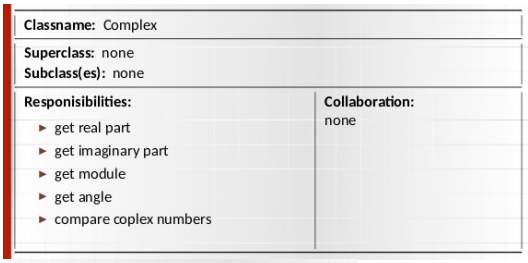
Diagramy

- ***Analiza czasownikowo-rzeczownikowa***
  - jednym kolorem należy zaznaczyć obiekty, a innym wykonywane na nich działania

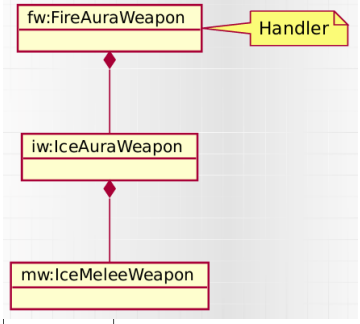
Potrzebny jest obiekt umożliwiający wykonywanie operacji na liczbach zespolonych. Chcemy, aby umożliwiał on wyłuskanie części rzeczywistej, części urojonej, kąta oraz modułu liczby zespolonej. Ponadto potrzebujemy możliwości sprawdzenia czy dwie liczby zespolone są sobie równe.



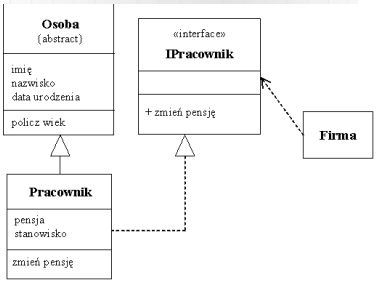
- ***Diagram przypadków użycia***
  - diagram pokazujący przypadki I możliwości użycia metod/klas przez aktora



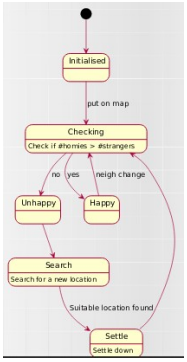
- ***Karta CRC (Class Responsibilities and Collaboration)***
  - karta przedstawiająca za co odpowiadają klasy I jakie mają połączenia z innymi



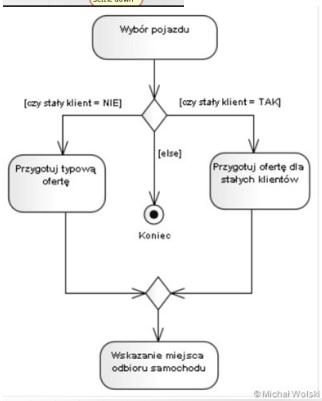
- ***Diagram obiektów***
  - obrazują obiekty występujące w systemie i ich związki. Są one zazwyczaj wykorzystywane do wyjaśnienia znaczenia diagramów klas[*]



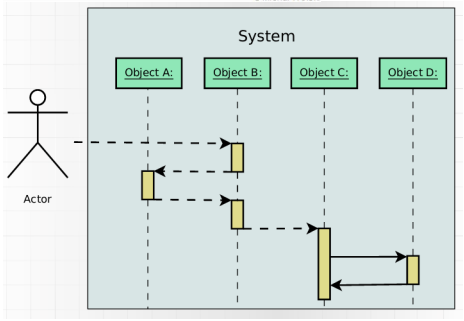
- ***Diagram klas***
  - diagram w którym zostają oddzielone zmienne I metody oraz pokazane połączenia między klasami wewnątrz paczki



- ***Diagram stanów***
  - Pokazuje przede wszystkim możliwe stany obiektu oraz przejścia, które powodują zmianę tego stanu[*]



- ***Diagram aktywności***
  - w języku UML służy do modelowania czynności i zakresu odpowiedzialności elementów bądź użytkowników systemu. Jest niejako podobny do diagramu stanu, jednak w odróżnieniu od niego nie opisuje działań związanych z jednym obiektem a wieloma, pomiędzy którymi może występować komunikacja przy wykonywaniu czynności.[*]



- ***Diagram sekwencji***
  - Diagram sekwencji służy do prezentowania interakcji pomiędzy obiektami wraz z uwzględnieniem w czasie komunikatów, jakie są przesyłane pomiędzy nimi.[*]
    - obiekty ułożone są wzdłuż osi X
    - komunikaty przesyłane są wzdłuż osi Y

Zasady programowania

- ***Zasada DRY (Don't Repeat Yourself)***
  - powtarzający się kod wydzielać do oddzielnych metod lub klas
  - unikanie powtarzania tych samych części kodu/logiki w różnych miejscach
- ***Keep it simple, stupid. (KISS)***
  - Jak coś można zrobić prościej, to tak należy zrobić

- ***SOLID***
  - **Zasada SRP (single responsibility principle)**
    - Nie powinno być więcej niż jednego powodu do modyfikacji klasy
    - Przypisuj odpowiedzialności do obiektu tak, aby spójność była jak największa.
    - Jedna klasa – jeden obiekt powinien być odpowiedzialny za jedną funkcjonalność (Trajdos)
  - **Open-Close Principle**
    - Klasy powinny być otwarte na rozszerzanie, a zamknięte na modyfikację
    - Polegaj na abstrakcji i polimorfizmie
    - otwarte na rozszerzenie, zamknięte na modyfikacje. Rozszerzanie przez kompozycję, dziedziczenie (Trajdos)
  - **Liskov Substitution Principle**
    - Klasy w programie powinny być podmienialne przez swoje podklasy bez naruszania poprawności programu, czyli klasa dziedzicząca musi być dobrym odpowiednikiem klasy bazowej.
    - Podklasa powinna robić więcej niż klasa bazowa
    - odwołanie do referencji/obiektu powinno być dostępne dla potomka (dziedziczenie) (Trajdos)
  - **Interface Segregation Principle**
    - Klasa powinna udostępniać drobnoziarniste interfejsy dostosowane do potrzeb jej klienta
    - Powinno się projektować małe i zwarte interfejsy
    - ujawanianie drobnoziarnistych, wyspecjalizowanych interfejsów (Trajdos)
  - **Dependency inversion Principle**
    - Wysokopoziomowe moduły nie powinny zależeć od modułów niskopoziomowych– zależności między nimi powinny wynikać z abstrakcji
    - relacje między klasami powinny odbywać się przez abstrajcję (Trajdos)
- ***G.R.A.S.P (General Responsibility Assignment Software Principles)***
  - **Creator**
    - kiedy jedna klasa tworzy obiekty innej klasy (Trajdos)
      - kompozycja, wzorzec fabryki

Nazwa	***Creator (twórca)***
Problem	Kto tworzy instancję klasy A?
Rozwiązanie	Przydziel zobowiązanie tworzenia instancji klasy A klasie B, jeżeli zachodzi jeden z warunków: • B "zawiera" A lub agreguje A (kompozycja) • B zapamiętuje A • B bezpośrednio używa A • B posiada dane inicjalizujące dla A

- **Information Expert**
  - działanie powinno być umieszczone w klasie najlepiej je spełniające (Trajdos)

Nazwa	***Information Expert***
Problem	Jak przydzielać obiektom zobowiązania?
Rozwiązanie	Przydziel zobowiązanie “ekspertowi” - tej klasie, która ma informacje konieczne do jego realizacji

- **Low Coupling**
  - jak najmniej powiązań, najlepiej przez abstrakcję (Trajdos)

Nazwa	***Low Coupling***
Problem	Jak zmniejszyć liczbę zależności I zasięg zmian, a zwiększyć możliwość ponownego wykorzystania kodu
Rozwiązanie	Przydziel odpowiedzialność tak, aby ograniczyć stopień sprzężenia (liczbę powiązań obiektu)

- **High Cohesion**
  - nie przeładowywać klasy odpowiedzialnościami, spójność (nie rozdrabniać bez uzasadnienia) (Trajdos)

Nazwa	***High Cohesion***
Problem	Jak sprawić by obiekt miał jasny cel, były zrozumiałe I łatwe w utrzymaniu
Rozwiązanie	Przydziel odpowiedzialność by spójność pozosawała wysoka

- **Polymorphism**
  - kiedy jest referencja do rodzica to można tam ustawić metodę rodzica (Trajdos)

Nazwa	***Polymorphism***
Problem	Jak obsługiwać warunki zależne od typu?
Rozwiązanie	Przydziel zobowiązania – przy użyciu operacji polimorficznych – typom dla których to zachowanie jest różne

- **Protected variations**
  - hermetyzacja (Trajdos)

Nazwa	***Protected variations***
Problem	Jak projektować obiekty, by ich zmienność nie wywierała szkodliwego wpływu na inne elementy?
Rozwiązanie	Rozpoznaj punkty zmienności I otocz je stabilnym interfejsem

Wzorce projektowe

Wzorzec kreacyjny – służy do tworzenia klas/struktur (Trajdos)

- ***Singleton***
  - celem jest ograniczenie możliwości tworzenia obiektu danej klasy do jednej instancji oraz zapewnienie globalnego dostępu do stworzonego obiektu
  - Trajdos
    - klasa, której chcemy mieć tylko jedną instancję
  - The Singleton pattern ensures that a class has only one instance and provides a global point of access to that instance. It is named after the singleton set, which is defined to be a set containing one element. The office of the President of the United States is a Singleton. The United States Constitution specifies the means by which a president is elected, limits the term of office, and defines the order of succession. As a result, there can be at most one active president at any given time. Regardless of the personal identity of the active president, the title, "The President of the United States" is a global point of access that identifies the person in the office.
  - implementacja
- ***Builder***
  - Trajdos
    - budowniczy – każdy z nich wie jak należy wykonać kroki (zna szczegóły)
    - architekt – co należy zrobić, w jakiej kolejności (zna ogóły)
  - The Builder pattern separates the construction of a complex object from its representation so that the same construction process can create different representations. This pattern is used by fast food restaurants to construct children's meals. Children's meals typically consist of a main item, a side item, a drink, and a toy (e.g., a hamburger, fries, Coke, and toy dinosaur). Note that there can be variation in the content of the children's meal, but the construction process is the same. Whether a customer orders a hamburger, cheeseburger, or chicken, the process is the same. The employee at the counter directs the crew to assemble a main item, side item, and toy. These items are then placed in a bag. The drink is placed in a cup and remains outside of the bag. This same process is used at competing restaurants.
    - Builder oddziela konstrukcję skomplikowanej formy od jego reprezentacji tak, że ten sam proces budowania może dawać różne obiekty końcowe
  - implementacja
- ***Abstract Factory***
  - celem jest dostarczenie interfejsu do tworzenia różnych obiektów jednego typu bez specyfikowania ich konkretnych klas (factory of factories)
  - Trajdos
    - umożliwia tworzenie interfejsów I za jego pomocą obiektów
    - celem jest rozluźnienie zależności
    - nie chcemy się wiązać z konkretną implementacją
  - The purpose of the Abstract Factory is to provide an interface for creating families of related objects, without specifying concrete classes. This pattern is found in the sheet metal stamping equipment used in the manufacture of Japanese automobiles. The stamping equipment is an Abstract Factory which creates auto body parts. The same machinery is used to stamp right hand doors, left hand doors, right front fenders, left front fenders, hoods, etc. for different models of cars. Through the use of rollers to change the stamping dies, the concrete classes produced by the machinery can be changed within three minutes.
  - implementacja

- ***Prototype***
  - celem jest umożliwienie tworzenia obiektów danej klasy bądź klas z wykorzystaniem już istniejącego obiektu, zwanego prototypem
  - Trajdos
    - wiązanie na poziomie interfejsu pozwala kopiować obiekty
    - obiekt, ktorego interfejs umożliwia kopiowanie tego interfejsu
    - nie chcemy się wiązać z konkretną implementacją
  - The Prototype pattern specifies the kind of objects to create using a prototypical instance. Prototypes of new products are often built prior to full production, but in this example, the prototype is passive and does not participate in copying itself. The mitotic division of a cell – resulting in two identical cells – is an example of a prototype that plays an active role in copying itself and thus, demonstrates the Prototype pattern. When a cell splits, two cells of identical genotype result. In other words, the cell clones itself.
  - implementacja
- ***Factory Method***
  - Trajdos
    - jedna metoda, którą przeciążamy w potomnych klasach
  - The Factory Method defines an interface for creating objects, but lets subclasses decide which classes to instantiate. Injection molding presses demonstrate this pattern. Manufacturers of plastic toys process plastic molding powder, and inject the plastic into molds of the desired shapes. The class of toy (car, action figure, etc.) is determined by the mold.
    - Factory Method definiuje interfejs do tworzenia obiektów, ale pozwala podklasom zdecydować, które klasy utworzyć
  - implementacja
  - ***vs Abstract Factory***
    - Factory Method jest rozszerzane poprzez dziedziczenie lub kompozycję (Trajdos)

Wzorzec strukturalny

- ***Wrapper***
  - celem jest umożliwienie współpracy dwóm klasom o niekompatybilnych interfejsach
  - kompozycja, dziedziczenie
  - The Adapter pattern allows otherwise incompatible classes to work together by converting the interface of one class into an interface expected by the clients. Socket wrenches provide an example of the Adapter. A socket attaches to a ratchet, provided that the size of the drive is the same. Typical drive sizes in the United States are 1/2" and 1/4". Obviously, a 1/2" drive ratchet will not fit into a 1/4" drive socket unless an adapter is used. A 1/2" to 1/4" adapter has a 1/2" female connection to fit on the 1/2" drive ratchet, and a 1/4" male connection to fit in the 1/4" drive socket.
  - implementacja

- ***Decorator***
  - pozwala na dodanie funkcji do istniejących klas dynamicznie podczas działania programu
  - Trajdos
    - obiekt może zawierać w sobie kolejne obiekty
    - celem jest rozszerzenie funkcjonalności
    - kompozycja
  - The Decorator attaches additional responsibilities to an object dynamically. The ornaments that are added to pine or fir trees are examples of Decorators. Lights, garland, candy canes, glass ornaments, etc., can be added to a tree to give it a festive look. The ornaments do not change the tree itself which is recognizable as a Christmas tree regardless of particular ornaments used. As an example of additional functionality, the addition of lights allows one to "light up" a Christmas tree.
    Another example: assault gun is a deadly weapon on it's own. But you can apply certain "decorations" to make it more accurate, silent and devastating.
  - implementacja
- ***Facade***
  - służy do ujednolicenia dostępu do złożonego systemu poprzez ustawienie uproszczonego, uporządkowanego interfejsu
  - Trajdos
    - wie jakie operacje są wykonywalne, jak należy to zrobić I udostępnia to klientowi (niejako main)
    - jak mamy wiele klas/zależności I chcemy zrobić coś bardziej złożonego I potrzebujemy zasłonić implementację. Musimy stworzyć Facade, która wie jakie są zależneści (bez cech szczególnych) między klasami
  - The Facade defines a unified, higher level interface to a subsystem that makes it easier to use. Consumers encounter a Facade when ordering from a catalog. The consumer calls one number and speaks with a customer service representative. The customer service representative acts as a Facade, providing an interface to the order fulfillment department, the billing department, and the shipping department
  - implementacja
- ***Composite***
  - celem jest składanie obiektów w taki sposób, aby klient widział wiele z nich jako pojedyńczy obiekt
  - The Composite composes objects into tree structures and lets clients treat individual objects and compositions uniformly. Although the example is abstract, arithmetic expressions are Composites. An arithmetic expression consists of an operand, an operator (+ - * /), and another operand. The operand can be a number, or another arithmetic expression. Thus, 2 + 3 and (2 + 3) + (4 * 6) are both valid expressions.
    - Composite komponuje obiekt do struktury drzewa I pozwala klientowi traktować pojedyńcze obiety I kompozycje tak samo
  - implementacja

- ***Bridge***
  - pozwala oddzielić abstrakcję od jego implementacji
  - Trajdos
    - pozwala rozddzielać operacje (chierarchia liczb zespolonych od rzeczywistych)
  - The Bridge pattern decouples an abstraction from its implementation, so that the two can vary independently. A household switch controlling lights, ceiling fans, etc. is an example of the Bridge. The purpose of the switch is to turn a device on or off. The actual switch can be implemented as a pull chain, simple two position switch, or a variety of dimmer switches.
  - implementacja
- ***Proxy***
  - celem jest utworzenie obiektu zastępującego inny obiekt
  - Trajdos
    - obiekt ograniczający dostęp i przechwytujący żadania
      - cache w dysku na wykładzie. Przechwytuje dane I sprawdza czy już wcześniej nie było do niego żądania
  - The Proxy provides a surrogate or place holder to provide access to an object. A check or bank draft is a proxy for funds in an account. A check can be used in place of cash for making purchases and ultimately controls access to cash in the issuer's account
  - implementacja
- ***Flyweight***
  - celem jest zmniejszenie wykorzystania pamięci poprzez poprawę efektywości obsługi dużych obiektów zbudowanych z wielu mniejszych elementów poprzez współdzielenie wspólnych małych elementów
  - Trajdos
    - jeśli mamy wiele obiektów o podobnych właściwościach, to chcemy kompozycją wydzielić dane ciężkie od lekkich
  - The Flyweight uses sharing to support large numbers of objects efficiently. Modern web browsers use this technique to prevent loading same images twice. When browser loads a web page, it traverse through all images on that page. Browser loads all new images from Internet and places them the internal cache. For already loaded images, a flyweight object is created, which has some unique data like position within the page, but everything else is referenced to the cached one.
  - implementacja

Wzorce behawioralne:

- ***Visitor***
  - zadaniem jest odseparowanie algorytmu od struktury obiektowej na której operuje
  - The Visitor pattern represents an operation to be performed on the elements of an object structure without changing the classes on which it operates. This pattern can be observed in the operation of a taxi company. When a person calls a taxi company (accepting a visitor), the company dispatches a cab to the customer. Upon entering the taxi the customer, or Visitor, is no longer in control of his or her own transportation, the taxi (driver) is.
  - implemetacja
- ***Chain of Responsibility***
  - żądanie może być przetwarzane przez różne obiekty, w zależności od jego typu
  - The Chain of Responsibility pattern avoids coupling the sender of a request to the receiver by giving more than one object a chance to handle the request. ATM use the Chain of Responsibility in money giving mechanism.
  - Implementacja

- ***Template Method***
  - Wzorzec metody szablonowej wykorzystuje klasę abstrakcyjną do zaimplementowania lub utworzenia deklaracji wspólnych metod oraz swoistą metodę run, która w odpowiedniej kolejności wywołuje wszystkie metody pośrednie.
  - The Template Method defines a skeleton of an algorithm in an operation, and defers some steps to subclasses. Home builders use the Template Method when developing a new subdivision. A typical subdivision consists of a limited number of floor plans with different variations available for each. Within a floor plan, the foundation, framing, plumbing, and wiring will be identical for each house. Variation is introduced in the later stages of construction to produce a wider variety of models.
    Another example: daily routine of a worker
  - Fajny link z implementacją
- ***Strategy***
  - definiuje rodzinę wymiennych algorytmów i kapsułkuje je w postaci klas. Umożliwia wymienne stosowanie każdego z nich w trakcie działania aplikacji niezależnie od korzystających z nich użytkowników.
  - A Strategy defines a set of algorithms that can be used interchangeably. Modes of transportation to an airport is an example of a Strategy. Several options exist such as driving one's own car, taking a taxi, an airport shuttle, a city bus, or a limousine service. For some airports, subways and helicopters are also available as a mode of transportation to the airport. Any of these modes of transportation will get a traveler to the airport, and they can be used interchangeably. The traveler must chose the Strategy based on trade-offs between cost, convenience, and time.
  - Implementacja
- ***State***
  - umożliwia zmianę zachowania obiektu poprzez zmianę jego stanu wewnętrznego. Innymi słowy – uzależnia sposób działania obiektu od stanu w jakim się aktualnie znajduje.
  - The State pattern allows an object to change its behavior when its internal state changes. This pattern can be observed in a vending machine. Vending machines have states based on the inventory, amount of currency deposited, the ability to make change, the item selected, etc. When currency is deposited and a selection is made, a vending machine will either deliver a product and no change, deliver a product and change, deliver no product due to insufficient currency on deposit, or deliver no product due to inventory depletion.
  - implementacja

- ***Mediator***
  - umożliwia zmniejszenie liczby powiązań między różnymi klasami, poprzez utworzenie mediatora będącego jedyną klasą, która dokładnie zna metody wszystkich innych klas, którymi zarządza.
  - The Mediator defines an object that controls how a set of objects interact. Loose coupling between colleague objects is achieved by having colleagues communicate with the Mediator, rather than with each other. The control tower at a controlled airport demonstrates this pattern very well. The pilots of the planes approaching or departing the terminal area communicate with the tower rather than explicitly communicating with one another. The constraints on who can take off or land are enforced by the tower. It is important to note that the tower does not control the whole flight. It exists only to enforce constraints in the terminal area.
  - implementacja