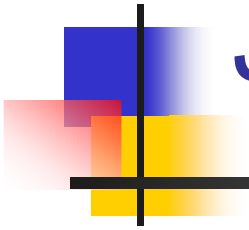
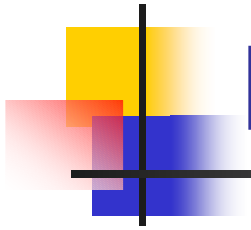


Java 1.5, wzorce i praktyki

Studium przypadku ObjectRADIUS

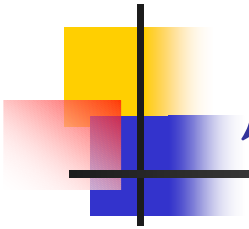
Przemysław Pokrywka

Michał Słociński



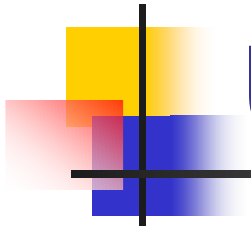
Kim jesteśmy

- Pasjonatami Javy od czasu studiów
- Sun Certified Java Programmers
- Programujemy obiektowo aplikacje rozproszone
- Spotkaliśmy się w Siemensie
- Pracujemy z biblioteką RADIUS starej generacji



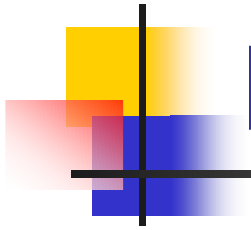
Pomysł, czyli „Dobry, Zły i Brzydki”

- Dostępne API do obsługi RADIUS w Javie
  - niektóre - **Złe**
  - niektóre - **Brzydkie**
  - brak naprawdę **Dobrych** implementacji
- Przemek:
  - reaguje alergicznie na kod zły i brzydki
  - planował publikację **artykułu** o ulepszeniu jej aspektów
- Michał:
  - myślał o **napisaniu** takiej biblioteki lepiej
- Co dwie głowy...



Użyte technologie i praktyki

- Java 5 generics
- Design Patterns
  - Factory (połączona z Composite)
  - Typesafe Enum (sic! Nie Java 5 enum)
- Generowanie kodu
- Immutability
- Fluent Interface
- Testy jednostkowe
- Proces Continuous Integration

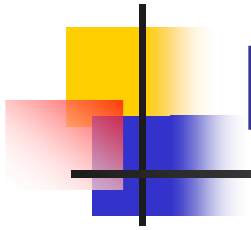


RADIUS – protokół AAA

- Authentication
  jesteś tym, za kogo się podajesz
- Authorization
  przyznaję Ci dostęp do moich zasobów
- Accounting
  monitoruję to, jak z nich korzystasz

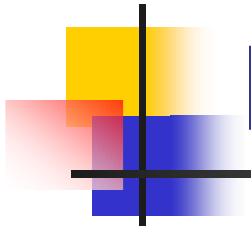
RADIUS - zastosowania

- Standard branży telekomunikacyjnej:
  - IETF
  - 3GPP (GSM, GPRS, UMTS, …)
  - WiMAX Forum
- Używają: Internet Service Providers (ISP)
- Klient – urządzenie dostępowe Network Access Server (NAS)
  - „półka” Neostrady / pula modemów
  - domowy router bezprzewodowy WI-FI
- Serwer – scentralizowana baza danych
  - Popularny FreeRADIUS



RADIUS w akcji

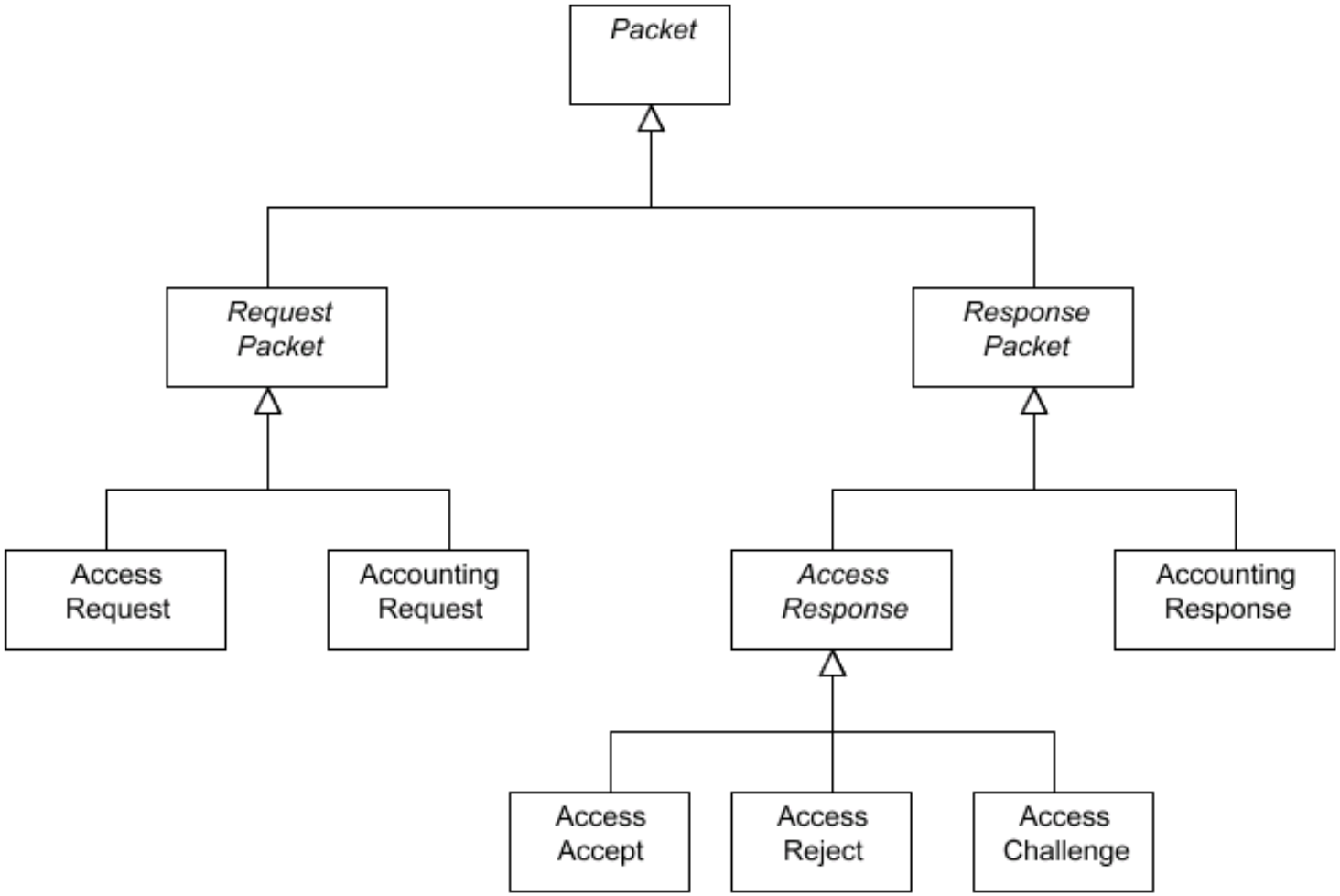
- Użytkownik próbuje połączyć się z Internetem
- NAS (Network Access Server) prosi o hasło
- **NAS przesyła login i hasło do serwera**
- **Serwer weryfikuje dane i zezwala NASowi wpuścić użytkownika**
- NAS wpuszcza użytkownika oraz ustala parametry połączenia (przepustowość, IP, ...)
- **NAS informuje serwer o wykorzystaniu łącza aż do końca sesji**



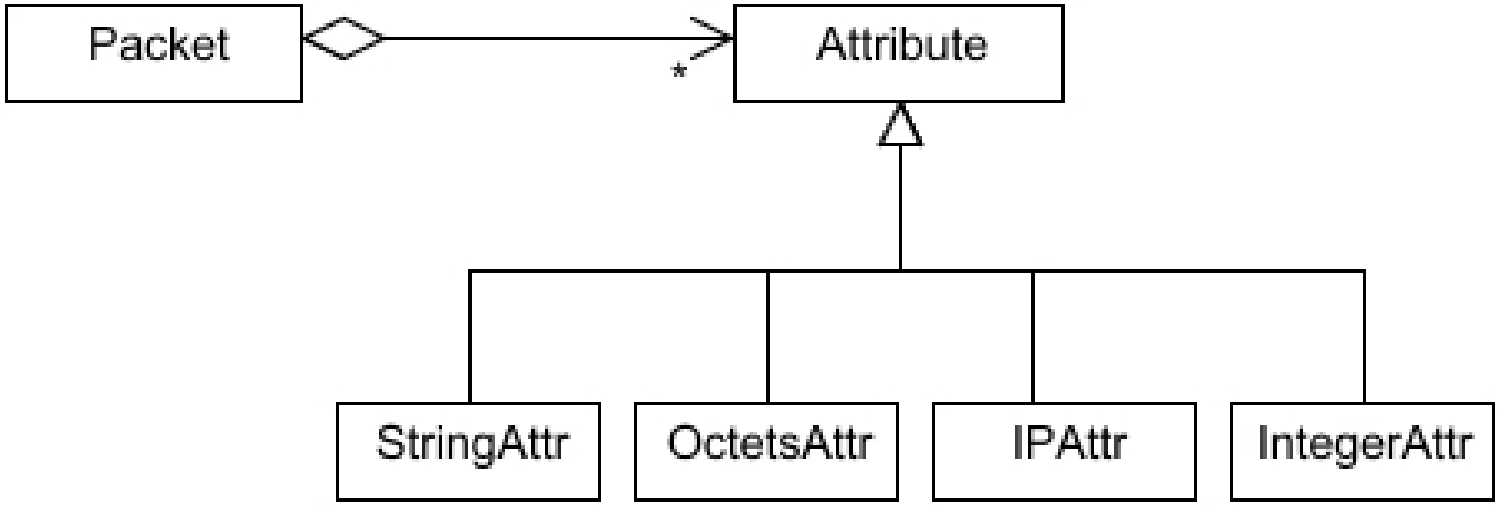
RADIUS – pakiety

- Kilkanaście typów, najczęściej stosowanych 6
- Typ pakietu - zasadnicze przesłanie (odpowiednik GET, POST, PUT i DELETE z HTTP)
- Pakiet zawiera:
  - Nagłówek (m.in. typ, ID, długość)
  - Listę atrybutów (pary klucz-wartość, odpowiednik nagłówków HTTP)

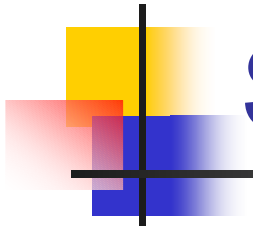
RADIUS – popularne pakiety



RADIUS - atrybuty



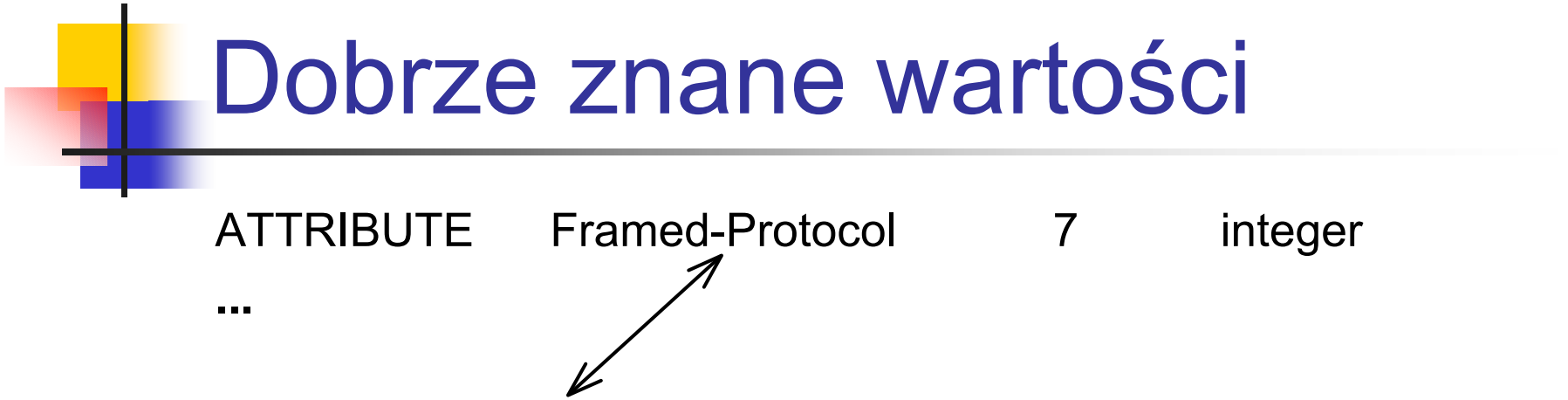
- Precyzują szczegóły (jak nagłówki w HTTP, np. Content-Type: text/html)
- Obecnie ~3000 rodzajów atrybutów
- Każdy posiada jeden ze zdefiniowanych typów



Słownik atrybutów

#Keyword	Attribute Name	Num	Type
ATTRIBUTE	User-Name	1	string
ATTRIBUTE	Password	2	string
ATTRIBUTE	CHAP-Password	3	string
ATTRIBUTE	Client-Id	4	ipaddr
ATTRIBUTE	Client-Port-Id	5	integer
ATTRIBUTE	Service-Type	6	integer
ATTRIBUTE	Framed-Protocol	7	integer
ATTRIBUTE	Framed-Address	8	ipaddr
ATTRIBUTE	Framed-Netmask	9	ipaddr
...	...	...	...
...	...	...	...

Dobrze znane wartości



ATTRIBUTE Framed-Protocol 7 integer

...

VALUE Framed-Protocol PPP 1

VALUE Framed-Protocol SLIP 2

VALUE Framed-Protocol ARAP 3

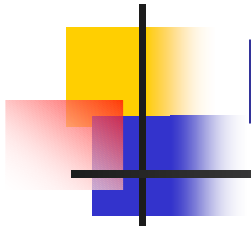
VALUE Framed-Protocol Gandalf-SLML 4

VALUE Framed-Protocol Xylogics-IPX-SLIP 5

VALUE Framed-Protocol X.75-Synchronous 6

... ...

... ...



RADIUS – pakiety AAA

→ Access-Request

(

User-Name = wroc3215

User-Password = kvFd3X5!

)

← Access-Accept

(

Session-Timeout = 24 h

Max-Bandwidth = 128 kbps

Download-Limit = 5 GB

)

RADIUS – pakiety AAA

→ Accounting-Request

(

Acct-Status-Type = Interim-Update

Acct-Output-Octets = 15211

Acct-Input-Octets = 4335211

)

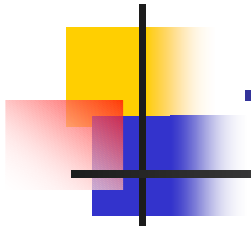
← Accounting-Response

(

« no attributes »

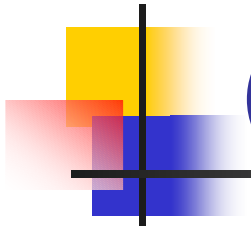
)

API biblioteki RADIUSa - minimalne wymagania



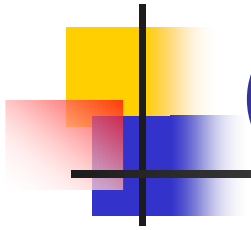
- Tworzenie atrybutów
- Tworzenie pakietów
  - Dodawanie atrybutów
- Odczytywanie atrybutów z pakietów
- Odczytywanie wartości atrybutów

Tworzenie atrybutu (Stara Szkoła)



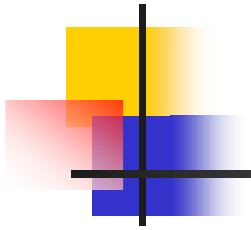
- new Attribute(
  Attribute.**User_Name**,
  „chuck”
)
- new Attribute(
  Attribute.**Framed_IP_Address**,
  InetAddress.getByName(**„1.2.0.1”**)
)
- new Attribute(
  Attribute.**Session_Timeout**,
  180
)

C.D. Tworzenia atrybutu (Stara Szkoła)



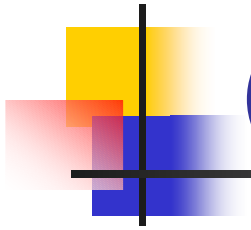
- Ale też:
  new Attribute(
  Attribute.**Session_Timeout**,
  „180 sec” // tu powinien być int
)
- Wymaganie, aby programista znał typ każdego atrybutu
  - W praktyce – posługiwał się referencją atrybutów
- „fail last” zamiast „fail fast”
- Problematyczny idiom „stałych typu int”

Tworzenie atrybutu - rozwiązanie



- Osobna klasa dla każdego atrybutu
  - Klasy można wygenerować automatycznie na podstawie słownika atrybutów
- Klasa zawiera informację o typie atrybutu
  - UserName(String username) { ... }
  - NasIpAddress(InetAddress ip) { ... }
  - SessionTimeout(long timeout) { ... }
- Bez dokumentacji można się już obejść
  - Autouzupełnianie w IDE

Odczyt wartości atrybutów (Stara Szkoła)

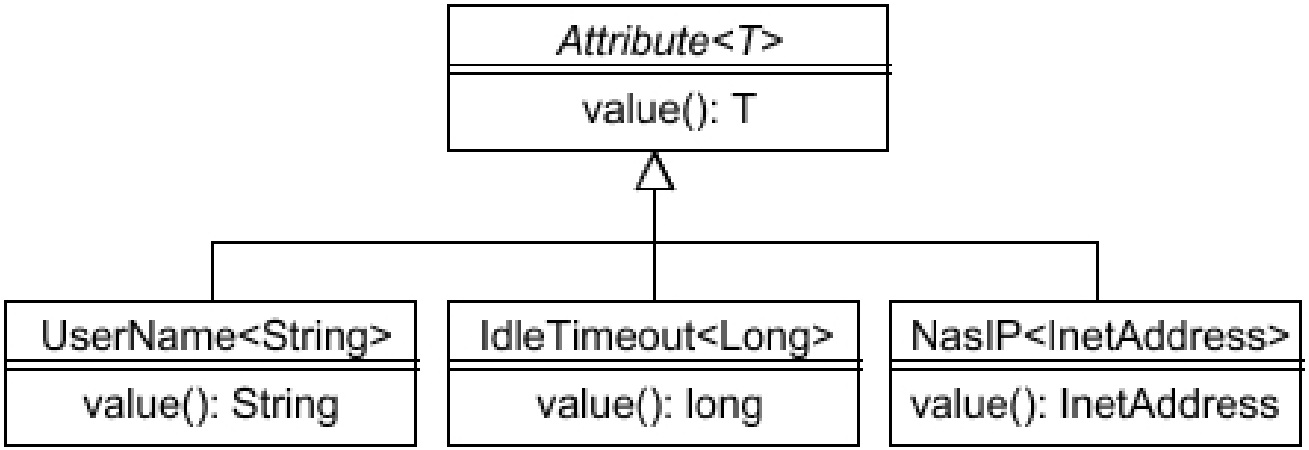


- W stylu retro (bez osobnych klas atrybutów)
  - name = usernameAttribute.valueAsString()
  - Wymaga znajomości typu atrybutu
- Rozwiązanie z klasami atrybutów
  - class Attribute { ...
    public Object value() { ... }
    }
  - name = (String) usernameAttribute.value()
  - nasIp = (InetAddress) nasIpAttribute.value()
  - Wymagane rzutowanie, ale typ wartości atrybutu można sprawdzić w jego klasie (z poziomu IDE)

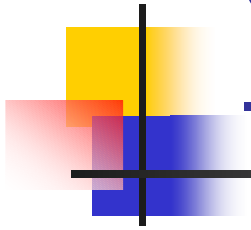
Odczyt wartości atrybutu - próbka Javy 1.5

- Generics
  - Każdy konkretny atrybut ma swój typ:

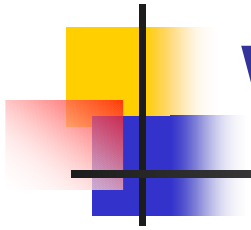
```
class UserName extends Attribute<String>
```



Odczyt wartości atrybutu
- efekt użycia generics



- UserName name = new UserName(„cnorris”);
  String username = name.value();
- IdleTimeout timeout = new IdleTimeout(5 * 60);
  long idleTimeout = timeout.value();
- NasIP serverIP =
  new NasIP(InetAddress.getByName(„10.1.0.1”));
  InetAddress accessServerIP = serverIP.value();
- Rzutowania stają się zbędne
  - Kompilator podpowiada właściwy typ



Wydobycie atrybutu z pakietu

- Stara szkoła – typ atrybutu określany liczbą:

```
class Packet {
      Attribute getAttribute(int type) { ... } ... }

UserName name = (UserName)
      packet.getAttribute(UserName.TYPE);

IdleTimeout timeout = (IdleTimeout)
      packet.getAttribute(IdleTimeout.TYPE);
```

- getAttribute(int) gubi informację o typie – konieczne jest użycie rzutowań

Wydobywanie atrybutów - bez rzutowań?

- Generics na ratunek

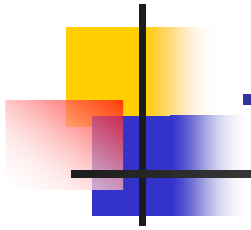
typem zwracanym metody może być
typ do niej przekazany jako argument

```
<A extends Attribute> A getAttribute(Class<A> attribType)
```

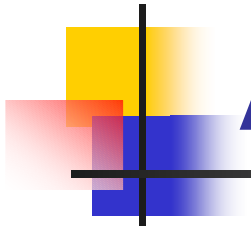
- Pomysł: określać typ klasą (typem) Javy

```
UserName name = p.getAttribute(UserName.class);
IdleTimeout idle = p.getAttribute(IdleTimeout.class);
NasIP nasIP = p.getAttribute(NasIP.class);
```

Pobieranie wartości atrybutu - jeszcze prościej



- Cel: pobieranie wartości atrybutu wprost z pakietu
  - Bez używania instancji atrybutu
- <T, A extends Attribute<T>>
  T **valueOf(**Class<A> attribType**)**
  - **String** name = p.valueOf(**UserName**.class)
  - **long** canStayIdle = p.valueOf(**IdleTimeout**.class)
  - **InetAddress** ip = p.valueOf(**NasIP**.class)

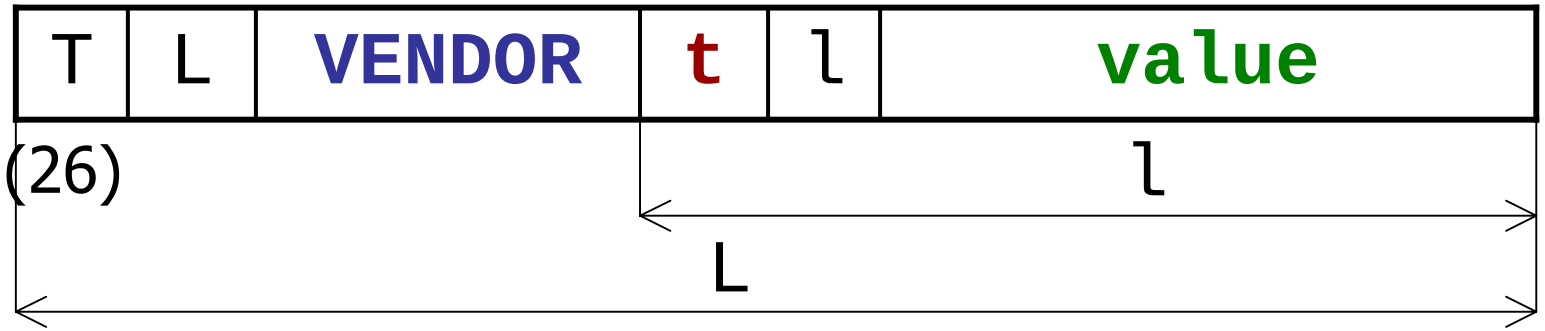


Atrybuty wyliczeniowe

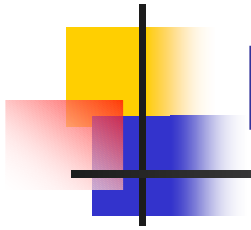
- Typesafe enum pattern
  - Dlaczego nie typ wyliczeniowy Javy 1.5
  - Dziedziczenie
  - Wartość „nieznana”
  - Zanieczyszczenie API metodami klasy Enum

Atrybuty Vendor-Specific

- Ekspansja przestrzeni atrybutów



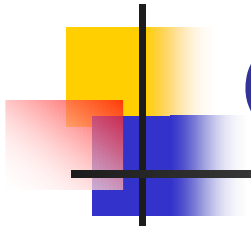
- Attribute(long **vendor**, int **type**, String **value**) ?
  - new Attribute(
    Vendors.WISPR,
    WISPr.LocationID,
    „**Krakow**”)
- new radius.attribute.**wispr**.**LocationID**(„**Krakow**”) !



Fluent Interface

- Zwracanie **this** przez metody zwracające do tej pory **void**
- Ułatwia zwięzłe konstruowanie wyrażeń:

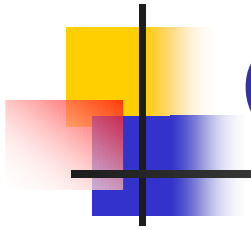
```
AccessAccept acceptPacket = request.
        accept().
        add(
                new ReplyMessage(„Welcome home”)
        );
```



Od zer i jedynek do obiektów

- *Packet* create*(ByteBuffer bytes)*

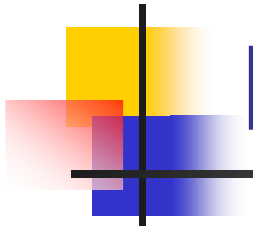
	prostota	elastyczność	obfuscators
switch	+	-	+
refleksja	+	+	-
composite factory	-	+	+



Composite factory

- Środki
  - Zastąp switch polimorfizmem (Fowler)
  - Drzewo fabryk
  - Uformuj metodę szablonu
- Efekt
  - Łatwa kontrola nad fabrykami
  - Bezproblemowe użycie obfuscatorów

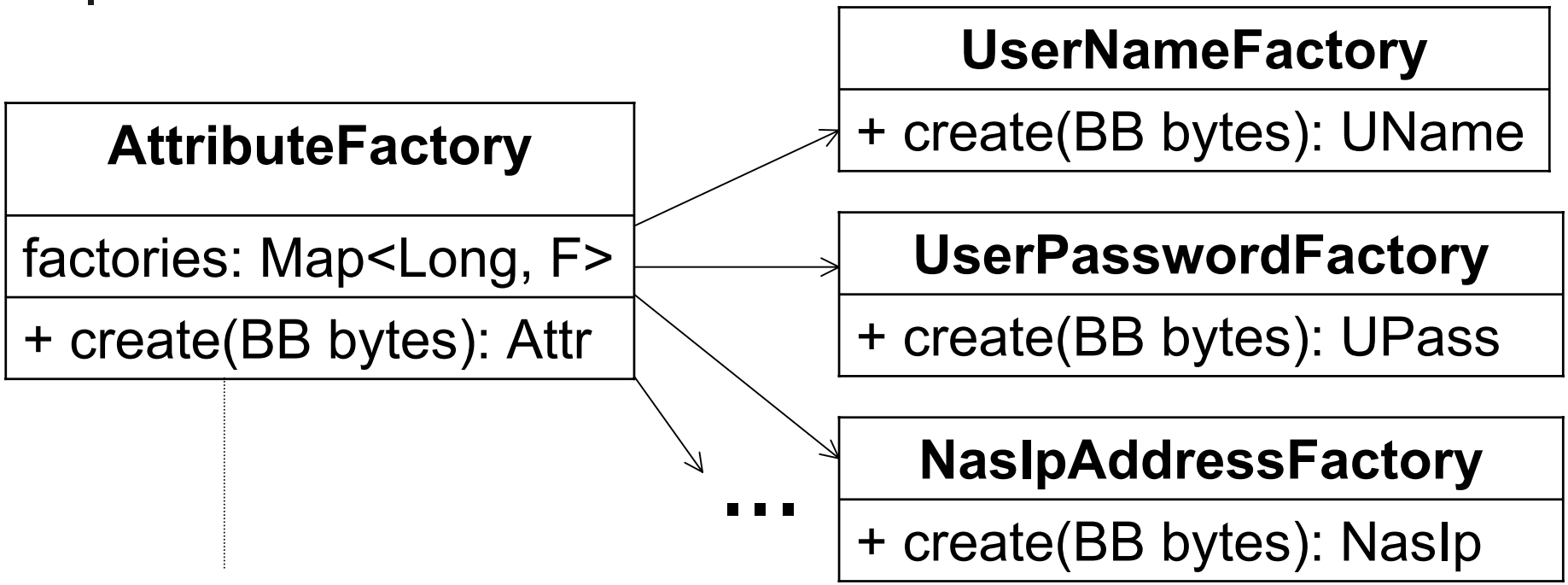
Punkt wyjścia - switch



AttributeFactory
+ create(ByteBuffer byteForm): Attribute

- switch (attributeType) {
  case UserName.TYPE:
  return new UserName(val);
  case UserPassword.TYPE:
  return new UserPassword(val);
  case NasIpAddress.TYPE:
  return new NasIpAddress(val);
  case ...

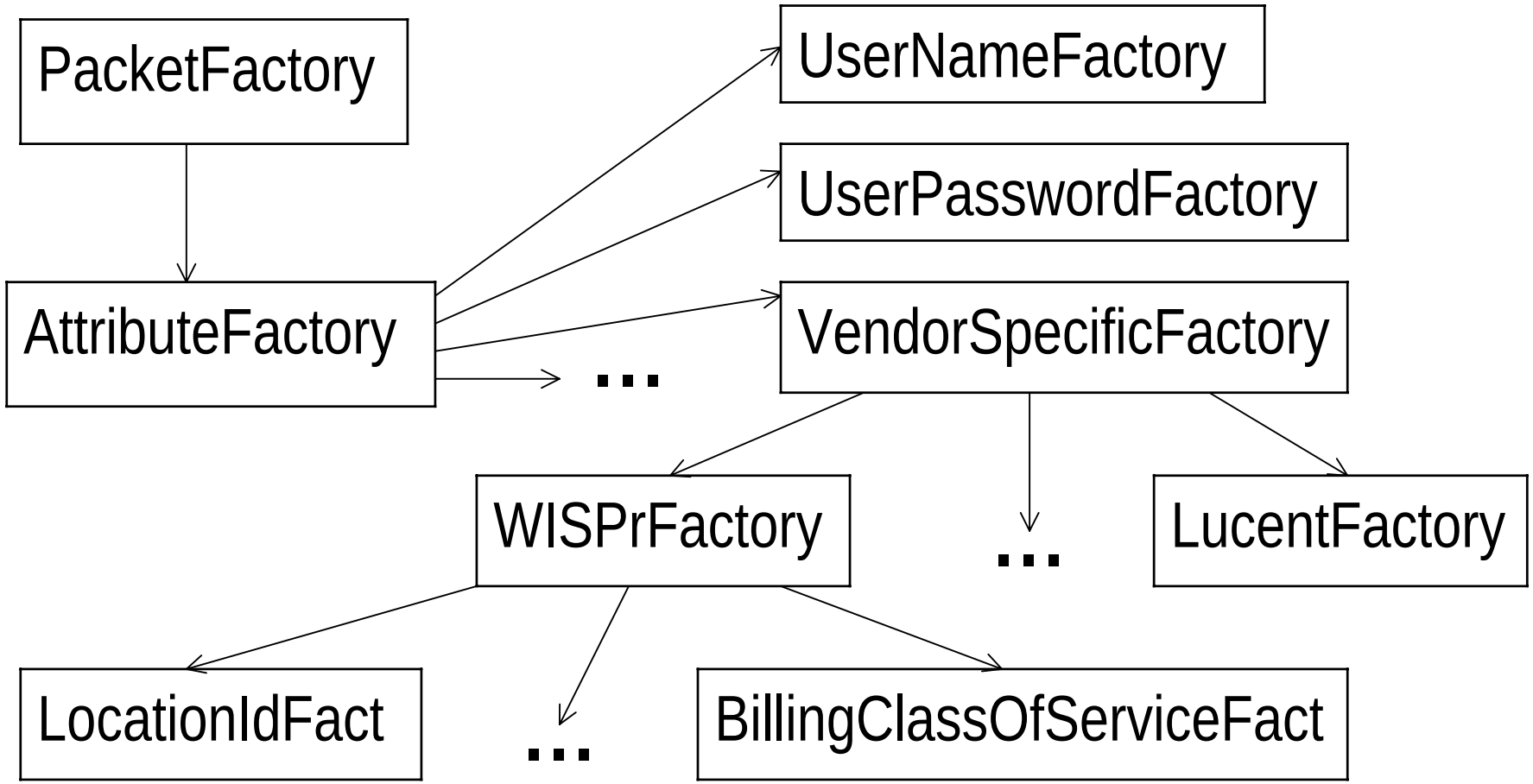
Zastąp switch polimorfizmem

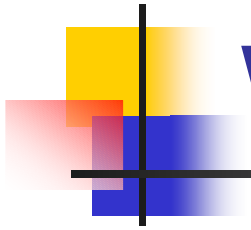



```
int nr = attributeType(bytes);
Factory f = factories.get(nr);
return f.create(value);
```

```
return new UserName(bytes);
```

Drzewo fabryk

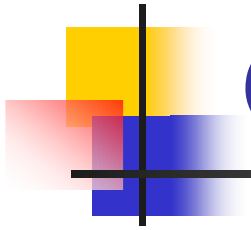




Wspólne cechy fabryk

- Rozpoznawanie typu tworzonego obiektu
- Rejestrowanie fabryk składowych
- Wybór i użycie odpowiedniej fabryki składowej

CompositeFactory<T>
+ create(BB bytes): T # readTypeSpecifier(BB bytes): long + getOwnTypeSpecifier(): long + registerSubFactory(CF subFact)

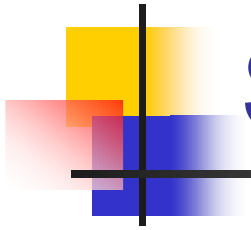


Osiągnięte korzyści

- API przyjazne dla użytkowników IDE
- Zredukowane możliwości pomyłek

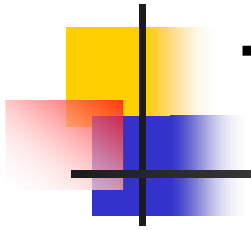
i...

- Radość z używania biblioteki! :-)



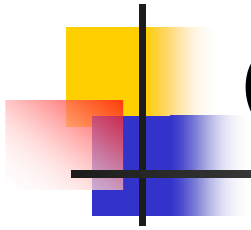
Strong vs loose typing

- Weryfikacja poprawności programu
  - Statyczna
  - Dynamiczna
- Produktywność
  - J. dynamiczne
    - Niska bariera wejścia
    - Potężne konstrukcje językowe
  - J. statycznie typizowane
    - IDE to w praktyce prerekwizyt
    - Potężne IDE i narzędzia wspomagające



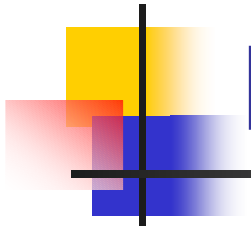
Testy jednostkowe

- Pozwalają weryfikować poprawność działania kodu
- Znakomicie dokumentują kod
- Dają pewność że pomimo wprowadzonych zmian, nasz stary kod działa tak jak powinien
- Możliwość śledzenia wartości % pokrycia kodu testami jednostkowymi



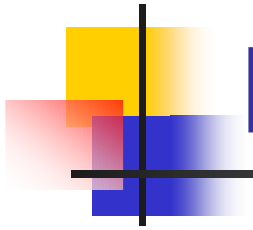
Continuous Integration

- Subversion + Apache Maven 2 + Apache Continuum
- Proces budowy (kompilacja, testowanie, pakowanie, raportowanie)
- Zautomatyzowany po wykryciu zmian w repozytorium kodu
- Publikacja wyników procesu budowania:
  - Status testów (JUnit)
  - Status pokrycia kodu testami (Emma)
  - Świeża dokumentacja Javadoc
  - Świeże pakiety JAR

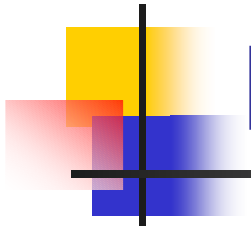


Dziękujemy za uwagę

- http://radius.safehaus.org

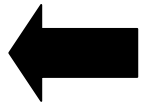


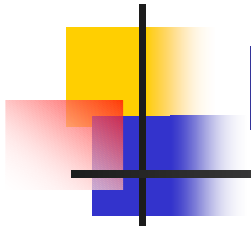
Pytania?



Fail fast

- W wypadku błędu, przerwanie bieżącej czynności zamiast próby kontynuowania
- Zgłoszenie szczegółowego błędu do wyższych warstw, odpowiedzialnych za jego obsługę
- http://www.martinfowler.com/ieeeSoftware/failFast.pdf
- http://en.wikipedia.org/wiki/Fail-fast





Idiom „stałych typu int”

- Problemy używania
  - Pierwszy kontakt – „co to za liczba?”
  - Następne kontakty – „w jakiej klasie były zdefiniowane te stałe?”
- Dobre zamienniki
  - Konstrukcja enum (Java >= 5)
  - Typesafe enum pattern
  - Wyjątek – środowiska z wieloma Classloaders