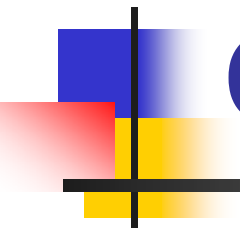
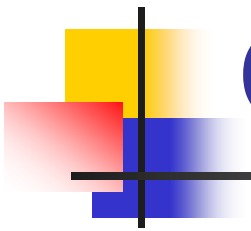


Mechanizmy refleksji w Javie i C#

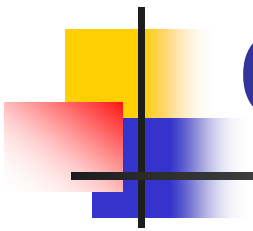
Autor: Aleksander Nałęczyński



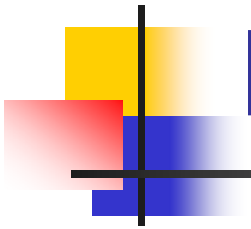
Co to jest refleksja? [1/2]

- Możliwość obserwowania lub manipulowania pracą programu od wewnątrz
- Problem
  - Zazwyczaj niskopoziomowa struktura programu różni się od wysokopoziomowej

Co to jest refleksja? [2/2]

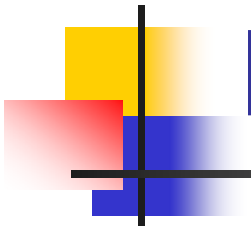


- System refleksyjny (reflective system)
  - System zawierający struktury reprezentujące aspekty samego siebie
  - System ma zawsze dokładną reprezentację samego siebie
  - Reprezentacja zawsze zgadza sie z aktualnym stanem systemu



Motywacja

- Tworzenie statystyk działania programu
- Debuggowanie
- Skrypty użytkownika
- Komunikacja z innymi systemami
- Systemy samokonfigurujące

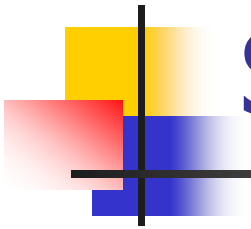


Historia

- Programy modyfikujące własny kod
- Forth
- Lisp (Scheme, CLOS)
- Prolog

Architektury refleksyjne

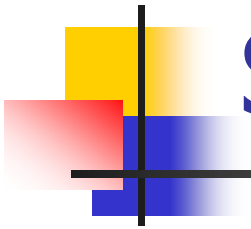
- meta-circular interpreter
  - 3-LISP
  - 3-KRS
  - FOL
  - TEIRESIAS



Scheme

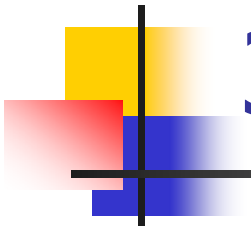
- Kontynuacje
- Refleksja proceduralna (procedural reflection)
  - Reprezentacja systemu w terminach programu implementującego system

```
(eval `(+ 1 2))
```



Smalltalk

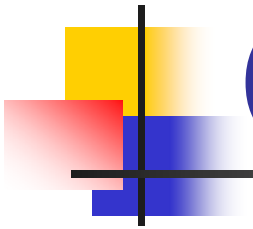
- Refleksja dobrze wpasowuje się w paradygmat obiektowy
- W pełni refleksyjny system
- Dynamiczny system typów
- Możliwość dodawania nowych klas w trakcie działania
- Smalltalk-80
  - Metaklasy



3-KRS

- Każdy obiekt jest związany z meta-obiektem
- Pełna, modyfikowalna samoreprezentacja programu

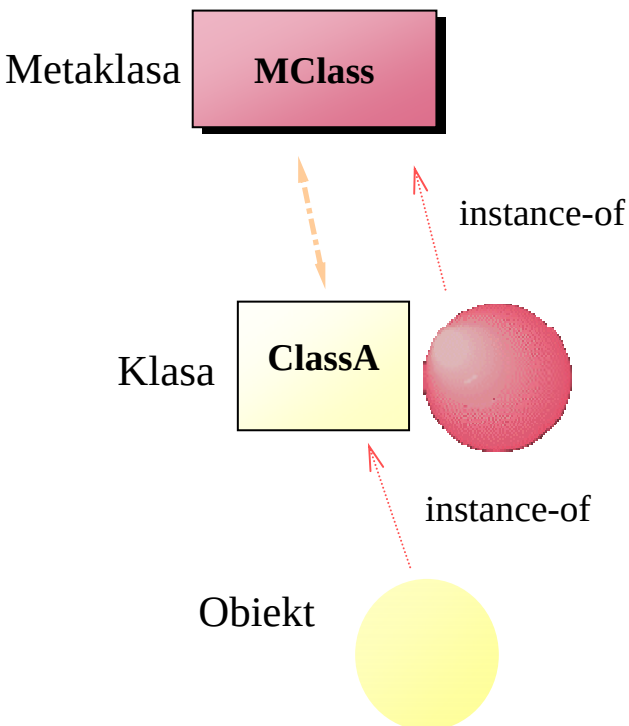
Protokoły metaobiektowe (MOP)



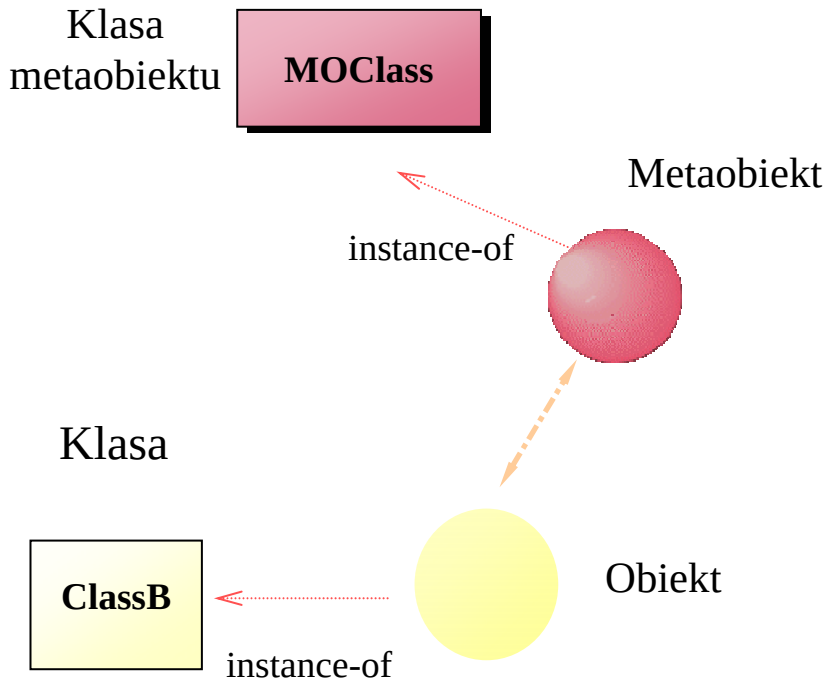
- Reprezentujemy instancje klas, metody i komunikaty jako dane
- Przechwytujemy komunikaty między obiektami, przekazujemy je do metaobiektu
- Kontynuujemy normalne wykonanie

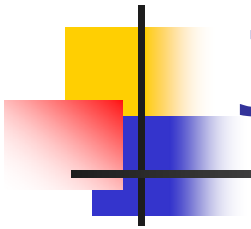
Modele refleksji

Metaklasy



Metaobiekty





Java i C#

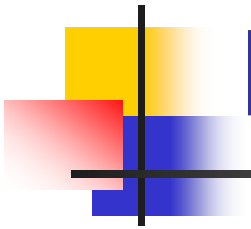
- Introspekcja, ale nie modyfikacja struktur
- Tworzenie instancji klasy
- Dynamiczne wołanie metod
- Zmiana wartości pól

java.lang.reflect vs System.Reflection

- java.lang.Class
- Field
- Method
- Constructor<T>

- Assembly
- System.Type
- FieldInfo
- MethodInfo
- PropertyInfo
- ConstructorInfo

Dynamiczne ładowanie klas



- Java

```
Class c = Class.forName("MojaKlasa");
```

- C#

```
Assembly a = Assembly.Load("assembly.dll");
Type t = a.getType("MojaKlasa");
```

Opóźnione wiązanie

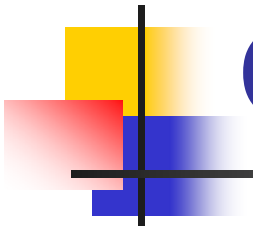
- Java

```
Object o = c.newInstance();
// wywołanie metody bezparametrowej
Method m = c.getMethod("toString", null);
String s = m.invoke(o, null);
```

- C#

```
Type theMathType = Type.GetType("System.Math");
Object theObj = Activator.CreateInstance(theMathType);
Type[] paramTypes = new Type[1]; paramTypes[0]=
     Type.GetType("System.Double");
MethodInfo CosineInfo =
     theMathType.GetMethod("Cos",paramTypes);
Object[] parameters = new Object[1];
parameters[0] = 45;
Object returnVal =  CosineInfo.Invoke(theObj,parameters);
```

C#: atrybuty [1/5]



- Wbudowane (intrinsic attributes)

```
using System.Security.Permissions;
[assembly:
FileIOPermission(SecurityAction.RequestMinimum)
]
public class FileManager {...}
```

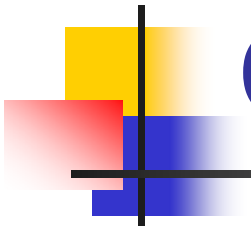
C#: atrybuty [2/5]

- Użytkownika (custom attributes)

```
public class BugFixAttribute: System.Attribute
{
   // przynajmniej 1 konstruktor musi być zdefiniowany
   // Tutaj są ustawiane wartości parametrów pozycyjnych
   public BugFixAttribute(int bugID, string programmer,
   string date) {
     this.bugID = bugID;
     this.programmer = programmer;
     this.date = date;
   }
```

C#: atrybuty [3/5]

```
// parametry nazwane są implementowane jako właściwości
   public string Comment {
        get {
               return comment;
        }
        set {
               comment = value;
        }
   }
} // BugFixAttribute
```



C#: atrybuty [4/5]

- Sposób użycia

```
/* Określenie wartości parametrów pozycyjnych
   jest obowiązkowe, opcjonalne dla parametrów
   nazwanych */
[BugFixAttribute(121,"Jesse Liberty","01/03/05")]
[BugFixAttribute(107,"Jesse Liberty","01/04/05",
   Comment="Fixed off by one errors")]
   public class MyMath {...}
```

C#: atrybuty [5/5]

- Meta-atrybuty

```
[AttributeUsage(AttributeTargets.Class |
AttributeTargets.Constructor |
AttributeTargets.Field |
AttributeTargets.Method |
AttributeTargets.Property,
AllowMultiple = true)]
public class BugFixAttribute : System.Attribute
{...}
```

Java: adnotacje [1/3]

- Wbudowane
  http://java.sun.com/j2se/1.5.0/docs/api/java/lang/annotation/Annotation.html

```
@Deprecated
public class SomeClass {...}

@Deprecated
public void method() {...}

@Override
public String toStirng() {...}
```

Java: adnotacje [2/3]

- Użytkownika

```
import java.lang.annotation.*;
public @interface BugFix {
     int id();
     String name() default "[anonymous]";
     String date();
}
@BugFix(
     id = 2868724,
     name = "G. Kiczales”
     date = "4/1/2007"
)
public static void takeYourTime() {...}
```

Java: adnotacje [3/3]

- Meta-adnotacje

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface Test {...}
```

- Deklaracje meta-adnotacji

```
// Adnotacja @Target opisuje również samą siebie
@Documented
@Retention(value=RUNTIME)
@Target(value=ANNOTATION_TYPE)
public @interface Target
```

Java: klasy proxy

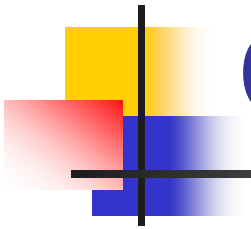
- java.lang.Reflect.Proxy
- Motywacja – np. stworzenie obiektu implementującego wiele interfejsów dziedziczących po EventListener

```
static Class<?> getProxyClass(ClassLoader loader,
    Class<?>... interfaces)

static Object newProxyInstance(ClassLoader loader,
    Class<?>[] interfaces, InvocationHandler h)
```

- Brak tego mechanizmu w C#

C#: Kompilacja [1/2]



- Zapisujemy kod C# do pliku
- Wywołujemy kompilator C#

```
ProcessStartInfo psi = new ProcessStartInfo( );
psi.FileName = "cmd.exe";
string compileString = "/c csc /optimize+ ";
compileString += " /target:library ";
compileString += "{0}.cs > compile.out";
psi.Arguments = String.Format(compileString,
fileName);
psi.WindowStyle = ProcessWindowStyle.Minimized;
Process proc = Process.Start(psi);
proc.WaitForExit( );
```

C#: Kompilacja [2/2]

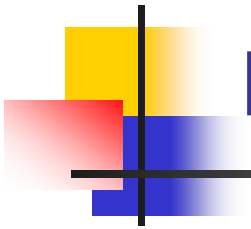
- Ładujemy powstałą bibliotekę

```
Assembly a = Assembly.LoadFrom(fileName + ".dll");
Object theObject = a.CreateInstance(className);
Type theType = a.GetType(className);
```

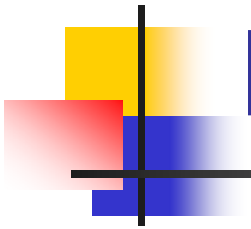
- Usuwamy plik z kodem

```
File.Delete(fileName + ".cs");
```

Słabości mechanizmów refleksji w Javie i C#

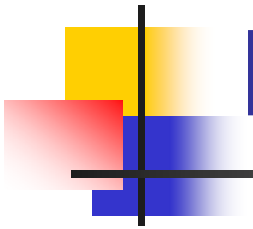


- Nie można dodawać metod, ani zmieniać kodu istniejących
- Nie można dziedziczyć po metaklasach
  - **Class**
  - **Method**

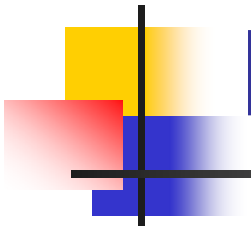


Rozszerzenia Javy

- MetaJava
- Guarana
- JMOP



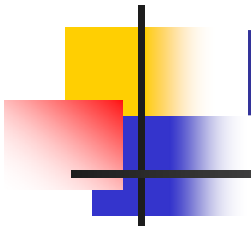
Refleksja a AOP



Bibliografia

- Gregory T. Sullivan, “AOP using Reflection and MetaObject Protocols”, CACM 2001

  people.csail.mit.edu/u/ g/gregs/public_html/cacm-sidebar.pdf
- Jesse Liberty, “Programming C#”, O’Reilly 2001
- MSDN Library – Kirk Radeck, „C# and Java: Comparing Programming Languages”



Bibliografia

- http://java.sun.com/j2se/1.5.0/docs/api/java/lang/reflect/package-summary.html
- http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpref/html/frlrfsystemreflection.asp
- http://java.sun.com/j2se/1.5.0/docs/guide/language/annotations.html
- http://java.sun.com/j2se/1.5.0/docs/guide/reflection/proxy.html