OO testování

Radek Mařík

ČVUT FEL, K13132

October 2, 2014

Obsah

Dědičnost [AO08, AO10]

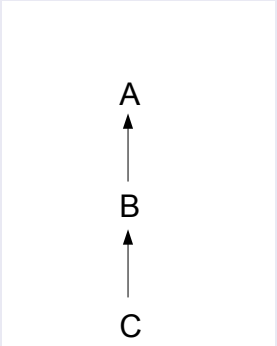
- Umožňuje, aby **společné vlastnosti** mnoha tříd byly definovány v jedné třídě
- **Odvozená** třída má vše, co má její bázová třída. Navíc může:
  - **vylepšit** odvozené vlastnosti (přepsáním)
  - **omezit** odvozené vlastnosti
  - **přidat** nové vlastnosti (rozšířením)

Dědičnost typů [AO08, AO10]

- **Deklarovaný typ** . . . typ specifikovaný v deklaraci reference na objekt
  - *Clock w1; // declared type Clock*
- **Skutečný typ** . . . typ skutečného objektu
  - *w1 = new Watch(); // actual type Watch*

Metoda v Java

Metoda, která se volá, je nejnižší verze metody definované mezi skutečným a deklarovaným typem v hierarchii dědičnosti.

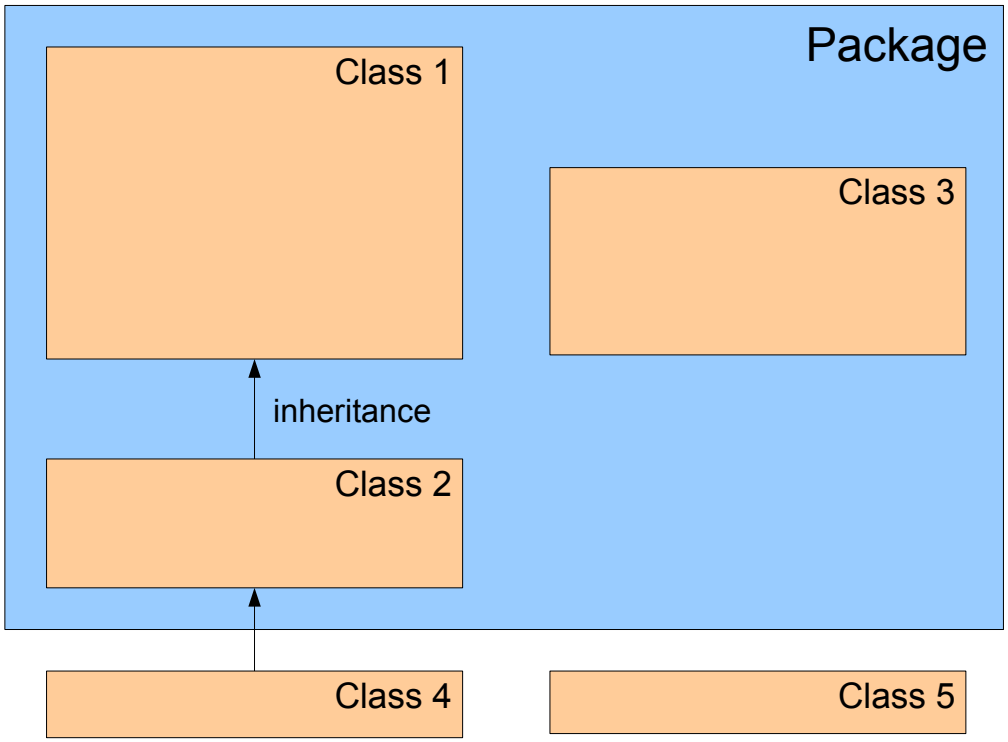


Dědičnost podtypu a podtřídy [AO08, AO10]

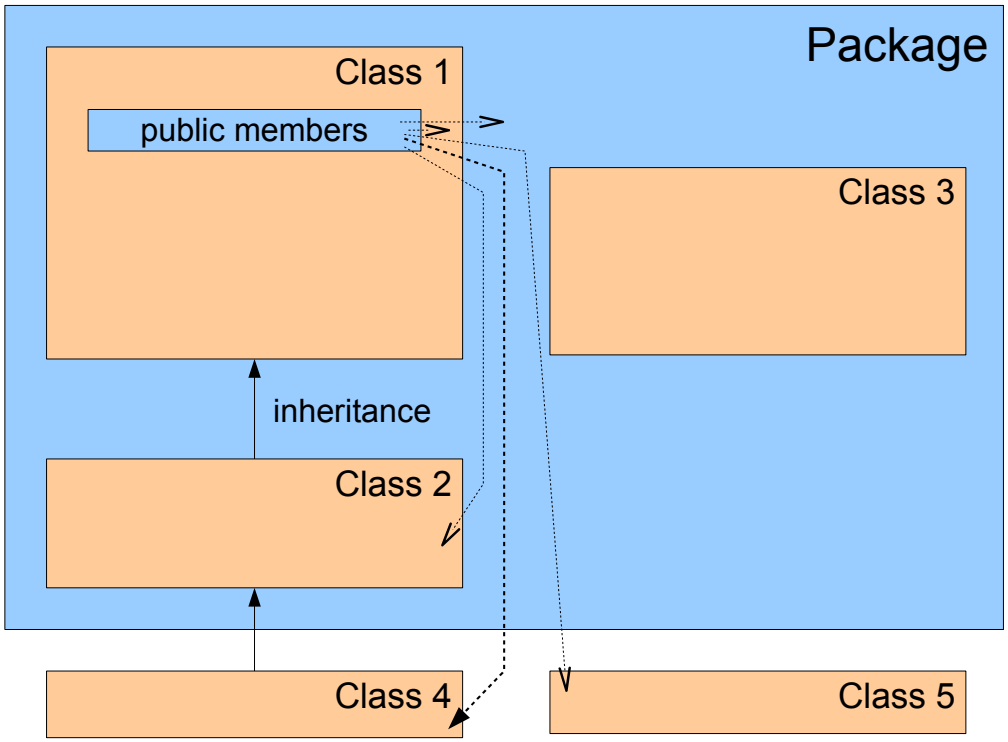
Jestliže B dědí z A, pak

- **Dědičnost podtypu** ... kterýkoliv objekt typu B může nahradit objekt typu A.
  - *Notebook* "je" speciálním typem *počítače*.
  - *nahraditelnost*.
- **Dědičnost podtřídy** ... objekty typu B nesmí být použity jako náhrady za objekty typu A.
  - Objekty typu B nemusí být "*typově kompatibilní*".
  - Stack dědí z Vector ... pohodlné pro implementaci, ale stack **není** určitě vektor.

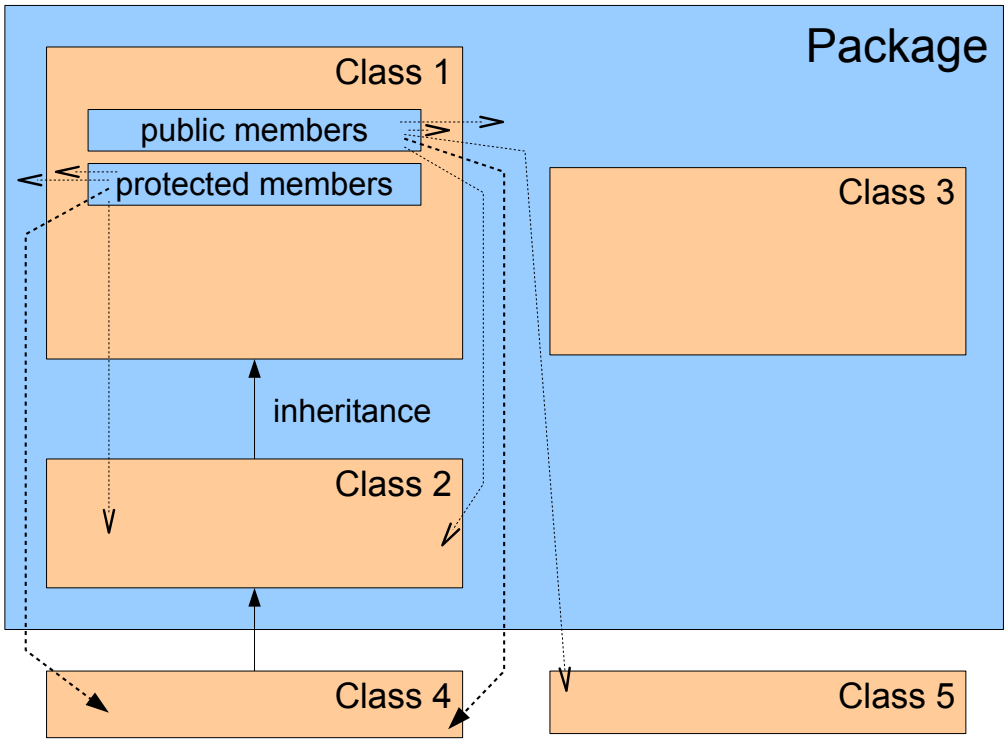
Řízení přístupu (v Java) [AO08, AO10]



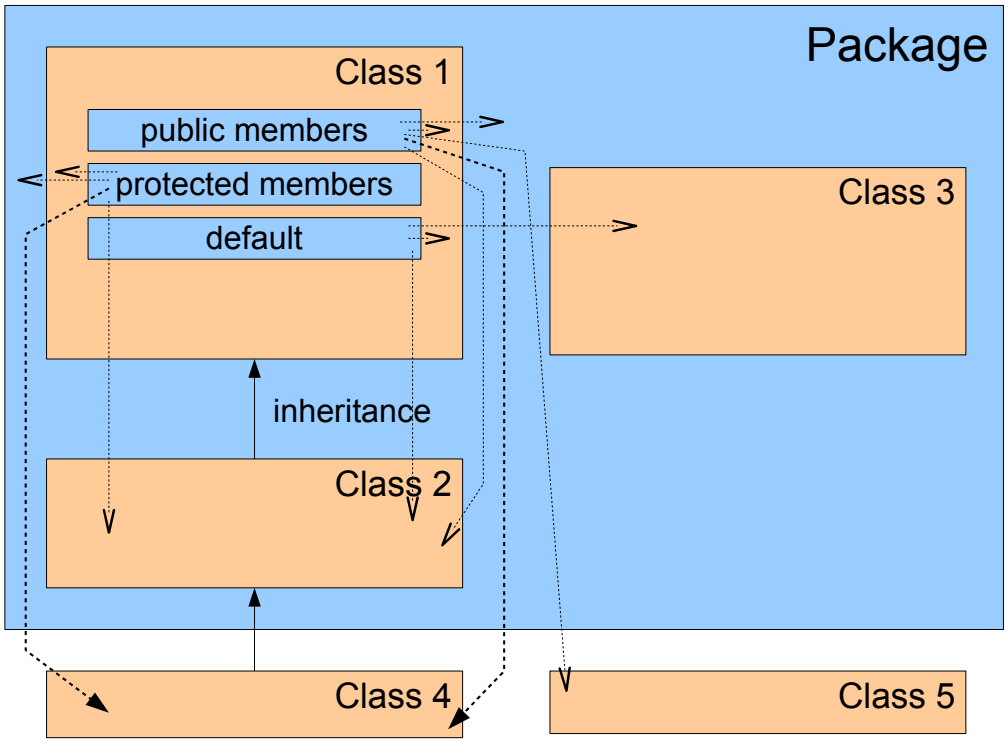
Řízení přístupu (v Java) [AO08, AO10]



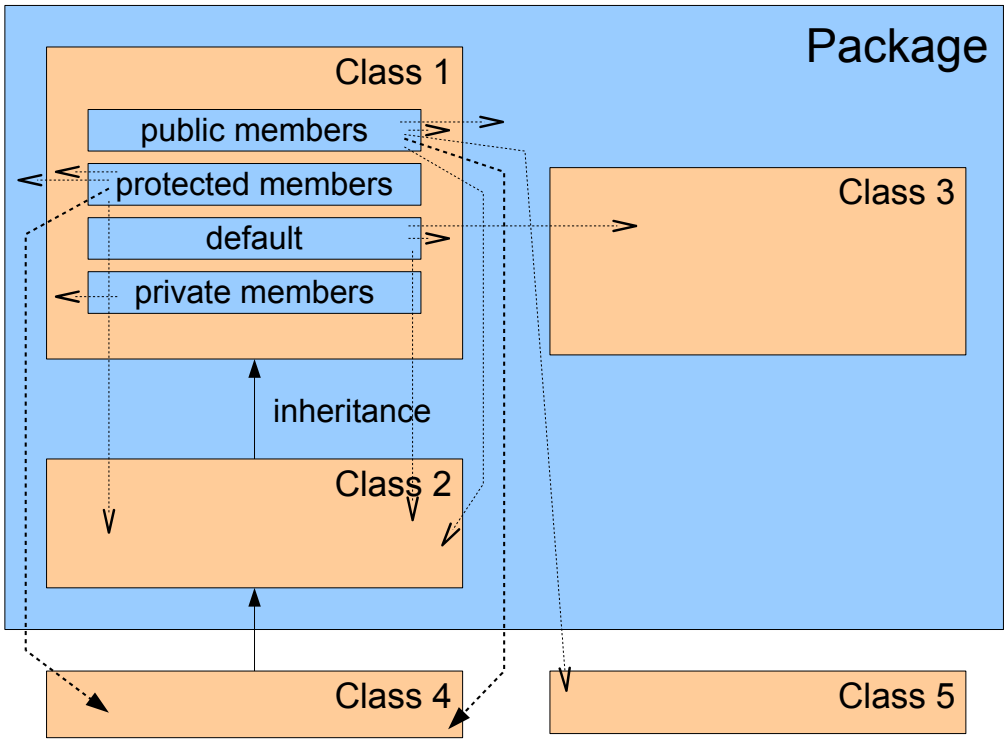
Řízení přístupu (v Java) [AO08, AO10]



Řízení přístupu (v Java) [AO08, AO10]



Řízení přístupu (v Java) [AO08, AO10]



Polymorfizmus [AO08, AO10]

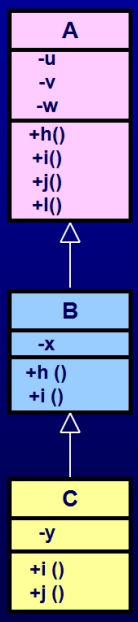
- Ta samá proměnná může mít **různé typy**, které závisí na běhu programů.
- Jestliže B dědí z A, potom objekt typu B **může být použit** tam, kde se očekává objekt typu A.
- Jestliže jak A tak B **definují tu samou metodu** M (B přepisuje A), potom ten samý příkaz může volat jak A verzi M, tak i B verzi.

Typy testování [AO08, AO10]

- **Testování intra-metod** . . . testování v rámci jedné metody jedné třídy.
- **Testování inter-metod** . . . testování v rámci několika metod jedné třídy.
- **Testování intra-třídy** . . . testování v rámci jedné třídy.
  - obvykle použitím sekvencí volání metod v rámci jedné třídy.
- **Testování inter-tříd** . . . testování několika tříd najednou.
  - ověřování možností integrace

DU páry a jejich možné anomálie 1 [AO08, AO10]

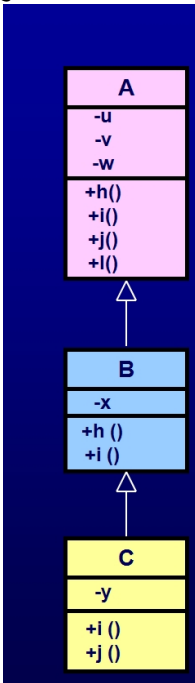
Přepisující metoda má jinou def-množinu než přepisovaná metoda?

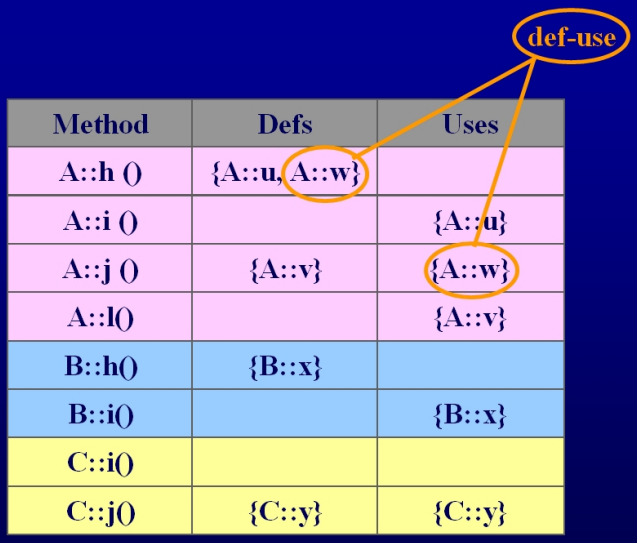


Method	Defs	Uses
A::h ()	{A::u, A::w}	
A::i ()		{A::u}
A::j ()	{A::v}	{A::w}
A::l()		{A::v}
B::h()	{B::x}	
B::i()		{B::x}
C::i()		
C::j()	{C::y}	{C::y}

DU páry a jejich možné anomálie 2 [AO08, AO10]

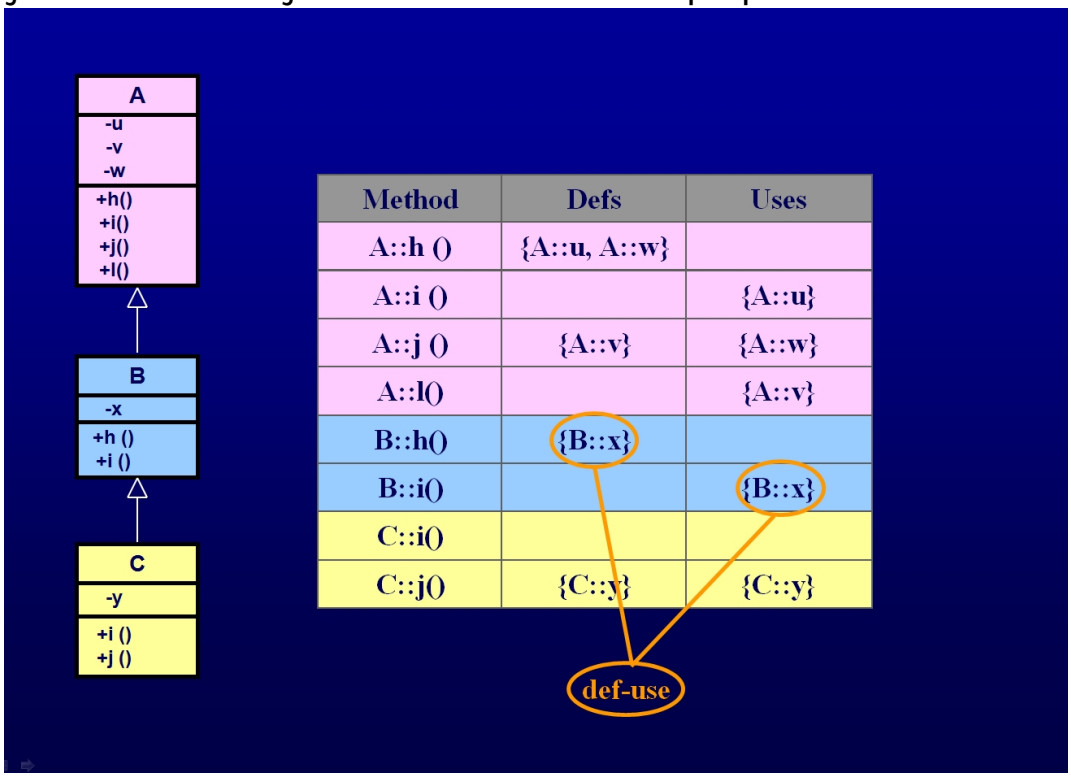
Přepisující metoda má jinou def-množinu než přepisovaná metoda?





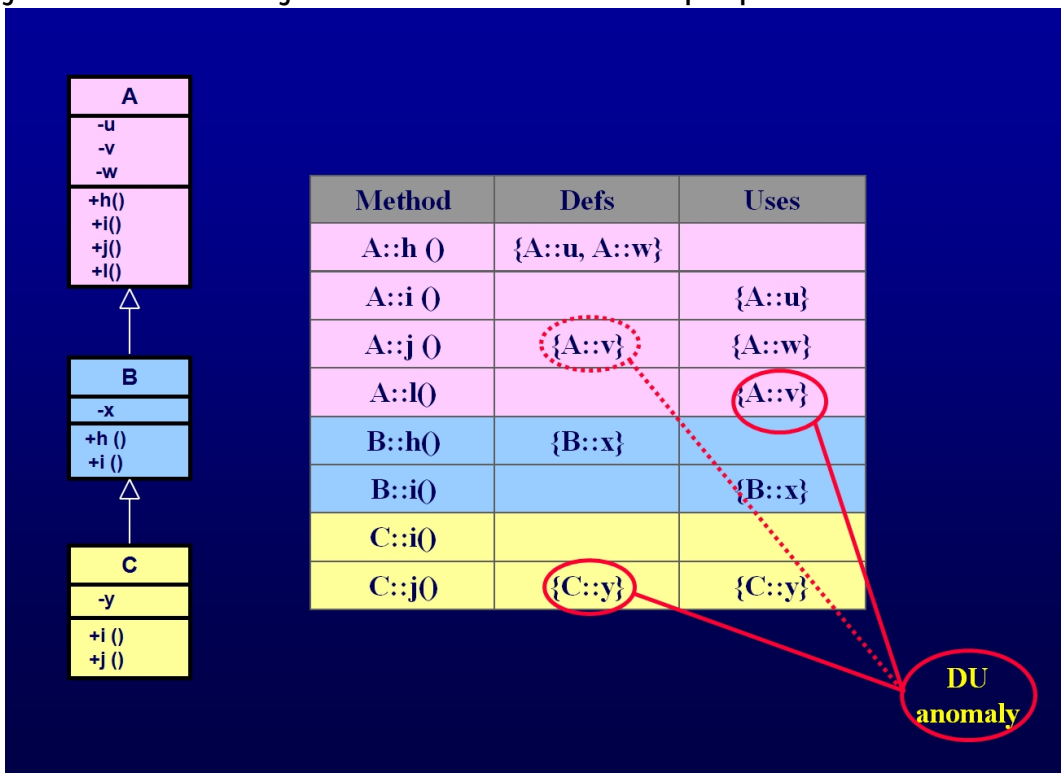
DU páry a jejich možné anomálie 3 [AO08, AO10]

Přepisující metoda má jinou def-množinu než přepisovaná metoda?



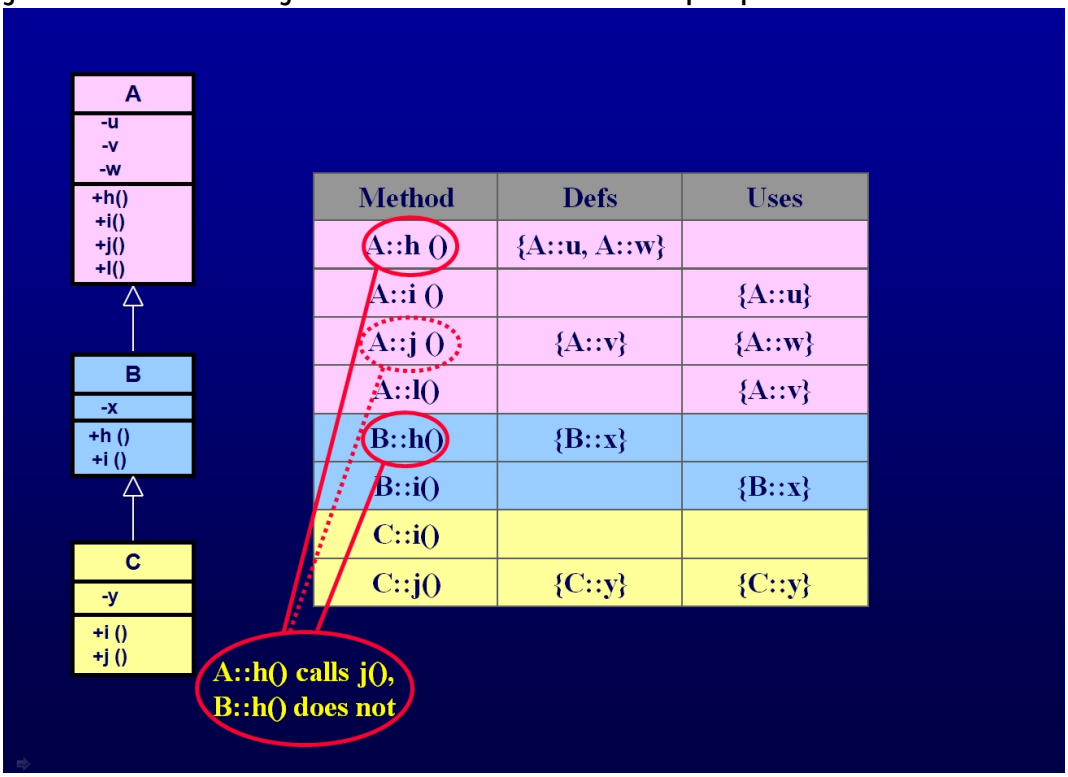
DU páry a jejich možné anomálie 4 [AO08, AO10]

Přepisující metoda má jinou def-množinu než přepisovaná metoda?

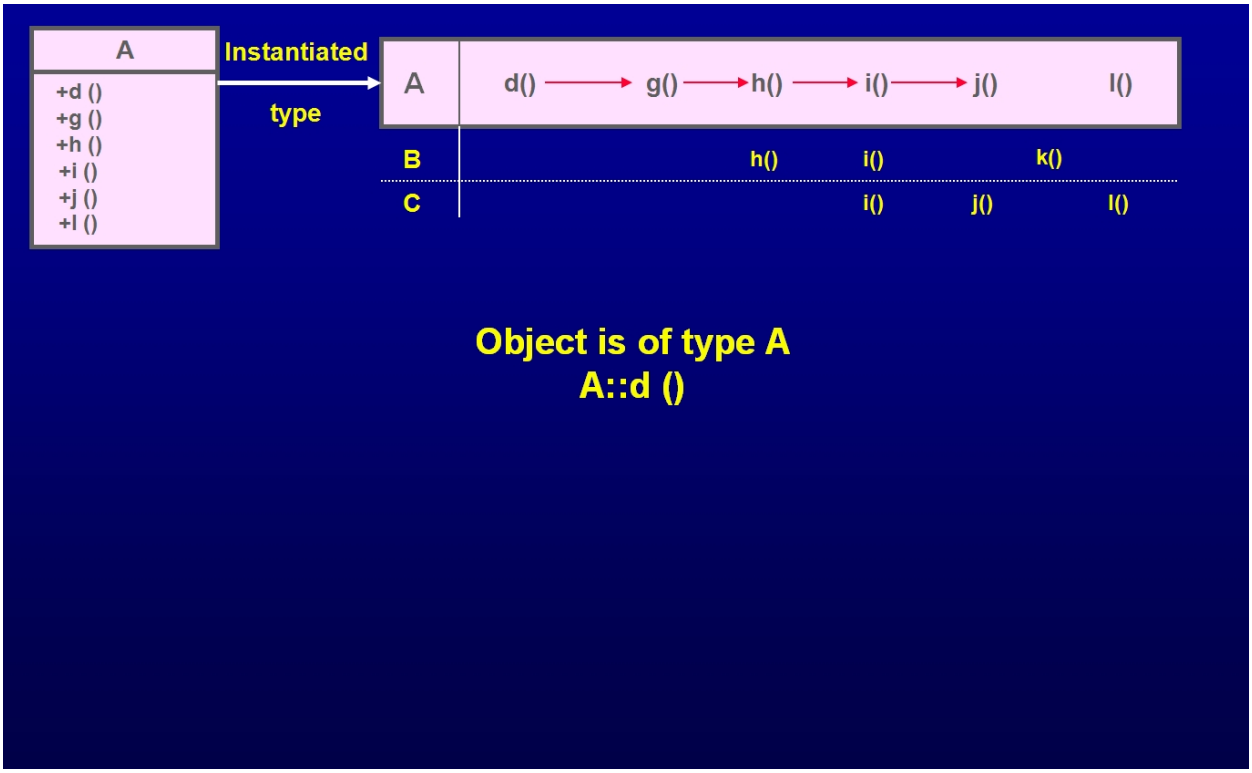


DU páry a jejich možné anomálie 5 [AO08, AO10]

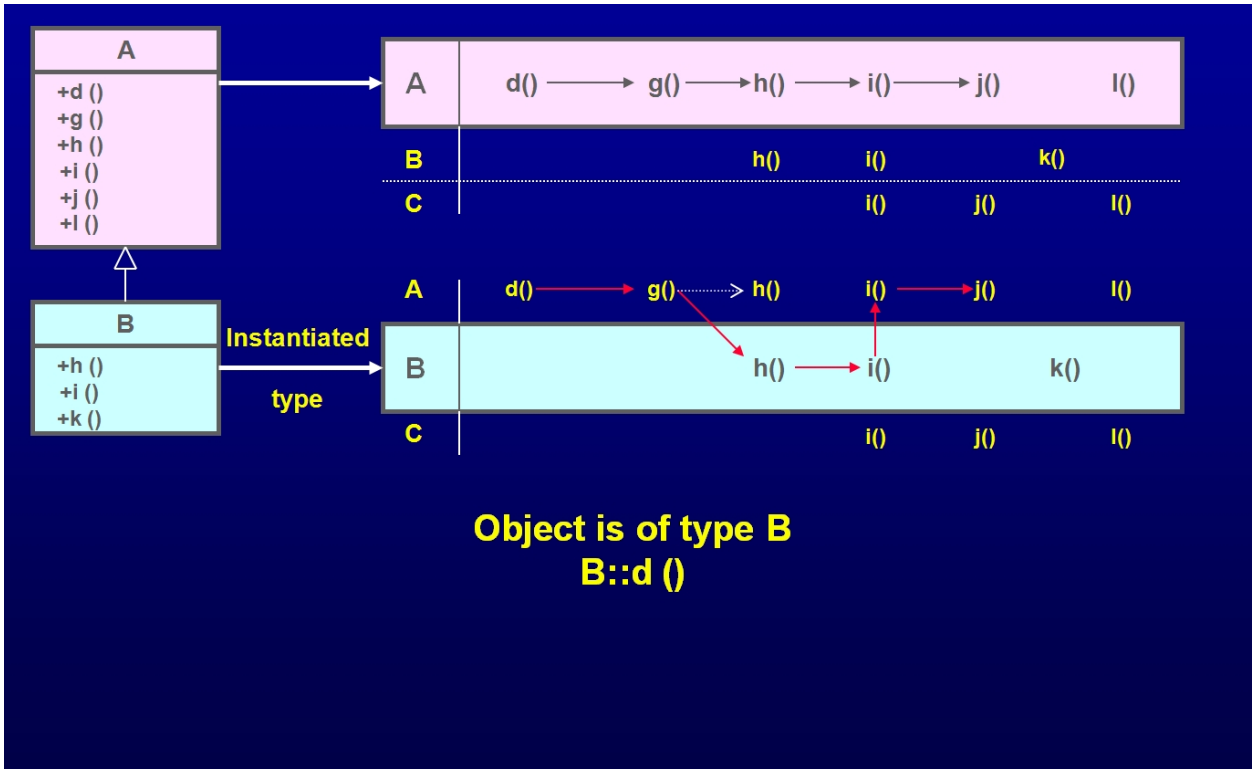
Přepisující metoda má jinou def-množinu než přepisovaná metoda?



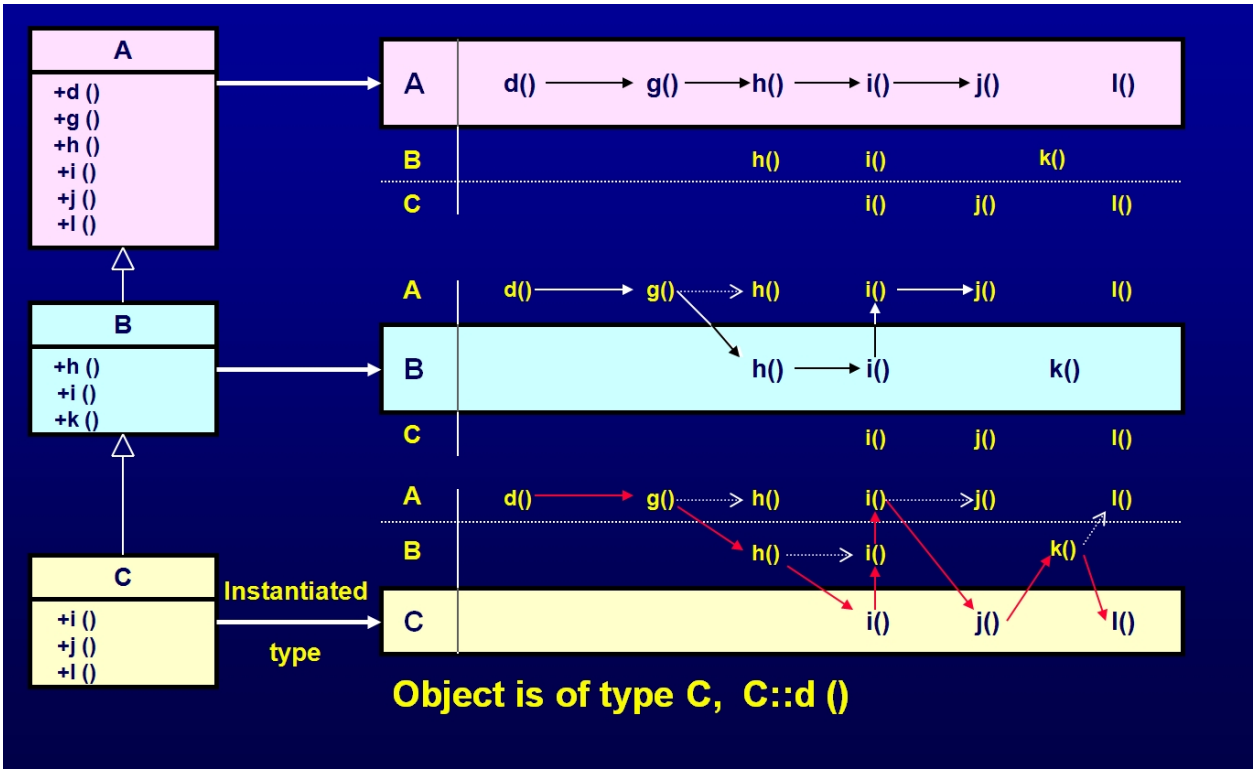
Anomálie polymorfizmu 1 [AO08, AO10]



Anomálie polymorfizmu 2 [AO08, AO10]



Anomálie polymorfizmu 3 [AO08, AO10]



Možné vady OO programů [AO08, AO10]

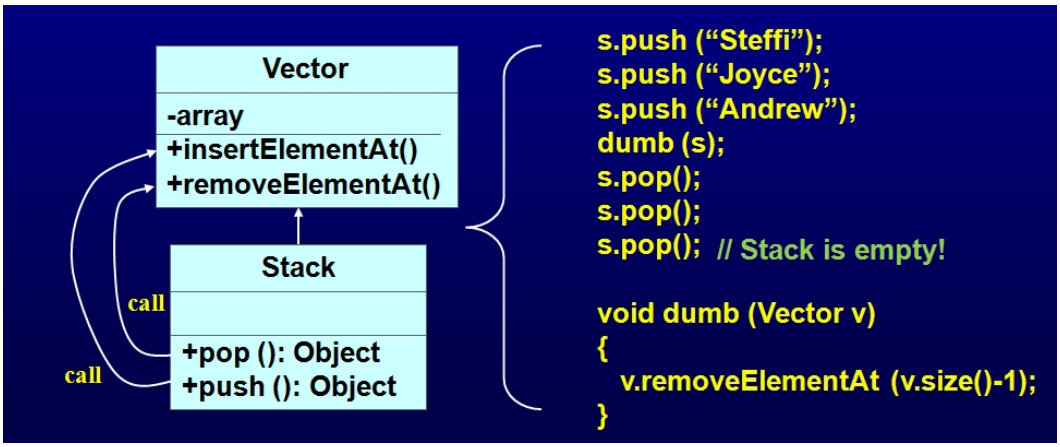
- Složitost vniká množstvím **propojení** komponent.
- **Statický determinismus** mizí - mnoho vad je možné detekovat pouze za běhu.
- Dědičnost a polymorfizmus umožňuje **vertiální** a **dynamickou** integraci.
- **Agregační** vazby a vazby **užití** jsou mnohem složitější.
- Návrháři ne vždy zacházejí opatrně s **viditelností** dat a metod.

OO vady a anomálie [AO08, AO10]

Zkratka	Vada/Anomálie
ITU*	Nekonzistetní užití typu
SDA*	Anomálie definice stavu
SDIH*	Nekonzistentní definice stavu
SDI	Nesprávná definice stavu
IISD	Nepřímá nekonzistentní definice stavu
ACB1*	Anomální chování konstrukce (1)
ACB2	Anomální chování konstrukce (2)
IC	Neúplná konstrukce
SVA*	Anomálie viditelnosti stavu

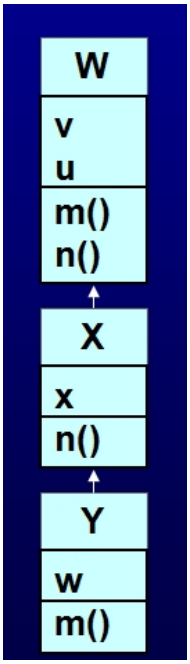
Nekonzistetní užití typu (ITU) [AO08, AO10]

- **nepřepisuje se** (nejedná se o polymorfizmus).
- C rozšiřuje T a C přidává nové metody (**rozšíření**
- objekt je použit "jako C", potom jako T, potom jako C
- metody T mohou přivést objekt do stavu **nekonzistentním** s C



Anomálie definice stavu (SDA) [AO08, AO10]

- X rozšíří W, a X **přepisuje** některé z metod.
- Přepisující metody v X **nenadefinují** některé proměnné, které jsou definovány přepsanými metodami v W

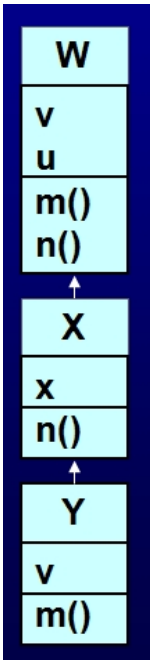


Selhání pro objekt Y při zavolání $m()$ následovaném $n()$

- $W :: m()$ definuje v a $W :: n()$ používá v
- $X :: n()$ používá v
- $Y :: m()$ nedefinuje v

Nekonzistence definice stavu (SDIH) [AO08, AO10]

- **Překrytí** proměnné, možná náhodně.
- Jestliže proměnná v potomku je definována, verze **v předchůdci** nemusí být.

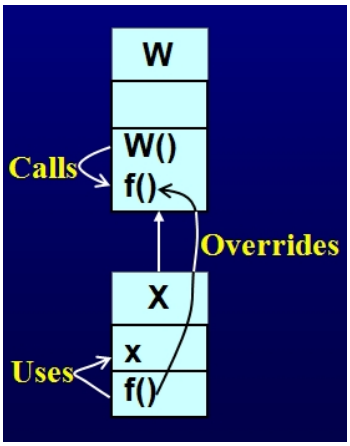


Selhání pro objekt Y při zavolání $m()$ následovaném $n()$

- Y přepíše W) verzi v
- $Y :: m()$ definuje $Y :: v$
- $X :: n()$ používá v

Anomálie v chování konstruktoru (ACB1) [AO08, AO10]

- **Konstruktor** W volá metodu $f()$.
- **Potomek** W, X, přepíše $f()$.
- $X :: f()$ používá **proměnné**, které by měly být definovány v konstruktoru X

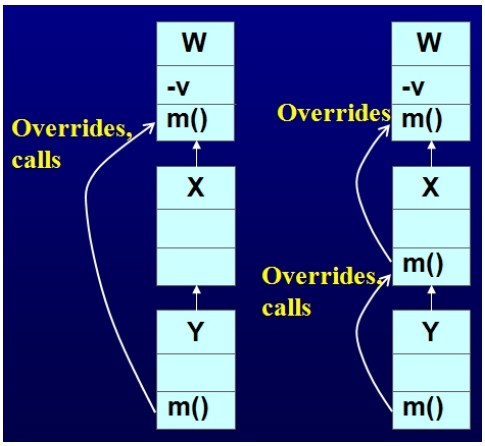


Selhání při konstrukci

- Když je konstruován objekt typu X, $W()$ se provádí před $X :: ()$
- Když $W()$ volá $X :: f()$, použije se x, ale té zatím nebyla dána hodnota!

Anomálie ve viditelnosti stavu (SVA) [AO08, AO10]

- Privátní proměnná v je **deklarována** v předchůdci W a v je definována $W :: m()$.
- X **rozšiřuje** W a Y rozšiřuje X.
- Y přepisuje $m()$ a **volá** $W :: m()$ k nadefinování v



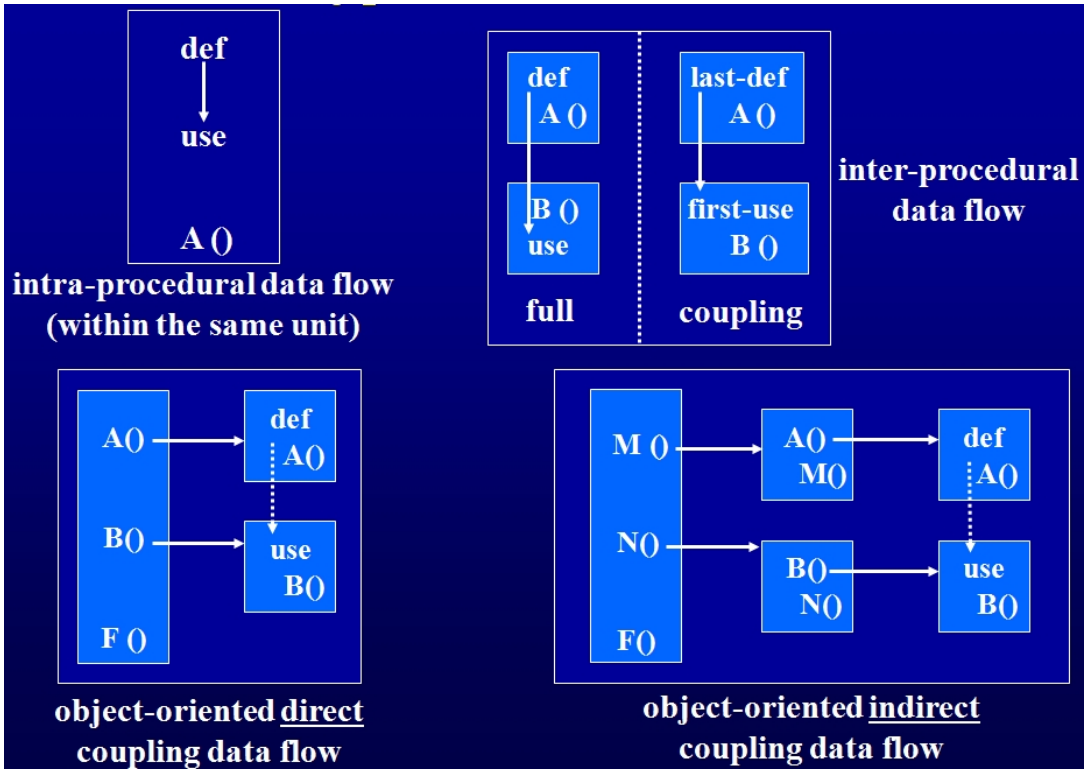
Selhání

- $X :: m()$ je přidána později.
- $Y :: m()$ nemůže dále již volat $W :: m()$!

Vazební sekvence [AO08, AO10]

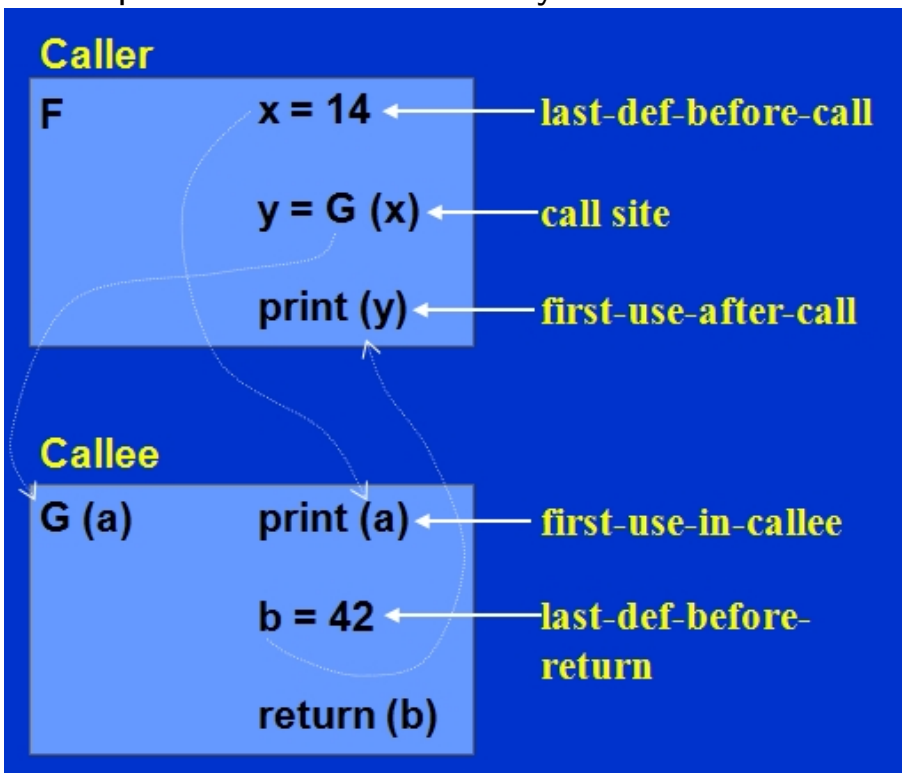
- **Páry** volání metod v rámci jedné testované metody:
  - společný **kontext instance**.
  - množina **stavových proměnných**, které jsou referencovány oběma metodami.
  - obsahuje alespoň jednu **vazební cestu** mezi volání obou metod vzhledem k dané stavové proměnné.
- Reprezentují možné **interakce stavových prostorů** mezi volanými metodami vzhledem k volající metodě.
- Používá se k identifikaci **bodů integrace** a testovacích požadavků.

Typy Def-Use párů [AO08, AO10]



Testování založené na vazbách [AO08, AO10]

Integrace nastává pomocí vazeb softwarových artefaktů.



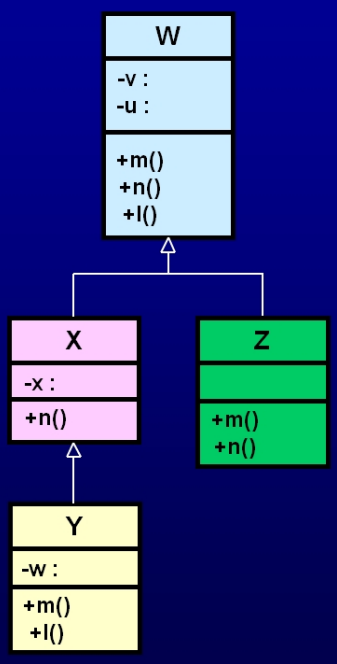
Množina polymorfních volání [AO08, AO10]

PCS

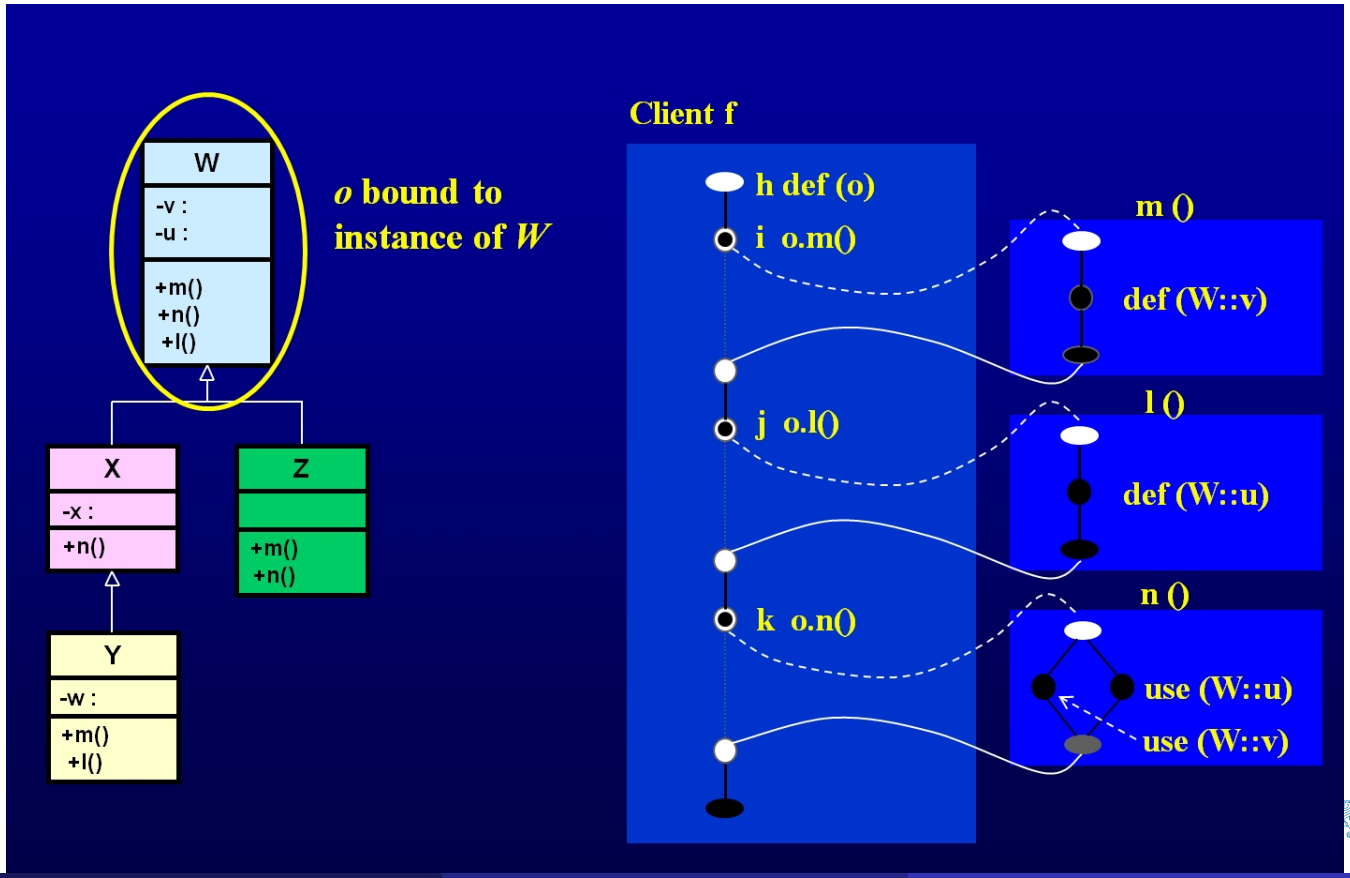
- množina metod, které mohou být **potenciálně** provedeny jako výsledek volání metody v rámci dané instance kontextu.
- $pcs(o :: m) = \{W :: m, Y :: m, X :: m\}$

```
public void f ( W o )
{
      ...
j     o.m();
      ...
l     o.l();
      ...
k     o.n();
 }
```

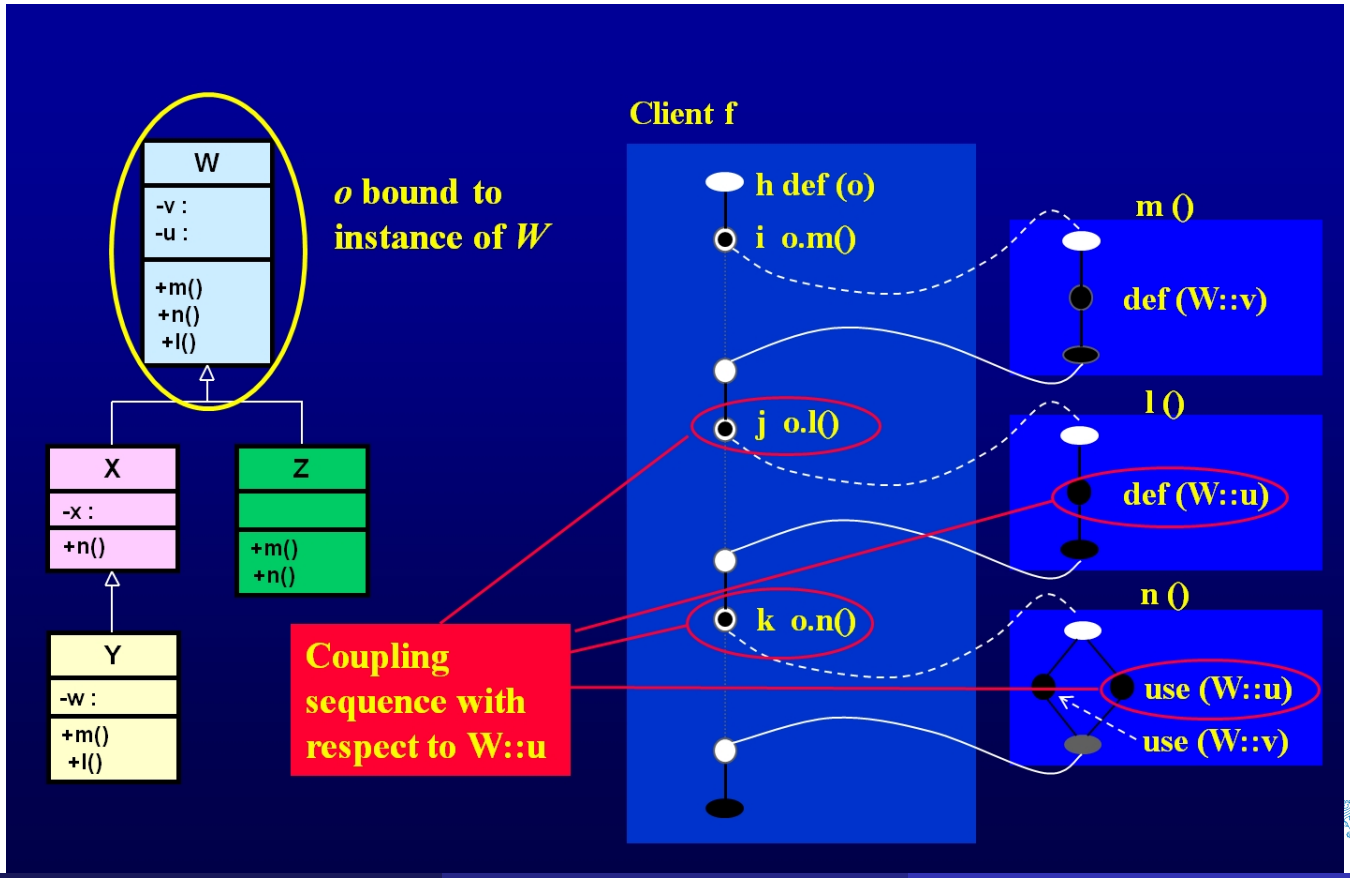

Příklad vazební sekvence 1 [AO08, AO10]



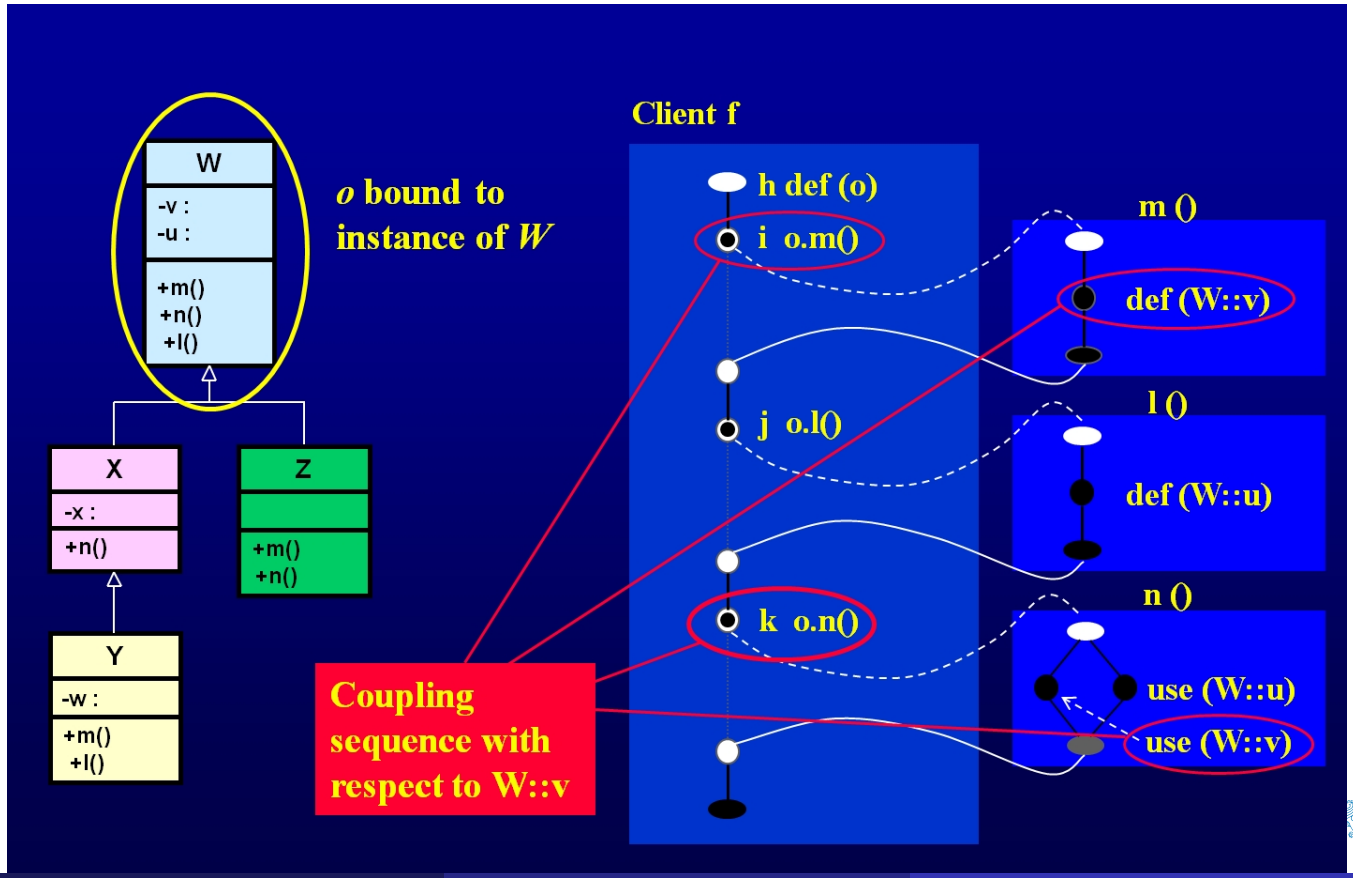
Příklad vazební sekvence 1 [AO08, AO10]



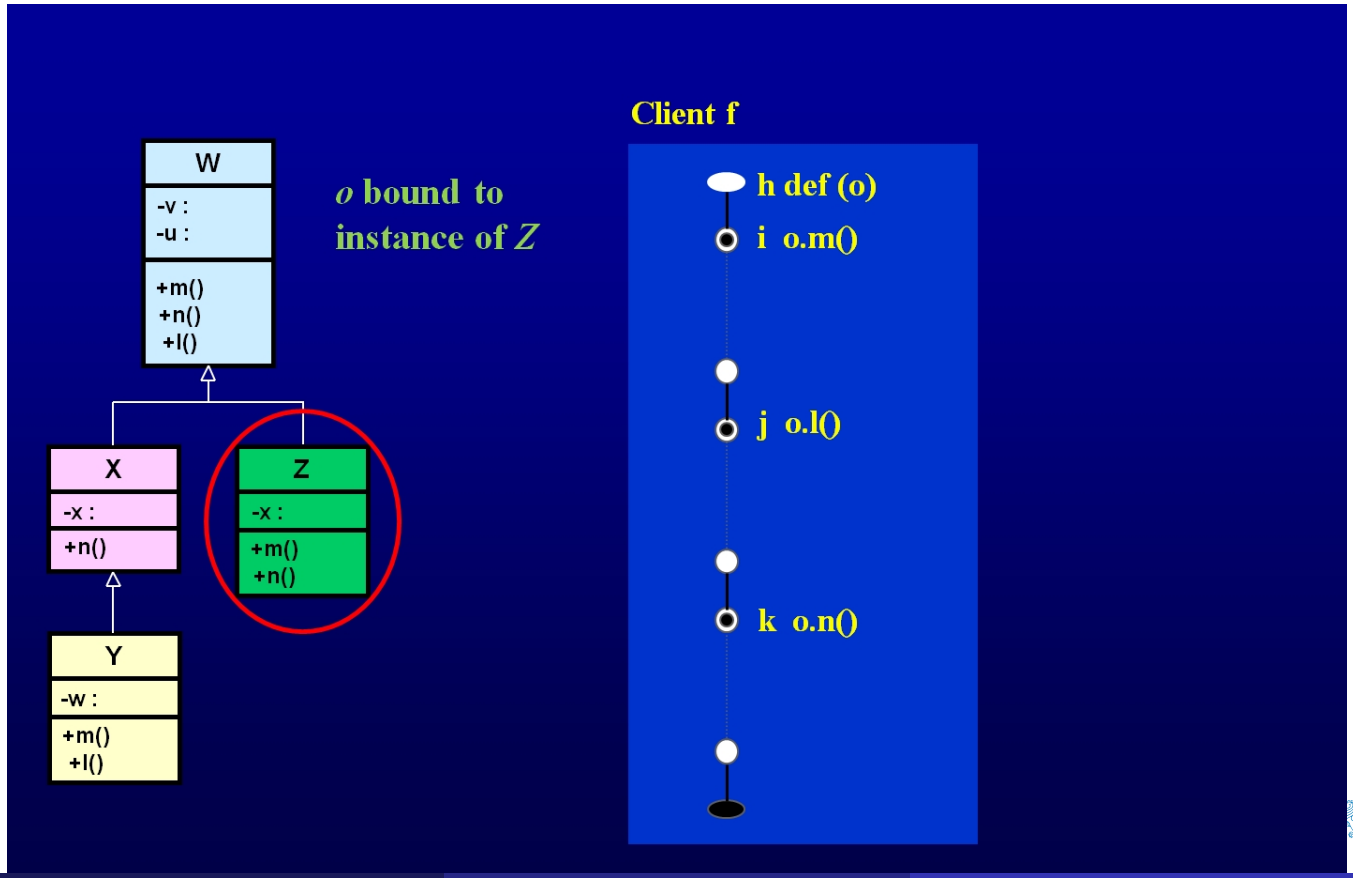
Příklad vazební sekvence 1 [AO08, AO10]



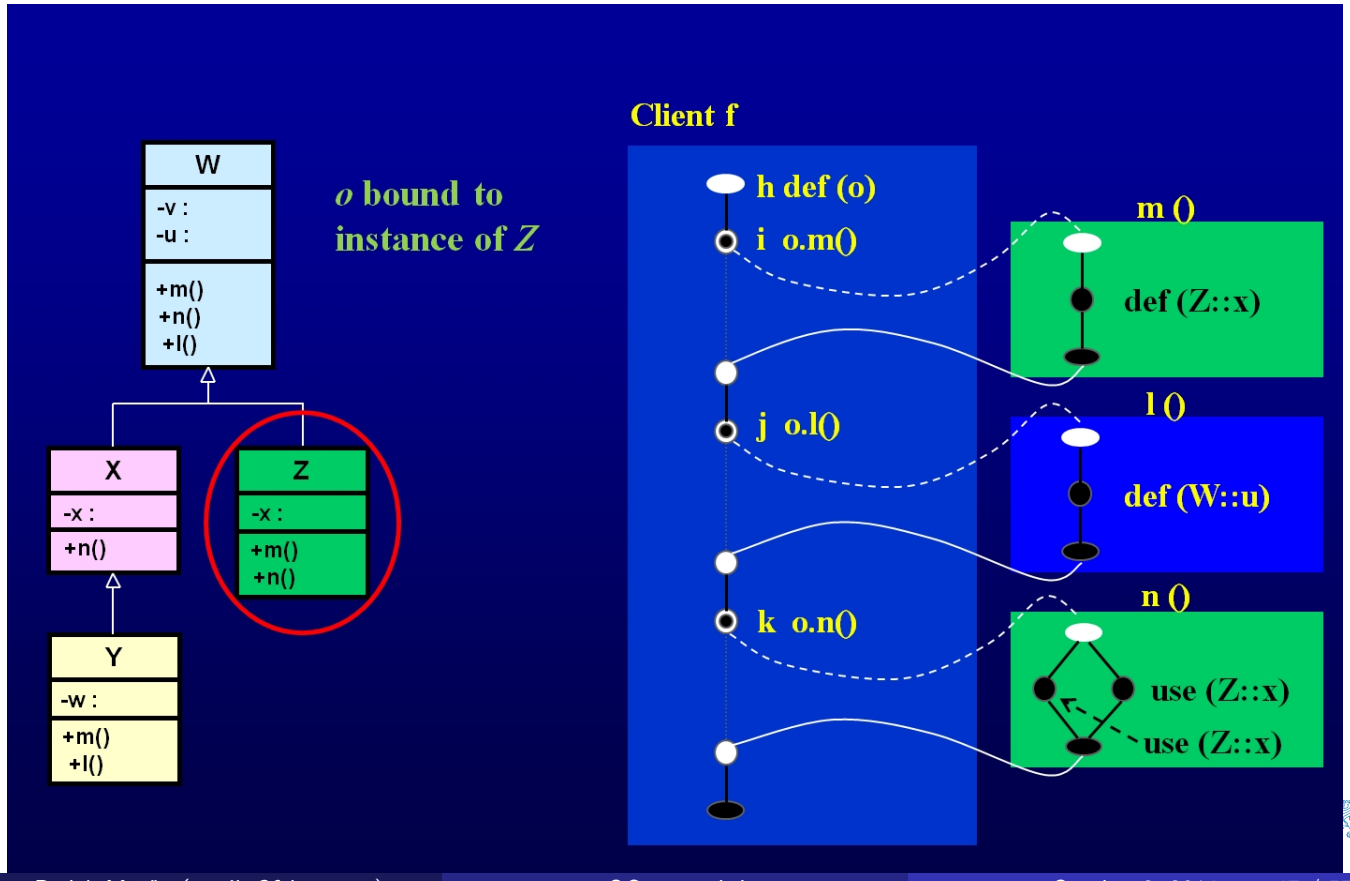
Příklad vazební sekvence 1 [AO08, AO10]



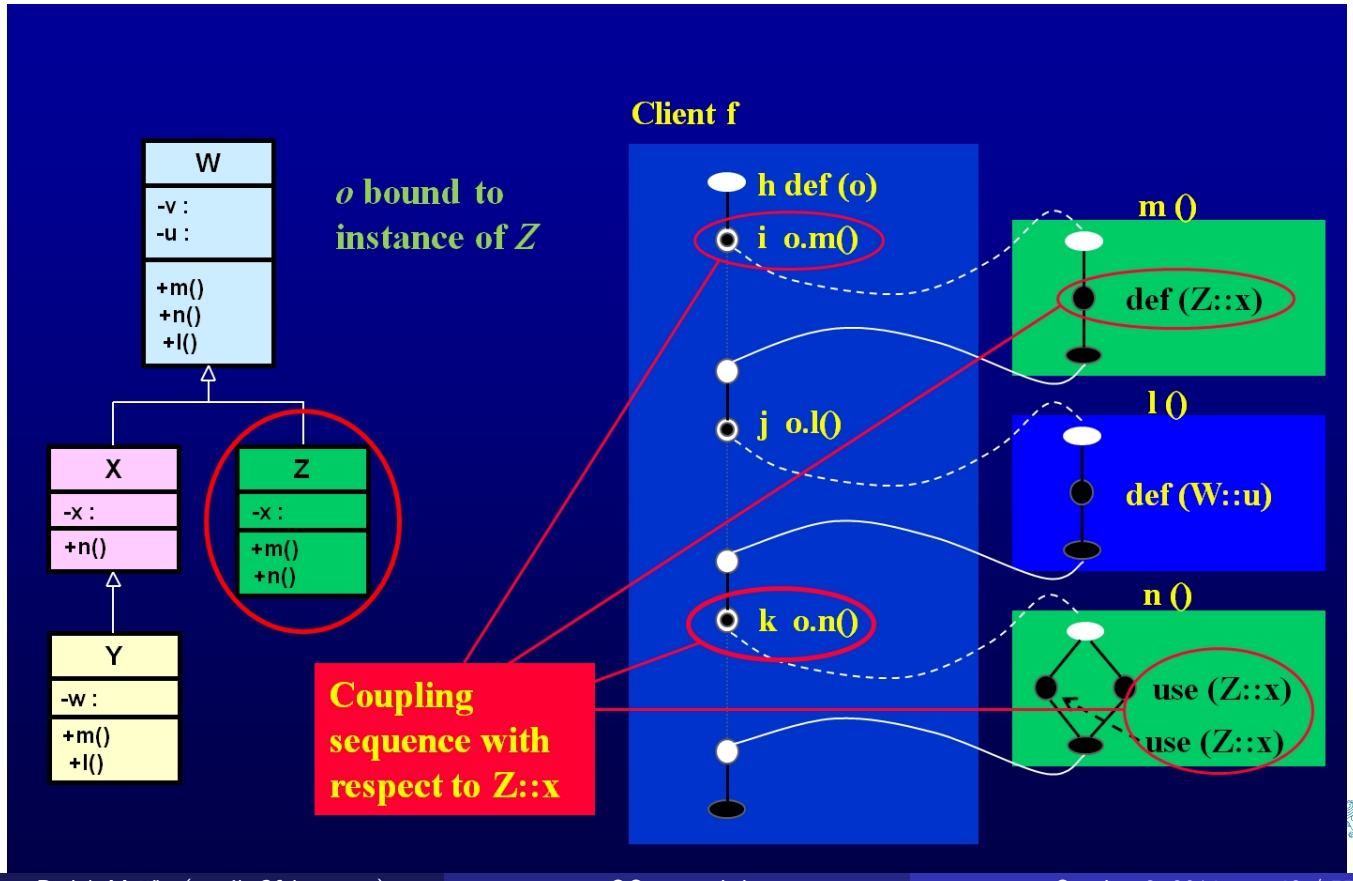
Příklad vazební sekvence 2 [AO08, AO10]



Příklad vazební sekvence 2 [AO08, AO10]



Příklad vazební sekvence 2 [AO08, AO10]



Cíle testování [AO08, AO10]

- Testujeme, jak na sebe **vzájemně působí** metoda a instance vázaná na objekt o:
  - interakce skrze **vazební sekvence**.
- Potřeba uvážit **množinu interakcí**:
  - Možné typy objektu o
  - Které metody mohou být ve skutečnosti provedeny (množiny polymorfních volání).
- Testují se **všechny vazby** se **všemi typy**.

Sekvence všech vazeb [AO08, AO10]

All-Coupling-Sequences (ACS)

- Pro všechny vazební sekvence $S_{j,k}$ v $f()$,
- existuje alespoň jeden testovací případ t takový, že existuje vazební cesta indukovaná $S_{j,k}$ taková, že je podcestou běhové trasy $f(t)$

- Alespoň jedna vazební cesta musí být provedena.
- Neuvažuje ani dědičnost ani polymorfismus.

Všechny poly-třídy [AO08, AO10]

All-Poly-Classes (APC)

- Pro všechny vazební sekvence $S_{j,k}$ v $f()$ a
- pro každou třídu rodiny typů definovanou kontextem $S_{j,k}$,
- existuje alespoň jeden testovací případ t takový, že existuje vazební cesta p indukovaná $S_{j,k}$ taková, že je podcestou běhové trasy $f(t)$

- Zahrnuje kontexty instací při volání.
- Nejméně jeden test for každý typ objektu.

Všechny vazební Def-Use [AO08, AO10]

All-Coupling-Defs-Uses (ACDU)

- Pro každou vazební proměnnou v
- každé vazební sekvence $S_{j,k}$ v $f()$,
- existuje alespoň jeden testovací případ t takový, že existuje vazební cesta p indukovaná $S_{j,k}$ taková, že je podcestou běhové trasy $f(t)$

- Každá poslední definice vazební proměnné dosahuje na všechny svá první použití.
- Neuvažuje dědičnost a polymorfismus.

Všechny polymorfní vazební Def-Use [AO08, AO10]

All-Poly-Coupling-Defs-Uses (APDU)

- Pro každou vazební proměnnou v,
- pro každou třídu rodiny typů definovanou kontextem $S_{j,k}$,
- každé vazební sekvence $S_{j,k}$ v $f()$,
- pro každý uzel m, který má poslední definici v a
- každý uzel n, který má první použití v,
- existuje alespoň jeden testovací případ t takový, že existuje vazební cesta p indukovaná $S_{j,k}$ taková, že je podcestou běhové trasy $f(t)$

- Každá poslední definice vazební proměnné dosahuje na všechny svá první použití pro každý možný typ.
- Pracuje s dědičnosti a polymorfismem.
- Uvažuje definice a použití proměnných.
- Kombinuje předchozí kritéria.

Literatura I

V této prezentaci je použita řada obrázků z níže uvedených původních anglických přednášek (©Ammann and Offutt).

Paul Ammann and Jeff Offutt.
Introduction to Software Testing.
Cambridge University Press, Cambridge, UK, first edition, February 2008.
ISBN 0-52188-038-1.

Paul Ammann and Jeff Offutt.
Introduction to software testing, powerpoint slides, August 2010.