Lund University
Computer Science Department

On Variance-Based Subtyping for Parametric Types

TAPL

Idriss Riouak

March 8, 2022

Overview

- ECOOP 2002 - Atsushi **Igarashi** and Mirko **Viroli**
- Igarashi is one of the authors of **Featherweight Java**
- **Motivations**
  - Java's designers initially decided to avoid generic features,

Overview

- ECOOP 2002 - Atsushi **Igarashi** and Mirko **Viroli**
- Igarashi is one of the authors of **Featherweight Java**
- **Motivations**
  - Java's designers initially decided to avoid generic features,
  - Only inclusive polymorphism supported by inheritance,

Overview

- ECOOP 2002 - Atsushi **Igarashi** and Mirko **Viroli**
- Igarashi is one of the authors of **Featherweight Java**
- **Motivations**
  - Java's designers initially decided to avoid generic features,
  - Only inclusive polymorphism supported by inheritance,
  - Java5: ↑ programmers' productivity; ↑ readability; ↑maintainability and ↑safety.

Overview

- ECOOP 2002 - Atsushi **Igarashi** and Mirko **Viroli**
- Igarashi is one of the authors of **Featherweight Java**
- **Motivations**
  - Java's designers initially decided to avoid generic features,
  - Only inclusive polymorphism supported by inheritance,
  - Java5: ↑ programmers' productivity; ↑ readability; ↑maintainability and ↑safety.
  - Not only *inheritance* subtyping but also **pointwise** subtyping `Stack<X> <: Vector<X>` $\Longrightarrow$ `Stack<String> <: Vector<String>`

Variance

- Necessity to introduce another subtyping scheme: **variance**.

Variance

- Necessity to introduce another subtyping scheme: **variance**.
- Defines the subtype relation between **different** instantiations of the **same** generic class.

Variance

- Necessity to introduce another subtyping scheme: **variance**.
- Defines the subtype relation between **different** instantiations of the **same** generic class.
- A generic class `C<X>` is said to be:

Variance

- Necessity to introduce another subtyping scheme: **variance**.
- Defines the subtype relation between **different** instantiations of the **same** generic class.
- A generic class `C<X>` is said to be:
  - **covariant** w.r.t. `X` if $\forall S, T : S <: T \implies$ `C<S><:C<T>`

Variance

- Necessity to introduce another subtyping scheme: **variance**.
- Defines the subtype relation between **different** instantiations of the **same** generic class.
- A generic class `C<X>` is said to be:
  - **covariant** w.r.t. `X` if $\forall S, T : S <: T \implies$ `C<S><:C<T>`
  - **contravariant** w.r.t. `X` if $\forall S, T : S <: T \implies$ `C<T><:C<S>`

Variance

- Necessity to introduce another subtyping scheme: **variance**.
- Defines the subtype relation between **different** instantiations of the **same** generic class.
- A generic class `C<X>` is said to be:
  - **covariant** w.r.t. `X` if $\forall S, T : S <: T \implies$ `C<S><:C<T>`
  - **contravariant** w.r.t. `X` if $\forall S, T : S <: T \implies$ `C<T><:C<S>`
  - **invariant** w.r.t. `X` if $\forall S, T :$ `C<S><:C<T>` $\implies S = T$

Variance

- Necessity to introduce another subtyping scheme: **variance**.
- Defines the subtype relation between **different** instantiations of the **same** generic class.
- A generic class `C<X>` is said to be:
  - **covariant** w.r.t. `X` if $\forall S, T : S <: T \implies$ `C<S><:C<T>`
  - **contravariant** w.r.t. `X` if $\forall S, T : S <: T \implies$ `C<T><:C<S>`
  - **invariant** w.r.t. `X` if $\forall S, T :$ `C<S><:C<T>` $\implies S = T$
- For the type system to be **sound**, *covariance* and *contravariance* are permitted under some constraint on the occurrences of `X` within `C<X>`' signature:

Variance

- Necessity to introduce another subtyping scheme: **variance**.
- Defines the subtype relation between **different** instantiations of the **same** generic class.
- A generic class `C<X>` is said to be:
  - **covariant** w.r.t. `X` if $\forall S, T : S <: T \implies$ `C<S><:C<T>`
  - **contravariant** w.r.t. `X` if $\forall S, T : S <: T \implies$ `C<T><:C<S>`
  - **invariant** w.r.t. `X` if $\forall S, T :$ `C<S><:C<T>` $\implies S = T$
- For the type system to be **sound**, *covariance* and *contravariance* are permitted under some constraint on the occurrences of `X` within `C<X>`' signature:
  - **covariant** if it is read-only

Variance

- Necessity to introduce another subtyping scheme: **variance**.
- Defines the subtype relation between **different** instantiations of the **same** generic class.
- A generic class `C<X>` is said to be:
  - **covariant** w.r.t. `X` if $\forall S, T : S <: T \implies$ `C<S><:C<T>`
  - **contravariant** w.r.t. `X` if $\forall S, T : S <: T \implies$ `C<T><:C<S>`
  - **invariant** w.r.t. `X` if $\forall S, T :$ `C<S><:C<T>` $\implies S = T$
- For the type system to be **sound**, *covariance* and *contravariance* are permitted under some constraint on the occurrences of `X` within `C<X>`' signature:
  - **covariant** if it is read-only
  - **contravariant** if it is write-only

Variance example


```
class Pair<X extends Object, Y extends Object> extends Object{
  private X fst;
  private Y snd;
  Pair(X fst,Y snd){ this.fst=fst; this.snd=snd; }
  void setFst(X fst){ this.fst=fst; }
  Y getSnd(){ return snd; }
}
```

This class can be safely considered

- *covariant* in type variable `Y`, and — *read-only*
- *contravariant* in type variable `X` — *write-only*

LTH
FACULTY OF ENGINEERING

Variance example cont'd

Any type `Pair<R,S>` can be safely considered a subtype of `Pair<String,Number>` when `R :> String` and `S <: Number`.

```
Number getAndSet(Pair<String,Number> c, String s){
 c.setFst(s);
 return c.getSnd();
}

Number n=getAndSet( new Pair<Object,Integer>(null, new Integer(1)),"1");
```

Problem

The type variables typically occur in such positions that forbid both covariance and contravariance

```
class Vector<X>{
  private X[] ar;
  Vector(int size){ ar=new X[size];}
  int size(){ return ar.length; }
  X getElementAt(int i){ return ar[i];}
  void setElementAt(X t,int i){ ar[i]=t;}
}
```

Variance cont'd

The main contribution of the paper:

- Specify variance of each type parameter when the type is **used**
- Not when the type is declared
- This should:
  - release the class designer from the burden of taking variance into account
  - improve reusability.

Variant Parametric Types

Let the programmer specify within parametric types whether the type argument should be *contravariant*, *covariant* or *invariant*

- Each type parameter may be associated with a **variance annotation**
  - '+': *covariance* `Vector<+String>`
  - '-': *contravariant* `Vector<-String>`
  - '*': *bivariance* `Vector<*String>` *or* `Vector<*>`
- If the outermost parametric type is without annotation then is called **invariant**.
  - `Vector<String>`
  - `Pair<Vector<+String>,Integer>`

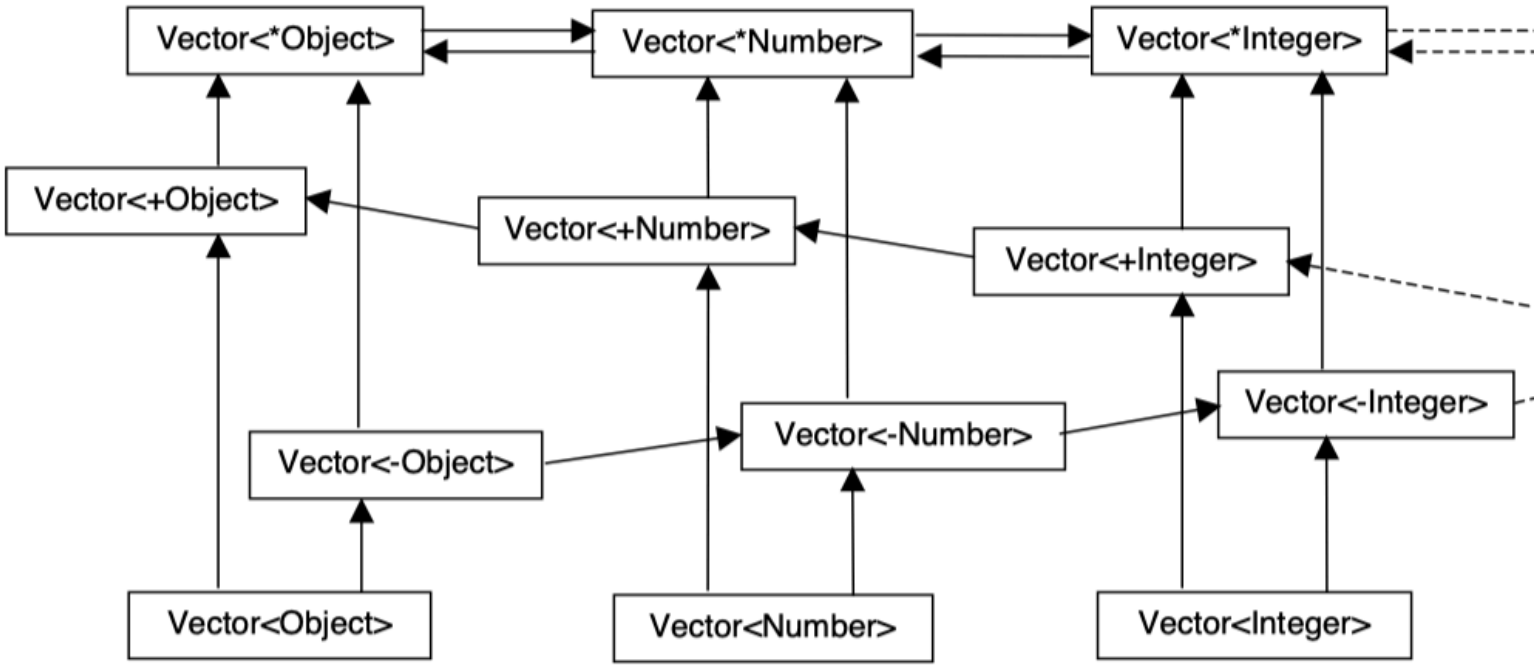
Simple interpretation

An interpretation of variant parametric types is given as a set of variant types.

- `C<+T>` $= \{$`C<S>` $\mid S <: T\}$
- `C<-T>` $= \{$`C<S>` $\mid S :> T\}$
- `C<*T>` $= \{$`C<S>` $\mid \forall\ S\}$ `C<*T>` $=$ `C<*>`
- An invariant type correspond to a singleton
  - `Vector<Integer>` $= \{$`Vector<Integer>`$\}$

Subtyping

$$\texttt{Integer} <: \texttt{Number} <: \texttt{Object}$$



Syntax

$\mathtt{N} ::= \mathtt{C<}\overline{\mathtt{vT}}\mathtt{>}$	variant parametric types
$\mathtt{T} ::= \mathtt{X} \mid \mathtt{N}$	types
$\mathtt{v} ::= \mathtt{o} \mid \mathtt{+} \mid \mathtt{-} \mid \mathtt{*}$	variance annotations
$\mathtt{L} ::= \mathtt{class\ C<}\overline{\mathtt{X}}\mathtt{\triangleleft}\overline{\mathtt{N}}\mathtt{>\triangleleft D<}\overline{\mathtt{S}}\mathtt{>\ \{\ }\overline{\mathtt{T}}\ \overline{\mathtt{f}}\mathtt{;}\ \overline{\mathtt{M}}\mathtt{\ \}}$	class definitions
$\mathtt{M} ::= \mathtt{T\ m(}\overline{\mathtt{T}}\ \overline{\mathtt{x}}\mathtt{)\{\ return\ e;\ \}}$	method definitions
$\mathtt{e} ::= \mathtt{x}$	variables
$\mid\ \mathtt{e.f}$	field access
$\mid\ \mathtt{e.m(}\overline{\mathtt{e}}\mathtt{)}$	method invocation
$\mid\ \mathtt{new\ C<}\overline{\mathtt{T}}\mathtt{>(}\overline{\mathtt{e}}\mathtt{)}$	object instantiation
$\mid\ \mathtt{(T)e}$	typecasts

LTH
FACULTY OF ENGINEERING

Auxiliary functions

Field lookup:

$$fields(\texttt{Object}) = \bullet$$

$$\frac{\texttt{class C<}\overline{\texttt{X}}\triangleleft\overline{\texttt{N}}\texttt{>}\triangleleft\texttt{D<}\overline{\texttt{U}}\texttt{>}\ \texttt{\{}\overline{\texttt{S}}\ \overline{\texttt{f}}\texttt{;}\ \overline{\texttt{M}}\texttt{\}} \qquad fields([\overline{\texttt{T}}/\overline{\texttt{X}}]\texttt{D<}\overline{\texttt{U}}\texttt{>}) = \overline{\texttt{V}}\ \overline{\texttt{g}}}{fields(\texttt{C<}\overline{\texttt{T}}\texttt{>}) = \overline{\texttt{V}}\ \overline{\texttt{g}},\ [\overline{\texttt{T}}/\overline{\texttt{X}}]\overline{\texttt{S}}\ \overline{\texttt{f}}}$$

Method type lookup:

$$\frac{\texttt{class C<}\overline{\texttt{X}}\triangleleft\overline{\texttt{N}}\texttt{>}\triangleleft\texttt{D<}\overline{\texttt{S}}\texttt{>}\ \texttt{\{...}\ \overline{\texttt{M}}\texttt{\}} \qquad \texttt{U}_0\ \texttt{m(}\overline{\texttt{U}}\ \overline{\texttt{x}}\texttt{)\{ return e; \}} \in \overline{\texttt{M}}}{mtype(\texttt{m}, \texttt{C<}\overline{\texttt{T}}\texttt{>}) = [\overline{\texttt{T}}/\overline{\texttt{X}}](\overline{\texttt{U}}\texttt{->}\texttt{U}_0)}$$

$$\frac{\texttt{class C<}\overline{\texttt{X}}\triangleleft\overline{\texttt{N}}\texttt{>}\triangleleft\texttt{D<}\overline{\texttt{S}}\texttt{>}\ \texttt{\{...}\ \overline{\texttt{M}}\texttt{\}} \qquad \texttt{m} \notin \overline{\texttt{M}}}{mtype(\texttt{m}, \texttt{C<}\overline{\texttt{T}}\texttt{>}) = mtype(\texttt{m}, [\overline{\texttt{T}}/\overline{\texttt{X}}]\texttt{D<}\overline{\texttt{S}}\texttt{>})}$$

Open:

$$\frac{\begin{array}{l}\text{if } (\texttt{v}_i, \texttt{T}_i) \neq (\texttt{w}_i, \texttt{U}_i), \text{then} \\ \quad \texttt{w}_i = \texttt{o} \qquad \texttt{U}_i = \texttt{X}_i \notin dom(\Delta) \cup \{\overline{\texttt{U}}\} \setminus \{\texttt{U}_i\} \qquad \Delta'(\texttt{X}_i) = (\texttt{v}_i, \texttt{T}_i)\end{array}}{\Delta \vdash \texttt{C<}\overline{\texttt{vT}}\texttt{>} \Uparrow^{\Delta'} \texttt{C<}\overline{\texttt{wU}}\texttt{>}}$$

Close:

$$\frac{\Delta(\texttt{X}) = (+, \texttt{T})}{\texttt{X} \Downarrow_\Delta \texttt{T}} \qquad \frac{\texttt{X} \notin dom(\Delta)}{\texttt{X} \Downarrow_\Delta \texttt{X}} \qquad \frac{\overline{\texttt{T}} \Downarrow_\Delta \overline{\texttt{T}}' \qquad \texttt{w}_i = \begin{cases} \texttt{v}_i & \text{if } \texttt{T}_i = \texttt{T}'_i \\ +\vee\texttt{v}_i & \text{otherwise} \end{cases}}{\texttt{C<}\overline{\texttt{vT}}\texttt{>} \Downarrow_\Delta \texttt{C<}\overline{\texttt{wT}}'\texttt{>}}$$

LTH
FACULTY OF ENGINEERING

Judgments

Subtyping:

$$\Delta \vdash \mathtt{T} <: \mathtt{T} \qquad \frac{\Delta \vdash \mathtt{S} <: \mathtt{T} \qquad \Delta \vdash \mathtt{T} <: \mathtt{U}}{\Delta \vdash \mathtt{S} <: \mathtt{U}} \qquad \frac{\Delta(\mathtt{X}) = (+, \mathtt{T})}{\Delta \vdash \mathtt{X} <: \mathtt{T}} \qquad \frac{\Delta(\mathtt{X}) = (-, \mathtt{T})}{\Delta \vdash \mathtt{T} <: \mathtt{X}}$$

$$\frac{\overline{\mathtt{v}} \leq \overline{\mathtt{w}} \qquad \text{if } \mathtt{w}_i \leq -, \text{ then } \Delta \vdash \mathtt{T}_i <: \mathtt{S}_i \qquad \text{if } \mathtt{w}_i \leq +, \text{ then } \Delta \vdash \mathtt{S}_i <: \mathtt{T}_i}{\Delta \vdash \mathtt{C}\langle\overline{\mathtt{vS}}\rangle <: \mathtt{C}\langle\overline{\mathtt{wT}}\rangle}$$

Type Well-formedness:

$$\Delta \vdash \mathtt{Object}\ \text{ok} \qquad \frac{\mathtt{X} \in \mathit{dom}(\Delta)}{\Delta \vdash \mathtt{X}\ \text{ok}} \qquad \frac{\begin{array}{c}\mathtt{class}\ \mathtt{C}\langle\overline{\mathtt{X}} \triangleleft \overline{\mathtt{N}}\rangle \triangleleft \mathtt{D}\langle\overline{\mathtt{S}}\rangle\ \{\ldots\} \\ \Delta \vdash \overline{\mathtt{T}}\ \text{ok} \qquad \Delta \vdash \overline{\mathtt{T}} <: [\overline{\mathtt{T}}/\overline{\mathtt{X}}]\overline{\mathtt{N}}\end{array}}{\Delta \vdash \mathtt{C}\langle\overline{\mathtt{vT}}\rangle\ \text{ok}}$$

LTH
FACULTY OF ENGINEERING

Typing

Expression Typing:

$$\Delta; \Gamma \vdash \mathtt{x} \in \Gamma(\mathtt{x})$$

$$\frac{\Delta; \Gamma \vdash \mathtt{e}_0 \in \mathtt{T}_0 \qquad \Delta \vdash \mathit{bound}_\Delta(\mathtt{T}_0) \Uparrow^{\Delta'} \mathtt{C<\overline{T}>} \qquad \mathit{fields}(\mathtt{C<\overline{T}>}) = \overline{\mathtt{S}}\ \overline{\mathtt{f}} \qquad \mathtt{S}_i \Downarrow_{\Delta'} \mathtt{T}}{\Delta; \Gamma \vdash \mathtt{e}_0.\mathtt{f}_i \in \mathtt{T}}$$

$$\frac{\begin{array}{c}\Delta; \Gamma \vdash \mathtt{e}_0 \in \mathtt{T}_0 \qquad \Delta \vdash \mathit{bound}_\Delta(\mathtt{T}_0) \Uparrow^{\Delta'} \mathtt{C<\overline{T}>} \qquad \mathit{mtype}(\mathtt{m}, \mathtt{C<\overline{T}>}) = \overline{\mathtt{U}}\mathtt{->}\mathtt{U}_0 \\ \Delta; \Gamma \vdash \overline{\mathtt{e}} \in \overline{\mathtt{S}} \qquad \Delta \vdash \overline{\mathtt{S}} \mathtt{<:} \overline{\mathtt{U}} \qquad \mathtt{U}_0 \Downarrow_{\Delta'} \mathtt{T}\end{array}}{\Delta; \Gamma \vdash \mathtt{e}_0.\mathtt{m}(\overline{\mathtt{e}}) \in \mathtt{T}}$$

$$\frac{\Delta \vdash \mathtt{C<\overline{T}>}\ \mathrm{ok} \qquad \mathit{fields}(\mathtt{C<\overline{T}>}) = \overline{\mathtt{U}}\ \overline{\mathtt{f}} \qquad \Delta; \Gamma \vdash \overline{\mathtt{e}} \in \overline{\mathtt{S}} \qquad \Delta \vdash \overline{\mathtt{S}} \mathtt{<:} \overline{\mathtt{U}}}{\Delta; \Gamma \vdash \mathtt{new}\ \mathtt{C<\overline{T}>}(\overline{\mathtt{e}}) \in \mathtt{C<\overline{T}>}}$$

$$\frac{\begin{array}{c}\Delta; \Gamma \vdash \mathtt{e}_0 \in \mathtt{T}_0 \\ \Delta \vdash \mathit{bound}_\Delta(\mathtt{T}_0) \mathtt{<:} \mathit{bound}_\Delta(\mathtt{T}) \quad \text{or} \quad \Delta \vdash \mathit{bound}_\Delta(\mathtt{T}) \mathtt{<:} \mathit{bound}_\Delta(\mathtt{T}_0)\end{array}}{\Delta; \Gamma \vdash (\mathtt{T})\mathtt{e}_0 \in \mathtt{T}}$$

Final check

Class Typing:

$$\frac{\overline{\mathtt{X}}\mathtt{<:}\overline{\mathtt{N}} \vdash \overline{\mathtt{N}}, \mathtt{D<}\overline{\mathtt{S}}\mathtt{>}, \overline{\mathtt{T}}\ \mathrm{ok} \qquad \overline{\mathtt{M}}\ \mathtt{OK\ IN}\ \mathtt{C<}\overline{\mathtt{X}} \triangleleft \overline{\mathtt{N}}\mathtt{>}}{\mathtt{class\ C<}\overline{\mathtt{X}} \triangleleft \overline{\mathtt{N}}\mathtt{>} \triangleleft \mathtt{D<}\overline{\mathtt{S}}\mathtt{>\ \{}\overline{\mathtt{T}}\ \overline{\mathtt{f}}\mathtt{;}\ \overline{\mathtt{M}}\mathtt{\}\ OK}}$$

LTH
FACULTY OF ENGINEERING