CSED-321

Featherweight Java

POSTECH

박성우

Featherweight Java

- Core calculus for Java
  - retains all the core features of Java
  - no side effects

- Consists of:
  - syntax
  - typing rules
  - reduction rules

- Every Featherweight Java program is a valid Java program

- You will implement Featherweight Java in Assignment #7.

Program = Collection of class declarations + an expression to evalute

- Every class must specify its supertype (superclass).
  - except the top-most **Object** class
- Constructor:
  - one argument for each field
  - super() should be called in the beginning.
- Method body:
  - **return <expression>;**
- Field access must specify the receiver.
  - Ex. **this.snd**

```
class A extends Object {
  A() { super(); }
}

class B extends Object {
  B() { super(); }
}

class Pair extends Object {
  Object fst;
  Object snd;
  Pair(Object fst, Object snd) {
    super(); this.fst = fst; this.snd = snd;
  }

  Pair setfst(Object newfst) {
    return new Pair(newfst, this.snd);
  }
}
```

Evaluation

new Pair(new A(), new B())
.setfst (new B())

→

new Pair(new B(), new B())

```
class A extends Object {
  A() { super(); }
}

class B extends Object {
  B() { super(); }
}

class Pair extends Object {
  Object fst;
  Object snd;
  Pair(Object fst, Object snd) {
    super(); this.fst = fst; this.snd = snd;
  }

  Pair setfst(Object newfst) {
    return new Pair(newfst, this.snd);
  }
}
```

Five forms of expressions in Featherweight Java

- #1. Object constructor
- #2. Method invocation
  - **new Pair(new A(), new B()) .setfst (new B())**
- #3. Field access
  - **new Pair(newfst, this.snd)**
- #4. Variables
  - **new Pair(newfst, this.snd)**
- #5. Cast

```
class A extends Object {
  A() { super(); }
}

class B extends Object {
  B() { super(); }
}

class Pair extends Object {
  Object fst;
  Object snd;
  Pair(Object fst, Object snd) {
    super(); this.fst = fst; this.snd = snd;
  }

  Pair setfst(Object newfst) {
    return new Pair(newfst, this.snd);
  }
}
```

#5. Cast

- Example

((Pair)new Pair(new Pair(new A(), new B()), new A()).fst).snd

- **new Pair(new Pair(new A(), new B()), new A()).fst** has type Object.
- Hence it should be cast to **Pair**.

- At runtime, we check whether **new Pair(new Pair(new A(), new B()), new A()).fst** can have type **Pair**.

Small-step reduction rules

- Evaluation can be described within the syntax of Featherweight Java.
  - because of no side effects
  - similarly to lambda-calculus
- Non-deterministic evaluation:
  - The order of evaluation does not affect the final result.

- Three kinds of reduction rules
  - #1. Field access
  - #2. Method invocation
  - #3. Cast

#1. Field access - Example

```
new Pair(new A(), new B()).snd → new B()
```

```
class A extends Object {
  A() { super(); }
}

class B extends Object {
  B() { super(); }
}
```

```
class Pair extends Object {
  Object fst;
  Object snd;
  Pair(Object fst, Object snd) {
    super(); this.fst = fst; this.snd = snd;
  }
  Pair setfst(Object newfst) {
    return new Pair(newfst, this.snd);
  }
}
```

#2. Method invocation - Example

```
new Pair(new A(), new B()).setfst(new B())
→ [          new B()/newfst,          ] new Pair(newfst, this.snd)
    new Pair(new A(), new B())/this
≡ new Pair(new B(), new Pair(new A(), new B()).snd)
```

```
class A extends Object {
  A() { super(); }
}

class B extends Object {
  B() { super(); }
}
```

```
class Pair extends Object {
  Object fst;
  Object snd;
  Pair(Object fst, Object snd) {
    super(); this.fst = fst; this.snd = snd;
  }
  Pair setfst(Object newfst) {
    return new Pair(newfst, this.snd);
  }
}
```

#3. Cast - Example (where the evaluation does not get stuck)

```
(Pair)new Pair(new A(), new B()) → new Pair(new A(), new B())
```

```
class A extends Object {
  A() { super(); }
}

class B extends Object {
  B() { super(); }
}
```

```
class Pair extends Object {
  Object fst;
  Object snd;
  Pair(Object fst, Object snd) {
    super(); this.fst = fst; this.snd = snd;
  }
  Pair setfst(Object newfst) {
    return new Pair(newfst, this.snd);
  }
}
```

Three kinds of casts: (C) e

- #1. Upcast
  e is a subclass of **C**
- #2. Downcast
  e is a superclass of **C**
- #3. Stupid cast
  e is unrelated to **C**

- Difference from Java
  - Java rejects expressions containing stupid casts.
  - Featherweight Java does NOT reject such expressions.

$$\texttt{(A)}\underline{\texttt{(Object)new B()}} \rightarrow \texttt{(A)new B()}$$

Example of evaluation

```
   ((Pair)new Pair(new Pair(new A(), new B()), new A()).fst).snd
→ ((Pair)new Pair(new A(), new B())).snd
→ new Pair(new A(), new B()).snd
→ new B()
```

```
class A extends Object {
  A() { super(); }
}

class B extends Object {
  B() { super(); }
}
```

```
class Pair extends Object {
  Object fst;
  Object snd;
  Pair(Object fst, Object snd) {
    super(); this.fst = fst; this.snd = snd;
  }
  Pair setfst(Object newfst) {
    return new Pair(newfst, this.snd);
  }
}
```

Three cases of evaluation getting stuck

- #1. Attempt to access a field not declared for the class
  - → never happens in well-typed programs
- #2. Attempt for invoke a method not declared for the class
  - → never happens in well-typed programs
- #3. Attempt to cast to a wrong class
  - → never happens in well-typed programs with :
    - **no downcasts**
    - **no stupid casts**

Type soundness

- If:
  - an expression **e** reduces to expression **e'**
  - e is well-typed
- Then:
  - **e'** is also well-typed
  - type of **e'** = a subtype of the type of **e**

- Q. What is the difference from type safety in simply-typed lambda-calculus?

Definition of Featherweight Jaava

Syntax

$$
\begin{array}{lcl}
\texttt{CL} & ::= & \texttt{class C extends C } \{\overline{\texttt{C}}\ \overline{\texttt{f}}\texttt{; K } \overline{\texttt{M}}\} \\
\texttt{K} & ::= & \texttt{C}(\overline{\texttt{C}}\ \overline{\texttt{f}})\ \{\texttt{super}(\overline{\texttt{f}})\texttt{; this.}\overline{\texttt{f}}\texttt{ = }\overline{\texttt{f}}\texttt{;}\} \\
\texttt{M} & ::= & \texttt{C m}(\overline{\texttt{C}}\ \overline{\texttt{x}})\ \{\texttt{return e;}\} \\
\texttt{e} & ::= & \texttt{x} \\
 & | & \texttt{e.f} \\
 & | & \texttt{e.m}(\overline{\texttt{e}}) \\
 & | & \texttt{new C}(\overline{\texttt{e}}) \\
 & | & \texttt{(C)e}
\end{array}
$$

Subtyping

$$\texttt{C} <: \texttt{C}$$

$$\frac{\texttt{C} <: \texttt{D} \qquad \texttt{D} <: \texttt{E}}{\texttt{C} <: \texttt{E}}$$

$$\frac{CT(\texttt{C}) = \texttt{class C extends D \{...\}}}{\texttt{C} <: \texttt{D}}$$

Computation - Reduction

$$\frac{\mathit{fields}(\mathtt{C}) = \overline{\mathtt{C}}\ \overline{\mathtt{f}}}{(\mathtt{new}\ \mathtt{C}(\overline{\mathtt{e}}))\mathtt{.f}_i \longrightarrow \mathtt{e}_i} \qquad (\text{R-FIELD})$$

$$\frac{\mathit{mbody}(\mathtt{m}, \mathtt{C}) = (\overline{\mathtt{x}}, \mathtt{e}_0)}{\begin{array}{c}(\mathtt{new}\ \mathtt{C}(\overline{\mathtt{e}}))\mathtt{.m}(\overline{\mathtt{d}}) \\ \longrightarrow [\overline{\mathtt{d}}/\overline{\mathtt{x}}, \mathtt{new}\ \mathtt{C}(\overline{\mathtt{e}})/\mathtt{this}]\mathtt{e}_0\end{array}} \qquad (\text{R-INVK})$$

$$\frac{\mathtt{C} <: \mathtt{D}}{(\mathtt{D})(\mathtt{new}\ \mathtt{C}(\overline{\mathtt{e}})) \longrightarrow \mathtt{new}\ \mathtt{C}(\overline{\mathtt{e}})} \qquad (\text{R-CAST})$$

Computation - Congruence

$$\frac{\mathtt{e_0} \longrightarrow \mathtt{e_0}'}{\mathtt{e_0.f} \longrightarrow \mathtt{e_0}'\mathtt{.f}} \quad \text{(RC-Field)}$$

$$\frac{\mathtt{e_0} \longrightarrow \mathtt{e_0}'}{\mathtt{e_0.m}(\overline{\mathtt{e}}) \longrightarrow \mathtt{e_0}'\mathtt{.m}(\overline{\mathtt{e}})} \quad \text{(RC-Invk-Recv)}$$

$$\frac{\mathtt{e}_i \longrightarrow \mathtt{e}_i'}{\begin{array}{c}\mathtt{e_0.m}(\ldots,\mathtt{e}_i,\ldots) \\ \longrightarrow \mathtt{e_0.m}(\ldots,\mathtt{e}_i',\ldots)\end{array}} \quad \text{(RC-Invk-Arg)}$$

$$\frac{\mathtt{e}_i \longrightarrow \mathtt{e}_i'}{\begin{array}{c}\mathtt{new\ C}(\ldots,\mathtt{e}_i,\ldots) \\ \longrightarrow \mathtt{new\ C}(\ldots,\mathtt{e}_i',\ldots)\end{array}} \quad \text{(RC-New-Arg)}$$

$$\frac{\mathtt{e_0} \longrightarrow \mathtt{e_0}'}{\mathtt{(C)e_0} \longrightarrow \mathtt{(C)e_0}'} \quad \text{(RC-Cast)}$$

Expression Typing

$$\Gamma \vdash \mathtt{x} \in \Gamma(\mathtt{x}) \qquad \text{(T-Var)}$$

$$\frac{\Gamma \vdash \mathtt{e}_0 \in \mathtt{C}_0 \qquad \mathit{fields}(\mathtt{C}_0) = \overline{\mathtt{C}}\ \overline{\mathtt{f}}}{\Gamma \vdash \mathtt{e}_0.\mathtt{f}_i \in \mathtt{C}_i} \qquad \text{(T-Field)}$$

$$\frac{\Gamma \vdash \mathtt{e}_0 \in \mathtt{C}_0 \quad \mathit{mtype}(\mathtt{m}, \mathtt{C}_0) = \overline{\mathtt{D}} \to \mathtt{C} \quad \Gamma \vdash \overline{\mathtt{e}} \in \overline{\mathtt{C}} \qquad \overline{\mathtt{C}} <: \overline{\mathtt{D}}}{\Gamma \vdash \mathtt{e}_0.\mathtt{m}(\overline{\mathtt{e}}) \in \mathtt{C}} \qquad \text{(T-Invk)}$$

$$\frac{\mathit{fields}(\mathtt{C}) = \overline{\mathtt{D}}\ \overline{\mathtt{f}} \quad \Gamma \vdash \overline{\mathtt{e}} \in \overline{\mathtt{C}} \qquad \overline{\mathtt{C}} <: \overline{\mathtt{D}}}{\Gamma \vdash \mathtt{new}\ \mathtt{C}(\overline{\mathtt{e}}) \in \mathtt{C}} \qquad \text{(T-New)}$$

$$\frac{\Gamma \vdash \mathtt{e}_0 \in \mathtt{D} \qquad \mathtt{D} <: \mathtt{C}}{\Gamma \vdash (\mathtt{C})\mathtt{e}_0 \in \mathtt{C}} \qquad \text{(T-UCast)}$$

$$\frac{\Gamma \vdash \mathtt{e}_0 \in \mathtt{D} \qquad \mathtt{C} <: \mathtt{D} \qquad \mathtt{C} \neq \mathtt{D}}{\Gamma \vdash (\mathtt{C})\mathtt{e}_0 \in \mathtt{C}} \qquad \text{(T-DCast)}$$

$$\frac{\Gamma \vdash \mathtt{e}_0 \in \mathtt{D} \qquad \mathtt{C} \not<: \mathtt{D} \qquad \mathtt{D} \not<: \mathtt{C} \qquad \textit{stupid warning}}{\Gamma \vdash (\mathtt{C})\mathtt{e}_0 \in \mathtt{C}} \qquad \text{(T-SCast)}$$

Method typing and Class typing

Method typing:

$$\frac{\overline{\mathtt{x}} : \overline{\mathtt{C}}, \mathtt{this} : \mathtt{C} \vdash \mathtt{e}_0 \in \mathtt{E}_0 \qquad \mathtt{E}_0 <: \mathtt{C}_0 \qquad CT(\mathtt{C}) = \mathtt{class\ C\ extends\ D\ \{...\}} \qquad \mathit{override}(\mathtt{m}, \mathtt{D}, \overline{\mathtt{C}} \to \mathtt{C}_0)}{\mathtt{C}_0\ \mathtt{m}\ (\overline{\mathtt{C}}\ \overline{\mathtt{x}})\ \{\mathtt{return\ e}_0\mathtt{;}\}\ \mathtt{OK\ IN\ C}} \text{(T-Method)}$$

Class typing:

$$\frac{\mathtt{K} = \mathtt{C}(\overline{\mathtt{D}}\ \overline{\mathtt{g}},\ \overline{\mathtt{C}}\ \overline{\mathtt{f}})\ \{\mathtt{super}(\overline{\mathtt{g}})\mathtt{;}\ \mathtt{this.}\overline{\mathtt{f}} = \overline{\mathtt{f}}\mathtt{;}\} \qquad \mathit{fields}(\mathtt{D}) = \overline{\mathtt{D}}\ \overline{\mathtt{g}} \qquad \overline{\mathtt{M}}\ \mathtt{OK\ IN\ C}}{\mathtt{class\ C\ extends\ D}\ \{\overline{\mathtt{C}}\ \overline{\mathtt{f}}\mathtt{;}\ \mathtt{K}\ \overline{\mathtt{M}}\}\ \mathtt{OK}} \text{(T-Class)}$$

Auxiliary functions

Field lookup:

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

$$\frac{CT(\texttt{C}) = \texttt{class C extends D } \{\overline{\texttt{C}}\ \overline{\texttt{f}}\texttt{; K } \overline{\texttt{M}}\} \qquad \mathit{fields}(\texttt{D}) = \overline{\texttt{D}}\ \overline{\texttt{g}}}{\mathit{fields}(\texttt{C}) = \overline{\texttt{D}}\ \overline{\texttt{g}}, \overline{\texttt{C}}\ \overline{\texttt{f}}}$$

Method type lookup:

$$\frac{CT(\texttt{C}) = \texttt{class C extends D } \{\overline{\texttt{C}}\ \overline{\texttt{f}}\texttt{; K } \overline{\texttt{M}}\} \qquad \texttt{B m } (\overline{\texttt{B}}\ \overline{\texttt{x}})\ \texttt{\{return e;\}} \in \overline{\texttt{M}}}{\mathit{mtype}(\texttt{m}, \texttt{C}) = \overline{\texttt{B}} \rightarrow \texttt{B}}$$

$$\frac{CT(\texttt{C}) = \texttt{class C extends D } \{\overline{\texttt{C}}\ \overline{\texttt{f}}\texttt{; K } \overline{\texttt{M}}\} \qquad \texttt{m is not defined in } \overline{\texttt{M}}}{\mathit{mtype}(\texttt{m}, \texttt{C}) = \mathit{mtype}(\texttt{m}, \texttt{D})}$$

Auxiliary functions

Method body lookup:

$$\frac{CT(\mathtt{C}) = \mathtt{class\ C\ extends\ D\ \{\overline{C}\ \overline{f};\ K\ \overline{M}\}} \quad \mathtt{B\ m\ (\overline{B}\ \overline{x})\ \{return\ e;\}} \in \overline{\mathtt{M}}}{mbody(\mathtt{m}, \mathtt{C}) = (\overline{\mathtt{x}}, \mathtt{e})}$$

$$\frac{CT(\mathtt{C}) = \mathtt{class\ C\ extends\ D\ \{\overline{C}\ \overline{f};\ K\ \overline{M}\}} \quad \mathtt{m} \text{ is not defined in } \overline{\mathtt{M}}}{mbody(\mathtt{m}, \mathtt{C}) = mbody(\mathtt{m}, \mathtt{D})}$$

Valid method overriding:

$$\frac{mtype(\mathtt{m}, \mathtt{D}) = \overline{\mathtt{D}} \to \mathtt{D}_0, \text{implies } \overline{\mathtt{C}} = \overline{\mathtt{D}} \text{ and } \mathtt{C}_0 = \mathtt{D}_0}{override(\mathtt{m}, \mathtt{D}, \overline{\mathtt{C}} \to \mathtt{C}_0)}$$