Object – Oriented Technology

Chapter 4: Type Systems

Suad Alagić

Springer 2016

ASSIGNMENT

$$\frac{\mathcal{T} \vdash x : A\ var, \mathcal{T} \vdash e : A}{\mathcal{T} \vdash x = e : Statement}$$

$$\frac{\mathcal{T} \vdash x : int\ var, \mathcal{T} \vdash x + 1 : int}{\mathcal{T} \vdash x = x + 1 : Statement}$$

CONDITIONAL STATEMENT

$$\frac{\mathcal{T} \vdash e : \mathit{boolean}, \mathcal{T} \vdash S1 : \mathit{Statement}, \mathcal{T} \vdash S2 : \mathit{Statement}}{\mathcal{T} \vdash \textbf{if } (e)\ S1 \textbf{ else } S2 : \mathit{Statement}}$$

$$\frac{\mathcal{T} \vdash x > 0 : \mathit{boolean}, \mathcal{T} \vdash y = x : \mathit{Statement}, \quad \mathcal{T} \vdash y = -x : \mathit{Statement}}{\mathcal{T} \vdash \textbf{if } (x > 0)\ y = x \textbf{ else } y = -x : \mathit{Statement}}$$

WHILE LOOP

$$\frac{\mathcal{T} \vdash e : \ \mathit{boolean}, \mathcal{T} \vdash S : \ \mathit{Statement}}{\mathcal{T} \vdash \textbf{while}\ (e)\ S : \ \mathit{Statement}}$$

$$\frac{\mathcal{T} \vdash x < \mathit{highValue} : \ \mathit{boolean}, \mathcal{T} \vdash x = x + 1 : \ \mathit{Statement}}{\mathcal{T} \vdash \textbf{while}\ (x < \mathit{highValue})\ x = x + 1 : \ \mathit{Statement}}$$

DO LOOP

$$\frac{\mathcal{T} \vdash e : \mathit{boolean}, \mathcal{T} \vdash S : \mathit{Statement}}{\mathcal{T} \vdash \textbf{do } S \textbf{ while } (e) : \mathit{Statement}}$$

EXPRESSIONS

$$\frac{\mathcal{T} \vdash e1 : \ int, \mathcal{T} \vdash e2 : \ int, \quad \mathcal{T} \vdash \textbf{op} \ \in \{+, -, *, /, \%\}}{\mathcal{T} \vdash \ e1 \ \textbf{op} \ e2 : \ int}$$

$$\frac{\mathcal{T} \vdash x : \ int, \mathcal{T} \vdash y : \ int}{\mathcal{T} \vdash \ x + y : \ int}$$

$$\frac{\mathcal{T} \vdash e1 : \mathit{boolean}, \mathcal{T} \vdash e2 : \mathit{boolean}, \quad \mathcal{T} \vdash \mathbf{op} \in \{\&, |, \&\&, ||\}}{\mathcal{T} \vdash e1\ \mathbf{op}\ e2 : \mathit{boolean}}$$

SAMPLE CLASS

```
class Collection {
  private int size;
  private Object[] elements;
  public Collection() {...}
  public boolean belongs(Object x){...}
  public int cardinality(){...}
  public void add(Object x){...}
  public void remove(Object x) {...}
}
```

CLASS TYPE SIGNATURE

$$fields(C) =< int\ size;\ Object[]\ elements >$$
$$constructors(C) =< Collection() >$$
$$methods(C) =< boolean\ belongs(Object);\ int\ cardinality();$$
$$void\ add(Object);\ void\ remove(Object) >$$

Type signature:

$$Sig(C) = \{fields(C); constructors(C); methods(C)\}$$

TYPE SIGNATURES

- Field signatures:
  $fields(A) =< F_i\ f_i >_{i=1}^{k^A}$ (a sequence of fields) with F_i the type of field f_i, and $f_i \neq f_j$ for $i \neq j$, where $j = 1, 2, \ldots, k^A$.
- Constructor signatures:
  $constructors(A) = \{A(C_{c_{i_1}}, C_{c_{i_2}}, \ldots, C_{c_{i_n}})\}_{i=1}^{c^A}$
  (a collection of constructors) where $C_{m_{i_j}}$ are types for $j = 1, 2, \ldots, n$.
- Method signatures:
  $methods(A) = \{C_{m_i}\ m_i(C_{m_{i_1}}, C_{m_{i_2}}, \ldots, C_{m_{i_n}})\}_{i=1}^{m^A}$
  (a collection of methods) where m_i is the method name, and C_{m_i}, $C_{m_{i_j}}$ are types where $j = 1, 2, \ldots, n$.

$$\frac{\mathcal{T} \cup \{\textbf{class } D \textbf{ extends } B\} \Rightarrow \mathcal{T}'}{\mathcal{T}' \vdash D <: B}$$

$$\frac{\mathcal{T} \vdash \ T1 \ <: T2, \ \mathcal{T} \vdash \ T2 \ <: T3}{\mathcal{T} \vdash T1 \ <: \ T3}$$

ASSIGNMENT

$$\frac{\mathcal{T} \vdash x : A\ var, \mathcal{T} \vdash e : B, \quad \mathcal{T} \vdash B <: A}{\mathcal{T} \vdash x = e : Statement}$$

ASSIGNMENT

$$\frac{\mathcal{T} \vdash x : \mathit{Person}\ \mathit{var}, \mathcal{T} \vdash y : \mathit{Employee}, \quad \mathcal{T} \vdash \mathit{Employee} <: \mathit{Person}}{\mathcal{T} \vdash x = y : \mathit{Statement}}$$

SUBSUMPTION

$$\frac{\mathcal{T} \vdash e : B, \quad \mathcal{T} \vdash B <: A,}{\mathcal{T} \vdash e : A}$$

$$\mathcal{T} \vdash D <: B,$$
$$\mathcal{T} \vdash fields(B) =< F_i^B \ f_i^B >_{i=1}^{k^B},$$
$$\mathcal{T} \vdash fields(D) =< F_i^D \ f_i^D >_{i=1}^{k^D},$$
$$k^B \leq k^D,$$
$$j \in \{1, 2, \ldots, k^B\}$$

$$\mathcal{T} \vdash < F_j^B \ f_j^B > = < F_j^D \ f_j^D >$$

$$\mathcal{T} \vdash \mathit{OrderedCollection} <: \mathit{Collection},$$
$$\mathcal{T} \vdash \mathit{fields}(\mathit{Collection}) =< \mathit{int\ size};\ \mathit{Object}[]\ \mathit{elements} >,$$
$$\mathcal{T} \vdash \mathit{fields}(\mathit{OrderedCollection}) =<$$
$$\mathit{int\ size}; \mathit{Object}[]\ \mathit{elements};\ \mathit{other\ fields} >$$

$$\frac{\mathcal{T} \vdash D <: B, \quad \mathcal{T} \vdash C_m\ m(C_{m_1}, C_{m_2}, \ldots, C_{m_n}) \in methods(B)}{\mathcal{T} \vdash C_m\ m(C_{m_1}, C_{m_2}, \ldots, C_{m_n}) \in methods(D)}$$

$$\frac{\mathcal{T} \vdash \mathit{Collection} <: \mathit{Object}, \quad \mathcal{T} \vdash \mathit{boolean\ equals}(\mathit{Object}) \in \mathit{methods}(\mathit{Object})}{\mathcal{T} \vdash \mathit{boolean\ equals}(\mathit{Object}) \in \mathit{methods}(\mathit{Collection})}$$

OVERRIDING

$$\mathcal{T} \vdash \mathit{Collection} <: \mathit{Object},$$
$$\mathcal{T} \vdash \mathit{Object}\ \mathit{clone}() \in \mathit{methods}(\mathit{Object})$$

$$\mathcal{T} \vdash \mathit{Collection}\ \mathit{clone}() \in \mathit{methods}(\mathit{Collection})$$

$$\frac{\mathcal{T} \vdash C_m\ m(C_{m_1}, C_{m_2}, \ldots, C_{m_n}) \in methods(A),\quad \mathcal{T} \vdash C'_m\ m(C'_{m_1}, C'_{m_2}, \ldots, C'\,m_n) \in methods(A)}{\mathcal{T} \vdash (\exists i)((i \in \{1, 2, \ldots, n\}), (C_{m_i} \neq C'_{m_i}))}$$

$$\frac{\begin{array}{c}\mathcal{T} \vdash x : A, \\ \mathcal{T} \vdash C_m\ m(C_{m_1}, C_{m_2}, \ldots, C_{m_n}) \in methods(A), \\ \mathcal{T} \vdash a_i : A_i, i \in \{1, 2, \ldots, n\}, \\ \mathcal{T} \vdash (\forall i)(i \in \{1, 2, \ldots, n\} \Rightarrow A_i <: C_{m_i})\end{array}}{\mathcal{T} \vdash x.m(a_1, a_2, \ldots, a_n) : C_m}$$

$$\frac{\mathcal{T} \vdash \mathit{employees} : \mathit{Collection\ var},\ \mathcal{T} \vdash \mathit{boolean\ add}(\mathit{Object}) \in \mathit{methods}(\mathit{Collection}),\ \mathcal{T} \vdash x : \mathit{Employee},\ \mathcal{T} \vdash \mathit{Employee} <: \mathit{Object}}{\mathcal{T} \vdash \mathit{employees.add}(x) : \mathit{boolean}}$$

TYPE CASTS

$$\frac{\mathcal{T} \vdash e : A, \mathcal{T} \vdash D <: A}{\mathcal{T} \vdash (D)e : D}$$

TYPE CASTS

$$\frac{\mathcal{T} \vdash e : \mathit{Person}, \mathcal{T} \vdash \mathit{Employee} <: \mathit{Person}}{\mathcal{T} \vdash (\mathit{Employee})e : \mathit{Employee}}$$

TYPE CASTS

$$\frac{\mathcal{T} \vdash e : A, \mathcal{T} \vdash A <: D}{\mathcal{T} \vdash (D)e : D}$$

STATIC and DYNAMIC TYPE

$$\frac{\mathcal{T} \vdash e : A}{\mathcal{T} \vdash eval(e.getClass()) <: eval(Class.forName(''A''))}$$

TYPE CASTS

$$\frac{\mathcal{T} \vdash e : A, \mathcal{T} \vdash D <: A, \quad \mathcal{T} \vdash eval(e.getClass()) \not<: eval(Class.forName(''D''))}{\mathcal{T} \vdash eval((D)e) : ClassCastException}$$

$$\frac{\mathcal{T} \vdash T <: Object \Rightarrow \mathcal{T} \vdash \textbf{class}\ C < T > \{fields(C);\ methods(C)\} :\ Class, \mathcal{T} \vdash D <: Object}{\mathcal{T} \vdash C < D > <: Object}$$

UNIVERSAL INSTANTIATION

$$\frac{\begin{array}{c}\mathcal{T} \vdash T <: \mathit{Object} \Rightarrow \\ \mathcal{T} \vdash \textbf{class}\ C < T > \{\mathit{fields}(C);\ \mathit{methods}(C)\} :\ \mathit{Class}, \\ \mathcal{T} \vdash D <: \mathit{Object}\end{array}}{\begin{array}{c}\mathcal{T} \vdash \mathit{Sig}(C < D >) = \\ \{\mathit{fields}(C)[D/T];\ \mathit{methods}(C)[D/T]\}\end{array}}$$

UNIVERSAL INSTANTIATION

Sig(Collection<T>) =
< T[] elements; boolean belongs(T);
void add(T); void remove(T) >.

Sig(Collection<Employee>) =
< Employee[] elements; boolean belongs(Employee);
void add(Employee); void remove(Employee) >.

BOUNDED PARAMETRIC

$$\frac{\begin{array}{c}\mathcal{T} \vdash B <: \mathit{Object}, \\ \mathcal{T} \vdash T <: B \Rightarrow \\ \mathcal{T} \vdash \textbf{class } C < T \textbf{ extends } B > \{\mathit{fields}(C);\ \mathit{methods}(C)\} :\ \mathit{Class}, \\ \mathcal{T} \vdash D <: B[D/T]\end{array}}{\mathcal{T} \vdash C < D > <: \mathit{Object}}$$

BOUNDED INSTANTIATION

Sig(OrderedCollection<T **extends** Comparable< *T* >>)
< T[] elements; boolean belongs(T);
void add(T); void remove(T) >.

Employee <: Comparable<Employee>

Sig(OrderedCollection<Employee>) =
< Employee[] elements; boolean belongs(Employee);
void add(Employee); void remove(Employee) >.

TYPE ERASURE: COLLECTION CLASS

```
class Collection {
  private Object[] elements;
  public boolean belongs(Object x){...}
  public int size(){...}
  public void add(Object x){...}
  public void remove(Object x) {...}
}
```

TYPE ERASURE: PARAMETRIC CLASS

```
class Collection<T> {
  private T[] elements;
  public boolean belongs(T x){...}
  public int size(){...}
  public void add(T x){...}
  public void remove(T x) {...}
}
```

TYPE ERASURE: SIGNATURES

boolean belongs(Object x)
void add(Object x)
void remove(Object x)

boolean belongs(Employee x)
void add(Employee x)
void remove(Employee x)

Collection $\not<:$ Collection<Employee>
Collection<Employee> $\not<:$ Collection

TYPE ERASURE: LEGACY CLASSES

```
public class LegacyClass {
  public void addEmployees(Collection c) {
  for (int i=0; i < 5; i++)
        c.add(new Employee());
  }
}
public class TestLegacy {
  public static void main(String[] args) {
  Collection<Department> c =
        new LinkedList<Department>();
        new LegacyClass().addEmployees(c);
  }
}
```

TYPE ERASURE: MESSAGE PASSING

```
public class MessagePassing {
  public static void main(String[] args) {
  LinkedList<Employee> employees =
        new LinkedList<Employee>();
  LinkedList<Department> departments =
        new LinkedList<Department>();
  updateCollection(employees,departments);
  }
  public static void updateCollection
        (LinkedList collectionA, LinkedList collectionB) {
  collectionA.addAll(collectionB);
  }
}
```

TYPE ERASURE: OVERLOADING

```
public class UpdateTransaction {
  public void update(Collection<Employee> employees) {...}
  public void update(Collection<Department> departments) {...}
}
```

TYPE ERASURE: OVERLOADING

```
public interface EmployeeView {
  public Collection<Employee>
        select(Collection<Employee> x);
}
public interface DepartmentView
        extends EmployeeView {
  public Collection<Department>
        select(Collection<Department> x);
  }
public class Company
        implements DepartmentView {
  public Collection<Employee>
        select(Collection<Employee> x) {...}
  // public Collection<Department>
        // select(Collection<Department> x) {...}
  }
```

TYPE ERASURE: ASSIGNMENTS

```
public class EmployeesAreDepartments {
  public static void main(String[] args) {
  LinkedList<Department> departments =
        new LinkedList<Department>();
  LinkedList<Employee> employees =
        new LinkedList<Employee>();
  LinkedList objects = employees;
  departments = objects;
  }
}
```

TYPE ERASURE: SUBTYPING

```
public class UpdateTransaction {
  public static void main(String[] args) {
  LinkedList<Employee> employees =
        new LinkedList<Employee>();
  employees.add(new Employee("Joe"));
  LinkedList objects = new LinkedList();
  objects.add(new Object());
  updateCollection(employees);
  updateDepartment(objects);
  }
  public static void updateCollection(
        LinkedList collection) {
  collection.add(new Department());
  }
  public static void updateDepartment(
        LinkedList<Department> collection) {
  Department d = collection.getFirst();
  }
```

TYPE ERASURE: DYNAMIC TYPECHECKING

```
public class UpdateTransaction {
  public static void main(String[] args) {
  LinkedList<Employee> employees =
        new LinkedList<Employee>();
  updateCollection(employees);
  Employee emp = employees.remove();
  // runtime exception here
  }
  public static void updateCollection(Object collection) {
  ((LinkedList<Department>)collection).add(new Department());
  }
}
```

TYPE ERASURE: SERIALIZIBILITY

```
public class StoreCollectionObject {
  public static void main(String[] args)
        throws Exception {
  FileOutputStream fileout =
        new FileOutputStream("employees");
  ObjectOutputStream out =
        new ObjectOutputStream(fileout);
  Collection<Employee> employees =
        new LinkedList<Employee>();
  employees.add(new Employee());
  out.writeObject(employees);
  }
}
```

TYPE ERASURE: SERIALIZIBILITY

```
public class ReadCollectionObject {
  public static void main(String[] args)
        throws Exception {
  FileInputStream filein =
        new FileInputStream("employees");
  ObjectInputStream in =
        new ObjectInputStream(filein);
  Collection<Department> departments = null;
  try { departments =
        (Collection<Department>) in.readObject();
        } catch (ClassCastException e) {exception handling }
}
```

```
Iterator<Department> it = departments.iterator();
Department d = it.next(); ???
```

TYPE ERASURE: REFLECTION

```
public class ReflectiveTransaction {
  public void updateEmployees(
        LinkedList<Employee> collection) {
        Employee e = collection.remove();
  // unexpected ClassCastException!? }
  public static void main(String[] args) {
  ReflectiveTransaction trans =
        new ReflectiveTransaction();
  Collection<Department> departments =
        new LinkedList<Department>();
  departments.add(new Department());
  try {
        Method method =
        trans.getClass().getMethod
        ("updateEmployees", departments.getClass());
        method.invoke(trans, departments);
        } catch (Exception e) {exception handling }
  }
}
```