CS 476 – Programming Language Design

William Mansky

Questions

Nobody has responded yet.

Hang tight! Responses are coming in.

Imperative Languages

- Arithmetic and boolean expressions
- Variables and assignment
- Control flow (conditionals, loops)
- Variable declarations
- Function declarations and calls

Object-Oriented Imperative Languages

- Arithmetic and boolean expressions
- Variables and assignment
- Control flow (conditionals, loops)
- Variable declarations
- Function declarations and calls
- ***Objects and classes***

- Exercise: What is an object in a programming language?

Object-Oriented Programming

- An *object* is a kind of value
- Objects have *fields* (object-specific variables) and *methods* (object-specific functions)

o

Name	Value
x	3
y	5
getx()	return x;
sety(n)	y := n;

```
ox := o.getx();
o.sety(ox);
```

Object-Oriented Programming

- An *object* is a kind of value
- Objects have *fields* (object-specific variables) and *methods* (object-specific functions)
- Different objects may provide the same method but have different code for it (dynamic dispatch)

o

Name	Value
x	3
y	3
getx()	return x;
sety(n)	y := n;

o2

Name	Value
r	4
theta	pi/4
getx()	return r * cos(theta);
sety(n)	…

```
o.getx();
o2.getx();
```

Object-Oriented Programming

- An *object* is a kind of value
- Objects have *fields* (object-specific variables) and *methods* (object-specific functions)
- Different objects may provide the same method but have different code for it (dynamic dispatch)
- We can only access a field/method by going through its object
- An object may belong to a *class*, which describes a list of fields and methods the object contains
- Classes may have *subclasses*, which extend them with more fields and methods

Java-Like Language: Classes

```
class Square extends Shape {
  int side;

  int area(){
    return this.side *
           this.side;
  }
}
```

```
Square s = new Square(3);
```

Name	Value
side	3

```
int x = s.area();
// x will be 9
```

Java-Like Language: Syntax

CL ::= `class` <id> `extends` <id> `{` *T* <id>`;` … `;` *T* <id>`;` *M* … *M* `}`

M ::= *T* <id>`(`*T* <id>`,` … `,` *T* <id>`){` *C* `}`

P ::= *CL* … *CL*

E ::= <#> | *E* `+` *E* | <id> | … | *E*`.`<id>

C ::= <id> `=` *E* | … | <id> `=` *E*`.`<id>`(`*E*`,` … `,` *E*`)`
| <id> `=` `new` <id>`(`*E*`,` … `,` *E*`)`

T ::= `int` | <id>

Java-Like Language: Syntax

CL ::= `class` <id> `extends` <id> `{` *T* <id>`;` … `;` *T* <id>`;` *M* … *M* `}`

M ::= *T* <id>`(`*T* <id>`,` … `,` *T* <id>`){` *C* `}`

P ::= *CL* … *CL*

E ::= <#> | *E* `+` *E* | <id> | … | *E*`.`<id>

C ::= <id> `=` *E* | … | <id> `=` *E*`.`<id>`(`*E*`,` … `,` *E*`)`
| <id> `=` `new` <id>`(`*E*`,` … `,` *E*`)`

T ::= `int` | <id>

Questions

Nobody has responded yet.

Hang tight! Responses are coming in.

Java-Like Language: Subtyping

```
class A extends Object {
  int x;

  int getx(){
    return this.x;
  }
}
```

```
class B extends A {
  int y;

  int gety(){
    return this.y;
  }
}
```

Java-Like Language: Subtyping

```
(new A(5)).getx();
(new B(5, 6)).getx();  // A has getx method,
                       // and B extends A
```

- Anywhere an object of class A is expected, an object of class B should work just as well!

$$\frac{\Gamma \vdash e : \tau_1 \quad \tau_1 <: \tau_2}{\Gamma \vdash e : \tau_2}$$

- "Anything of type B is also of type A"
- $\tau_1 <: \tau_2$ means τ_2 is in τ_1's list of superclasses

Questions

Nobody has responded yet.

Hang tight! Responses are coming in.

Java-Like Language: Types

- Types: int, any class name
- Rules:

$$\frac{\Gamma \vdash e_1 : \text{int} \quad \Gamma \vdash e_2 : \text{int}}{\Gamma \vdash e_1 + e_2 : \text{int}} \qquad \frac{\Gamma \vdash e : \tau_1 \quad \tau_1 <: \tau_2}{\Gamma \vdash e : \tau_2}$$

$$\frac{?}{\Gamma \vdash e.f : \tau} \qquad \frac{?}{\Gamma \vdash \text{new } C(e_1, \ldots e_n) : \tau}$$

$$\frac{?}{\Gamma \vdash x = e.m(e_1, \ldots, e_n) : \text{ok}}$$

Java-Like Language: Types

- Types: int, any class name

$$\frac{?}{\Gamma \vdash e.f : \tau}$$

- Exercise: When is a field access type-correct?

```
class A extends Object {
  int x;
  …
}

objA.x : int
```

Java-Like Language: Types

- Types: int, any class name

$$\frac{?}{\Gamma \vdash e.f : \tau}$$

- e is an object of a class C
- C has a field f of type τ

```
class A extends Object {
  int x;
  …
}

objA.x : int
```

Java-Like Language: Types

- Types: int, any class name

$$\frac{\Gamma \vdash e : C \quad (\text{fields}(\Gamma, C) = \cdots, \tau\ f)}{\Gamma \vdash e.f : \tau}$$

Lists all the fields of C, its superclass, its superclass's superclass, and so on up to Object

- e is an object of a class C
- C has a field f of type τ

```
class A extends Object {
  int x;
  …
}

objA.x : int
```

Java-Like Language: Types

- Types: int, any class name

$$\frac{?}{\Gamma \vdash \texttt{new}\ C(e_1, \ldots e_n) : \tau}$$

```
class A extends Object
{
  int x;
}

new A(5) : ?
```

Java-Like Language: Types

- Types: int, any class name

$$\frac{?}{\Gamma \vdash \texttt{new}\ C(e_1, \ldots e_n) : C}$$

- $e_1, \ldots e_n$ have the right types
- Assuming default constructor takes initial values for each field in order

```
class A extends Object
{
  int x;
}

new A(5) : ?
```

Java-Like Language: Types

- Types: int, any class name

$$\frac{(\text{fields}(\Gamma, C) = \tau_1\ f_1, \ldots, \tau_n\ f_n) \quad \Gamma \vdash e_1 : \tau_1 \ \ldots\ \Gamma \vdash e_n : \tau_n}{\Gamma \vdash \texttt{new}\ C(e_1, \ldots e_n) : C}$$

- $e_1, \ldots e_n$ have the right types (the types of the fields of C, in order)

```
class A extends Object
{
  int x;
}

new A(5) : ?
```

Java-Like Language: Types

- Types: int, any class name

$$\frac{?}{\Gamma \vdash x = e.m(e_1, \ldots e_n) : \text{ok}}$$

- e is an object of a class C
- C has a method m
- args and return of m are well typed

```
class A extends Object {
  int x;
  int getx(){
     return this.x; }
}

y = objA.getx();
```

Java-Like Language: Types

- Types: int, any class name

$$\frac{\Gamma \vdash e : C \quad \left(\text{methods}(\Gamma, C) = \cdots, \tau\ m(\tau_1\ x_1, \ldots, \tau_n\ x_n)\right) \qquad \Gamma \vdash e_1 : \tau_1\ \ldots\ \Gamma \vdash e_n : \tau_n \quad \Gamma(x) = \tau}{\Gamma \vdash x = e.m(e_1, \ldots e_n) : \text{ok}}$$

- e is an object of a class C
- C has a method m
- args and return of m are well typed

```
class A extends Object {
   int x;
   int getx(){
      return this.x; }
}

y = objA.getx();
```

Questions

Nobody has responded yet.

Hang tight! Responses are coming in.

Java-Like Language: Declarations

- In a Java-like language, all the code is in class declarations

CL ::= `class` <id> `extends` <id> `{` *T* <id>`;` … `;` *T* <id>`;` *M* … *M* `}`

- We need to store all the class declarations in the context Γ, and then check that each method is type-correct

Java-Like Language: Declarations

```
class A extends Object {
  int x;
  int addx(int y){
    return this.x + y;
  }
}
```

$$\frac{?}{\Gamma \vdash \tau\ m(\tau_1\ x_1, \ldots, \tau_n\ x_n)\{\ c\ \} : \mathrm{ok}}$$

Java-Like Language: Declarations

```
class A extends Object {
  int x;
  int addx(int y){
    return this.x + y;
  }
}
```

$$\frac{\Gamma[x_1 \mapsto \tau_1, \ldots, x_n \mapsto \tau_n] \vdash c : \text{ok}}{\Gamma \vdash \tau\ m(\tau_1\ x_1, \ldots, \tau_n\ x_n)\{\ c\ \} : \text{ok}}$$

- A method declaration is typed like a function declaration: add the parameters to the context, then typecheck the body

Java-Like Language: Declarations

```
class A extends Object {
  int x;
  int addx(int y){
    return this.x + y;
  }
}
```

$$\frac{\Gamma[x_1 \mapsto \tau_1, \ldots, x_n \mapsto \tau_n] \vdash c : \text{ok}}{\Gamma \vdash \tau\ m(\tau_1\ x_1, \ldots, \tau_n\ x_n)\{\ c\ \} : \text{ok}}$$

- A method declaration is typed like a function declaration: add the parameters to the context, then typecheck the body

Java-Like Language: Declarations

```
class A extends Object {
  int x;
  int addx(int y){
    return this.x + y;
  }
}
```

$$\frac{\Gamma[\texttt{this} \mapsto ?, x_1 \mapsto \tau_1, \ldots, x_n \mapsto \tau_n] \vdash c : \text{ok}}{\Gamma \vdash \tau\ m(\tau_1\ x_1, \ldots, \tau_n\ x_n)\{\ c\ \} : \text{ok}}$$

- A method declaration is typed like a function declaration: add the parameters to the context, then typecheck the body
- ~~Exercise: What is the type of "this"?~~

Java-Like Language: Declarations

```
class A extends Object {
  int x;
  int addx(int y){
    return this.x + y;
  }
}
```

$$\frac{\Gamma[\texttt{this} \mapsto C, x_1 \mapsto \tau_1, \ldots, x_n \mapsto \tau_n] \vdash c : \text{ok}}{\Gamma, C \vdash \tau\ m(\tau_1\ x_1, \ldots, \tau_n\ x_n)\{\ c\ \} : \text{ok}}$$

- A method declaration is typed like a function declaration
- Except that it is called on an object
- And it might be overriding a superclass's method – some languages have additional restrictions in this case

Java-Like Language: Overriding

```
class A extends Object {
  int x;

  int getval(){
    return this.x;
  }
}
```

```
class B extends A {
  int y;

  int getval(){
    return this.x +
           this.y;
  }
}
```

Java-Like Language: Overriding

```
class A extends Object {
  int x;

  int getval(){
    return this.x;
  }
}
```

```
class B extends A {
  int y;

  int getval(int x){
    return this.x +
           this.y + z;
  }
}
```

Java-Like Language: Overriding

```
class A extends Object {
  int x;

  int getval(A z){
    return this.x;
  }
}
```

```
class B extends A {
  int y;

  int getval(B z){
    return this.x +
           this.y + z;
  }
}
```

Java-Like Language: Declarations

$$\frac{\Gamma[\texttt{this} \mapsto C, x_1 \mapsto \tau_1, \ldots, x_n \mapsto \tau_n] \vdash c : \text{ok} \quad \text{can\_override}\big(\Gamma, C, \tau\ m(\tau_1\ x_1, \ldots, \tau_n\ x_n)\big)}{\Gamma, C \vdash \tau\ m(\tau_1\ x_1, \ldots, \tau_n\ x_n)\{\ c\ \} : \text{ok}}$$

- A method declaration is typed like a function declaration
- Except that it is called on an object
- And it might be overriding a superclass's method – some languages have additional restrictions in this case

Questions

Nobody has responded yet.

Hang tight! Responses are coming in.

Java-Like Language: Declarations

$$\frac{\Gamma, C \vdash M_1 : \text{ok} \;\ldots\; \Gamma, C \vdash M_j : \text{ok}}{\Gamma \vdash \texttt{class}\; C\; \texttt{extends}\; D \left\{ \tau_1\; f_1\texttt{;}\; \ldots\; \texttt{;}\; \tau_n\; f_n\texttt{;}\; M_1 \ldots M_j \right\} : \text{ok}}$$

- A class declaration is well typed if all its methods are well typed

Questions

Nobody has responded yet.

Hang tight! Responses are coming in.

Object-Oriented Programming

- An *object* is a kind of value
- Objects have *fields* (object-specific variables) and *methods* (object-specific functions)
- Different objects may provide the same method but have different code for it (dynamic dispatch)

➢We can only access a field/method by going through its object

- An object may belong to a *class*, which describes a list of fields and methods the object contains
- Classes may have *subclasses*, which extend them with more fields and methods

Java-Like Language: Access Modifiers

$$\frac{\Gamma \vdash e : C \qquad (\text{fields}(\Gamma, C) = \cdots, \tau\ f)}{\Gamma \vdash e.f : \tau}$$

```
B b1 = new B(1, 2);
b1.y = 5; // error
```

```
class B extends A {
  private int y;

  public int gety(){
    return this.y;
  }
  public int addy(B b){
    this.y = this.y + b.y;
  }
}
```

Java-Like Language: Access Modifiers

$$\frac{\Gamma \vdash e : C \quad (\text{fields}(\Gamma, C) = \cdots, \texttt{public}\ \tau\ f)}{\Gamma \vdash e.f : \tau}$$

$$\frac{\Gamma, C \vdash e : C \quad (\text{fields}(\Gamma, C) = \cdots, \texttt{private}\ \tau\ f)}{\Gamma, C \vdash e.f : \tau}$$

```
class B extends A {
  private int y;

  public int gety(){
    return this.y;
  }
  public int addy(B b){
    this.y = this.y + b.y;
  }
}
```

Java-Like Language: Access Modifiers

$$\frac{\Gamma, D \vdash e : C \quad (\text{fields}(\Gamma, C) = \cdots, \texttt{public}\ \tau\ f)}{\Gamma, D \vdash e.f : \tau}$$

$$\frac{\Gamma, C \vdash e : C \quad (\text{fields}(\Gamma, C) = \cdots, \texttt{private}\ \tau\ f)}{\Gamma, C \vdash e.f : \tau}$$

$$\frac{\Gamma[\texttt{this} \mapsto C, x_1 \mapsto \tau_1, \ldots] \vdash c : \text{ok}}{\Gamma, C \vdash \tau\ m(\tau_1\ x_1, \ldots)\{\ c\ \} : \text{ok}}$$

```
class B extends A {
  private int y;

  public int gety(){
    return this.y;
  }
  public int addy(B b){
    this.y = this.y + b.y;
  }
}
```

Java-Like Language: Access Modifiers

$$\frac{\Gamma, D \vdash e : C \quad (\text{fields}(\Gamma, C) = \cdots, \texttt{public}\ \tau\ f)}{\Gamma, D \vdash e.f : \tau}$$

$$\frac{\Gamma, C \vdash e : C \quad (\text{fields}(\Gamma, C) = \cdots, \texttt{private}\ \tau\ f)}{\Gamma, C \vdash e.f : \tau}$$

$$\frac{\Gamma[\texttt{this} \mapsto C, x_1 \mapsto \tau_1, \ldots], C \vdash c : \text{ok}}{\Gamma, C \vdash \tau\ m(\tau_1\ x_1, \ldots)\{\ c\ \} : \text{ok}}$$

```
class B extends A {
  private int y;

  public int gety(){
    return this.y;
  }
  public int addy(B b){
    this.y = this.y + b.y;
  }
}
```

Questions

Nobody has responded yet.

Hang tight! Responses are coming in.