Questions?
Imperative Languages

• Arithmetic and boolean expressions
• Variables and assignment
• Control flow (conditionals, loops)
• Variable declarations
• Exceptions and exception handling
• Function declarations and calls
Object-Oriented Imperative Languages

• Arithmetic and boolean expressions
• Variables and assignment
• Control flow (conditionals, loops)
• Variable declarations
• Exceptions and exception handling
• 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)

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>3</td>
</tr>
<tr>
<td>y</td>
<td>5</td>
</tr>
<tr>
<td>getx()</td>
<td>return x;</td>
</tr>
<tr>
<td>sety(n)</td>
<td>y := n;</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>3</td>
</tr>
<tr>
<td>y</td>
<td>3</td>
</tr>
<tr>
<td>getx()</td>
<td>return x;</td>
</tr>
<tr>
<td>sety(n)</td>
<td>y := n;</td>
</tr>
</tbody>
</table>
```

```
o2

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>4</td>
</tr>
<tr>
<td>theta</td>
<td>pi/4</td>
</tr>
<tr>
<td>getx()</td>
<td>return r * cos(theta);</td>
</tr>
<tr>
<td>sety(n)</td>
<td>...</td>
</tr>
</tbody>
</table>
```

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;
    Square(int side) {
        super();
        this.side = side;
    }
    int area() {
        return this.side * this.side;
    }
}

Square s = new Square(3);
int x = s.area();
// x will be 9
Java-Like Language: Syntax

CL ::= class <id> extends <id> { T <id>, ..., T <id>; K M ... M }
K ::= <id>(T <id>, ..., T <id>){
    super(<id>, ..., <id>); this.<id> = <id>; ...; this.<id> = <id>; }
M ::= T <id>(T <id>, ..., T <id>){ C }

E ::= <#> | E + E | <id> | E.<id>
C ::= <id> = E; | <id> = E.<id>(E, ..., E); | <id> = new <id>(E, ..., E)
T ::= int | <id>
P ::= CL ... CL C
Java-Like Language: Syntax

\( CL ::= \) class <id> extends <id> \{ \( T \) <id>, ..., \( T \) <id>; \( K M \) ... \( M \) \}

\( K ::= \) <id>(\( T \) <id>, ..., \( T \) <id>)\{
      super(<id>, ..., <id>); this.<id> = <id>; ...; this.<id> = <id>; \}

\( M ::= \) \( T \) <id>(\( T \) <id>, ..., \( T \) <id>)\{ \( C \) \}

\( E ::= \) ... \mid E.<id>

\( C ::= \) ... \mid <id> = E.<id>(E, ..., E); \mid <id> = new <id>(E, ..., E)

\( T ::= \) int \mid <id>

\( P ::= \) CL ... CL C
class A extends Object {
    int x;
    A(int x){
        super(); this.x = x;
    }
    int getx(){
        return this.x;
    }
}

class B extends A {
    int y;
    B(int x, int y){
        super(x); this.y = 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!

\[ \Gamma \vdash e : \tau_1 \quad \tau_1 \leq \tau_2 \]
\[ \Gamma \vdash e : \tau_2 \]

• “Anything of type B is also of type A”
Java-Like Language: Subtyping

\[
\Gamma \vdash C <: C \\

(\Gamma(C) = \text{class } C \text{ extends } D \{ \ldots \}) \\
\Gamma \vdash C <: D \\
\Gamma \vdash C <: D \quad \Gamma \vdash D <: E \\
\Gamma \vdash C <: E
\]
Java-Like Language: Subtyping

class Shape extends Object { ... }
class Square extends Shape { ... }

\[ \Gamma \vdash C <: D \quad \Gamma \vdash D <: E \]

\[ \frac{\Gamma \vdash C <: E}{\Gamma \vdash \text{Square} <: \text{Object}} \]
Java-Like Language: Subtyping

class Shape extends Object { ... }
class Square extends Shape { ... }

\[
\Gamma \vdash C <: D \quad \Gamma \vdash D <: E
\]
\[
\Gamma \vdash C <: E
\]

\[
\Gamma \vdash \text{Square} <: ? \quad \Gamma \vdash ? <: \text{Object}
\]
\[
\Gamma \vdash \text{Square} <: \text{Object}
\]
Java-Like Language: Subtyping

class Shape extends Object { ... }
class Square extends Shape { ... }

\[(\Gamma(C) = \text{class } C \text{ extends } D \{ ... \})\]
\[\Gamma \vdash C <: D\]

\[\Gamma \vdash \text{Square} <: ? \quad \Gamma \vdash ? <: \text{Object}\]
\[\Gamma \vdash \text{Square} <: \text{Object}\]
Java-Like Language: Subtyping

class Shape extends Object { ... }
class Square extends Shape { ... }

\[
\Gamma(C) = \text{class } C \text{ extends } D \{ \ldots \} \\
\Gamma \vdash C <: D
\]

\[
\Gamma(Square) = \ldots \quad (\Gamma(\text{Shape}) = \ldots)
\]
\[
\Gamma \vdash \text{Square} <: \text{Shape} \\
\Gamma \vdash \text{Shape} <: \text{Object}
\]
\[
\Gamma \vdash \text{Square} <: \text{Object}
\]
Java-Like Language: Types

• Types: int, any class name

• Rules:

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

\[ \frac{?}{\Gamma \vdash e.f : \tau} \quad \frac{?}{\Gamma \vdash \text{new } C(e_1, ..., e_n) : \tau} \]

\[ \frac{?}{\Gamma \vdash x = e.m(e_1, ..., e_n) : \text{ok}} \]
Java-Like Language: Types

• Types: int, any class name

\[ \Gamma \vdash e. f : \tau \]

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

```java
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 \(\tau\)

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

\texttt{objA.x : int}
Java-Like Language: Types

• Types: int, any class name

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

• \( e \) is an object of a class \( C \)
• \( C \) has a field \( f \) of type \( \tau \)

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

objA.x : int
Java-Like Language: Types

• Types: int, any class name

\[
\begin{align*}
\Gamma & \vdash \text{new } C(e_1, \ldots, e_n) : \tau \\
\text{class } A \text{ extends } \text{Object} \{ \\
& \quad \text{int } x; \\
& \quad A(int x)\{ \text{super(); this.x = x;} \} \\
& \}
\end{align*}
\]

\text{new } A(5) : ?
Java-Like Language: Types

• Types: int, any class name

\[
? \quad \Gamma \vdash \text{new } C(e_1, \ldots, e_n) : C
\]

• \(e_1, \ldots, e_n\) have the right types

```java
class A extends Object {
    int x;
    A(int x) {
        super();
        this.x = x;
    }
}
```

```java
new A(5) : A
```
Java-Like Language: Types

• Types: int, any class name

\[(\text{fields}(\Gamma, C) = \tau_1 f_1, \ldots, \tau_n f_n) \quad \Gamma \vdash e_1 : \tau_1 \quad \ldots \quad \Gamma \vdash e_n : \tau_n \]

\[\Gamma \vdash \text{new } C(e_1, \ldots, e_n) : C\]

• \(e_1, \ldots, e_n\) have the right types (the types of the fields of \(C\))

```java
class A extends Object {
    int x;
    A(int x){ super(); this.x = x; }
}
```

```java
new A(5) : A
```
Java-Like Language: Types

• Types: int, any class name

\[ \Gamma \vdash x = e.m(e_1, \ldots, e_n); : \text{ok} \]

• \(e\) is an object of a class \(C\)
• \(C\) has a method \(m\)
• args and return of \(m\) are well typed

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

y = objA.getx();
```
Java-Like Language: Types

• Types: int, any class name

\[ \Gamma \vdash e : C \quad (\text{methods}(\Gamma, C) = \cdots, \tau m(\tau_1 x_1, \ldots, \tau_n x_n)) \]

\[ \Gamma \vdash e_1 : \tau_1 \ldots \Gamma \vdash e_n : \tau_n \quad \Gamma(x) = \tau \]

\[ \Gamma \vdash x = e.m(e_1, \ldots e_n); : \text{ok} \]

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

y = objA.getx();

• \( e \) is an object of a class \( C \)
• \( C \) has a method \( m \)
• args and return of \( m \) are well typed
Questions?
Java-Like Language: Declarations

• In a Java-like language, most of the code is in class declarations

\[ CL ::= \text{class } <\text{id}> \text{ extends } <\text{id}> \{ \ T <\text{id}>, \ldots, \ T <\text{id}>; \ K M \ldots M \} \]

• We need to store all the class declarations in the context $\Gamma$, and then check that each method is type-correct
class A extends Object {
    int x;
    int addx(int y){
        return this.x + y;
    }
}

\[
\frac{?}{\Gamma \vdash \tau \, m(\tau_1 \, x_1, \ldots, \tau_n \, x_n)\{c\} : \text{ok}}
\]
Java-Like Language: Declarations

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

\[
\Gamma[x_1 \mapsto \tau_1, \ldots, x_n \mapsto \tau_n] \vdash c : \text{ok}
\]

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

- 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;
    }
}

• 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;
    }
}

• 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;
    }
}

\[
\begin{align*}
\Gamma[\text{this} \mapsto C, x_1 \mapsto \tau_1, \ldots, x_n \mapsto \tau_n] & \vdash c : \text{ok} \\
\Gamma, C & \vdash \tau m(\tau_1 x_1, \ldots, \tau_n x_n)\{c\} : \text{ok}
\end{align*}
\]

• 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
class A extends Object {
    int x;
    A(int x){
        super(); this.x = x;
    }
    int getval(){
        return this.x;
    }
}

class B extends A {
    int y;
    B(int x, int y){
        super(x); this.y = y;
    }
    int getval(){
        return this.x + this.y;
    }
}
class A extends Object {
    int x;
    A(int x) {
        super(); this.x = x;
    }
    int getval() {
        return this.x;
    }
}

class B extends A {
    int y;
    B(int x, int y) {
        super(x); this.y = y;
    }
    int getval(int z) {
        return this.x + this.y + z;
    }
}
Java-Like Language: Declarations

\[
\begin{align*}
\Gamma[\text{this} \mapsto C, x_1 \mapsto \tau_1, \ldots, x_n \mapsto \tau_n] & \vdash c : \text{ok} \\
(\Gamma(C) = \text{class } C \text{ extends } D \{\ldots\}) & \overline{\text{(override_safe}(\tau m(\tau_1 x_1, \ldots, \tau_n x_n), \Gamma, D))} \\
\Gamma, C & \vdash \tau m(\tau_1 x_1, \ldots, \tau_n x_n)\{c\} : \text{ok}
\end{align*}
\]

• 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, in which case we need to check methods w/ same name in the superclass
Questions?
Java-Like Language: Declarations

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

\[
\Gamma, C \vdash K : \text{ok} \quad \Gamma, C \vdash M_1 : \text{ok} \quad \ldots \quad \Gamma, C \vdash M_j : \text{ok}
\]

\[
\Gamma \vdash \text{class } C \text{ extends } D \{ \tau_1 f_1; \ldots; \tau_n f_n; K M_1 \ldots M_j \} : \text{ok}
\]

• A class declaration is well typed if all its methods are well typed.
Questions?
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

\[
\Gamma \vdash e : C \\
\text{(fields}(\Gamma, C) = \ldots, \tau f) \\
\Gamma \vdash e.f : \tau
\]

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

\[
\begin{align*}
\Gamma \vdash e : C \\
\text{(fields}(\Gamma, C) = \ldots, \text{public } \tau f) \\
\hline
\Gamma \vdash e.f : \tau \\
\Gamma \vdash e : C \\
\text{(fields}(\Gamma, C) = \ldots, \text{private } \tau f) \\
\hline
\Gamma, C \vdash e.f : \tau
\end{align*}
\]

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

\[(\Gamma, D \vdash e : C, (\text{fields}(\Gamma, C) = \ldots, \text{public } \tau f)) \quad \Gamma, C \vdash e.f : \tau\]\n
\[(\Gamma, C \vdash e : C, (\text{fields}(\Gamma, C) = \ldots, \text{private } \tau f)) \quad \Gamma, C \vdash e.f : \tau\]\n
\[\Gamma[\text{this }\mapsto C, x_1 \mapsto \tau_1, \ldots] \vdash c : \text{ok}\]

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

class B extends A {
    private int y;

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

class B extends A {
    public int addy(B b){
        this.y = this.y + b.y;
    }
}
Java-Like Language: Access Modifiers

\[\Gamma, D \vdash e : C \]
(fields(\Gamma, C) = \ldots, \text{public } \tau \ f)
\[\Gamma, D \vdash e. f : \tau\]

\[\Gamma, C \vdash e : C \]
(fields(\Gamma, C) = \ldots, \text{private } \tau \ f)
\[\Gamma, C \vdash e. f : \tau\]

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

class B extends A {
    private int y;

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

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

46
Questions?