

- *Class may extend another class or default to extending Object*

$\langle \text{class} \rangle \Rightarrow$
 $\quad \textbf{class } \langle \text{class id} \rangle \langle \text{extend} \rangle \{ \langle \text{class section} \rangle^* \}$
 $\langle \text{extend} \rangle \Rightarrow$
 $\quad \epsilon$
 $\quad | \textbf{extends } \langle \text{class id} \rangle$

- *Sections – private protected public refinements and main*

$\langle \text{class section} \rangle \Rightarrow$
 $\quad \langle \text{refinement} \rangle$
 $\quad | \langle \text{access group} \rangle$
 $\quad | \langle \text{main} \rangle$

- *Refinements are named method dot refinement*

$\langle \text{refinement} \rangle \Rightarrow$
 $\quad \textbf{refinement } \{ \langle \text{refine} \rangle^* \}$
 $\langle \text{refine} \rangle \Rightarrow$
 $\quad \langle \text{return type} \rangle \langle \text{var id} \rangle . \langle \text{var id} \rangle \langle \text{params} \rangle \{ \langle \text{statement} \rangle^* \}$

- *Access groups contain all the members of a class*

$\langle \text{access group} \rangle \Rightarrow$
 $\quad \langle \text{access type} \rangle \{ \langle \text{member} \rangle^* \}$
 $\langle \text{access type} \rangle \Rightarrow$
 $\quad \textbf{private}$
 $\quad | \textbf{protected}$
 $\quad | \textbf{public}$
 $\langle \text{member} \rangle \Rightarrow$
 $\quad \langle \text{var decl} \rangle$
 $\quad | \langle \text{method} \rangle$
 $\quad | \langle \text{init} \rangle$
 $\langle \text{method} \rangle \Rightarrow$
 $\quad \langle \text{return type} \rangle \langle \text{var id} \rangle \langle \text{params} \rangle \{ \langle \text{statement} \rangle^* \}$
 $\langle \text{init} \rangle \Rightarrow$
 $\quad \textbf{init } \langle \text{params} \rangle \{ \langle \text{statement} \rangle^* \}$

- *Main is special – not instance data starts execution*

$\langle \text{main} \rangle \Rightarrow$
 $\quad \textbf{main } (\textbf{String}[] \langle \text{var id} \rangle) \{ \langle \text{statement} \rangle^* \}$

- *Finally the meat and potatoes*

$\langle \text{statement} \rangle \Rightarrow$
 $\quad \langle \text{var decl} \rangle ;$

| $\langle \text{assignment} \rangle$;
 | $\langle \text{conditional} \rangle$
 | $\langle \text{loop} \rangle$
 | $\langle \text{expression} \rangle$;

- *Accessing a variable possibly by index*

$\langle \text{lvalue} \rangle \Rightarrow$
 $\langle \text{var id} \rangle$
 | $\langle \text{lvalue} \rangle [\langle \text{expression} \rangle]$

- *Accessing a member's member's member etc etc*

$\langle \text{lvalue} \rangle \Rightarrow$
 $\langle \text{lvalue} \rangle$
 | $\langle \text{lvalue} \rangle . \langle \text{lvalue} \rangle$

- *Assignment – lvalues receive the results of expressions*

$\langle \text{assignment} \rangle \Rightarrow$
 $\langle \text{lvalue} \rangle := \langle \text{expression} \rangle$

- *Basic control structures*

$\langle \text{conditional} \rangle \Rightarrow$
 if ($\langle \text{expression} \rangle$) { $\langle \text{statement} \rangle^*$ } **else** { $\langle \text{statement} \rangle^*$ }
 $\langle \text{loop} \rangle \Rightarrow$
 while ($\langle \text{expression} \rangle$) { $\langle \text{statement} \rangle^*$ }

- *Anything that can result in a value*

$\langle \text{expression} \rangle \Rightarrow$
 $\langle \text{invocation} \rangle$
 | $\langle \text{field} \rangle$
 | $\langle \text{variable} \rangle$
 | $\langle \text{arithmetic} \rangle$
 | $\langle \text{test} \rangle$
 | $\langle \text{instantiate} \rangle$
 | $\langle \text{refine expr} \rangle$
 | $\langle \text{literal} \rangle$
 | ($\langle \text{expression} \rangle$)
 | **null**

- *Method invocations always have a receiver*

$\langle \text{invocation} \rangle \Rightarrow$
 $\langle \text{expression} \rangle . \langle \text{invoke} \rangle$
 $\langle \text{invoke} \rangle \Rightarrow$

$\langle \text{var id} \rangle ()$
 $| \langle \text{var id} \rangle (\langle \text{args} \rangle)$

- *Field of some foreign object (or this)*

$\langle \text{field} \rangle \Rightarrow$
 $\langle \text{expression} \rangle . \langle \text{variable} \rangle$

- *Variable values can be indexed or not*

$\langle \text{variable} \rangle \Rightarrow$
 $\langle \text{var id} \rangle$
 $| \langle \text{variable} \rangle [\langle \text{expression} \rangle]$

- *Basic arithmetic can and will be done!*

$\langle \text{arithmetic} \rangle \Rightarrow$
 $\langle \text{expression} \rangle \langle \text{bin op} \rangle \langle \text{expression} \rangle$
 $| \langle \text{unary op} \rangle \langle \text{expression} \rangle$
 $\langle \text{bin op} \rangle \Rightarrow$
 $+$
 $| -$
 $| *$
 $| /$
 $| \%$
 $\langle \text{unary op} \rangle \Rightarrow$
 $-$

- *Common boolean predicates*

$\langle \text{test} \rangle \Rightarrow$
 $\langle \text{expression} \rangle \langle \text{bin pred} \rangle \langle \text{expression} \rangle$
 $| \langle \text{unary pred} \rangle \langle \text{expression} \rangle$
 $| \text{refinable} (\langle \text{var id} \rangle)$
 $\langle \text{bin pred} \rangle \Rightarrow$
 and
 $| \text{or}$
 $| \text{xor}$
 $| \text{nand}$
 $| \text{nor}$
 $| <$
 $| <=$
 $| =$
 $| !=$
 $| >=$
 $| >$
 $\langle \text{unary pred} \rangle \Rightarrow$

!

- *Making something*

$\langle \text{instantiate} \rangle \Rightarrow$
 $\langle \text{object instantiate} \rangle$
 | $\langle \text{array instantiate} \rangle$
 $\langle \text{object instantiate} \rangle \Rightarrow$
 new $\langle \text{class id} \rangle$
 | **new** $\langle \text{class id} \rangle$ ($\langle \text{args} \rangle$)
 $\langle \text{array instantiate} \rangle \Rightarrow$
 new $\langle \text{type} \rangle$ [$\langle \text{expression} \rangle$]

- *Refinement takes a specific specialization and notes the required return type*

$\langle \text{refine expr} \rangle \Rightarrow$
 refine $\langle \text{specialize} \rangle$ **to** $\langle \text{type} \rangle$
 $\langle \text{specialize} \rangle \Rightarrow$
 $\langle \text{var id} \rangle$ ()
 | $\langle \text{var id} \rangle$ ($\langle \text{args} \rangle$)

- *Literally necessary*

$\langle \text{literal} \rangle \Rightarrow$
 $\langle \text{int lit} \rangle$
 | $\langle \text{bool lit} \rangle$
 | $\langle \text{float lit} \rangle$
 | $\langle \text{string lit} \rangle$
 $\langle \text{float lit} \rangle \Rightarrow$
 $\langle \text{digit} \rangle^+ . \langle \text{digit} \rangle^+$
 $\langle \text{int lit} \rangle \Rightarrow$
 $\langle \text{digits} \rangle^+$
 $\langle \text{bool lit} \rangle \Rightarrow$
 true
 | **false**
 $\langle \text{string lit} \rangle \Rightarrow$
 “ $\langle \text{string escape seq} \rangle$ ”

- *Params and args are as expected*

$\langle \text{params} \rangle \Rightarrow$
 ($\langle \text{paramlist} \rangle$)
 $\langle \text{paramlist} \rangle \Rightarrow$
 $\langle \text{var decl} \rangle$
 | $\langle \text{paramlist} \rangle$, $\langle \text{var decl} \rangle$
 $\langle \text{args} \rangle \Rightarrow$

$\langle \text{expression} \rangle$
 $| \langle \text{args} \rangle , \langle \text{expression} \rangle$

- *All the basic stuff we've been saving up until now*

$\langle \text{var decl} \rangle \Rightarrow$
 $\langle \text{type} \rangle \langle \text{var id} \rangle$
 $\langle \text{return type} \rangle \Rightarrow$
 \mathbf{unit}
 $| \langle \text{type} \rangle$
 $\langle \text{type} \rangle \Rightarrow$
 $\langle \text{class id} \rangle$
 $| \langle \text{type} \rangle []$
 $\langle \text{class id} \rangle \Rightarrow$
 $\langle \text{upper} \rangle \langle \text{alphanum} \rangle^*$
 $\langle \text{var id} \rangle \Rightarrow$
 $\langle \text{lower} \rangle \langle \text{alphanum} \rangle^*$