Programs

<program> ::= <block> | <program> <block>

Type

Blocks and Commands

<block> ::= { <block\_statements>? }

<block\_statements>::= <block\_statement> | <block\_statements> <block\_statement>

<block\_statement>::=<decl\_statement>|<functional\_statement> | <misc\_statement>

<functional\_statement> ::= empty

<decl\_statement> ::= Type identifier

<misc\_statement> ::= <block> | <empty\_statement> | <expression\_statement> | <reserved\_statement>

<empty\_statement> ::= ;

<reserved\_statement>::=<break\_statement>|<continue\_statement>|<return\_statement>

<break\_statement> ::= break;

<continue\_statement> ::= continue;

<return\_statement> ::= return <expression>;

Expressions

<expression> ::= <assignment\_expression>

<assignment\_expression> ::= <conditional\_expression> | <assignment>

<assignment>::=<left\_hand\_side> <assignment\_operator> <assignment\_expression>

<left\_hand\_side> ::= <expression name>

<assignment operator> ::= =

<conditional\_ expression> ::= <conditional\_or\_expression>

<conditional\_or\_expression>::=<conditional\_and\_expression>|<conditional\_or\_expression> || <conditional\_and\_expression>

<conditional\_and\_expression>::=<equality\_expression> | conditional\_expression> && <equality\_expression>

<equality\_expression> ::= <relational\_expression> | <equality\_expression> == <relational\_expression> | <equality\_expression> != <relational\_expression>

<relational\_expression> ::= <additive\_expression> | <relational\_expression> > <additive\_expression> | <relational\_expression> <= <additve\_expression> | <relational\_expression> > <additive\_expression> | <relational\_expression> >= <additive\_expression>

<additive\_expression> ::= <multiplicative\_expression> | <additive\_expression> +<multiplicative\_expression> | <additive\_expression> - <multiplicative\_expression>

<multiplicative expression>::= <unary\_expression> | <multiplicative\_expression> \* <unary\_expression> | <multiplicative\_expression> / <unary\_expression> | <multiplicative\_expression> % <unary\_expression>

Lexical

Tokens

<keyword> ::= abstract | boolean | break | byte | case | catch | char | class | const | continue | default | do | double | else | extends | final | finally | float | for | goto | if | implements |import | instanceof | int | interface | long | native | new | package | private | protected | public | return | short | static | super | switch | synchronized | this | throw | throws |transient | try | void | volatile | while