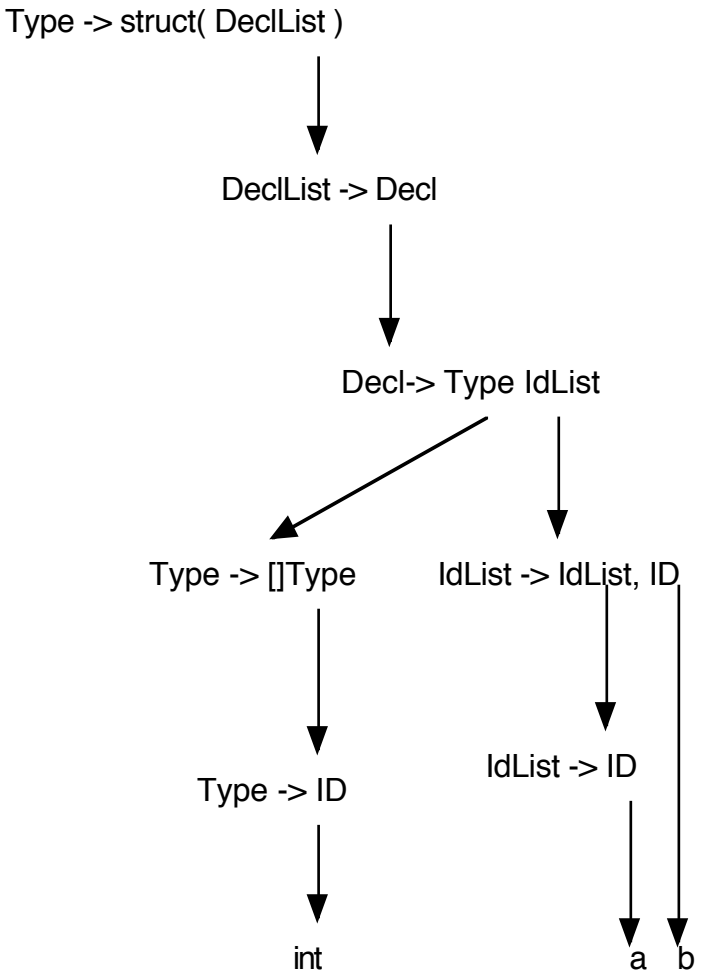
1. Bottom Up LALR(1) Parsing

1(a)

Stack							Token	Action
$0	□	□	□	□	□	□	struct	shift
$0	struct 2	□	□	□	□	□	(	shift
$0	struct 2	(7	□	□	□	□	[	shift
$0	struct 2	(7	[4	□	□	□	]	shift
$0	struct 2	(7	[4	] 5	□	□	ID int	shift
$0	struct 2	(7	[4	] 5	ID 3	□	ID a	reduce
$0	struct 2	(7	[4	] 5	Type 6	□	□	reduce
$0	struct 2	(7	Type 9	□	□	□	□	reduce
$0	struct 2	(7	Type 9	ID 11	□	□	,	reduce
$0	struct 2	(7	Type 9	IdList 12	□	□	□	shift
$0	struct 2	(7	Type 9	IdList 12	, 13	□	ID b	shift
$0	struct 2	(7	Type 9	IdList 12	, 13	ID 14	)	reduce
$0	struct 2	(7	Type 9	IdList 12	□	□	□	reduce
$0	struct 2	(7	Decl 10	□	□	□	□	reduce
$0	struct 2	(7	DeclList 8	□	□	□	□	shift
$0	struct 2	(7	DeclList 8	) 16	□	□	$	reduce
$0	Type 1	□	□	□	□	□	□	shift
$0	Type 1	$ 18	□	□	□	□	□	Accept

1(b)



1(c)

lalr_state [7]: {
  [Type ::= STRUCT LEFT (*) DeclList RIGHT , {EOF IDENT }]
  [DeclList ::= (*) Decl , {RIGHT SEMICOLON }]
  [DeclList ::= (*) DeclList SEMICOLON Decl , {RIGHT SEMICOLON }]
  [Decl ::= (*) Type IdentList , {RIGHT SEMICOLON }]
  [Type ::= (*) LEFTSQ RIGHTSQ Type , {IDENT }]
  [Type ::= (*) IDENT , {IDENT }]
  [Type ::= (*) STRUCT LEFT DeclList RIGHT , {IDENT }]
}

2. Write a grammar definition

```
terminal
     LEFT, RIGHT, LEFTBRACE, RIGHTBRACE, COMMA, DOTDOT, UNION, INTERSECT;

terminal
     String INTCONST;

non terminal
     Set1, Set2, Set3, BasicSet, ComponentList, Component;

start with Set1;

Set1::=
          Set1 UNION Set2
     |
          Set2
     ;

Set2::=
          Set2 INTERSECT Set3
     |
          Set3
     ;

Set3::=
          LEFT Set1 RIGHT
     |
          BasicSet
     ;

BasicSet::=
          LEFTBRACE RIGHTBRACE
     |
          LEFTBRACE INTCONST DOTDOT RIGHTBRACE
     |
          LEFTBRACE ComponentList RIGHTBRACE
     |
          LEFTBRACE ComponentList COMMA INTCONST DOTDOT RIGHTBRACE
     ;

ComponentList::=
          Component
     |
          ComponentList COMMA Component
     ;

Component::=
          INTCONST
     |
          INTCONST DOTDOT INTCONST
     ;
```

3. Interpretation

3 (a)

```
0       n = 3
10      for i = 1 to n
20      print i
30      next i
40
```

Need an environment, containing the mapping of identifiers to values, reference to next line, etc.

Need a control stack containing saved information when enter control structure, such as "for" loop.

For "for" loops, push information on the stack:

- Indication is for loop.
- Name of for variable.
- Final value.
- Reference to next statement after the for statement, so know where to branch back to.

For "for" statement, evaluate initial and final value expressions. Assign initial value to variable. Save required info on the stack.

For "next" statement, check have "for" info on the stack, with the same "for" variable. Increment for variable. Compare with final value. If variable <= final value, set next line to line in the for info, else pop for info off the stack.

e.g., push on (FORINFO, "i", 3, line 20) when enter for loop. Set i = 1.

Modify i each time through the loop, until has value of 4, when pop info off the control stack.

3(b)

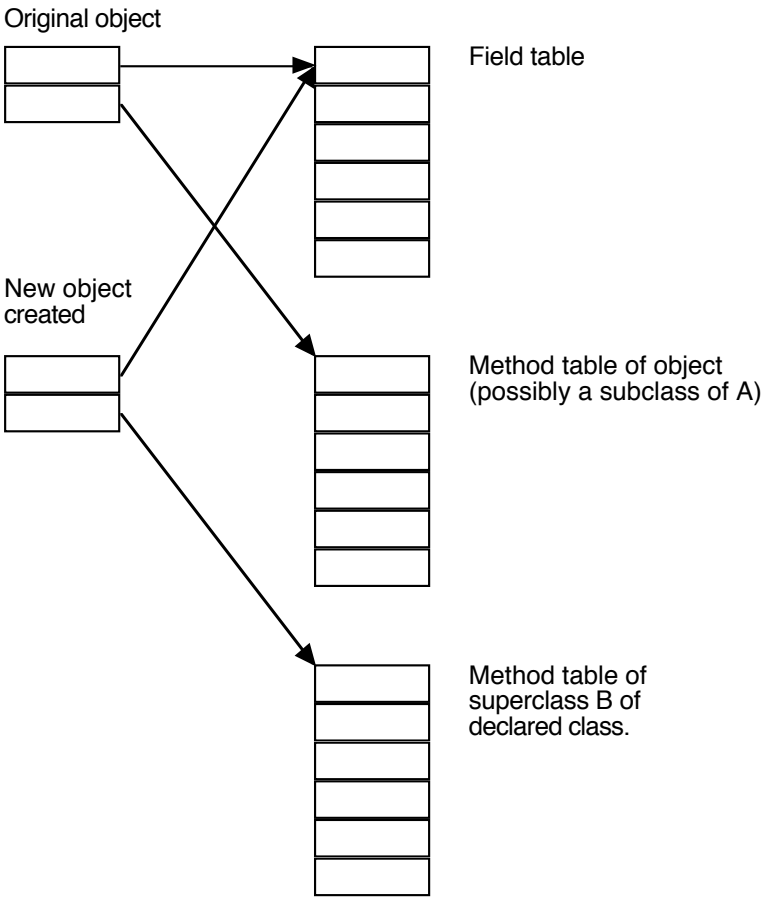
have notation object.className

e.g.,

```
class A {
        void method() { ... }
        }
class B extends A {
        void method() {this.A.method(); ... }
        }
```

so in the above, we are invoking the method declared in the class A, for an object of type B.

This generates a new object with the same fields as the specified object, but method table of the specified class, which must be a superclass of the declared type of object.

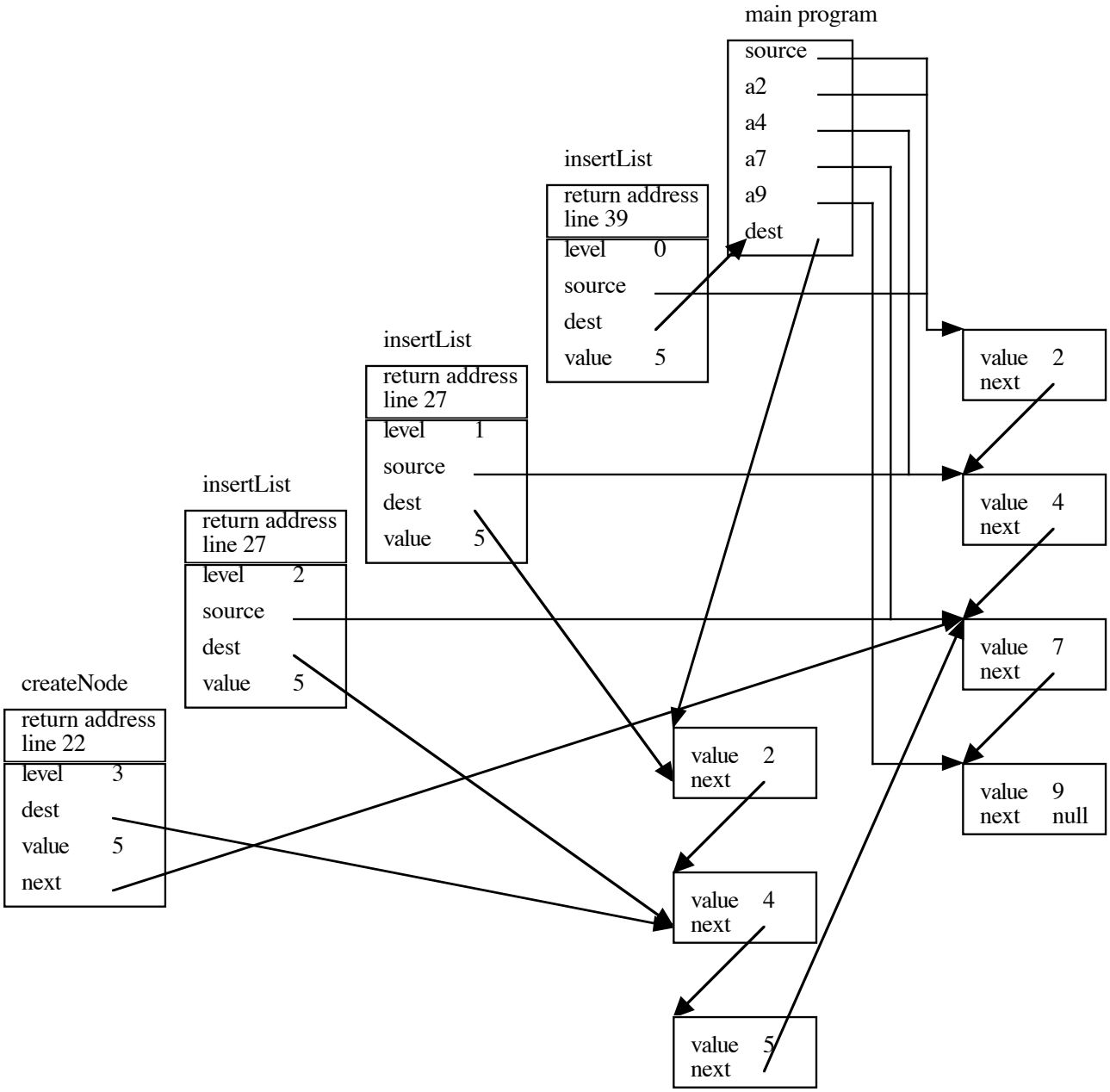

Original object
Field table
New object
created
Method table of object
(possibly a subclass of A)
Method table of
superclass B of
declared class.

4. Show the run time stack.

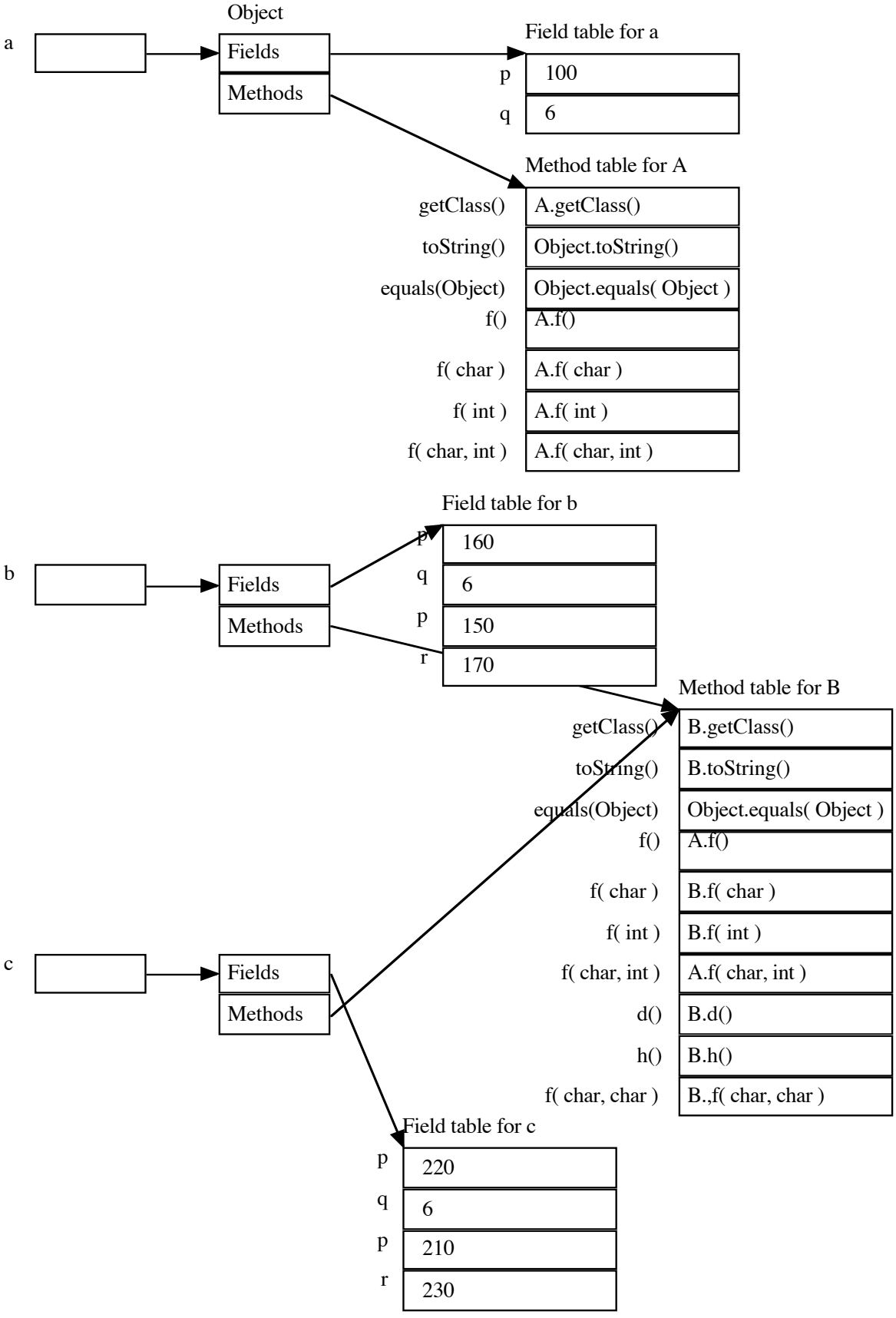
Output:

{ 2, 4, 7, 9 }

{ 2, 4, 5, 7, 9 }



5. Implementation of object oriented languages



a.p = 100

a.q = 6

b.p = 150

b.q = 6

b.r = 170

c.p = 220

c.q = 6

c = B

b.f('B') = B.f('B')

c.f('B') = B.f('B')

b.f('A', 'B') = B.f('A', 'B')

c.f('A', 'B') = A.f('A', 66)

c.g('A') = A.g(65)

6. Code generation

6(a)

```
{
  ldq $t0, a_3($fp);
  ldq $t1, b_4($fp);
  addq $t0, $t1, $t0;
  ldq $t1, c_5($fp);
  stq $t0, 0($t1);
}
```

6(b)

```
{
  lda $sp, -invoc.act3($sp);
  ldq $t0, INST.fields($ip);
  ldq $t0, main.subClass.B_3.object.field.x_1($t0);
  stq $t0, invoc.act0($sp);
  ldiq $t0, 1;
  stq $t0, invoc.act1($sp);
  ldq $t0, INST.fields($ip);
  lda $t0, main.subClass.B_3.object.field.y_2($t0);
  stq $t0, invoc.act2($sp);
  ldq $t0, INST.fields($ip);
  ldq $t0, main.subClass.B_3.object.field.a_0($t0);
  mov $t0, $nip;
  ldq $at, INST.methods($nip);
  ldq $pv, main.subClass.A_2.object.method.f_3($at);
  jsr $ra, ($pv);
  lda $sp, +invoc.act3($sp);
}
```