1. JFlex [10 Marks]

Indicate at least 10 different kinds of errors in the following fragment of JFlex code. ("..." just means omitted code). Assume line breaks **are** syntactically important, but spaces **are not**. Assume comments **can** be nested.

Should not have [] in newline.

Missed out [A-Za-z] for identifier.

Should be 0-9 in [1-9]+

(,),. need quotes.

Should not return symbol for start of comment.

Should change into comment state for start of comment.

Should not match "*/" in normal mode.

{ident}, not ident, {number}, not number.

Should return token for newline in normal.

Should not have quotes in ".".

Should return token for lex error.

Should have if statement in "*/" in comment state.

Should have lineCount++ for newline in comment.

Should not return token for . in comment.

(14 errors)

2. Bottom Up LALR(1) Parsing [23 Marks]

Consider the CUP grammar in the **Appendix For Question 2**. Note that some rules are left recursive, while other rules are right recursive. Also note the rule for "ElementListOpt" that expands to empty.

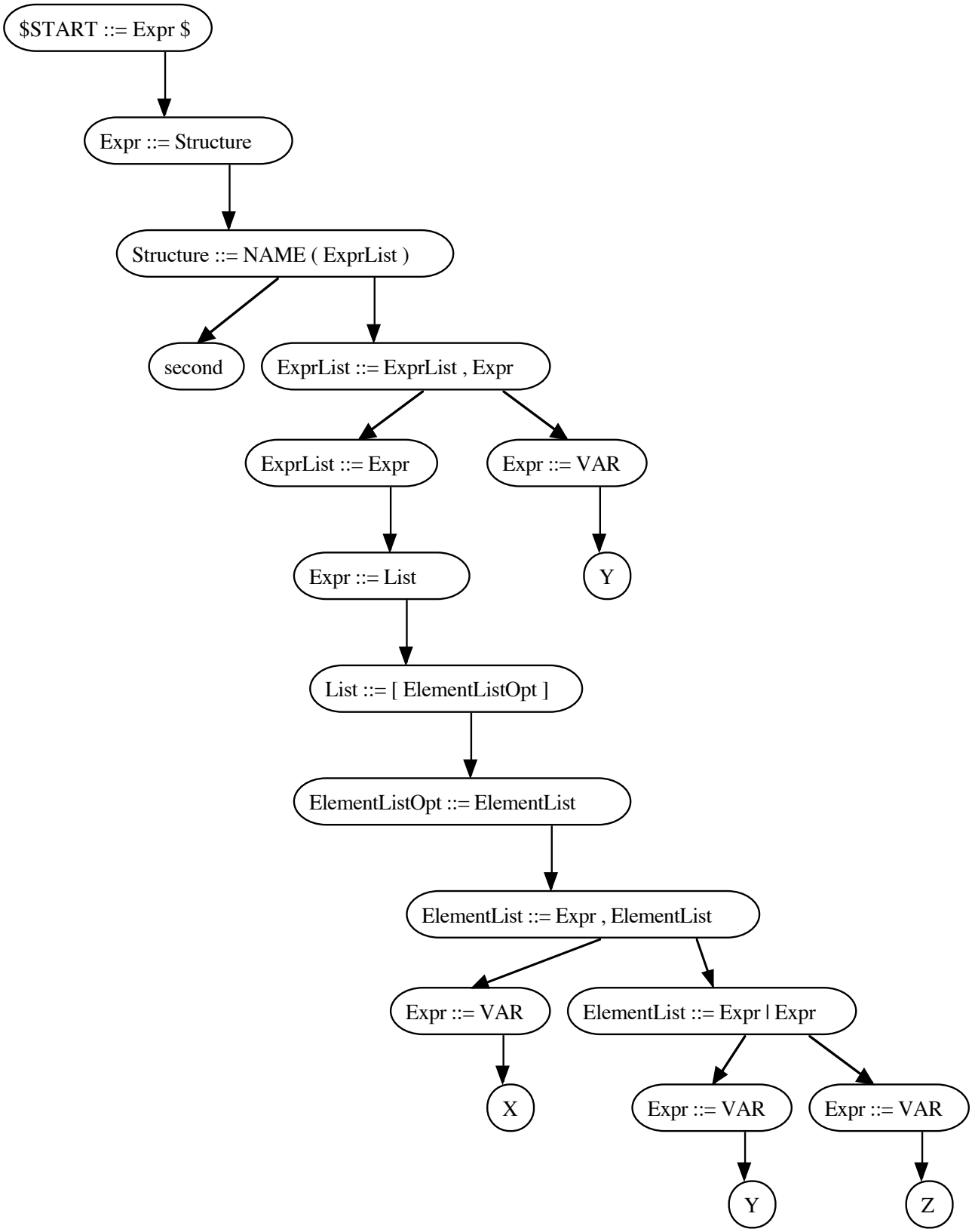
(a) Using the information provided in the appendix, perform a shift-reduce LALR(1) parse of the input

```
second( [ X, Y | Z ], Y )
```

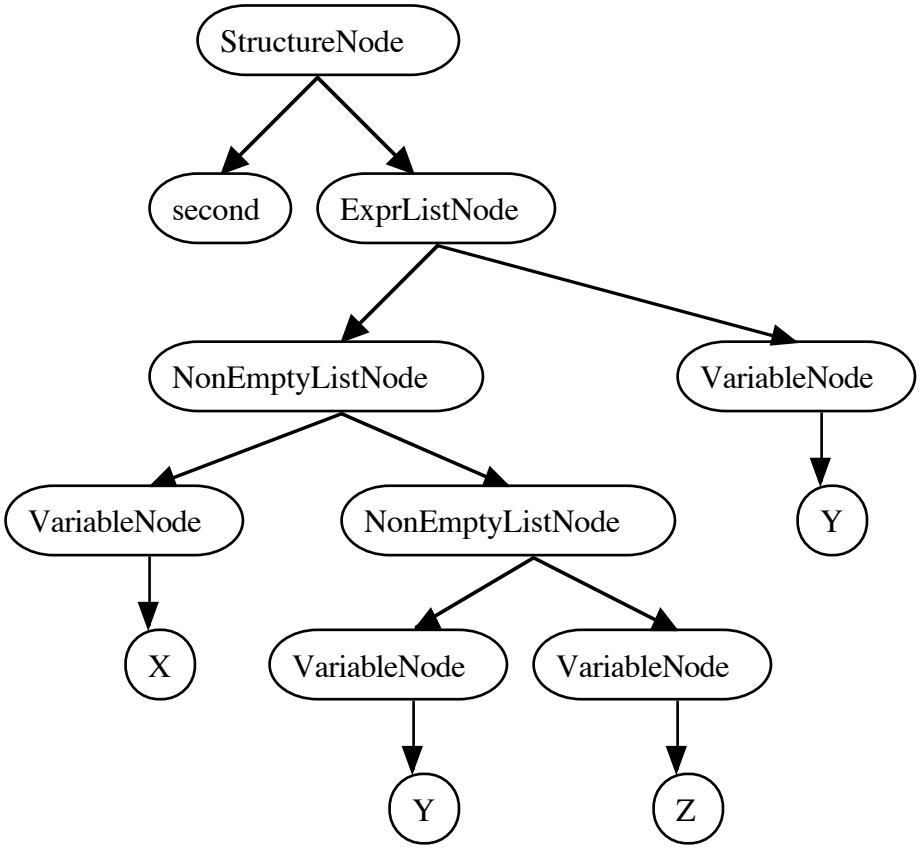
Assume "`second`" matches `NAME`; "`X`", "`Y `", "`Z`", match `VARIABLE`, and "`,`", "`(`", "`)`", "`[`", "`]`" and "`|`" match `COMMA`, `LEFT`, `RIGHT`, `LEFTSQ`, `RIGHTSQ` and `BAR`, respectively.

Stack									Token	Action	
$0									NME sec	Shift	
$0	NME 5								(	Shift	
$0	NME 5	(16							[	Shift	
$0	NME 5	(16	[6						VAR X	Shift	
$0	NME 5	(16	[6	VAR 7					,	Reduce	E ::= VAR
$0	NME 5	(16	[6	E 9						Shift	
$0	NME 5	(16	[6	E 9	, 11				VAR Y	Shift	
$0	NME 5	(16	[6	E 9	, 11	VAR 7			|	Reduce	E ::= VAR
$0	NME 5	(16	[6	E 9	, 11	E 9				Shift	
$0	NME 5	(16	[6	E 9	, 11	E 9	| 12		VAR Z	Shift	
$0	NME 5	(16	[6	E 9	, 11	E 9	| 12	VAR 7	]	Reduce	E ::= VAR
$0	NME 5	(16	[6	E 9	, 11	E 9	| 12	E 13		Reduce	EIL ::= E | E
$0	NME 5	(16	[6	E 9	, 11	EL 14				Reduce	EIL ::= E , EIL
$0	NME 5	(16	[6	EIL 10						Reduce	EILO ::= EIL
$0	NME 5	(16	[6	EILO 8						Shift	
$0	NME 5	(16	[6	EILO 8	] 15				,	Reduce	L ::= [EILO]
$0	NME 5	(16	L 2							Reduce	E ::= L
$0	NME 5	(16	E 18							Reduce	ExL ::= E
$0	NME 5	(16	ExL 17							Shift	
$0	NME 5	(16	ExL 17	, 19					VAR Y	Shift	
$0	NME 5	(16	ExL 17	, 19	VAR 7				)	Reduce	E ::= VAR
$0	NME 5	(16	ExL 17	, 19	E 21					Reduce	ExL ::= ExL , E
$0	NME 5	(16	ExL 17							Shift	
$0	NME 5	(16	ExL 17	) 20					$	Reduce	S ::= NME(ExL)
$0	S 1									Reduce	E ::= S
$0	E 3									Shift	
$0	E 3	$ 22							$	Reduce	$Start ::= E $
$0	$Start -1									Accept	

(b) Draw the parse tree corresponding to the grammar rules used to parse this input



(c) Draw the abstract syntax tree built by the actions associated with the grammar rules.



3. Write a grammar for class declarations [15 Marks]

```
ClassDecl::=
          "class" IDENT ExtendsOpt ImplementsOpt "{" MemberDeclList "}"
     ;
ExtendsOpt ::=
     |
          "extends" IDENT
     ;
ImplementsOpt::=
     |
          "implements" IdentList
     ;
IdentList::=
          IDENT
     |
          IdentList "," IDENT
     ;
MemberDeclList::=
     |
          MemberDeclList MemberDecl
     ;
MemberDecl::=
          VarDecl
     |
          MethodDecl
     ;
VarDecl::=
          Type IdentList ";"
     ;
MethodDecl::=
          Type IDENT "(" VarDeclList ")" "{" LocalDeclStmtList "}"
     ;
VarDeclList::=
     |
          VarDeclList VarDecl
     ;
LocalDeclStmtList::=
     |
          LocalDeclStmtList LocalDeclStmt
     ;
LocalDeclStmt::=
          VarDecl
     |
          Stmt
     ;
Type::=
          IDENT
     |
          "[" "]" TYPE
     ;
```

4. Write a BHP function **[7 Marks]**

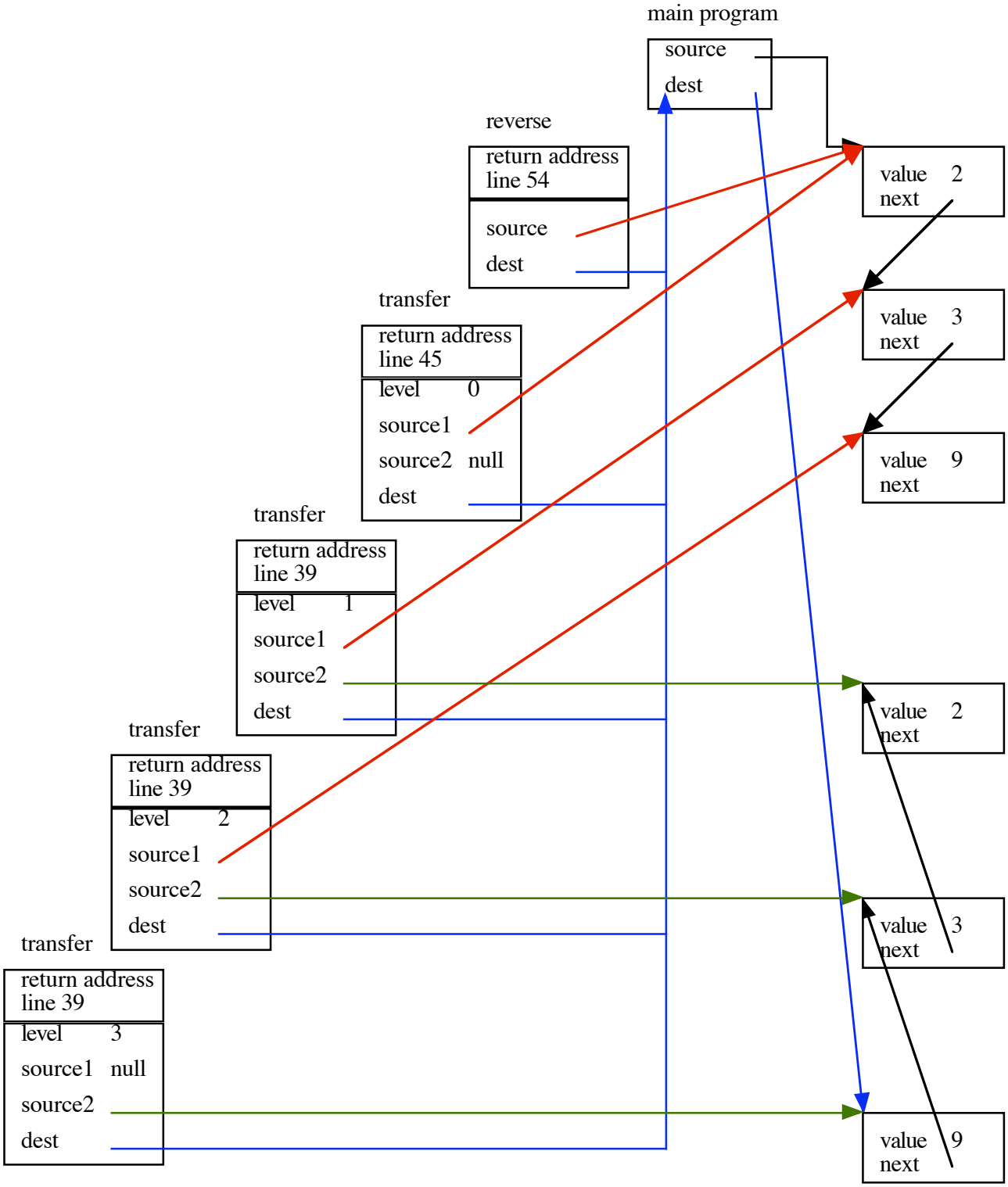
Consider the BHP language of Assignment 2.

Write a BHP function to clone a compound object.

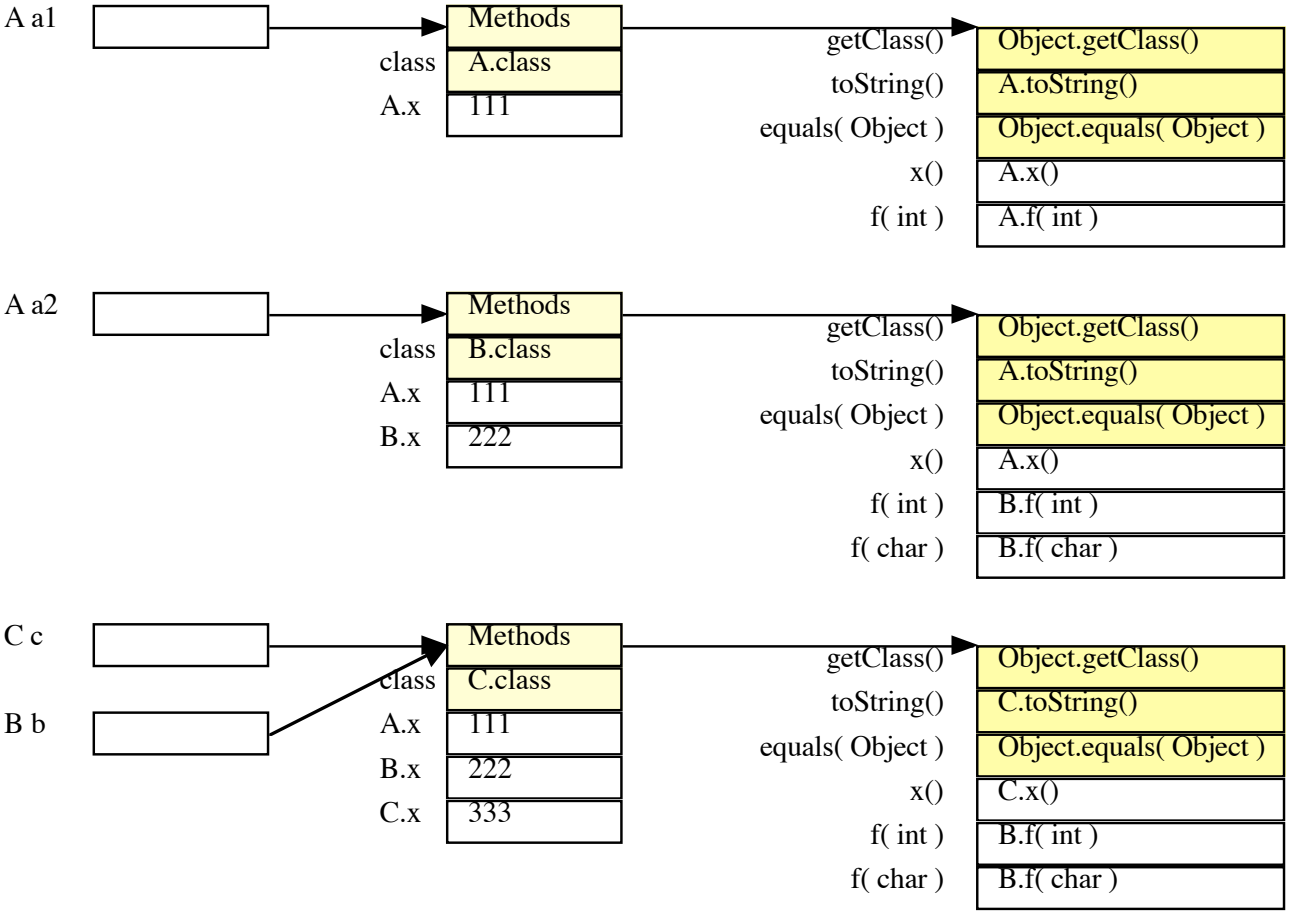
```
function clone( $oldEnv )
     begin
          var $newEnv = new();
          var $key;
          $newEnv.0 = $oldEnv.0;
          $newEnv.# = $oldEnv.#;
          iterate $key through $oldEnv do
               var $newEnv[ $key ] = $oldEnv[ $key ];
          end
          return $newEnv;
     end
```

5. Show the run time stack for a B-- program **[14 Marks]**

```
{ 2, 3, 9 }
{ 9, 3, 2 }
```



6. Implementation of object oriented languages [16 Marks]




```
Constructor A
Constructor A
Constructor B
Constructor A
Constructor B
Constructor C
a1.getClass() = class A
a2.getClass() = class B
b.getClass() = class C
c.getClass() = class C
a1 = A
a2 = A
b = C
c = C
a1.x = 111
a2.x = 111
b.x = 222
c.x = 333
a1.x() = 111
a2.x() = 111
b.x() = 333
c.x() = 333
a1.f( 'a' ) = A.f( 97 )
a2.f( 'a' ) = B.f( 97 )
b.f( 'a' ) = B.f( 'a' )
c.f( 'a' ) = B.f( 'a' )
```

7. Code generation [15 Marks]

(a)

```
public a_10:
        space 0x18;
```

(b)

```
            const {
            public enter:
                quad main.A_0.methodCode.get_0.enter;
                quad main.A_0.methodCode.set_8.enter;
            } const
```

(c)

```
                {
                    ldiq $t0, 0x3;
                    stq $t0, field.x_8($ip);
                }
                {
                    ldiq $t0, 0x4;
                    stq $t0, field.y_10($ip);
                }
```

(d)

```
                {
                    ldiq $t0, main.a_10;
                    lda $nip, ($t0);
                    ldiq $at, main.A_0.methodTable.enter;
                    stq $at, ($nip);
                    bsr main.A_0.initInstance.enter;
                }
```

(e)

```
                {
                    lda $sp, -invoc.act1($sp);
                    ldiq $t0, 0x5;
                    stq $t0, invoc.act0($sp);
                    ldiq $t0, main.a_10;
                    mov $t0, $nip;
                    ldq $at, main.A_0.field.methodTablePtr($nip);
                    ldq $pv, main.A_0.method.set_8($at);
                    jsr;
                    lda $sp, +invoc.act1($sp);
```