```
public class Blah {
    static interface I {
        int m(int x);
    }
    static class A implements I {
        private int y;
        public void init(int a) { y = a; }
        public int m(int x) {
            return y + x;
        }
    }
    static class B implements I {
        private int y;
        public void create(int b) { y = b; }
        public int m(int x) {
            return y * x;
        }
    }

    public static void main(String[] args) {
        I i;
        int j, z;
        j = 0;
        while (j < 5) {
            if (j - 2 * (j / 2) == 0) {
                i = new A();
                ((A) i).init(j);
            }
            else {
                i = new B();
                ((B) i).create(j);
            }
            System.out.println(i.m(j + 3));
            j = j + 1;
        }
    }
}
```

OJay

Program	→	`public class` ID '{' *Class** *GlobalDeclarations* `public static void main ( String[] args )` '{' *Declarations Statements* '}' '}'
GlobalDeclarations	→	*GlobalDeclartion* *
GlobalDeclaration	→	`static` *Type Identifiers* `;`
Class	→	`static class` ID '{' *Field*Method**'}'
Field	→	`private` *Type Identifiers* `;`
Method	→	`public` *ReturnType* ID `(` *FormParams*? `)` '{' *Declarations Statements* '}'
Declarations	→	*Declartion* *
Declaration	→	*Type Identifiers* `;`
Type	→	*PrimitiveType* | `ID`
PrimitiveType	→	`int` | `boolean`
Identifiers	→	ID { `,` ID } *
Statements	→	*Statement* *
Statement	→	`;` | *Block* | *Assignment* | *IfStatement* | *WhileStatement* | *PrintStatement* | *InvocationStatement* | *ReturnStatement*
Block	→	'{' *Declarations Statements* '}'
Assignment	→	ID `=` *Expression* `;`
IfStatement	→	`if (` *Expression* `)` *Statement* { `else` *Statement* }?
WhileStatement	→	`while (` *Expression* `)` *Statement*
PrintStatement	→	`System.out.println (` *Expression* `) ;`
InvocationStatement	→	*Identifier* `.` *Identifier* `(` *ActParams* `) ;`
ActParams	→	*Expression* { `,` *Expression* } *
ReturnStatement	→	return *Expression*? `;`
Expression	→	*Conjunction* { `||` *Conjunction* }*
Conjunction	→	*Relation* { `&&` *Relation* }*
Relation	→	*Addition* { *RelOp Addition* }?
RelOp	→	`<` | `<=` | `>` | `>=` | `==` | `!=`
Addition	→	*Term* { *AddOp Term* } *
AddOp	→	`+` | `-`
Term	→	*Negation* { *MulOp Negation* } *
MulOp	→	'*' | `/`
Negation	→	*NegOp*? *Factor*
NegOp	→	`!` | `-`
Factor	→	ID | LITERAL | `(` *Expression* `)` | *Invocation* | *Instantiation*
Invocation	→	*Identifier* `.` *Identifier* `(` *ActParams* `)`
Instantiation	→	`new ID()`
ReturnType	→	*Type* | `void`
FormParams	→	*FormParam* { `,` *FormParam* } *
FormParam	→	*Type* ID

POJay

Program → `public class` ID '{' *Interface*`*` *Class*`*`*GlobalDeclarations*
`public static void main ( String[] args )`
'{' *Declarations Statements* '}' '}'

Interface → `static interface` ID '{' *Signature*`*` '}'

Signature → *Type* ID `(` *FormParams*`?` `)` `;`

GlobalDeclarations → *GlobalDeclartion* `*`

GlobalDeclaration → `static` *Type Identifiers* `;`

Class → `static class` ID `implements` ID'{' *Field*`*`*Method*`*`'}'

Field → `private` *Type Identifiers* `;`

Method → `public` *ReturnType* ID `(` *FormParams*`?` `)` '{' *Declarations Statements* '}'

Declarations → *Declartion* `*`

Declaration → *Type Identifiers* `;`

Type → *PrimitiveType* | `ID`

PrimitiveType → `int` | `boolean`

Identifiers → ID { `,` ID } `*`

Statements → *Statement* `*`

Statement → `;` | *Block* | *Assignment* | *IfStatement* | *WhileStatement* | *PrintStatement* | *InvocationStatement* | *ReturnStatement*

Block → '{' *Declarations Statements* '}'

Assignment → ID `=` *Expression* `;`

IfStatement → `if` `(` *Expression* `)` *Statement* { `else` *Statement* }?

WhileStatement → `while` `(` *Expression* `)` *Statement*

PrintStatement → `System.out.println` `(` *Expression* `)` `;`

InvocationStatement → *Identifier* `.` *Identifier* `(` *ActParams* `)` `;`

ActParams → *Expression* { `,` *Expression* } `*`

ReturnStatement → return *Expression*`?` `;`

Expression → *Conjunction* { `||` *Conjunction* }`*`

Conjunction → *Relation* { `&&` *Relation* }`*`

Relation → *Addition* { *RelOp Addition* }?

RelOp → `<` | `<=` | `>` | `>=` | `==` | `!=`

Addition → *Term* { *AddOp Term* } `*`

AddOp → `+` | `-`

Term → *Negation* { *MulOp Negation* } `*`

MulOp → '`*`' | `/`

Negation → *NegOp*`?` *Factor*

NegOp → `!` | `-`

Factor → ID | LITERAL | `(` *Expression* `)` |
Invocation | *Instantiation* | *Cast*

Invocation → *Identifier* `.` *Identifier* `(` *ActParams* `)`

Instantiation → `new ID()`

Cast → `(` ID `)` *Factor*

ReturnType → *Type* | `void`

FormParams → *FormParam* { `,` *FormParam* } `*`

FormParam → *Type* ID