Solution Exercise 1

Prof. Dr. Bernhard Rumpe, Oliver Kautz, Christian Kirchhof
Software Engineering
RWTH Aachen

http://www.se-rwth.de/

Exercise 1.1: Develop your own grammar

- *In this task you will develop your own grammar “Appointments”.*
- *There are 4 example models in* “src/main/resources/examples” *that are valid models of the grammar you are asked to develop in this task.*

1. Extend the given grammar such that the given example appointments can be parsed. Keep in mind that similar appointment models, e.g. different time slots, other participants etc. must be supported as well. For this only use the Nonterminals provided, i.e. do not create new nonterminal productions but develop the bodies of the given nonterminal productions (und thus their composition).

2. Extend the JUnit Test AppointmentsParserTest such that it comprises one testcase per example ensuring these examples are models of your developed language. For each model instantiate the parser, parse one of the models and check whether the resulting Optional is present.

Exercise 1.1: Develop your own grammar

MG

```
grammar Appointments extends de.monticore.literals.MCCommonLiterals {
  start Appointment;

  Appointment = ;

  Start = ;
  End = ;
  Date = ;
  Time = ;
  Break = ;
}
```

Appointments

```
appointment "Kuchen" {
  start: 01/10/19 13:00
  end:   13:30
  participants: "Peter"
  once
}
```

Exercise 1.1: Develop your own grammar

MG

```
grammar Appointments extends de.monticore.literals.MCCommonLiterals {
  start Appointment;

  Appointment = "appointment" name:String "{„
                "}";
  Start = ;
  End = ;
  Date = ;
  Time = ;
  Break = ;
}
```

Appointments

```
appointment "Kuchen" {
  start: 01/10/19 13:00
  end:   13:30
  participants: "Peter"
  once
}
```

Exercise 1.1: Develop your own grammar

MG

```
grammar Appointments extends de.monticore.literals.MCCommonLiterals {
  start Appointment;

  Appointment = "appointment" name:String "{"
                Start
                "}";
  Start = "start" ":" Date? Time;
  End = ;
  Date = day:NatLiteral "/" month:NatLiteral  "/" year:NatLiteral;
  Time = hour:NatLiteral ":" min:NatLiteral;
  Break = ;
}
```

Appointments

```
appointment "Kuchen" {
  start: 01/10/19 13:00
  end:   13:30
  participants: "Peter"
  once
}
```

Exercise 1.1: Develop your own grammar

MG

```
grammar Appointments extends de.monticore.literals.MCCommonLiterals {
  start Appointment;

  Appointment = "appointment" name:String "{"
                Start
                End
                "}";
  Start = "start" ":" Date? Time;
  End = "end" ":" Date? Time;
  Date = day:NatLiteral "/" month:NatLiteral  "/" year:NatLiteral;
  Time = hour:NatLiteral ":" min:NatLiteral;
  Break = ;
}
```

Appointments

```
appointment "Kuchen" {
  start: 01/10/19 13:00
  end:   13:30
  participants: "Peter"
  once
}
```

Exercise 1.1: Develop your own grammar

MG

```
grammar Appointments extends de.monticore.literals.MCCommonLiterals {
  start Appointment;

  Appointment = "appointment" name:String "{"
                Start
                End
                "}";
  Start = "start" ":" Date? Time;
  End = "end" ":" Date? Time;
  Date = day:NatLiteral "/" month:NatLiteral  "/" year:NatLiteral;
  Time = hour:NatLiteral ":" min:NatLiteral;
  Break = ;
}
```

Appointments

```
appointment "Kuchen" {
  start: 01/10/19 13:00
  end:   13:30
  participants: "Peter"
  once
}
```

Exercise 1.1: Develop your own grammar

MG

```
grammar Appointments extends de.monticore.literals.MCCommonLiterals {
  start Appointment;

  Appointment = "appointment" name:String "{"
                Start
                End
                "participants" ":" participant:String("," participant:String)*
                "}";
  Start = "start" ":" Date? Time;
  End = "end" ":" Date? Time;
  Date = day:NatLiteral "/" month:NatLiteral  "/" year:NatLiteral;
  Time = hour:NatLiteral ":" min:NatLiteral;
  Break = ;
}
```

Appointments

```
appointment "Kuchen" {
  start: 01/10/19 13:00
  end:   13:30
  participants: "Peter"
  once
}
```

Exercise 1.1: Develop your own grammar

MG

```
grammar Appointments extends de.monticore.literals.MCCommonLiterals {
  start Appointment;

  Appointment = "appointment" name:String "{"
                Start
                End
                "participants" ":" participant:String("," participant:String)*
                repetition:["once"]?
                "}";
  Start = "start" ":" Date? Time;
  End = "end" ":" Date? Time;
  Date = day:NatLiteral "/" month:NatLiteral  "/" year:NatLiteral;
  Time = hour:NatLiteral ":" min:NatLiteral;
  Break = ;
}
```

Appointments

```
appointment "Kuchen" {
  start: 01/10/19 13:00
  end:   13:30
  participants: "Peter"
  once
}
```

Exercise 1.1: Develop your own grammar

MG

```
grammar Appointments extends de.monticore.literals.MCCommonLiterals {
  start Appointment;

  Appointment = "appointment" name:String "{"
                Start
                End
                "participants" ":" participant:String("," participant:String)*
                repetition:["once"|"weekly"]?
                "}";
  Start = "start" ":" Date? Time;
  End = "end" ":" Date? Time;
  Date = day:NatLiteral "/" month:NatLiteral  "/" year:NatLiteral;
  Time = hour:NatLiteral ":" min:NatLiteral;
  Break = ;
}
```

Appointments

```
appointment "Konferenz" {
  start : 01/10/19 8:15
  end   : 02/10/19 16:15
  participants: "Peter Stein", "Paula Schmidt", "Tina Berg (Personal)"
  weekly
}
```

Exercise 1.1: Develop your own grammar

MG

```
grammar Appointments extends de.monticore.literals.MCCommonLiterals {
  start Appointment;

  Appointment = "appointment" name:String "{"
                Start
                End
                "participants" ":" participant:String("," participant:String)*
                repetition:["once"|"weekly"|"daily"]?
                "}";
  Start = "start" ":" Date? Time;
  End = "end" ":" Date? Time;
  Date = day:NatLiteral "/" month:NatLiteral  "/" year:NatLiteral;
  Time = hour:NatLiteral ":" min:NatLiteral;
  Break = ;
}
```

Appointments

```
appointment "Mittagspause" {
  start: 01/10/19 8:15
  end:   9:15
  participants: "Peter Stein", "Paula Schmidt"
  daily
}
```

Exercise 1.1: Develop your own grammar

MG

```
grammar Appointments extends de.monticore.literals.MCCommonLiterals {
  start Appointment;

  Appointment = "appointment" name:String "{"
                Start
                End
                "participants" ":" participant:String("," participant:String)*
                repetition:["once"|"weekly"|"daily"]?
                Break?
                "}";
  Start = "start" ":" Date? Time;
  End = "end" ":" Date? Time;
  Date = day:NatLiteral "/" month:NatLiteral  "/" year:NatLiteral;
  Time = hour:NatLiteral ":" min:NatLiteral;
  Break = "break" "{" Start End "}";
}
```

Appointments

```
appointment "Workshop" {
  /* ... */
  break {
    start : 11:30
    end   : 12:00
  }
}
```

Exercise 1.1: Develop your own grammar

- *In this task you will develop your own grammar “Appointments”.*
- *There are 4 example models in* “src/main/resources/examples” *that are valid models of the grammar you are asked to develop in this task.*

1. Extend the given grammar such that the given example appointments can be parsed. Keep in mind that similar appointment models, e.g. different time slots, other participants etc. must be supported as well. For this only use the Nonterminals provided, i.e. do not create new nonterminal productions but develop the bodies of the given nonterminal productions (und thus their composition).

2. Extend the JUnit Test AppointmentsParserTest such that it comprises one testcase per example ensuring these examples are models of your developed language. For each model instantiate the parser, parse one of the models and check whether the resulting Optional is present.

Exercise 1.1: Develop your own grammar

Java

```java
public class AppointmentParserTest { /* ... */
    @Nested class Parse {
    @ParameterizedTest
    @CsvSource({ "replaceMe" })
    void shouldParseValidModel(String modelFilePath) {
      //given
      AppointmentsParser p = new AppointmentsParser();

      //when
      //TODO: write me

      //then
      //TODO: write me
    }
  }
}
```

Exercise 1.1: Develop your own grammar

Java

```java
public class AppointmentParserTest { /* ... */
  @Nested class Parse {
  @ParameterizedTest
  @CsvSource({"src/main/resources/example/Example1.ap",
              "src/main/resources/example/Example2.ap",
              "src/main/resources/example/Example3.ap",
              "src/main/resources/example/Example4.ap" })
  void shouldParseValidModel(String modelFilePath) throws IOException {
      //given
      AppointmentsParser p = new AppointmentsParser();

      //when
      Optional<ASTAppointment> result = p.parse(modelFilePath);

      //then
      assertThat(result).isPresent();
    }
  }
}
```

Exercise 1.2: Improve your grammar

- Now that you have a working grammar it's time to flexibilize it!

- Introduce flexibility in modelling appointments by using an interface nonterminal for the appointment's elements such as break, participants and repetition.
- For this purpose, create a new grammar AppointmentsFlexibilized.
- Adapt the existing nonterminal productions to use/implement the interface nonterminal.

- Create a JUnit test AppointmentsFlexibilizedParserTest that test the new parser for the models in
  - "src/main/resources/example" as well as the model in
  - "src/main/resources/flexibilized".

Exercise 1.2: Improve your grammar

- Now that you have a working grammar it's time to flexibilize it!

- Introduce flexibility in modelling appointments by using an interface nonterminal for the appointment's elements such as **break**, **participants** and **repetition**.
- For this purpose, create a new grammar AppointmentsFlexibilized.
- Adapt the existing nonterminal productions to use/implement the interface nonterminal.

- Create a JUnit test AppointmentsFlexibilizedParserTest that test the new parser for the models in
  - "src/main/resources/example" as well as the model in
  - "src/main/resources/flexibilized".

Exercise 1.2: Improve your grammar

MG

```
grammar AppointmentsFlexibilized extends de.monticore.literals.MCCommonLiterals {
  start Appointment;

  Appointment = "appointment" name:String "{"
                Start
                End
                "participants" ":" participant:String("," participant:String)*
                repetition:["once"|"weekly"|"daily"]?
                "}";
  Start = "start" ":" Date? Time;
  End = "end" ":" Date? Time;
  Date = day:NatLiteral "/" month:NatLiteral  "/" year:NatLiteral;
  Time = hour:NatLiteral ":" min:NatLiteral;
  Break = "break" "{" Start End "}";
}
```

Exercise 1.2: Improve your grammar

MG

```
grammar AppointmentsFlexibilized extends de.monticore.literals.MCCommonLiterals {
  start Appointment;

  Appointment = "appointment" name:String "{"
                Start
                End
                Participants
                Repetition?
                "}";
  Start = "start" ":" Date? Time;
  End = "end" ":" Date? Time;
  Date = day:NatLiteral "/" month:NatLiteral  "/" year:NatLiteral;
  Time = hour:NatLiteral ":" min:NatLiteral;
  Break = "break" "{" Start End "}";
  Participants = "participants" ":" participant:String("," participant:String)*;
  Repetition = repetition:["once"|"weekly"|"daily"];
}
```

Exercise 1.2: Improve your grammar

MG

```
grammar AppointmentsFlexibilized extends de.monticore.literals.MCCommonLiterals {
  start Appointment;

  Appointment = "appointment" name:String "{"
                Element*
                "}";

  interface Element;

  Start implements Element = "start" ":" Date? Time;
  End implements Element = "end" ":" Date? Time;
  Date = day:NatLiteral "/" month:NatLiteral  "/" year:NatLiteral;
  Time = hour:NatLiteral ":" min:NatLiteral;
  Break implements Element = "break" "{" Start End "}";
  Participants implements Element = "participants" ":" participant:String("," participant:String)*;
  Repetition implements Element = repetition:["once"|"weekly"|"daily"];
}
```

Exercise 1.2: Improve your grammar

- Now that you have a working grammar it's time to flexibilize it!

- Introduce flexibility in modelling appointments by using an interface nonterminal for the appointment's elements such as break, participants and repetition.
- For this purpose, create a new grammar AppointmentsFlexibilized.
- Adapt the existing nonterminal productions to use/implement the interface nonterminal.

- Create a JUnit test AppointmentsFlexibilizedParserTest that test the new parser for the models in
  - "src/main/resources/example" as well as the model in
  - "src/main/resources/flexibilized".

Exercise 1.2: Improve your grammar

MG

```
public class AppointmentsFlexibilizedParserTest  { /* ... */
  @Nested class Parse {
  @ParameterizedTest
  @CsvSource({"src/main/resources/example/Example1.ap",
              "src/main/resources/example/Example2.ap",
              "src/main/resources/example/Example3.ap",
              "src/main/resources/example/Example4.ap",
              "src/main/resources/flexibilized/Example5.ap" })
  void shouldParseValidModel(String modelFilePath) throws IOException {
      //given
      AppointmentsFlexibilizedParser p = new AppointmentsFlexibilizedParser();

      //when
      Optional<ASTAppointment> result = p.parse(modelFilePath);

      //then
      assertThat(result).isPresent();
    }
  }
}
```

Exercise 1.3: Extend your language

a. Create a new grammar and name it Calendars. A model of this grammar holds an owner and a list of appointments. Create an extension point to delay the decision on how concrete appointments are modelled. A model without any Appointment could be modelled as follows:

```
Peter`s calendar:
```

b. Create a third grammar CalendarsWithAppointments that fills the extension point in your Calendars grammar with the Appointment Nonterminal of the Grammar Appointments. Create a JUnit Test CalendarsParserTest with one parser test for each of the following models located in “src/main/resources/calendar” that should be valid models of your language:

```
/*...*/
```

```
/*...*/
```

Exercise 1.3: Extend your language

```
component grammar Calendars extends de.monticore.literals.MCCommonLiterals {

  start Calendar;

  Calendar =
    Name "`s" "calendar" ":" App* ;


  external App;
}
```

MG

Peter`s calendar:

Exercise 1.3: Extend your language

a. Create a new grammar and name it Calendars. A model of this grammar holds an owner and a list of appointments. Create an extension point to delay the decision on how concrete appointments are modelled. A model without any Appointment could be modelled as follows:

```
Peter`s calendar:
```

b. Create a third grammar CalendarsWithAppointments that fills the extension point in your Calendars grammar with the Appointment Nonterminal of the Grammar Appointments. Create a JUnit Test CalendarsParserTest with one parser test for each of the following models located in “src/main/resources/calendar” that should be valid models of your language:

```
/*...*/
```

```
/*...*/
```

Exercise 1.3: Extend your language

```
grammar CalendarsWithAppointments extends Appointments, Calendars {

  start Calendar;

  App = Appointment;
}
```

MG

Exercise 1.3: Extend your language

a. Create a new grammar and name it Calendars. A model of this grammar holds an owner and a list of appointments. Create an extension point to delay the decision on how concrete appointments are modelled. A model without any Appointment could be modelled as follows:

```
Peter`s calendar:
```

b. Create a third grammar CalendarsWithAppointments that fills the extension point in your Calendars grammar with the Appointment Nonterminal of the Grammar Appointments. Create a JUnit Test CalendarsParserTest with one parser test for each of the following models located in “src/main/resources/calendar” that should be valid models of your language:

```
/*...*/
```

```
/*...*/
```

Exercise 1.3: Extend your language

Java

```java
public class CalendarsParserTest { /* ... */
  @Nested class Parse {
  @ParameterizedTest
  @CsvSource({ "src/main/resources/calendar/Peter.cal",
               "src/main/resources/calendar/Tina.cal" })
    void shouldParseValidModel(String modelFilePath) throws IOException {
      //given
      CalendarsWithAppointmentsParser p = new CalendarsWithAppointmentsParser();

      //when
      Optional<ASTCalendar> result = p.parse(modelFilePath);

      //then
      assertThat(result).isPresent();
    }
  }
}
```

Exercise 1.4: Types and Expressions

a. Draw the AST structure of the grammars MCBasicTypes and MCCollectionTypes.

b. Draw the tree structure of AST nodes that is created if this Expression is parsed by an empty grammar that extends CommonExpressions and MCCommonLiterals:

```
((4 > 17) || (call(25) + 3 != 7)) && !(a || false)
```

Exercise 1.4: Types and Expressions

MG

```
component grammar MCBasicTypes extends de.monticore.MCBasics, de.monticore.types.TypeSymbols {
  interface MCType;

  MCQualifiedName =
    part:(Name || ".")+;

  MCImportStatement =
    "import" MCQualifiedName ("." Star:["*"])? ";" ;

  MCPrimitiveType implements MCType =
    primitive: [ "boolean" | "byte" | "short" | "int"
               | "long" | "char" |"float" | "double" ];

  interface MCObjectType extends MCType;

  MCQualifiedType implements MCObjectType = MCQualifiedName;

  MCReturnType = MCVoidType | MCType;

  MCVoidType = "void";
}
```

Exercise 1.4: Types and Expressions

```
component grammar MCBasicTypes extends de.monticore.MCBasics, de.monticore.types.TypeSymbols {
  interface MCType;

  MCQualifiedName =
    part:(Name || ".")+;

  MCImportStatement =
    "import" MCQualifiedName ("." Star:["*"])? ";" ;

  MCPrimitiveType implements MCType =
    primitive: [ "boolean" | "byte" | "short" | "int"
               | "long" | "char" |"float" | "double" ];

  interface MCObjectType extends MCType;

  MCQualifiedType implements MCObjectType = MCQualifiedName;

  MCReturnType = MCVoidType | MCType;

  MCVoidType = "void";
}
```

MG

«interface»
ASTMCType

Exercise 1.4: Types and Expressions

```
component grammar MCBasicTypes extends de.monticore.MCBasics, de.monticore.types.TypeSymbols {
  interface MCType;

  MCQualifiedName =
    part:(Name || ".")+;

  MCImportStatement =
    "import" MCQualifiedName ("." Star:["*"])? ";" ;

  MCPrimitiveType implements MCType =
    primitive: [ "boolean" | "byte" | "short" | "int"
               | "long" | "char" |"float" | "double" ];

  interface MCObjectType extends MCType;

  MCQualifiedType implements MCObjectType = MCQualifiedName;

  MCReturnType = MCVoidType | MCType;

  MCVoidType = "void";
}
```

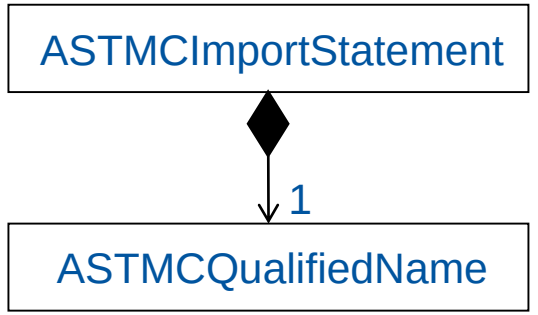
MG

ASTMCQualifiedName

Exercise 1.4: Types and Expressions

```
component grammar MCBasicTypes extends de.monticore.MCBasics, de.monticore.types.TypeSymbols {
  interface MCType;

  MCQualifiedName =
    part:(Name || ".")+;

  MCImportStatement =
    "import" MCQualifiedName ("." Star:["*"])? ";" ;

  MCPrimitiveType implements MCType =
    primitive: [ "boolean" | "byte" | "short" | "int"
               | "long" | "char" |"float" | "double" ];

  interface MCObjectType extends MCType;

  MCQualifiedType implements MCObjectType = MCQualifiedName;

  MCReturnType = MCVoidType | MCType;

  MCVoidType = "void";
}
```

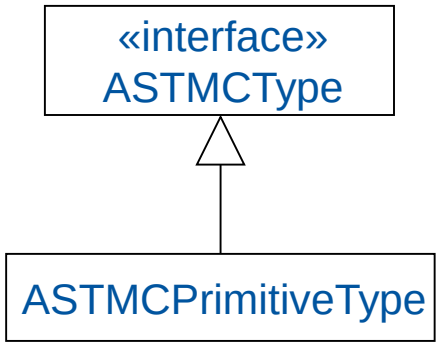
MG



Exercise 1.4: Types and Expressions

```
component grammar MCBasicTypes extends de.monticore.MCBasics, de.monticore.types.TypeSymbols {
  interface MCType;

  MCQualifiedName =
    part:(Name || ".")+;

  MCImportStatement =
    "import" MCQualifiedName ("." Star:["*"])? ";" ;

  MCPrimitiveType implements MCType =
    primitive: [ "boolean" | "byte" | "short" | "int"
               | "long" | "char" |"float" | "double" ];

  interface MCObjectType extends MCType;

  MCQualifiedType implements MCObjectType = MCQualifiedName;

  MCReturnType = MCVoidType | MCType;

  MCVoidType = "void";
}
```

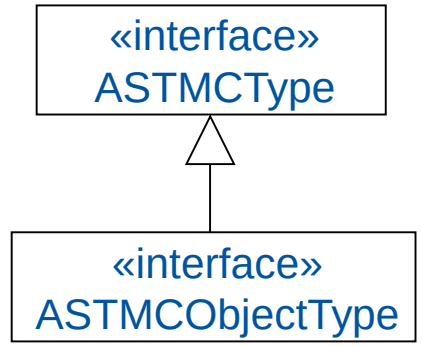
MG



Exercise 1.4: Types and Expressions

```
component grammar MCBasicTypes extends de.monticore.MCBasics, de.monticore.types.TypeSymbols {
  interface MCType;

  MCQualifiedName =
    part:(Name || ".")+;

  MCImportStatement =
    "import" MCQualifiedName ("." Star:["*"])? ";" ;

  MCPrimitiveType implements MCType =
    primitive: [ "boolean" | "byte" | "short" | "int"
               | "long" | "char" |"float" | "double" ];

  interface MCObjectType extends MCType;

  MCQualifiedType implements MCObjectType = MCQualifiedName;

  MCReturnType = MCVoidType | MCType;

  MCVoidType = "void";
}
```

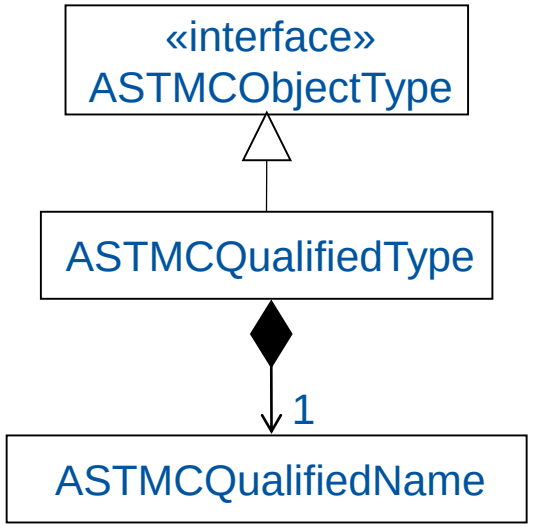
MG



Exercise 1.4: Types and Expressions

```
component grammar MCBasicTypes extends de.monticore.MCBasics, de.monticore.types.TypeSymbols {
  interface MCType;

  MCQualifiedName =
    part:(Name || ".")+;

  MCImportStatement =
    "import" MCQualifiedName ("." Star:["*"])? ";" ;

  MCPrimitiveType implements MCType =
    primitive: [ "boolean" | "byte" | "short" | "int"
               | "long" | "char" |"float" | "double" ];

  interface MCObjectType extends MCType;

  MCQualifiedType implements MCObjectType = MCQualifiedName;

  MCReturnType = MCVoidType | MCType;

  MCVoidType = "void";
}
```

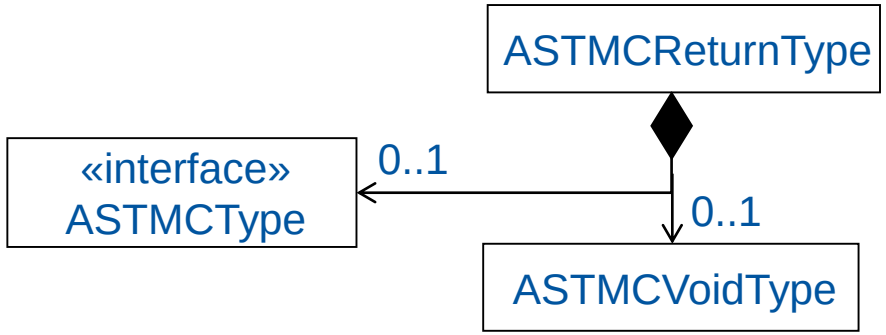
MG



Exercise 1.4: Types and Expressions

```
component grammar MCBasicTypes extends de.monticore.MCBasics, de.monticore.types.TypeSymbols {
  interface MCType;

  MCQualifiedName =
    part:(Name || ".")+;

  MCImportStatement =
    "import" MCQualifiedName ("." Star:["*"])? ";" ;

  MCPrimitiveType implements MCType =
    primitive: [ "boolean" | "byte" | "short" | "int"
               | "long" | "char" |"float" | "double" ];

  interface MCObjectType extends MCType;

  MCQualifiedType implements MCObjectType = MCQualifiedName;

  MCReturnType = MCVoidType | MCType;

  MCVoidType = "void";
}
```

MG

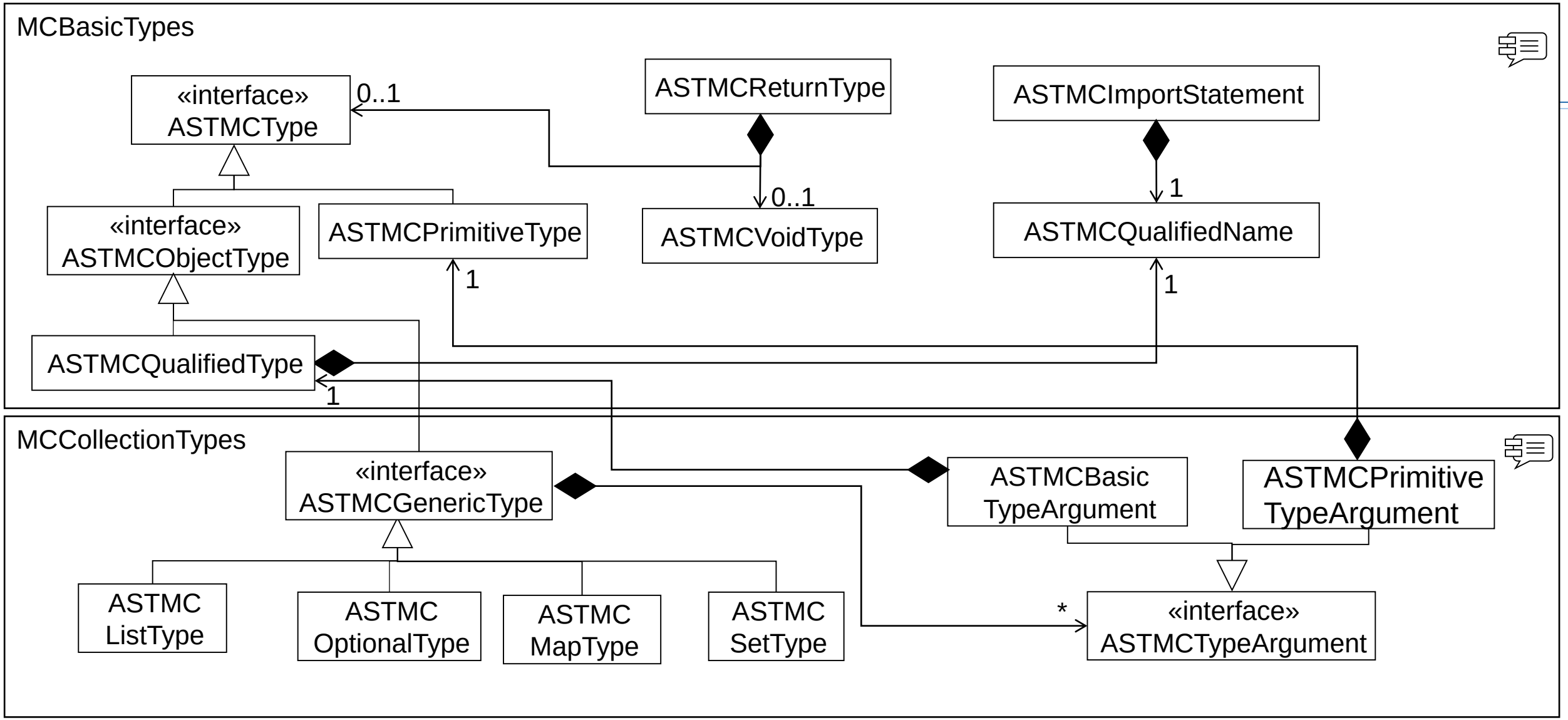


Exercise 1.4: Types and Expressions

```
component grammar MCBasicTypes extends de.monticore.MCBasics, de.monticore.types.TypeSymbols {
  interface MCType;

  MCQualifiedName =
    part:(Name || ".")+;

  MCImportStatement =
    "import" MCQualifiedName ("." Star:["*"])? ";" ;

  MCPrimitiveType implements MCType =
    primitive: [ "boolean" | "byte" | "short" | "int"
               | "long" | "char" |"float" | "double" ];

  interface MCObjectType extends MCType;

  MCQualifiedType implements MCObjectType = MCQualifiedName;

  MCReturnType = MCVoidType | MCType;

  MCVoidType = "void";
}
```

MG

ASTMCVoidType


MCBasicTypes
«interface»
ASTMCType
0..1
ASTMCReturnType
ASTMCImportStatement
«interface»
ASTMCObjectType
ASTMCPrimitiveType
0..1
ASTMCVoidType
1
ASTMCQualifiedName
1
1
ASTMCQualifiedType
1
MCCollectionTypes
«interface»
ASTMCGenericType
ASTMCBasic
TypeArgument
ASTMCPrimitive
TypeArgument
ASTMC
ListType
ASTMC
OptionalType
ASTMC
MapType
ASTMC
SetType
*
«interface»
ASTMCTypeArgument

Exercise 1.4: Types and Expressions

a. Draw the AST structure of the grammars MCBasicTypes and MCCollectionTypes.

b. Draw the tree structure of AST nodes that is created if this Expression is parsed by an empty grammar that extends CommonExpressions and MCCommonLiterals:

```
((4 > 17) || (call(25) + 3 != 7)) && !(a || false)
```

Exercise 1.4: Types and Expressions

((4 > 17) || (call(25) + 3 != 7)) && !(a || false)

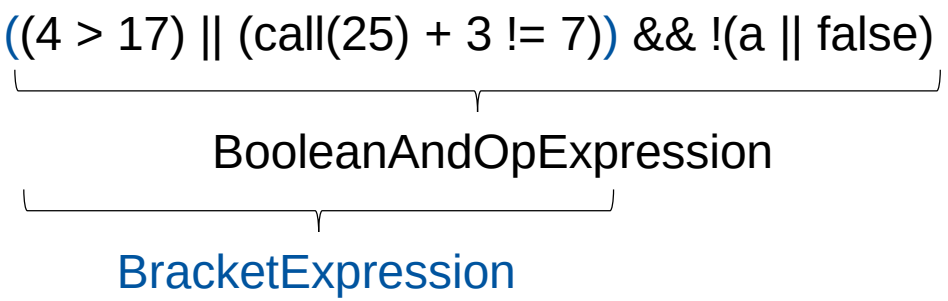
Exercise 1.4: Types and Expressions

((4 > 17) || (call(25) + 3 != 7)) && !(a || false)

BooleanAndOpExpression

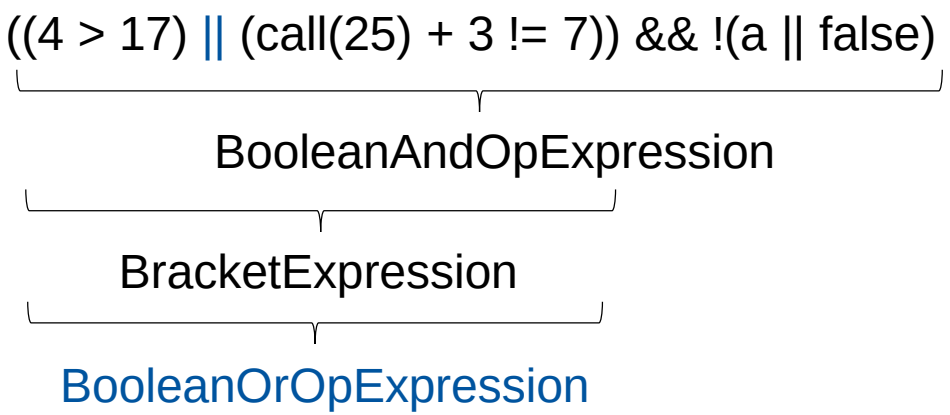
```
BooleanAndOpExpression implements Expression <120>, InfixExpression =
        left:Expression operator:"&&" right:Expression;
```

Exercise 1.4: Types and Expressions



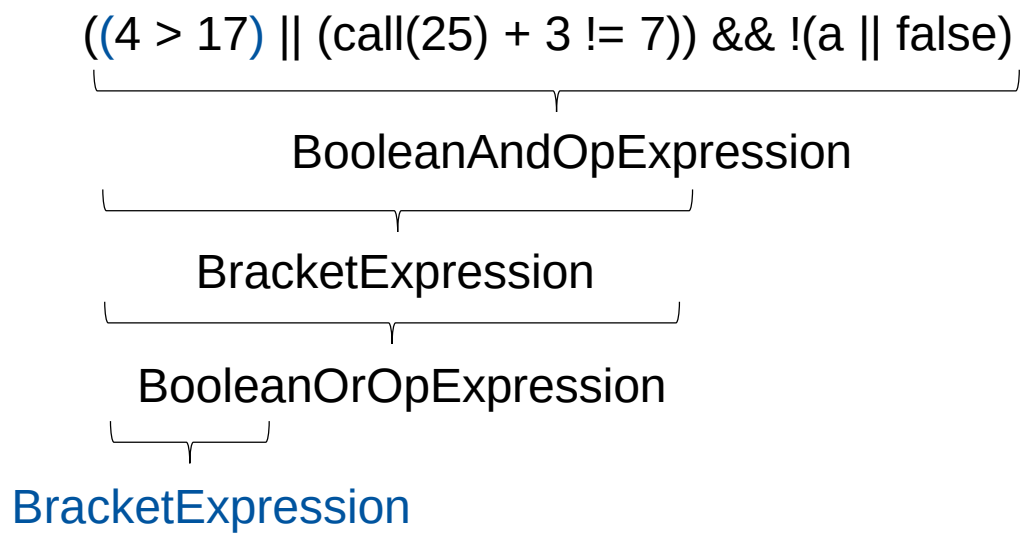

```
BracketExpression implements Expression <310> =  "(" Expression ")";
```

Exercise 1.4: Types and Expressions



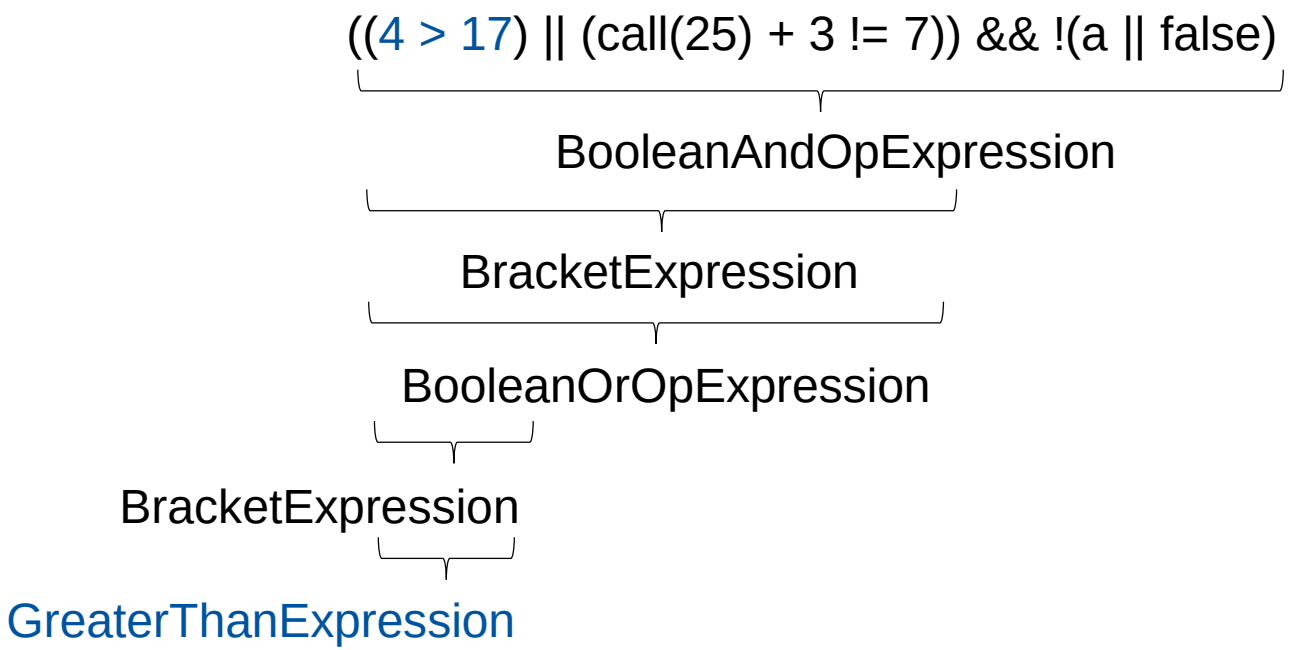

```
BooleanOrOpExpression implements Expression <117>, InfixExpression =
      left:Expression operator:"||" right:Expression;
```

Exercise 1.4: Types and Expressions



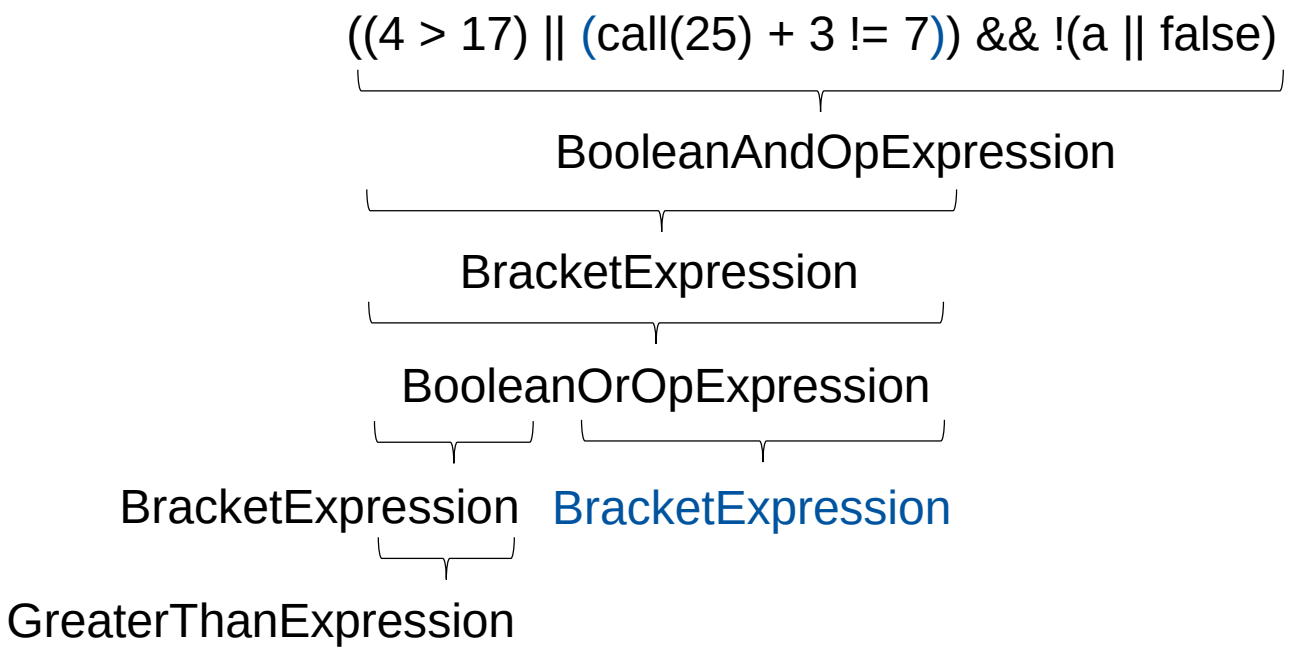

```
BracketExpression implements Expression <310> =  "(" Expression ")";
```

Exercise 1.4: Types and Expressions



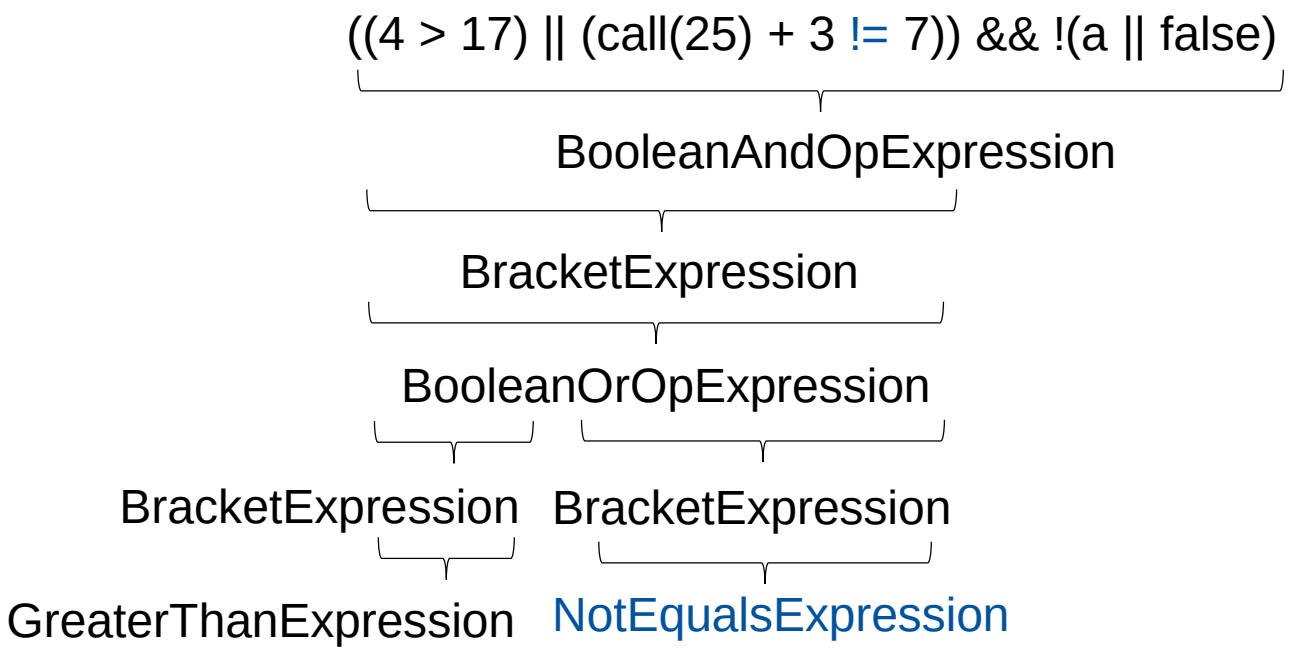

```
GreaterThanExpression implements Expression <150>, InfixExpression =
      left:Expression operator:">" right:Expression;
```

Exercise 1.4: Types and Expressions



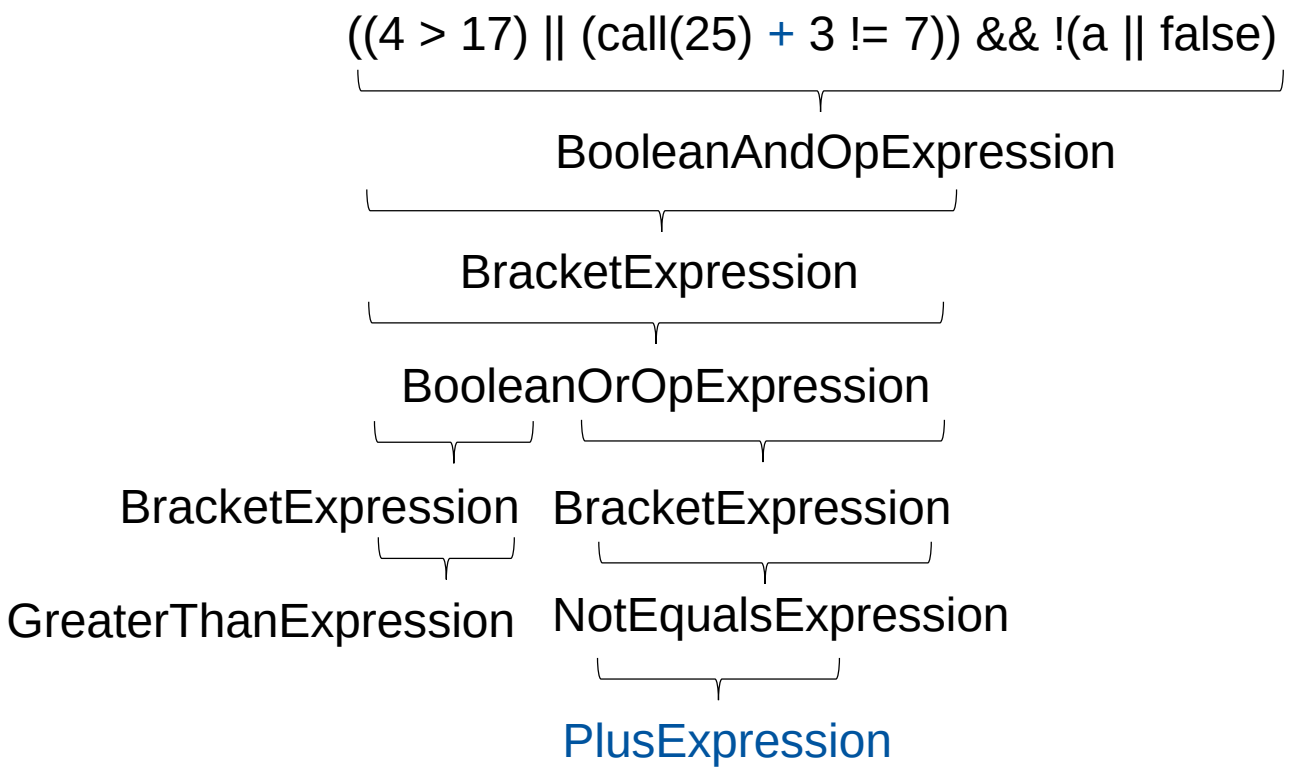

```
BracketExpression implements Expression <310> =  "(" Expression ")";
```

Exercise 1.4: Types and Expressions



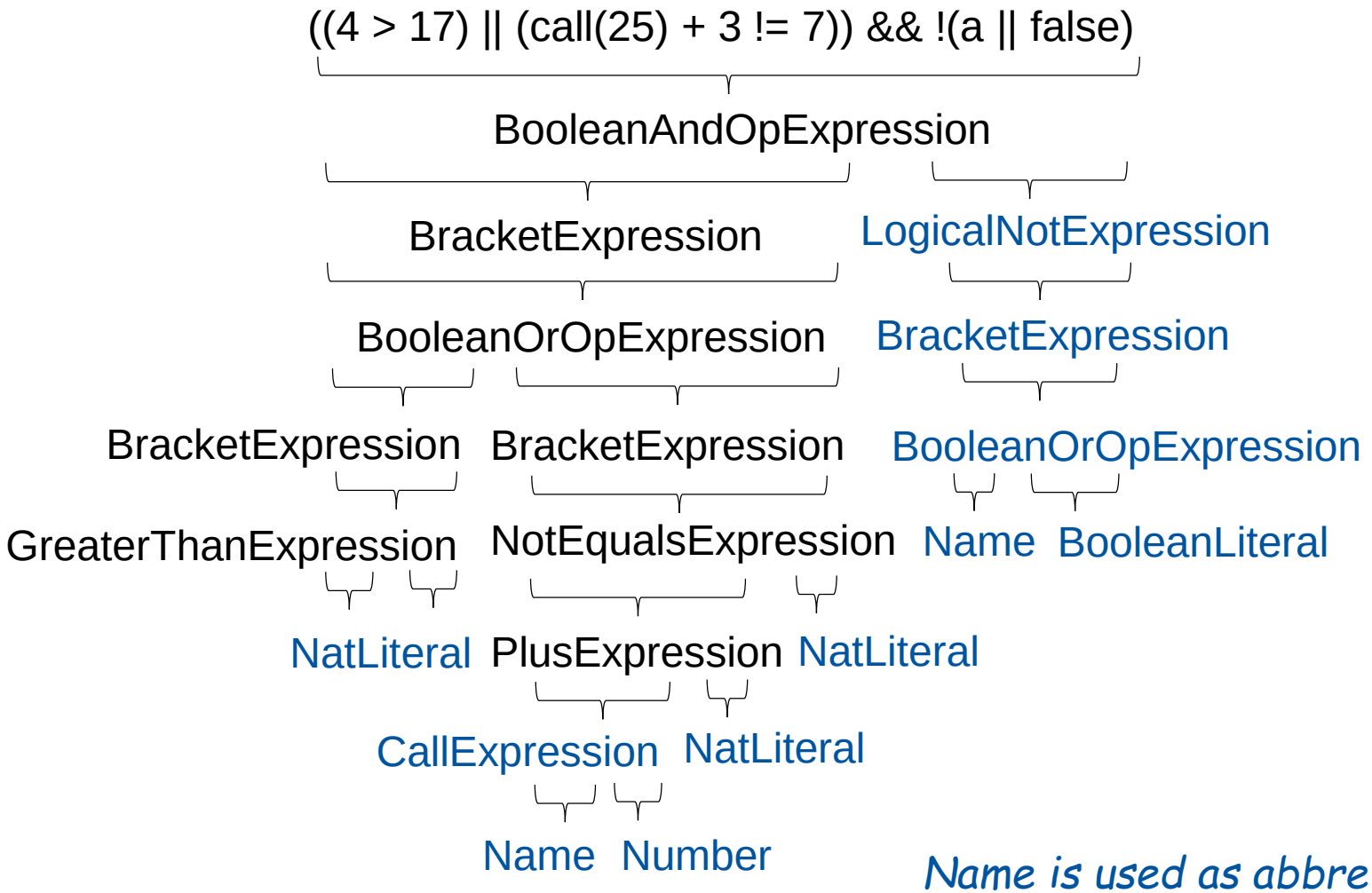

```
NotEqualsExpression implements Expression <130>, InfixExpression =
        left:Expression operator:"!=" right:Expression;
```

Exercise 1.4: Types and Expressions



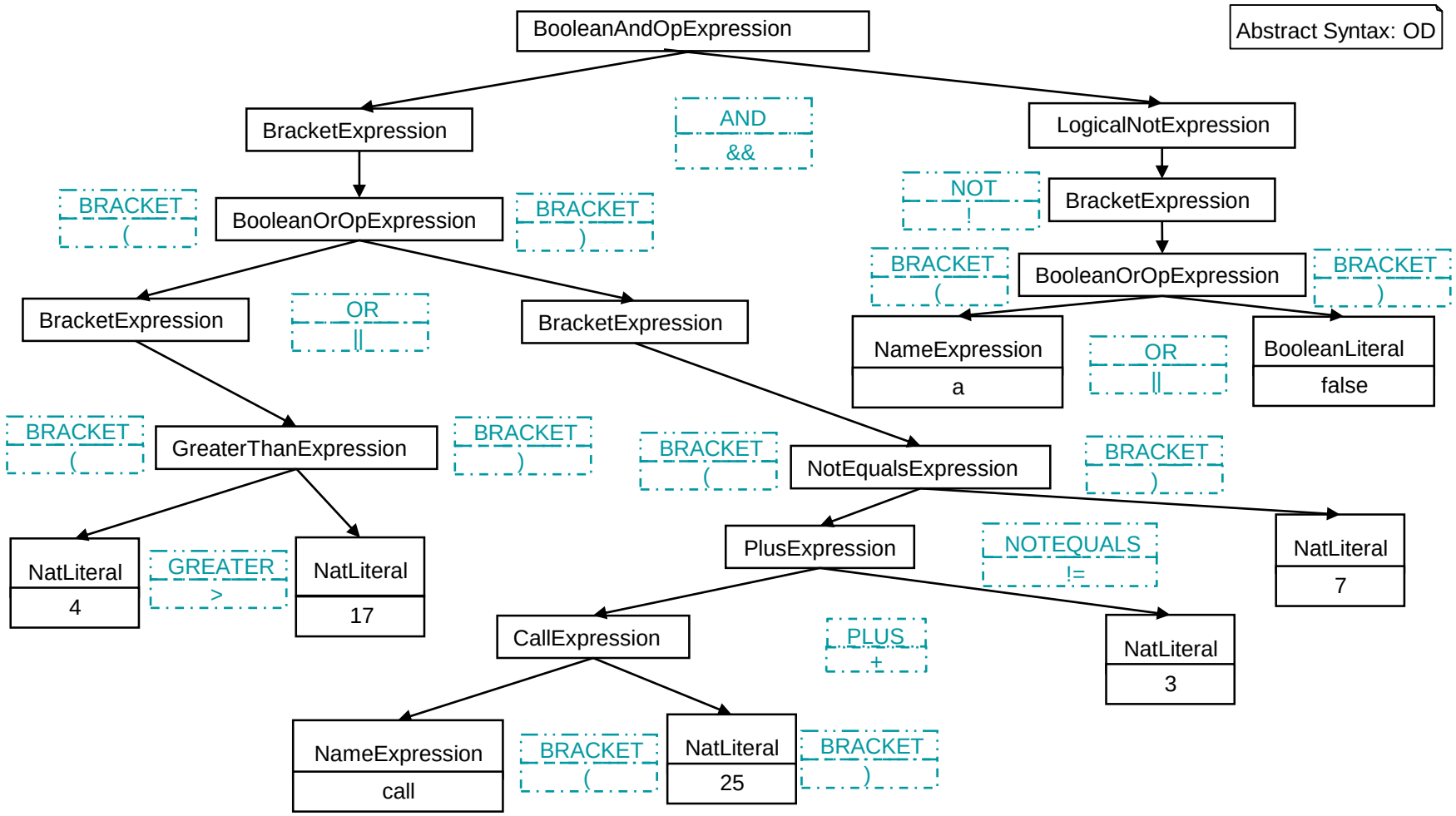

```
PlusExpression implements Expression <170>, InfixExpression =
      left:Expression operator:"+" right:Expression;
```

Exercise 1.4: Types and Expressions



Name is used as abbreviation for NameLiteral

Exercise 1.4: Types and Expressions



Questions?