Rapport des erreurs contextuelles

Équipe (43 G8) :
EL ARABI Mohamed
ARDAN Fatima-Azzahra
EL GOUIJ Faical
MOUNTASSIR Hamza
TOUATI Ayoub
Tuteur :
M. NICOLLIN Xavier
Date :
17 Janvier 2026


Introduction

Ce document regroupe les erreurs contextuelles implémentées dans le compilateur `Deca`. Pour chaque règle :

- l'énoncé de la règle est prise du poly ;
- les erreurs correspondantes sont listées avec le fichier Java, le code Java déclenchant l'erreur (`throw new ContextualError`), et des exemples Deca.

Règle 3.16 — list_decl_var (construction d'environnement)

$$\textbf{list\_decl\_var} \downarrow env\_types \downarrow env\_exp\_sup \downarrow env\_exp \downarrow class \uparrow env\_expr$$
$$\rightarrow \left\{ env\_expr := env\_exp \right\}$$
$$\left[(\textbf{decl\_var} \downarrow env\_types \downarrow env\_exp\_sup \downarrow env\_expr \downarrow class \uparrow env\_expr) \right]^*$$

Erreur (redéclaration au même niveau) — `DeclVar.java` / `EnvironmentExp`

Fichier Java : `DeclVar.java` (déclaration locale) + `EnvironmentExp` (détection du doublon)

Code Java (détection doublon) :

```java
public void declare(Symbol name, ExpDefinition def) throws DoubleDefException {
    if (this.env.containsKey(name)) {
        throw new DoubleDefException();
    } else {
        this.env.put(name, def);
    }
}
```

Code Java (conversion en ContextualError) :

```java
try {
    localEnv.declare(symbol, def);
} catch (EnvironmentExp.DoubleDefException e) {
    throw new ContextualError(
        "Variable already declared: " + symbol.getName(),
        varName.getLocation()
    );
}
```

Exemple incorrect

```
{
    int x;
    float x;
}
```

Exemple correct

```
{
    int x;
    {
        float x;
    }
}
```

Règle 3.17 — decl_var (type non void)

decl_var $\downarrow env\_types \downarrow env\_exp\_sup \downarrow env\_exp \downarrow class \uparrow \{name \mapsto (var, type)\} \oplus env\_exp$

$\rightarrow$ **DeclVar**$\Big[$

type $\downarrow env\_types \uparrow type$

Identifier $\uparrow name$

initialization $\downarrow env\_types \downarrow env\_exp/env\_exp\_sup \downarrow class \downarrow type$ $\Big]$

condition $type \neq void$

Erreur (variable de type void) — `DeclVar.java`

Fichier Java : `DeclVar.java`

Code Java :

```java
if (ttype.isVoid()) {
    throw new ContextualError(
        "Variable cannot be declared with type void",
        type.getLocation()
    );
}
```

Exemple incorrect

```
{
    void x;
}
```

Règle 3.28 — rvalue (compatibilité d'affectation)

rvalue $\downarrow env\_types \downarrow env\_exp \downarrow class \downarrow type_1$

$\rightarrow$ **expr** $\downarrow env\_types \downarrow env\_exp \downarrow class \uparrow type_2$

Erreur (affectation incompatible) — `Assign.java`

Fichier Java : `Assign.java`

Code Java (tel que fourni) :

```java
Type leftOperand_Type = this.getLeftOperand().verifyExpr(compiler, localEnv, currentClass);
Type rightOperand_Type = this.getRightOperand().verifyExpr(compiler, localEnv, currentClass);

// Rule: types must match for assignment
if (!rightOperand_Type.sameType(leftOperand_Type)) {
    throw new ContextualError(
        "Incompatible assignment: cannot assign " + rightOperand_Type.getName()
        + " to " + leftOperand_Type.getName(),
        this.getLocation()
    );
}
```

Exemple incorrect

```
{
   int a;
   float b;
   a = b;
}
```

Exemple correct

```
{
   float x;
   x = 3;
}
```

Erreur (rvalue incompatible avec expectedType) — `AbstractExpr.java`

Fichier Java : `AbstractExpr.java` (méthode `verifyRValue`)

Code Java :

```
Type rvalueType = this.verifyExpr(compiler, localEnv, currentClass);

if (!expectedType.assignCompatible(rvalueType)) {
    throw new ContextualError(
        "Incompatible assignment: expected " + expectedType + ", got " + rvalueType,
        this.getLocation()
    );
}
```

Exemple incorrect

```
{
    int x;
    x = true;
}
```

Exemple correct

```
{
    float x;
    x = 1;
}
```

Règle 3.8 — initialization

initialization $\downarrow env\_types \downarrow env\_exp \downarrow class \downarrow type$

$\rightarrow$ **Initialization**$\Big[$

rvalue $\downarrow env\_types \downarrow env\_exp \downarrow class \downarrow type$

$\Big]$

condition $assign\_compatible(env\_types, type_1, type_2)$

Erreur (initialisation incompatible) — `Initialization.java`

Fichier Java : `Initialization.java`

Code Java :

```
Type expression_type = this.expression.verifyExpr(compiler, localEnv, currentClass);
if (!expression_type.assignCompatible(t)) {
    throw new ContextualError(
        "Incompatible initialization: cannot initialize a variable of type "
        + t.getName() + " with an expression of type " + expression_type.getName(),
        this.getLocation()
    );
}
```

Exemple incorrect

```
{
   int a = 1.2;
}
```

Exemple correct

```
{
   float a = 5;
}
```

Règle 3.22 — `IfThenElse.java`

Fichier Java : `IfThenElse.java`

inst $\downarrow env\_types \downarrow env\_exp \downarrow class \downarrow return$

$\rightarrow$ **IfThenElse**$\Big[$

condition $\downarrow env\_types \downarrow env\_exp \downarrow class$

list_inst $\downarrow env\_types \downarrow env\_exp \downarrow class \downarrow return$

list_inst $\downarrow env\_types \downarrow env\_exp \downarrow class \downarrow return$

$\Big]$

condition $\downarrow env\_types \downarrow env\_exp \downarrow class$

$\rightarrow$ **expr** $\uparrow boolean$

Code Java :

```
Type cond_type = condition.verifyExpr(compiler, localEnv, currentClass);
if (!cond_type.isBoolean()) {
    throw new ContextualError(
        "If condition must be of type boolean",
        condition.getLocation()
    );
}
```

Exemple incorrect

```
{
   if (3) {
      inst1
   } else {
      inst2
   }
}
```

Exemple correct

```
{
   if (true) { inst1 }
   else { inst2 }
}
```

Règle 3.24 — `While.java`

Fichier Java : `While.java`

inst $\downarrow env\_types \downarrow env\_exp \downarrow class \downarrow return$

$\rightarrow$ **While**$\Big[$

condition $\downarrow env\_types \downarrow env\_exp \downarrow class$

list_inst $\downarrow env\_types \downarrow env\_exp \downarrow class \downarrow return$

$\Big]$

condition $\downarrow env\_types \downarrow env\_exp \downarrow class$

$\rightarrow$ **expr** $\uparrow boolean$

Code Java :

```
Type condType = condition.verifyExpr(compiler, localEnv, currentClass);
if (!condType.isBoolean()) {
    throw new ContextualError(
        "While condition must be of type boolean",
        getCondition().getLocation()
    );
}
```

Exemple incorrect

```
{
   while ("Kamehameha") {
      print(1);
   }
}
```

Exemple correct

```
{
   while (x < 67) {
      x = x + 1;
   }
}
```

Règle 3.33 — op_bin (type_binary_op)

\begin{quote}

$$\textbf{expr} \downarrow env\_types \downarrow env\_exp \downarrow class \uparrow type$$
$$\rightarrow \textbf{op\_bin} \uparrow op \left[\begin{array}{l} \textbf{expr} \uparrow type_1 \\ \textbf{expr} \uparrow type_2 \end{array} \right]$$
$$\textbf{affectation}\ type := type\_binary\_op(op, type_1, type_2)$$

Erreur (arithmétique sur types non numériques) — `AbstractOpArith.java`

Fichier Java : `AbstractOpArith.java`

Code Java :

```java
// Arithmetic ops only allowed between int/float
if (leftOperand_Type.isInt()) {
    if (rightOperand_Type.isInt()) { ... }
    else if (rightOperand_Type.isFloat()) { ... }
    else {
        throw new ContextualError(
            "Invalid right operand type for arithmetic operation: expected int or float",
            this.getLocation()
        );
    }
} else if (leftOperand_Type.isFloat()) {
    if (rightOperand_Type.isFloat()) { ... }
    else if (rightOperand_Type.isInt()) { ... }
    else {
        throw new ContextualError(
            "Invalid right operand type for arithmetic operation: expected int or float",
            this.getLocation()
        );
    }
} else {
    throw new ContextualError(
        "Invalid left operand type for arithmetic operation: expected int or float",
        this.getLocation()
    );
}
```

Exemple incorrect

```
{
   int a = 1 + true;
}
```

Exemple correct

```
{
   float a = 1 + 2.0;
}
```

Erreur (opérations booléennes sur non-bool) — `AbstractOpBool.java`

Fichier Java : `AbstractOpBool.java`

Code Java :

```java
if (!leftOperand_Type.isBoolean()) {
    throw new ContextualError(
        "Invalid left operand type for boolean operation: expected boolean",
        this.getLocation()
```

```
    );
}

if (!rightOperand_Type.isBoolean()) {
    throw new ContextualError(
        "Invalid right operand type for boolean operation: expected boolean",
        this.getLocation()
    );
}
```

Exemple incorrect

```
{
   boolean a = 1 || true;
}
```

Exemple correct

```
{
   if ((hot = cold) && (cool != lame)) { ... }
}
```

Erreur (modulo sur non-entiers) — `Modulo.java`

Fichier Java : `Modulo.java`

Code Java :

```
if (!leftOperand_Type.isInt()) {
    throw new ContextualError(
        "Invalid left operand type for modulo operation: expected int",
        this.getLocation()
    );
}
if (!rightOperand_Type.isInt()) {
    throw new ContextualError(
        "Invalid right operand type for modulo operation: expected int",
        this.getLocation()
    );
}
```

Exemple incorrect

```
{
   int a = 5;
   float b = 12.0;
   b % a;
}
```

Exemple correct

```
{
   int a = 5 % 2;
}
```

Règle 3.37 — op_un (type_unary_op)

expr $\downarrow env\_types \downarrow env\_exp \downarrow class \uparrow type$

$\rightarrow$ **op_un** $\uparrow op$ $\Big[$

expr $\uparrow type_1$

$\Big]$

affectation $type := type\_unary\_op(op, type_1)$

Erreur (Not sur non-bool) — `Not.java`

Fichier Java : `Not.java`

Code Java :

```
Type operand_Type = this.getOperand().verifyExpr(compiler, localEnv, currentClass);
if (!operand_Type.isBoolean()) {
    throw new ContextualError(
        "Invalid operand type for Not operation: expected boolean",
        this.getLocation()
    );
}
```

Exemple incorrect

```
{
   int a = 5;
   boolean smart = Not a;
}
```

Exemple correct

```
{
   boolean sigma = false;
   boolean not_sigma = Not sigma;
}
```

Erreur (moins unaire sur non-numérique) — `UnaryMinus.java`

Fichier Java : `UnaryMinus.java`

Code Java :

```
Type otype = getOperand().verifyExpr(compiler, localEnv, currentClass);
if (!otype.isInt() && !otype.isFloat()) {
    throw new ContextualError(
        "Unary minus can only be applied to int or float",
        getOperand().getLocation()
    );
}
```

Exemple incorrect

```
{
    boolean b;
    b = true;
    println(-b);
}
```

Exemple correct

```
{
    int x;
    x = 5;
    println(-x);
}
```

Règles 3.56–3.61 — Comparaisons

op_bin $\uparrow eq \rightarrow$ **Equals**
$\uparrow neq \rightarrow$ **NotEquals**
$\uparrow gt \rightarrow$ **Greater**
$\uparrow geq \rightarrow$ **GreaterOrEquals**
$\uparrow lt \rightarrow$ **Lower**
$\uparrow leq \rightarrow$ **LowerOrEquals**

Erreur (égalité sur types incompatibles) — `AbstractOpExact.java`

Fichier Java : `AbstractOpExact.java`

Code Java :

```
Type leftOperand_Type = this.getLeftOperand().verifyExpr(compiler, localEnv, currentClass); // les
    verifyExpr retournent tous les types
Type rightOperand_Type = this.getRightOperand().verifyExpr(compiler, localEnv, currentClass);
if ((!leftOperand_Type.assignCompatible(rightOperand_Type)) &&
    !(rightOperand_Type.assignCompatible(leftOperand_Type))){
          // Règle : les types doivent être compatibles pour une comparaison
            throw new ContextualError("Incompatible Types : cannot compare " +
                leftOperand_Type.getName() + " with " + rightOperand_Type.getName(),
                this.getLocation());
        }
```

Exemple incorrect (Equals)

```
{
   1 == true;
}
```

Exemple correct (Equals)

```
{
   1.0 == 2;
}
```

Exemple incorrect (NotEquals)

```
{
   1 != true;
}
```

Exemple correct (NotEquals)

```
{
   1 != 1;
}
```

Erreur (inégalités sur non-numériques) — `AbstractOpIneq.java`

Fichier Java : `AbstractOpIneq.java`

Code Java :

```
Type leftOperand_Type = this.getLeftOperand().verifyExpr(compiler, localEnv, currentClass);
Type rightOperand_Type = this.getRightOperand().verifyExpr(compiler, localEnv, currentClass);

if (leftOperand_Type.isInt() || leftOperand_Type.isFloat()) {
    if (!rightOperand_Type.isInt() && !rightOperand_Type.isFloat()) {
        throw new ContextualError(
            "Incompatible types: cannot apply inequality between "
            + leftOperand_Type.getName() + " and " + rightOperand_Type.getName()
            + " (right operand is not int/float)",
            this.getLocation()
        );
    }
    // implicit int->float conversions possible...
} else {
    throw new ContextualError(
        "Incompatible types: cannot apply inequality between "
        + leftOperand_Type.getName() + " and " + rightOperand_Type.getName()
        + " (left operand is not int/float)",
        this.getLocation()
    );
}
```

Exemples incorrects

```
{ 1 > false; }
{ true >= true; }
{ 1 < false; }
{ 1 <= true; }
```

Exemples corrects

```
{ 2 > 1; }
{ 2 >= 1; }
{ 1 < 2; }
{ 1 <= 2; }
```

Règle 3.65 — Sélection de champ (public)

$$\textbf{lvalue} \rightarrow \textbf{Selection}\Big[\begin{array}{l}\textbf{expr} \uparrow type\_class(class_2)\\ \textbf{field\_ident} \uparrow public \uparrow \_ \uparrow type\end{array}\Big]$$

$$\textbf{condition}\ \ (class(\_, env\_exp_2), \_) \in env\_types(class_2)$$

Erreur (sélection sur non-classe) — `Selection.java`

Fichier Java : `Selection.java`

Code Java :

```
Type objectExprType = objectExpr.verifyExpr(compiler, localEnv, currentClass);
if (!objectExprType.isClass()) {
    throw new ContextualError(
        "Cannot select a field from " + objectExpr + ": it must be a class instance",
```

```
        getLocation()
    );
}
```

Exemple incorrect

```
{
    int x;
    int z = x.y;
}
```

Exemple correct

```
class A {
        public int y;
        public void f() { this.y = 3; }
    }
{
    A a = new A();
    int z = a.y;
}
```

Erreur (sélection d'une méthode au lieu d'un champ) — `Selection.java`

Fichier Java : `Selection.java`

Code Java :

```
Definition def = field.verifyIdent(envExp2);

if (!def.isField()) {
    throw new ContextualError(
        field.getName() + " is not a field",
        field.getLocation()
    );
}
```

Exemple incorrect

```

class A {
        public int oneForAll;
        public int getOneForAll() { return this.oneForAll;
    }
}
{
    A a = new A();
    int x = a.getOneForAll;
}
```

Règle 3.66 — Sélection de champ (protected)

$$\textbf{Selection}\Big[\ \textbf{field\_ident}\ \uparrow protected\ \uparrow class\_field\ \uparrow type\Big]$$

$$\textbf{condition}\ \ (class(\_, env\_exp_2), \_) \in env\_types(class_2)$$
$$\wedge\ subtype(env\_types, type\_class(class_2), type\_class(class))$$
$$\wedge\ subtype(env\_types, type\_class(class), type\_class(class\_field))$$

Erreur (protected dans main) — `Selection.java`

Fichier Java : `Selection.java`

Code Java :

```
if (field_ident.getVisibility() == Visibility.PROTECTED) {
    if (currentClass == null) {
        throw new ContextualError(
            "Access to " + field + " is not allowed in main because it is protected",
            getLocation()
        );
    }
}
```

Erreur (subtype(class2, class) violée) — `Selection.java`

Fichier Java : `Selection.java`

Code Java :

```
if (!objectExprType.isSubTypeOf(currentClass.getType())) {
    throw new ContextualError(
        "Access to " + field + " is not allowed: " + class2
        + " is not a subtype of " + currentClass,
        field.getLocation()
    );
}
```

Erreur (subtype(class, class_field) violée) — `Selection.java`

Fichier Java : `Selection.java`

Code Java :

```
if (!currentClass.getType().isSubTypeOf(field_ident.getContainingClass().getType())) {
    throw new ContextualError(
        "Access to " + field + " is not allowed: " + currentClass
        + " is not a subtype of the field's declaring class",
        field.getLocation()
    );
}
```

Règle 3.42 — new

expr $\downarrow env\_types \uparrow type$

$\rightarrow$ **New** $\Big[$ **type** $\downarrow env\_types \uparrow type$ $\Big]$

condition $type = type\_class(\_)$

Erreur (new sur non-classe) — `New.java`

Fichier Java : `New.java`

Code Java :

```
Type ttype = type.verifyType(compiler);
if (!ttype.isClass()) {
    throw new ContextualError(
        "new can only be applied to a class",
        type.getLocation()
    );
}
```

Exemple incorrect

```
{
    int x = new int;
}
```

Exemple correct

```

class A {};
{
    A a = new A();
}
```

Règle 3.43 — this

expr $\downarrow env\_types \uparrow type$

$\rightarrow$ **New**$\Big[$ **type** $\downarrow env\_types \uparrow type$ $\Big]$

condition $type = type\_class(\_)$

Erreur (this hors classe) — `This.java`

Fichier Java : `This.java`

Code Java :

```
if (currentClass == null) {
    throw new ContextualError(
        "Token 'this' can only be used inside a class",
        getLocation()
    );
}
```

Exemple incorrect

```
{
    this;
}
```

Exemple correct

```

class A {
        void f() {
            println(this);
        }
    }
```

Règles 3.71–3.73 — Appels de méthodes

$$\textbf{method\_call} \downarrow env\_types \downarrow env\_exp \downarrow class \uparrow type$$
$$\rightarrow \textbf{MethodCall}\Big[$$
$$\textbf{expr} \uparrow type\_class(class2)$$
$$\textbf{method\_ident} \uparrow sig \uparrow type$$
$$\textbf{rvalue\_star} \downarrow sig \Big]$$
$$\textbf{condition}\ type = type\_class(\_)$$

Erreur (appel sur non-classe) — `MethodCall.java`

Fichier Java : `MethodCall.java`

Code Java :

```
Type objectExprType = objectExpr.verifyExpr(compiler, localEnv, currentClass);
if (!objectExprType.isClass()) {
    throw new ContextualError(
        "Cannot call a method on a non-class object",
        objectExpr.getLocation()
    );
}
```

Exemple incorrect

```
{
    int x;
    x.f();
}
```

Exemple correct

```
class A {
        public void One() { println(1); }
    }
{
    A a = new A();
    a.One();
}
```

Erreur (identifiant de méthode n'est pas une méthode) — `MethodCall.java`

Fichier Java : `MethodCall.java`

Code Java :

```
if (!def.isMethod()) {
    throw new ContextualError(
        methodIdent.getName() + " is not a method",
        methodIdent.getLocation()
    );
}
```

Exemple incorrect

```
class A { int x; }
{
    A a = new A();
    a.x();
}
```

Erreur (mauvais nombre d'arguments) — `ListExpr.java`

Fichier Java : `ListExpr.java`

Code Java :

```
if (this.size() != signature.size()) {
    throw new ContextualError(
        "Incorrect number of arguments: expected " + signature.size()
        + ", got " + this.size(),
        getLocation()
    );
}
```

Exemple incorrect

```
class A { void f(int x, String s) {} }
{
    A a = new A();
    a.f(1);
}
```

Exemple correct

```

class A { void f(int x, String s) {} }
{
    A a = new A();
    a.f(1, "Himsagi Himyoichi");
}
```

Règle 3.24 — Return

$$\textbf{inst} \downarrow env\_types \downarrow env\_exp \downarrow class \downarrow return$$

$$\rightarrow \textbf{Return}\Big[\ \ \textbf{rvalue} \downarrow env\_types \downarrow env\_exp \downarrow class \downarrow return\ \ \Big]$$

$$\textbf{condition}\ \ return \neq void$$

Erreur (return dans méthode void) — `Return.java`

Fichier Java : `Return.java`

Code Java :

```
if (returnType.isVoid()) {
    throw new ContextualError(
        "Return value is not allowed in a void method",
        getLocation()
    );
}
```

Exemple incorrect

```
class A {
    void f() { return 1; }
    }
```

Exemple correct

```
class A {
    void f() { return; }
}
```

Erreur (type retourné incompatible) — `Return.java`

Fichier Java : `Return.java`

Code Java :

```java
if (!returnType.assignCompatible(rvalueType)) {
    throw new ContextualError(
        "Incompatible return type: expected " + returnType + ", got " + rvalueType,
        rvalue.getLocation()
    );
}
```

Exemple incorrect

```
class A {
  int f() { return 1.5; }
}
```

Exemple correct

```

class A {
    float f() { return 1; }
}
```

Erreur (return interdit dans main) — `Return.java`

Fichier Java : `Return.java`

Code Java :

```java
if (currentClass == null) {
    throw new ContextualError(
        "Return statement is not allowed in main",
        getLocation()
    );
}
```

Exemples incorrects

```
{ return; }
{ return 3; }
```

Exemple correct

```
{
    class A {
        public int f() { return 3; }
    }
}
```

Règle 2.9 — Paramètre non void

decl_param $\downarrow env\_types \uparrow \{name \mapsto (param, type)\}$

$\rightarrow$ **DeclParam** $\Big[$ **type** $\downarrow env\_types \uparrow type$ **Identifier** $\uparrow name$ $\Big]$

condition $type \neq void$

Erreur (paramètre de type void) — `DeclParam.java`

Fichier Java : `DeclParam.java`

```
if (ttype.isVoid()) {
    throw new ContextualError(
        "Parameter cannot be of type void",
        type.getLocation()
    );
}
```

Exemple incorrect

```
{
    int f(void x) { return 0; }
}
```

Exemple correct

```
{
    int f(int x) { return x; }
}
```

Règle 3.12 — Environnement des paramètres (pas de doublons)

$$\begin{aligned}\textbf{list\_decl\_param}\ &\downarrow env\_types\ \uparrow env\_expr \\ &\rightarrow \left\{ env\_expr := \emptyset \right\} \\ &\quad \Big[\ (\textbf{decl\_param}\ \downarrow env\_types\ \uparrow env\_exp\ ) \left\{ env\_expr := env\_expr \oplus env\_exp \right\} \Big]^{*}\end{aligned}$$

Erreur (paramètre déclaré deux fois) — `DeclParam.java`

Fichier Java : `DeclParam.java`

```
try {
    localEnv.declare(symbol, def);
} catch (EnvironmentExp.DoubleDefException e) {
    throw new ContextualError(
        "Parameter already declared: " + symbol.getName(),
        paramName.getLocation()
    );
}
```

Exemple incorrect

```
{
    int f(int x, float x) { return 0; }
}
```

Exemple correct

```
{
    int f(int x, float y) { return x; }
}
```

Règle 3.5 — Déclaration de classe

decl_class $\downarrow env\_types$

$\rightarrow$ **DeclClass** $\Big[$

Identifier $\uparrow class$

Identifier $\uparrow super$

list_decl_field $\downarrow env\_types$ $\downarrow env\_exp$ $\downarrow class$

list_decl_method $\downarrow env\_types$ $\downarrow env\_exp$ $\downarrow class$

$\Big]$

condition $(class(\_, env\_exp), \_) \in env\_types(class)$

Erreur (classe déjà définie) — `DeclClass.java`

Fichier Java : `DeclClass.java`

```java
if (compiler.environmentType.defOfType(nomClasse) != null) {
    throw new ContextualError(
        "Class already defined: " + nomClasse.getName(),
        name.getLocation()
    );
}
```

Exemple incorrect

```
class A {}
class A {}
```

Erreur (super-classe inexistante) — `DeclClass.java`

Fichier Java : `DeclClass.java`

```java
if (superTypeDEF == null) {
    throw new ContextualError(
        "Superclass does not exist: " + nomClasseMere.getName(),
        superClass.getLocation()
    );
}
```

Exemple incorrect

```
class B extends Inconnue {}
```

Erreur (super n'est pas une classe) — `DeclClass.java`

Fichier Java : `DeclClass.java`

```java
if (!type.isClass()) {
    throw new ContextualError(
        "Superclass is not a class",
        superClass.getLocation()
    );
}
```

Exemple incorrect

```
int x;
class B extends x{}
```

Exemple correct

```
class A{};
class B extends A{}
```

Règle 3.7 — Déclaration de champ

decl_field $\downarrow env\_types \downarrow env\_exp \downarrow class$

$\rightarrow$ **DeclField**$\Big[$

type $\downarrow env\_types \uparrow type$

Identifier

initialization $\downarrow env\_types \downarrow env\_exp \downarrow class \downarrow type$

$\Big]$

Erreur (champ de type void) — `DeclField.java`

Fichier Java : `DeclField.java`

```
if (ttype.isVoid()) {
    throw new ContextualError(
        "Field cannot be declared with type void",
        type.getLocation()
    );
}
```

Exemple incorrect

```
class A { void x; }
```

Exemple correct

```
class A { int x; }
```

Erreur (cacher une méthode de la super-classe par un champ) — `DeclField.java`

Fichier Java : `DeclField.java`

```
ExpDefinition methodWithname = superClass.getMembers().get(fieldName.getName());
if (methodWithname != null && methodWithname.isMethod()) {
    throw new ContextualError(
        "Name already used for a method in super class: you cannot hide a method with a field",
        fieldName.getLocation()
    );
}
```

Exemple incorrect

```
class A { int f() { return 0; } }
class B extends A { int f; }
```

Erreur (champ déjà déclaré dans la classe) — `DeclField.java`

Fichier Java : `DeclField.java`

```
try {
    currentClass.getMembers().declare(symbol, def);
} catch (EnvironmentExp.DoubleDefException e) {
    throw new ContextualError(
        "Field already declared in this class: " + symbol.getName(),
        fieldName.getLocation()
    );
}
```

Exemple incorrect

```
class A { int x; float x; }
```

Exemple correct

```
class A { int x; float y; }
```

Règles 2.7 et 3.10–3.11 — Déclaration de méthode / redéfinition

Si une méthode est redéfinie : même signature, et type de retour sous-type du type hérité.

Erreur (méthode cache un champ de la super-classe) — `DeclMethod.java`

Fichier Java : `DeclMethod.java`

```
if (mfWithName != null) {
    if (mfWithName.isField()) {
        throw new ContextualError(
            "Name already used for a field in super class: you cannot hide a field with a method",
            methodName.getLocation()
        );
    }
}
```

Exemple incorrect

```
class A { int x; }
class B extends A { int x() { return 0; } }
```

Erreur (signature différente de la super-classe) — `DeclMethod.java`

Fichier Java : `DeclMethod.java`

```
if (!signature.equals(methodToOverride.getSignature())) {
    throw new ContextualError(
        "Overridden method signature does not match the one in the super class",
        methodName.getLocation()
    );
}
```

Erreur (type de retour pas sous-type de celui de la méthode hérité) — `DeclMethod.java`

Fichier Java : `DeclMethod.java`

```java
if (!ttype.isSubTypeOf(methodToOverride.getType())) {
    throw new ContextualError(
        "Overridden method return type must be a subtype of the super class method return type",
        getLocation()
    );
}
```

Exemple incorrect

```
class A {
    float f() { return 1.0; }
}

class B extends A {
    int f() { return 1; }
}
```

```
Class C;
Class D extends C;

class A {
    D f(){return new C()}
}

class B extends A {
    C f(){return new D()} <-- C n'est pas une sous classe de D
            }
```

Exemple correct

```
Class C;
class A {
    Object f() { return new Object(); }
}

class B extends A {
    C f() { return new C(); } <-- C est une sous classe de Object
}
```

Erreur (méthode déjà déclarée dans la classe) — `DeclMethod.java`

Fichier Java : `DeclMethod.java`

```java
try {
    currentClass.getMembers().declare(symbol, def);
} catch (EnvironmentExp.DoubleDefException e) {
    throw new ContextualError(
        "Method already declared: " + symbol.getName(),
        methodName.getLocation()
    );
}
```

Règles 3.67–3.69 / 3.72 — Identificateurs

identifier doit être défini dans env_exp ; et selon le contexte, correspondre à var/param/field/method.

Erreur (identifiant non déclaré dans env_exp et parents) — `Identifier.java`

Fichier Java : `Identifier.java`

Code Java :

```
Definition ident_def = localEnv.get(name);
if (ident_def == null) {
    throw new ContextualError(
        "Identifier " + name.toString()
        + " is not defined in the current scope nor in its parent scopes",
        this.getLocation()
    );
}
```

Exemple incorrect

```
{
   int a;
   b = a + 1;
}
```

Exemple correct

```
{
   int a;
   {
      float b;
      b = a + 1;
   }
}
```

Erreur (identifiant pas une expression) — `Identifier.java`

Code Java :

```
if (!(ident_def instanceof ExpDefinition)) {
    throw new ContextualError(
        "Identifier " + name.toString() + " is not an expression",
        this.getLocation()
    );
}
```

Erreur (type non défini) — `Identifier.java`

Code Java :

```
TypeDefinition ident_def = compiler.environmentType.defOfType(name);
if (ident_def == null) {
    throw new ContextualError(
        "Type " + name.toString() + " is not defined in the type environment",
        this.getLocation()
    );
}
```

Erreur (verifyIdent : identifiant non défini dans env_exp) — `Identifier.java`

```
Definition def = localEnv.get(name);
if (def == null) {
    throw new ContextualError(
        "Identifier " + name + " is not defined",
```

```
        getLocation()
    );
}
```

Exemple incorrect

```
{
    { int x = 3; }
    println(x);
}
```

Exemple correct

```
{
    int x = 3;
    { println(x); }
}
```

Erreur (identifiant utilisé déjà utilisé comme nom de la méthode contenant sa redéfinission) — `Identifier.java`

Fichier Java : `Identifier.java`

Code Java :

```
d{lstlisting}
        if (ident_def.isMethod()){
            throw new ContextualError("Identifier " + name.toString() + " is used as a method name",
            this.getLocation());
        }
```

Exemple incorrect

```
class A {
    int m() {
        int x = m;
    }
}
```

Règles 3.70 / 3.72 / 3.65–3.71 — Aides de typage (`asFieldDefinition`, `asMethodDefinition`, `asClassType`)

Erreur (utiliser une méthode comme champ) — `Definition.java`

```
public FieldDefinition asFieldDefinition(String errorMessage, Location l)
        throws ContextualError {
    throw new ContextualError(errorMessage, l);
}
```

Erreur (utiliser un champ comme méthode) — `Definition.java`

```
public MethodDefinition asMethodDefinition(String errorMessage, Location l)
        throws ContextualError {
    throw new ContextualError(errorMessage, l);
}
```

Erreur (utiliser un non-classe comme classe) — `Type.java`

```
public ClassType asClassType(String errorMessage, Location l)
        throws ContextualError {
    throw new ContextualError(errorMessage, l);
}
```

Transtypage

Erreur (Transtypage avec des types incompatibles) — `Cast.java`

```
Type castingType = typeCast.verifyType(compiler);
Type exprType = expr.verifyExpr(compiler, localEnv, currentClass);

      if(!(castingType.isCastCompatible(exprType))){
          throw new ContextualError("Incompatible cast, can not cast " + exprType + " to " +
              castingType, getLocation());
      }
      }
}
```

Exemple incorrect

```
{
    boolean b = true;
    int x = (int) (b);
}
```

```
class A {}
class B {}

{
    A a = new A();
    B b = (B) (a);
}
```

Erreur (Transtypage avec void) — `Cast.java`

```
Type castingType = typeCast.verifyType(compiler);
Type exprType = expr.verifyExpr(compiler, localEnv, currentClass);

        if (castingType.isVoid()){
            throw new ContextualError("Can not cast using a void Type", getLocation());
        }
```

Exemple incorrect

```
class A {
   void m() { }
}

{
   A a = new A();

   (void)(a.m());
}
```

Est Instance de

Erreur (L'expression n'est ni une classe ni Null) — `InstanceOf.java`

```
Type exprType = expr.verifyExpr(compiler, localEnv, currentClass);
Type instanceType = typeInstance.verifyType(compiler);

if(!(exprType.isClass() || exprType.isNull())){
    throw new ContextualError("Instanceof could only be applied to a class or nullType, but a " +
        exprType + " was given.", getLocation());
}
```

Erreur (Le type de l'instance n'est pas celui d'une classe) — `InstanceOf.java`

```

if (!(instanceType.isClass())){
    throw new ContextualError("Inctanceof expected a class type but a " + instanceType + " was
        given.", getLocation());
}
```

Exemple incorrect

```
class A {}

{
    float f = 2.71;
    if (f instanceof A) {
        println("ça marche pas");
    }
}
```

```
{
    int x = 3;

    if (x instanceof int) {
        println("ça marche pas");
    }
}
```

Exemple correct

```
class A {}
class B {}

{
    A a;
    a = new A();

    if (a instanceof B) {
        println("true");
    } else {
        println("false");
    }
}
```

```
class A {}

{

    if (null instanceof A) {
        println("true");
    } else {
        println("false");
    }
}
```