[CI2] Cours 5: Traduction de programmes II

Daniela Petrișan
Université Paris Cité, IRIF

Dans ce cours

Nous poursuivons la traduction de programmes et nous nous intéressons aux appels de fonctions.

Dans ce cours

Nous poursuivons la traduction de programmes et nous nous intéressons aux appels de fonctions.

Nous avons déjà vu que lors d'un appel de fonction, il est nécessaire de transmettre les paramètres (appel par valeur) et de mémoriser une adresse de retour.

Dans ce cours

Nous poursuivons la traduction de programmes et nous nous intéressons aux appels de fonctions.

Nous avons déjà vu que lors d'un appel de fonction, il est nécessaire de transmettre les paramètres (appel par valeur) et de mémoriser une adresse de retour.

Dans le cours précédent, on a fait la traduction d'un appel de fonction très simple, pour une fonction sans paramètres. Dans ce cas, il suffisait de conserver l'adresse de retour dans une pile d'entiers.

```java
public class Tiny {
  public static void f() {
    System.out.println("Hello"); // ic = 500
  } // ic = 501 : la sortie de l'appel de f

  public static void main(String[] args) {
    f(); // ic = 0 : appel a la fonction f
         // ic = 1 : point de retour, nettoyage
  } // ic = 2 : sortie du programme
}
```

```java
import java.util.Stack;

public class TinyTrad {
  public static void main(String[] args) {
    int ic = 0;   //compteur instruction
    Stack<Integer> p = new Stack<Integer>();

    while(true)
    switch(ic){
      case 0: p.push(ic+1); ic = 500; break;
              //@retour=1 sur la pile +
              //saut incond. vers 500
      case 1: p.pop(); ic++; break;
              //on dépile
      case 2: System.exit(0);

      case 500: System.out.println("Hello");
                ic++; break;
      case 501: ic = p.peek(); break;
} } }
```

Bloc d'activation

En général, chaque appel de fonction nécessite la mémorisation de certaines informations :

- l'adresse de retour, c'est-à-dire le numéro de l'instruction qui doit être exécutée au retour de l'appel
- les paramètres de la fonction initialisés avec les valeurs des paramètres effectifs
- la valeur de retour de la fonction (si le type de retour n'est pas vide)
- les variables locales de la fonction appelée

Bloc d'activation

En général, chaque appel de fonction nécessite la mémorisation de certaines informations :

- l'adresse de retour, c'est-à-dire le numéro de l'instruction qui doit être exécutée au retour de l'appel
- les paramètres de la fonction initialisés avec les valeurs des paramètres effectifs
- la valeur de retour de la fonction (si le type de retour n'est pas vide)
- les variables locales de la fonction appelée

Ces informations forment le bloc d'activation (stack frame) de l'appel de fonction.

Bloc d'activation

En général, chaque appel de fonction nécessite la mémorisation de certaines informations :

- l'adresse de retour, c'est-à-dire le numéro de l'instruction qui doit être exécutée au retour de l'appel
- les paramètres de la fonction initialisés avec les valeurs des paramètres effectifs
- la valeur de retour de la fonction (si le type de retour n'est pas vide)
- les variables locales de la fonction appelée

Ces informations forment le bloc d'activation (stack frame) de l'appel de fonction.

A priori, le compilateur ne connaît pas le nombre d'appels de fonction, donc ces blocs d'activation seront stockés dans une pile d'exécution (ou pile d'appels).

Bloc d'activation

En général, chaque appel de fonction nécessite la mémorisation de certaines informations :

- l'adresse de retour, c'est-à-dire le numéro de l'instruction qui doit être exécutée au retour de l'appel
- les paramètres de la fonction initialisés avec les valeurs des paramètres effectifs
- la valeur de retour de la fonction (si le type de retour n'est pas vide)
- les variables locales de la fonction appelée

Ces informations forment le bloc d'activation (stack frame) de l'appel de fonction.

A priori, le compilateur ne connaît pas le nombre d'appels de fonction, donc ces blocs d'activation seront stockés dans une pile d'exécution (ou pile d'appels).

La durée de vie d'un bloc d'activation dépend de l'appel de fonction auquel il correspond. Le bloc est empilé lorsque l'appel de fonction est effectué et détruit après la fin de l'appel.

Bloc d'activation minimal

Le bloc d'activation minimal ne contient que l'adresse de retour, par exemple pour une fonction sans paramètres, avec le type de retour void et sans variables locales

```
public class Bloc {
  private int adresseRetour;

  public Bloc(int adresseRetour) {
    this.adresseRetour = adresseRetour;
  }

  public int getAdresse() {
    return this.adresseRetour;
  }
}
```

La structure générale de la traduction d'un appel de fonction simple

```
public class AppelSimpleTrad {
  static int ic = 0;
  static Stack<Bloc> p = new Stack<Bloc>();
  public static void main(String[] args) {
   while(true) {
     switch(ic){
       ...
       case n: p.push(new Bloc(ic+1));   // l'appel de fonction est lancé: on prépare le retour et
               ic = m; break;                                          //on fait un saut inconditionnel
       case n+1: p.pop();      //le point de retour après l'appel et nettoyage : on dépile le bloc
                 ic++; break;
       ...

       case m:                                                          //début de la fonction appelée
       ...
       case m':  ic = p.peek().getAdress(); break;    //sortie de la fonction appelée: on récupère
                                                                  //l'adresse de retour et on fait le saut
}}}}
```

Un exemple de traduction d'un appel de fonction simple

```
public class AppelSimple {
  public static void f() {
    System.out.println("Entree f()");
    g();
    System.out.println("Sortie f()");
  }

  public static void g() {
    System.out.println("Je suis g()");
  }

  public static void main(String[] args) {
    System.out.println("Entree main");
    f();
    System.out.println("Sortie main");
  }
}
```

Un exemple de traduction d'un appel de fonction simple

```
public class AppelSimple {
  public static void f() {
    System.out.println("Entree f()");
    g();
    System.out.println("Sortie f()");
  }

  public static void g() {
    System.out.println("Je suis g()");
  }

  public static void main(String[] args) {
    System.out.println("Entree main");
    f();
    System.out.println("Sortie main");
  }
}
```

Un exemple de traduction d'un appel de fonction simple

```
public class AppelSimple {
  public static void f() {
    System.out.println("Entree f()");
    g();
    System.out.println("Sortie f()");
  }

  public static void g() {
    System.out.println("Je suis g()");
  }

  public static void main(String[] args) {
    System.out.println("Entree main");                    // 0
    f();
    System.out.println("Sortie main");
  }
}
```

Un exemple de traduction d'un appel de fonction simple

```
public class AppelSimple {
  public static void f() {
    System.out.println("Entree f()");
    g();
    System.out.println("Sortie f()");
  }

  public static void g() {
    System.out.println("Je suis g()");
  }

  public static void main(String[] args) {
    System.out.println("Entree main");                              // 0
    f();       // 1 empiler le  bloc d'activation et saut inconditionnel
                     // 2 point de retour : dépiler le bloc d'activation
    System.out.println("Sortie main");
  }
}
```

Un exemple de traduction d'un appel de fonction simple

```
public class AppelSimple {
  public static void f() {
    System.out.println("Entree f()");
    g();
    System.out.println("Sortie f()");
  }

  public static void g() {
    System.out.println("Je suis g()");
  }

  public static void main(String[] args) {
    System.out.println("Entree main");                              // 0
    f();       // 1 empiler le  bloc d'activation et saut inconditionnel
                    // 2 point de retour : dépiler le bloc d'activation
    System.out.println("Sortie main");                              // 3
  }
}
```

Un exemple de traduction d'un appel de fonction simple

```
public class AppelSimple {
  public static void f() {
    System.out.println("Entree f()");
    g();
    System.out.println("Sortie f()");
  }

  public static void g() {
    System.out.println("Je suis g()");
  }

  public static void main(String[] args) {
    System.out.println("Entree main");                              // 0
    f();       // 1 empiler le  bloc d'activation et saut inconditionnel
                    // 2 point de retour : dépiler le bloc d'activation
    System.out.println("Sortie main");                              // 3
  }                                                          // 4 sortie de main
}
```

Un exemple de traduction d'un appel de fonction simple

```
public class AppelSimple {
  public static void f() {
    System.out.println("Entree f()");                          // 100
    g();
    System.out.println("Sortie f()");
  }

  public static void g() {
    System.out.println("Je suis g()");
  }

  public static void main(String[] args) {
    System.out.println("Entree main");                          // 0
    f();       // 1 empiler le  bloc d'activation et saut inconditionnel
                    // 2 point de retour : dépiler le bloc d'activation
    System.out.println("Sortie main");                          // 3
  }                                                  // 4 sortie de main
}
```

Un exemple de traduction d'un appel de fonction simple

```
public class AppelSimple {
  public static void f() {
    System.out.println("Entree f()");                              // 100
    g();      // 101 empiler le bloc d'activation et saut inconditionnel
                  // 102 point de retour: dépiler le bloc d'activation
    System.out.println("Sortie f()");
  }

  public static void g() {
    System.out.println("Je suis g()");
  }

  public static void main(String[] args) {
    System.out.println("Entree main");                               // 0
    f();       // 1 empiler le  bloc d'activation et saut inconditionnel
                   // 2 point de retour : dépiler le bloc d'activation
    System.out.println("Sortie main");                              // 3
  }                                                          // 4 sortie de main
}
```

Un exemple de traduction d'un appel de fonction simple

```
public class AppelSimple {
  public static void f() {
    System.out.println("Entree f()");                              // 100
    g();      // 101 empiler le bloc d'activation et saut inconditionnel
                  // 102 point de retour: dépiler le bloc d'activation
    System.out.println("Sortie f()");                              // 103
  }                // 104 sortie de f (on récupère l'adresse de retour)

  public static void g() {
    System.out.println("Je suis g()");
  }

  public static void main(String[] args) {
    System.out.println("Entree main");                              // 0
    f();       // 1 empiler le  bloc d'activation et saut inconditionnel
                   // 2 point de retour : dépiler le bloc d'activation
    System.out.println("Sortie main");                              // 3
  }                                                  // 4 sortie de main
}
```

Un exemple de traduction d'un appel de fonction simple

```
public class AppelSimple {
  public static void f() {
    System.out.println("Entree f()");                              // 100
    g();      // 101 empiler le bloc d'activation et saut inconditionnel
                  // 102 point de retour: dépiler le bloc d'activation
    System.out.println("Sortie f()");                              // 103
  }               // 104 sortie de f (on récupère l'adresse de retour)

  public static void g() {
    System.out.println("Je suis g()");                             // 200
  }              // 201 sortie de g (on récupère l'adresse de retour)

  public static void main(String[] args) {
    System.out.println("Entree main");                              // 0
    f();       // 1 empiler le  bloc d'activation et saut inconditionnel
                   // 2 point de retour : dépiler le bloc d'activation
    System.out.println("Sortie main");                              // 3
  }                                                  // 4 sortie de main
}
```

```java
public class AppelSimple {
  public static void f() {
    System.out.println("Entree f()"); //100
    g();  // 101 empiler bloc + saut incond.
          // 102 point de retour:
          // dépiler le bloc d'activation
    System.out.println("Sortie f()"); //103
  }                        //104 sortie de f

  public static void g() {
    System.out.println("Je suis g()");//200
  }                        //201 sortie de g

  public static void main(String[] args) {
    System.out.println("Entree main"); //0
    f();   //1 empiler bloc + saut incond.
           //2 point de retour :
           //  dépiler le bloc d'activation
    System.out.println("Sortie main"); //3
  }                       //4 sortie de main
}
```

```java
import java.util.Stack;
public class AppelSimpleTrad {
  static int ic = 0;
  static Stack<Bloc> p = new Stack<Bloc>();
  public static void main(String[] args) {
   while(true) {
    switch(ic) {
    case 0:
    case 1:
    case 2:
    case 3:
    case 4:

    case 100:
    case 101:
    case 102:
    case 103:
    case 104:

    case 200:
    case 201:
} } } }
```

```java
public class AppelSimple {
  public static void f() {
    System.out.println("Entree f()"); //100
    g();  // 101 empiler bloc + saut incond.
          // 102 point de retour:
          // dépiler le bloc d'activation
    System.out.println("Sortie f()"); //103
  }                        //104 sortie de f

  public static void g() {
    System.out.println("Je suis g()");//200
  }                        //201 sortie de g

  public static void main(String[] args) {
    System.out.println("Entree main"); //0
    f();   //1 empiler bloc + saut incond.
           //2 point de retour :
           //  dépiler le bloc d'activation
    System.out.println("Sortie main"); //3
  }                       //4 sortie de main
}
```

```java
import java.util.Stack;
public class AppelSimpleTrad {
  static int ic = 0;
  static Stack<Bloc> p = new Stack<Bloc>();
  public static void main(String[] args) {
   while(true) {
    switch(ic) {
    case 0: ...println("Entree main"); ic++; break;
    case 1:
    case 2:
    case 3:
    case 4:

    case 100:
    case 101:
    case 102:
    case 103:
    case 104:

    case 200:
    case 201:
} } } }
```

```java
public class AppelSimple {
  public static void f() {
    System.out.println("Entree f()"); //100
    g();  // 101 empiler bloc + saut incond.
          // 102 point de retour:
          // dépiler le bloc d'activation
    System.out.println("Sortie f()"); //103
  }                          //104 sortie de f

  public static void g() {
    System.out.println("Je suis g()");//200
  }                          //201 sortie de g

  public static void main(String[] args) {
    System.out.println("Entree main"); //0
    f();   //1 empiler bloc + saut incond.
           //2 point de retour :
           //  dépiler le bloc d'activation
    System.out.println("Sortie main"); //3
  }                        //4 sortie de main
}
```

```java
import java.util.Stack;
public class AppelSimpleTrad {
  static int ic = 0;
  static Stack<Bloc> p = new Stack<Bloc>();
  public static void main(String[] args) {
   while(true) {
    switch(ic) {
    case 0: ...println("Entree main"); ic++; break;
    case 1: p.push(new Bloc(ic+1)); ic = 100; break;
    case 2:
    case 3:
    case 4:

    case 100:
    case 101:
    case 102:
    case 103:
    case 104:

    case 200:
    case 201:
} } } }
```

```java
public class AppelSimple {
  public static void f() {
    System.out.println("Entree f()"); //100
    g();  // 101 empiler bloc + saut incond.
          // 102 point de retour:
          // dépiler le bloc d'activation
    System.out.println("Sortie f()"); //103
  }                          //104 sortie de f

  public static void g() {
    System.out.println("Je suis g()");//200
  }                          //201 sortie de g

  public static void main(String[] args) {
    System.out.println("Entree main"); //0
    f();   //1 empiler bloc + saut incond.
           //2 point de retour :
           //  dépiler le bloc d'activation
    System.out.println("Sortie main"); //3
  }                        //4 sortie de main
}
```

```java
import java.util.Stack;
public class AppelSimpleTrad {
  static int ic = 0;
  static Stack<Bloc> p = new Stack<Bloc>();
  public static void main(String[] args) {
   while(true) {
    switch(ic) {
    case 0: ...println("Entree main"); ic++; break;
    case 1: p.push(new Bloc(ic+1)); ic = 100; break;
    case 2: p.pop(); ic++; break;
    case 3:
    case 4:

    case 100:
    case 101:
    case 102:
    case 103:
    case 104:

    case 200:
    case 201:
} } } }
```

```java
public class AppelSimple {
  public static void f() {
    System.out.println("Entree f()"); //100
    g();  // 101 empiler bloc + saut incond.
          // 102 point de retour:
          // dépiler le bloc d'activation
    System.out.println("Sortie f()"); //103
  }                          //104 sortie de f

  public static void g() {
    System.out.println("Je suis g()");//200
  }                          //201 sortie de g

  public static void main(String[] args) {
    System.out.println("Entree main"); //0
    f();   //1 empiler bloc + saut incond.
           //2 point de retour :
           //  dépiler le bloc d'activation
    System.out.println("Sortie main"); //3
  }                        //4 sortie de main
}
```

```java
import java.util.Stack;
public class AppelSimpleTrad {
  static int ic = 0;
  static Stack<Bloc> p = new Stack<Bloc>();
  public static void main(String[] args) {
   while(true) {
    switch(ic) {
    case 0: ...println("Entree main"); ic++; break;
    case 1: p.push(new Bloc(ic+1)); ic = 100; break;
    case 2: p.pop(); ic++; break;
    case 3: ...println("Sortie main"); ic++; break;
    case 4:

    case 100:
    case 101:
    case 102:
    case 103:
    case 104:

    case 200:
    case 201:
} } } }
```

```java
public class AppelSimple {
  public static void f() {
    System.out.println("Entree f()"); //100
    g();  // 101 empiler bloc + saut incond.
          // 102 point de retour:
          // dépiler le bloc d'activation
    System.out.println("Sortie f()"); //103
  }                          //104 sortie de f

  public static void g() {
    System.out.println("Je suis g()");//200
  }                          //201 sortie de g

  public static void main(String[] args) {
    System.out.println("Entree main"); //0
    f();   //1 empiler bloc + saut incond.
           //2 point de retour :
           //  dépiler le bloc d'activation
    System.out.println("Sortie main"); //3
  }                        //4 sortie de main
}
```

```java
import java.util.Stack;
public class AppelSimpleTrad {
  static int ic = 0;
  static Stack<Bloc> p = new Stack<Bloc>();
  public static void main(String[] args) {
   while(true) {
    switch(ic) {
    case 0: ...println("Entree main"); ic++; break;
    case 1: p.push(new Bloc(ic+1)); ic = 100; break;
    case 2: p.pop(); ic++; break;
    case 3: ...println("Sortie main"); ic++; break;
    case 4: System.exit(0);

    case 100:
    case 101:
    case 102:
    case 103:
    case 104:

    case 200:
    case 201:
} } } }
```

```java
public class AppelSimple {
  public static void f() {
    System.out.println("Entree f()"); //100
    g();  // 101 empiler bloc + saut incond.
          // 102 point de retour:
          // dépiler le bloc d'activation
    System.out.println("Sortie f()"); //103
  }                          //104 sortie de f

  public static void g() {
    System.out.println("Je suis g()");//200
  }                          //201 sortie de g

  public static void main(String[] args) {
    System.out.println("Entree main"); //0
    f();   //1 empiler bloc + saut incond.
           //2 point de retour :
           //  dépiler le bloc d'activation
    System.out.println("Sortie main"); //3
  }                        //4 sortie de main
}
```

```java
import java.util.Stack;
public class AppelSimpleTrad {
  static int ic = 0;
  static Stack<Bloc> p = new Stack<Bloc>();
  public static void main(String[] args) {
   while(true) {
    switch(ic) {
    case 0: ...println("Entree main"); ic++; break;
    case 1: p.push(new Bloc(ic+1)); ic = 100; break;
    case 2: p.pop(); ic++; break;
    case 3: ...println("Sortie main"); ic++; break;
    case 4: System.exit(0);

    case 100: ...println("Entree f()"); ic++; break;
    case 101:
    case 102:
    case 103:
    case 104:

    case 200:
    case 201:
} } } }
```

```java
public class AppelSimple {
  public static void f() {
    System.out.println("Entree f()"); //100
    g();  // 101 empiler bloc + saut incond.
          // 102 point de retour:
          // dépiler le bloc d'activation
    System.out.println("Sortie f()"); //103
  }                          //104 sortie de f

  public static void g() {
    System.out.println("Je suis g()");//200
  }                          //201 sortie de g

  public static void main(String[] args) {
    System.out.println("Entree main"); //0
    f();   //1 empiler bloc + saut incond.
           //2 point de retour :
           //  dépiler le bloc d'activation
    System.out.println("Sortie main"); //3
  }                        //4 sortie de main
}
```

```java
import java.util.Stack;
public class AppelSimpleTrad {
  static int ic = 0;
  static Stack<Bloc> p = new Stack<Bloc>();
  public static void main(String[] args) {
   while(true) {
    switch(ic) {
    case 0: ...println("Entree main"); ic++; break;
    case 1: p.push(new Bloc(ic+1)); ic = 100; break;
    case 2: p.pop(); ic++; break;
    case 3: ...println("Sortie main"); ic++; break;
    case 4: System.exit(0);

    case 100: ...println("Entree f()"); ic++; break;
    case 101: p.push(new Bloc(ic+1)); ic = 200; break;
    case 102:
    case 103:
    case 104:

    case 200:
    case 201:
} } } }
```

```java
public class AppelSimple {
  public static void f() {
    System.out.println("Entree f()"); //100
    g();  // 101 empiler bloc + saut incond.
          // 102 point de retour:
          // dépiler le bloc d'activation
    System.out.println("Sortie f()"); //103
  }                          //104 sortie de f

  public static void g() {
    System.out.println("Je suis g()");//200
  }                          //201 sortie de g

  public static void main(String[] args) {
    System.out.println("Entree main"); //0
    f();   //1 empiler bloc + saut incond.
           //2 point de retour :
           //  dépiler le bloc d'activation
    System.out.println("Sortie main"); //3
  }                        //4 sortie de main
}
```

```java
import java.util.Stack;
public class AppelSimpleTrad {
  static int ic = 0;
  static Stack<Bloc> p = new Stack<Bloc>();
  public static void main(String[] args) {
   while(true) {
    switch(ic) {
    case 0: ...println("Entree main"); ic++; break;
    case 1: p.push(new Bloc(ic+1)); ic = 100; break;
    case 2: p.pop(); ic++; break;
    case 3: ...println("Sortie main"); ic++; break;
    case 4: System.exit(0);

    case 100: ...println("Entree f()"); ic++; break;
    case 101: p.push(new Bloc(ic+1)); ic = 200; break;
    case 102: p.pop(); ic++; break;
    case 103:
    case 104:

    case 200:
    case 201:
} } } }
```

```java
public class AppelSimple {
  public static void f() {
    System.out.println("Entree f()"); //100
    g();  // 101 empiler bloc + saut incond.
          // 102 point de retour:
          // dépiler le bloc d'activation
    System.out.println("Sortie f()"); //103
  }                          //104 sortie de f

  public static void g() {
    System.out.println("Je suis g()");//200
  }                          //201 sortie de g

  public static void main(String[] args) {
    System.out.println("Entree main"); //0
    f();   //1 empiler bloc + saut incond.
           //2 point de retour :
           //  dépiler le bloc d'activation
    System.out.println("Sortie main"); //3
  }                        //4 sortie de main
}
```

```java
import java.util.Stack;
public class AppelSimpleTrad {
  static int ic = 0;
  static Stack<Bloc> p = new Stack<Bloc>();
  public static void main(String[] args) {
   while(true) {
    switch(ic) {
    case 0: ...println("Entree main"); ic++; break;
    case 1: p.push(new Bloc(ic+1)); ic = 100; break;
    case 2: p.pop(); ic++; break;
    case 3: ...println("Sortie main"); ic++; break;
    case 4: System.exit(0);

    case 100: ...println("Entree f()"); ic++; break;
    case 101: p.push(new Bloc(ic+1)); ic = 200; break;
    case 102: p.pop(); ic++; break;
    case 103:
    case 104:

    case 200:
    case 201:
} } } }
```

```java
public class AppelSimple {
  public static void f() {
    System.out.println("Entree f()"); //100
    g();  // 101 empiler bloc + saut incond.
          // 102 point de retour:
          // dépiler le bloc d'activation
    System.out.println("Sortie f()"); //103
  }                          //104 sortie de f

  public static void g() {
    System.out.println("Je suis g()");//200
  }                          //201 sortie de g

  public static void main(String[] args) {
    System.out.println("Entree main"); //0
    f();   //1 empiler bloc + saut incond.
           //2 point de retour :
           //  dépiler le bloc d'activation
    System.out.println("Sortie main"); //3
  }                        //4 sortie de main
}
```

```java
import java.util.Stack;
public class AppelSimpleTrad {
  static int ic = 0;
  static Stack<Bloc> p = new Stack<Bloc>();
  public static void main(String[] args) {
   while(true) {
    switch(ic) {
    case 0: ...println("Entree main"); ic++; break;
    case 1: p.push(new Bloc(ic+1)); ic = 100; break;
    case 2: p.pop(); ic++; break;
    case 3: ...println("Sortie main"); ic++; break;
    case 4: System.exit(0);

    case 100: ...println("Entree f()"); ic++; break;
    case 101: p.push(new Bloc(ic+1)); ic = 200; break;
    case 102: p.pop(); ic++; break;
    case 103: ...println("Sortie f()"); ic++; break;
    case 104:

    case 200:
    case 201:
} } } }
```

```java
public class AppelSimple {
  public static void f() {
    System.out.println("Entree f()"); //100
    g();  // 101 empiler bloc + saut incond.
          // 102 point de retour:
          // dépiler le bloc d'activation
    System.out.println("Sortie f()"); //103
  }                          //104 sortie de f

  public static void g() {
    System.out.println("Je suis g()");//200
  }                          //201 sortie de g

  public static void main(String[] args) {
    System.out.println("Entree main"); //0
    f();   //1 empiler bloc + saut incond.
           //2 point de retour :
           //  dépiler le bloc d'activation
    System.out.println("Sortie main"); //3
  }                        //4 sortie de main
}
```

```java
import java.util.Stack;
public class AppelSimpleTrad {
  static int ic = 0;
  static Stack<Bloc> p = new Stack<Bloc>();
  public static void main(String[] args) {
   while(true) {
    switch(ic) {
    case 0: ...println("Entree main"); ic++; break;
    case 1: p.push(new Bloc(ic+1)); ic = 100; break;
    case 2: p.pop(); ic++; break;
    case 3: ...println("Sortie main"); ic++; break;
    case 4: System.exit(0);

    case 100: ...println("Entree f()"); ic++; break;
    case 101: p.push(new Bloc(ic+1)); ic = 200; break;
    case 102: p.pop(); ic++; break;
    case 103: ...println("Sortie f()"); ic++; break;
    case 104: ic = p.peek().getAdresse(); break;

    case 200:
    case 201:
} } } }
```

```java
public class AppelSimple {
  public static void f() {
    System.out.println("Entree f()"); //100
    g();  // 101 empiler bloc + saut incond.
          // 102 point de retour:
          // dépiler le bloc d'activation
    System.out.println("Sortie f()"); //103
  }                          //104 sortie de f

  public static void g() {
    System.out.println("Je suis g()");//200
  }                          //201 sortie de g

  public static void main(String[] args) {
    System.out.println("Entree main"); //0
    f();   //1 empiler bloc + saut incond.
           //2 point de retour :
           //  dépiler le bloc d'activation
    System.out.println("Sortie main"); //3
  }                        //4 sortie de main
}
```

```java
import java.util.Stack;
public class AppelSimpleTrad {
  static int ic = 0;
  static Stack<Bloc> p = new Stack<Bloc>();
  public static void main(String[] args) {
   while(true) {
    switch(ic) {
    case 0: ...println("Entree main"); ic++; break;
    case 1: p.push(new Bloc(ic+1)); ic = 100; break;
    case 2: p.pop(); ic++; break;
    case 3: ...println("Sortie main"); ic++; break;
    case 4: System.exit(0);

    case 100: ...println("Entree f()"); ic++; break;
    case 101: p.push(new Bloc(ic+1)); ic = 200; break;
    case 102: p.pop(); ic++; break;
    case 103: ...println("Sortie f()"); ic++; break;
    case 104: ic = p.peek().getAdresse(); break;

    case 200: ...println("Je suis g()"); ic++; break;
    case 201:
} } } }
```

```java
public class AppelSimple {
  public static void f() {
    System.out.println("Entree f()"); //100
    g();  // 101 empiler bloc + saut incond.
          // 102 point de retour:
          // dépiler le bloc d'activation
    System.out.println("Sortie f()"); //103
  }                          //104 sortie de f

  public static void g() {
    System.out.println("Je suis g()");//200
  }                          //201 sortie de g

  public static void main(String[] args) {
    System.out.println("Entree main"); //0
    f();   //1 empiler bloc + saut incond.
           //2 point de retour :
           //  dépiler le bloc d'activation
    System.out.println("Sortie main"); //3
  }                        //4 sortie de main
}
```

```java
import java.util.Stack;
public class AppelSimpleTrad {
  static int ic = 0;
  static Stack<Bloc> p = new Stack<Bloc>();
  public static void main(String[] args) {
   while(true) {
    switch(ic) {
    case 0: ...println("Entree main"); ic++; break;
    case 1: p.push(new Bloc(ic+1)); ic = 100; break;
    case 2: p.pop(); ic++; break;
    case 3: ...println("Sortie main"); ic++; break;
    case 4: System.exit(0);

    case 100: ...println("Entree f()"); ic++; break;
    case 101: p.push(new Bloc(ic+1)); ic = 200; break;
    case 102: p.pop(); ic++; break;
    case 103: ...println("Sortie f()"); ic++; break;
    case 104: ic = p.peek().getAdresse(); break;

    case 200: ...println("Je suis g()"); ic++; break;
    case 201: ic = p.peek().getAdresse(); break;
} } } }
```

Deuxième exemple de traduction d'un appel de fonction simple

```
public class AppelSimpleDeux {
  static int k = 1;

  public static void g() {
    System.out.println(k+ " éléphants se balançaient\n"
        + "Sur une toile, toile, toile\n"
        + "Toile d'araignée");
    k++;
  }

  public static void main(String[] args) {
    while (k <= 10 ) {
      g();
    }
    System.out.println("C'était un jeu tellement\n"
        + "Tellement amusant\n"
        + "Que tout à coup\n"
        + "Ba da boum");
  } }
```

Deuxième exemple de traduction d'un appel de fonction simple

```
public class AppelSimpleDeux {
  static int k = 1;             // 0: init mem[0]

  public static void g() {
    System.out.println(k+ " éléphants se balançaient\n"
        + "Sur une toile, toile, toile\n"
        + "Toile d'araignée");
    k++;
  }

  public static void main(String[] args) {
    while (k <= 10 ) {
      g();
    }
    System.out.println("C'était un jeu tellement\n"
        + "Tellement amusant\n"
        + "Que tout à coup\n"
        + "Ba da boum");
  } }
```

Deuxième exemple de traduction d'un appel de fonction simple

```
public class AppelSimpleDeux {
  static int k = 1;             // 0: init mem[0]

  public static void g() {
    System.out.println(k+ " éléphants se balançaient\n"
        + "Sur une toile, toile, toile\n"
        + "Toile d'araignée");
    k++;
  }

  public static void main(String[] args) {
    while (k <= 10 ) {          // 1: saut conditionnel
      g();
    }
    System.out.println("C'était un jeu tellement\n"
        + "Tellement amusant\n"
        + "Que tout à coup\n"
        + "Ba da boum");
  } }
```

Deuxième exemple de traduction d'un appel de fonction simple

```
public class AppelSimpleDeux {
  static int k = 1;            // 0: init mem[0]

  public static void g() {
    System.out.println(k+ " éléphants se balançaient\n"
        + "Sur une toile, toile, toile\n"
        + "Toile d'araignée");
    k++;
  }

  public static void main(String[] args) {
    while (k <= 10 ) {         // 1: saut conditionnel
      g();                     // 2: appel à g
                               // 3: point de retour après l'appel
    }
    System.out.println("C'était un jeu tellement\n"
        + "Tellement amusant\n"
        + "Que tout à coup\n"
        + "Ba da boum");
  } }
```

Deuxième exemple de traduction d'un appel de fonction simple

```
public class AppelSimpleDeux {
  static int k = 1;            // 0: init mem[0]

  public static void g() {
    System.out.println(k+ " éléphants se balançaient\n"
        + "Sur une toile, toile, toile\n"
        + "Toile d'araignée");
    k++;
  }

  public static void main(String[] args) {
    while (k <= 10 ) {         // 1: saut conditionnel
      g();                     // 2: appel à g
                               // 3: point de retour après l'appel
    }                          // 4: saut inconditionnel
    System.out.println("C'était un jeu tellement\n"
        + "Tellement amusant\n"
        + "Que tout à coup\n"
        + "Ba da boum");
  } }
```

Deuxième exemple de traduction d'un appel de fonction simple

```
public class AppelSimpleDeux {
  static int k = 1;            // 0: init mem[0]

  public static void g() {
    System.out.println(k+ " éléphants se balançaient\n"
        + "Sur une toile, toile, toile\n"
        + "Toile d'araignée");
    k++;
  }

  public static void main(String[] args) {
    while (k <= 10 ) {         // 1: saut conditionnel
      g();                     // 2: appel à g
                               // 3: point de retour après l'appel
    }                          // 4: saut inconditionnel
    System.out.println("C'était un jeu tellement\n"
        + "Tellement amusant\n"
        + "Que tout à coup\n"
        + "Ba da boum");       // 5: affichage
  } }                          // 6: sortie
```

Deuxième exemple de traduction d'un appel de fonction simple

```
public class AppelSimpleDeux {
  static int k = 1;            // 0: init mem[0]

  public static void g() {
    System.out.println(k+ " éléphants se balançaient\n"
        + "Sur une toile, toile, toile\n"
        + "Toile d'araignée"); // 100: affichage
    k++;
  }

  public static void main(String[] args) {
    while (k <= 10 ) {         // 1: saut conditionnel
      g();                     // 2: appel à g
                               // 3: point de retour après l'appel
    }                          // 4: saut inconditionnel
    System.out.println("C'était un jeu tellement\n"
        + "Tellement amusant\n"
        + "Que tout à coup\n"
        + "Ba da boum");       // 5: affichage
  } }                          // 6: sortie
```

Deuxième exemple de traduction d'un appel de fonction simple

```
public class AppelSimpleDeux {
  static int k = 1;            // 0: init mem[0]

  public static void g() {
    System.out.println(k+ " éléphants se balançaient\n"
        + "Sur une toile, toile, toile\n"
        + "Toile d'araignée"); // 100: affichage
    k++;                       // 101: modif mem[0]
  }

  public static void main(String[] args) {
    while (k <= 10 ) {         // 1: saut conditionnel
      g();                     // 2: appel à g
                               // 3: point de retour après l'appel
    }                          // 4: saut inconditionnel
    System.out.println("C'était un jeu tellement\n"
        + "Tellement amusant\n"
        + "Que tout à coup\n"
        + "Ba da boum");       // 5: affichage
  } }                          // 6: sortie
```

Deuxième exemple de traduction d'un appel de fonction simple

```
public class AppelSimpleDeux {
  static int k = 1;            // 0: init mem[0]

  public static void g() {
    System.out.println(k+ " éléphants se balançaient\n"
        + "Sur une toile, toile, toile\n"
        + "Toile d'araignée"); // 100: affichage
    k++;                       // 101: modif mem[0]
  }                            // 102: sortie de g

  public static void main(String[] args) {
    while (k <= 10 ) {         // 1: saut conditionnel
      g();                     // 2: appel à g
                               // 3: point de retour après l'appel
    }                          // 4: saut inconditionnel
    System.out.println("C'était un jeu tellement\n"
        + "Tellement amusant\n"
        + "Que tout à coup\n"
        + "Ba da boum");       // 5: affichage
  } }                          // 6: sortie
```

```java
public class AppelSimpleDeux {
  static int k = 1;   // 0: init mem[0]

  public static void g() {
    System.out.println(k+ " éléphants ...
    ↪ d'araignée"); // 100
    k++;          // 101: modif mem[0]
  }               // 102: sortie g

  public static void main(String[] args) {
    while (k <= 10 ) { // 1: saut cond.
      g();        // 2: appel à g
                  // 3: point de retour
    }             // 4: saut incond.
    System.out.println("C'était un jeu
    ↪ tellement\n"
        + "Tellement amusant\n"
        + "Que tout à coup\n"
        + "Ba da boum");  // 5: affichage
  } }             // 6: sortie
```

```java
import java.util.Stack;
public class AppelSimpleDeuxTrad {
  static int ic = 0; int[] mem = new int[1000];
  static Stack<Bloc> p = new Stack<Bloc>();
  public static void main(String[] args) {
   while(true) {
    switch(ic) {
    case 0: mem[0] = 1;  ic++; break;   // init. de k
    case 1:

    case 2:

    case 3:
    case 4:
    case 5:
    case 6:
    case 100:
    case 101:
    case 102:

} } } }
```

```java
public class AppelSimpleDeux {
  static int k = 1;   // 0: init mem[0]

  public static void g() {
    System.out.println(k+ " éléphants ...
    ↪ d'araignée"); // 100
    k++;          // 101: modif mem[0]
  }               // 102: sortie g

  public static void main(String[] args) {
    while (k <= 10 ) { // 1: saut cond.
      g();        // 2: appel à g
                  // 3: point de retour
    }             // 4: saut incond.
    System.out.println("C'était un jeu
    ↪ tellement\n"
        + "Tellement amusant\n"
        + "Que tout à coup\n"
        + "Ba da boum");  // 5: affichage
  } }             // 6: sortie
```

```java
import java.util.Stack;
public class AppelSimpleDeuxTrad {
  static int ic = 0; int[] mem = new int[1000];
  static Stack<Bloc> p = new Stack<Bloc>();
  public static void main(String[] args) {
   while(true) {
    switch(ic) {
    case 0: mem[0] = 1;  ic++; break;   // init. de k
    case 1: if(mem[0] > 10){ ic += 4;} else {ic++;};
            break;                //saut cond.
    case 2:

    case 3:
    case 4:
    case 5:
    case 6:
    case 100:
    case 101:
    case 102:

} } } }
```

```java
public class AppelSimpleDeux {
  static int k = 1;   // 0: init mem[0]

  public static void g() {
    System.out.println(k+ " éléphants ...
    ↪ d'araignée"); // 100
    k++;          // 101: modif mem[0]
  }               // 102: sortie g

  public static void main(String[] args) {
    while (k <= 10 ) { // 1: saut cond.
      g();        // 2: appel à g
                  // 3: point de retour
    }             // 4: saut incond.
    System.out.println("C'était un jeu
    ↪ tellement\n"
        + "Tellement amusant\n"
        + "Que tout à coup\n"
        + "Ba da boum");  // 5: affichage
  } }             // 6: sortie
```

```java
import java.util.Stack;
public class AppelSimpleDeuxTrad {
  static int ic = 0; int[] mem = new int[1000];
  static Stack<Bloc> p = new Stack<Bloc>();
  public static void main(String[] args) {
   while(true) {
    switch(ic) {
    case 0: mem[0] = 1;  ic++; break;   // init. de k
    case 1: if(mem[0] > 10){ ic += 4;} else {ic++;};
            break;                //saut cond.
    case 2: p.push(new Bloc(ic+1)); ic = 100; break;
            //empiler le  bloc d'act. + saut incond.
    case 3:
    case 4:
    case 5:
    case 6:
    case 100:
    case 101:
    case 102:

} } } }
```

```java
public class AppelSimpleDeux {
  static int k = 1;   // 0: init mem[0]

  public static void g() {
    System.out.println(k+ " éléphants ...
    ↪ d'araignée"); // 100
    k++;          // 101: modif mem[0]
  }               // 102: sortie g

  public static void main(String[] args) {
    while (k <= 10 ) { // 1: saut cond.
      g();        // 2: appel à g
                  // 3: point de retour
    }             // 4: saut incond.
    System.out.println("C'était un jeu
    ↪ tellement\n"
        + "Tellement amusant\n"
        + "Que tout à coup\n"
        + "Ba da boum");  // 5: affichage
  } }             // 6: sortie
```

```java
import java.util.Stack;
public class AppelSimpleDeuxTrad {
  static int ic = 0; int[] mem = new int[1000];
  static Stack<Bloc> p = new Stack<Bloc>();
  public static void main(String[] args) {
   while(true) {
    switch(ic) {
    case 0: mem[0] = 1;  ic++; break;   // init. de k
    case 1: if(mem[0] > 10){ ic += 4;} else {ic++;};
            break;                //saut cond.
    case 2: p.push(new Bloc(ic+1)); ic = 100; break;
            //empiler le  bloc d'act. + saut incond.
    case 3: p.pop(); ic++; break;// dépiler le bloc
    case 4:
    case 5:
    case 6:
    case 100:
    case 101:
    case 102:

} } } }
```

```java
public class AppelSimpleDeux {
  static int k = 1;   // 0: init mem[0]

  public static void g() {
    System.out.println(k+ " éléphants ...
    ↪ d'araignée"); // 100
    k++;          // 101: modif mem[0]
  }               // 102: sortie g

  public static void main(String[] args) {
    while (k <= 10 ) { // 1: saut cond.
      g();        // 2: appel à g
                  // 3: point de retour
    }             // 4: saut incond.
    System.out.println("C'était un jeu
    ↪ tellement\n"
        + "Tellement amusant\n"
        + "Que tout à coup\n"
        + "Ba da boum");  // 5: affichage
  } }             // 6: sortie
```

```java
import java.util.Stack;
public class AppelSimpleDeuxTrad {
  static int ic = 0; int[] mem = new int[1000];
  static Stack<Bloc> p = new Stack<Bloc>();
  public static void main(String[] args) {
   while(true) {
    switch(ic) {
    case 0: mem[0] = 1; ic++;  break;   // init. de k
    case 1: if(mem[0] > 10){ ic += 4;} else {ic++;};
            break;                //saut cond.
    case 2: p.push(new Bloc(ic+1)); ic = 100; break;
            //empiler le  bloc d'act. + saut incond.
    case 3: p.pop(); ic++; break;// dépiler le bloc
    case 4: ic -= 3; break;       //saut incond.
    case 5:
    case 6:
    case 100:
    case 101:
    case 102:

} } } }
```

```java
public class AppelSimpleDeux {
  static int k = 1;   // 0: init mem[0]

  public static void g() {
    System.out.println(k+ " éléphants ...
    ↪ d'araignée"); // 100
    k++;          // 101: modif mem[0]
  }               // 102: sortie g

  public static void main(String[] args) {
    while (k <= 10 ) { // 1: saut cond.
      g();        // 2: appel à g
                  // 3: point de retour
    }             // 4: saut incond.
    System.out.println("C'était un jeu
    ↪ tellement\n"
        + "Tellement amusant\n"
        + "Que tout à coup\n"
        + "Ba da boum");  // 5: affichage
  } }             // 6: sortie
```

```java
import java.util.Stack;
public class AppelSimpleDeuxTrad {
  static int ic = 0; int[] mem = new int[1000];
  static Stack<Bloc> p = new Stack<Bloc>();
  public static void main(String[] args) {
   while(true) {
    switch(ic) {
    case 0: mem[0] = 1;  ic++; break;   // init. de k
    case 1: if(mem[0] > 10){ ic += 4;} else {ic++;};
            break;                //saut cond.
    case 2: p.push(new Bloc(ic+1)); ic = 100; break;
            //empiler le  bloc d'act. + saut incond.
    case 3: p.pop(); ic++; break;// dépiler le bloc
    case 4: ic -= 3; break;       //saut incond.
    case 5: S.out.println("C'était ..."); ic++; break;
    case 6:
    case 100:
    case 101:
    case 102:

} } } }
```

```java
public class AppelSimpleDeux {
  static int k = 1;   // 0: init mem[0]

  public static void g() {
    System.out.println(k+ " éléphants ...
    ↪ d'araignée"); // 100
    k++;          // 101: modif mem[0]
  }               // 102: sortie g

  public static void main(String[] args) {
    while (k <= 10 ) { // 1: saut cond.
      g();        // 2: appel à g
                  // 3: point de retour
    }             // 4: saut incond.
    System.out.println("C'était un jeu
    ↪ tellement\n"
        + "Tellement amusant\n"
        + "Que tout à coup\n"
        + "Ba da boum");  // 5: affichage
  } }             // 6: sortie
```

```java
import java.util.Stack;
public class AppelSimpleDeuxTrad {
  static int ic = 0; int[] mem = new int[1000];
  static Stack<Bloc> p = new Stack<Bloc>();
  public static void main(String[] args) {
   while(true) {
    switch(ic) {
    case 0: mem[0] = 1;  ic++; break;   // init. de k
    case 1: if(mem[0] > 10){ ic += 4;} else {ic++;};
            break;               //saut cond.
    case 2: p.push(new Bloc(ic+1)); ic = 100; break;
            //empiler le  bloc d'act. + saut incond.
    case 3: p.pop(); ic++; break;// dépiler le bloc
    case 4: ic -= 3; break;      //saut incond.
    case 5: S.out.println("C'était ..."); ic++; break;
    case 6: System.exit(0);
    case 100:
    case 101:
    case 102:

} } } }
```

```java
public class AppelSimpleDeux {
  static int k = 1;   // 0: init mem[0]

  public static void g() {
    System.out.println(k+ " éléphants ...
    ↪ d'araignée"); // 100
    k++;          // 101: modif mem[0]
  }               // 102: sortie g

  public static void main(String[] args) {
    while (k <= 10 ) { // 1: saut cond.
      g();        // 2: appel à g
                  // 3: point de retour
    }             // 4: saut incond.
    System.out.println("C'était un jeu
    ↪ tellement\n"
        + "Tellement amusant\n"
        + "Que tout à coup\n"
        + "Ba da boum");  // 5: affichage
  } }             // 6: sortie
```

```java
import java.util.Stack;
public class AppelSimpleDeuxTrad {
  static int ic = 0; int[] mem = new int[1000];
  static Stack<Bloc> p = new Stack<Bloc>();
  public static void main(String[] args) {
   while(true) {
    switch(ic) {
    case 0: mem[0] = 1; ic++;  break;   // init. de k
    case 1: if(mem[0] > 10){ ic += 4;} else {ic++;};
            break;                //saut cond.
    case 2: p.push(new Bloc(ic+1)); ic = 100; break;
            //empiler le  bloc d'act. + saut incond.
    case 3: p.pop(); ic++; break;// dépiler le bloc
    case 4: ic -= 3; break;       //saut incond.
    case 5: S.out.println("C'était ..."); ic++; break;
    case 6: System.exit(0);
    case 100: S.o.println(mem[0]+"éléphants...");
     ↪ ic++; break;
    case 101:
    case 102:

} } } }
```

```java
public class AppelSimpleDeux {
  static int k = 1;   // 0: init mem[0]

  public static void g() {
    System.out.println(k+ " éléphants ...
    ↪ d'araignée"); // 100
    k++;          // 101: modif mem[0]
  }               // 102: sortie g

  public static void main(String[] args) {
    while (k <= 10 ) { // 1: saut cond.
      g();        // 2: appel à g
                  // 3: point de retour
    }             // 4: saut incond.
    System.out.println("C'était un jeu
    ↪ tellement\n"
        + "Tellement amusant\n"
        + "Que tout à coup\n"
        + "Ba da boum");  // 5: affichage
  } }             // 6: sortie
```

```java
import java.util.Stack;
public class AppelSimpleDeuxTrad {
  static int ic = 0; int[] mem = new int[1000];
  static Stack<Bloc> p = new Stack<Bloc>();
  public static void main(String[] args) {
   while(true) {
    switch(ic) {
    case 0: mem[0] = 1;  ic++; break;   // init. de k
    case 1: if(mem[0] > 10){ ic += 4;} else {ic++;};
            break;                //saut cond.
    case 2: p.push(new Bloc(ic+1)); ic = 100; break;
            //empiler le  bloc d'act. + saut incond.
    case 3: p.pop(); ic++; break;// dépiler le bloc
    case 4: ic -= 3; break;       //saut incond.
    case 5: S.out.println("C'était ..."); ic++; break;
    case 6: System.exit(0);
    case 100: S.o.println(mem[0]+"éléphants...");
     ↪  ic++; break;
    case 101: mem[0]++; ic++; break;
    case 102:

} } } }
```

```java
public class AppelSimpleDeux {
  static int k = 1;   // 0: init mem[0]

  public static void g() {
    System.out.println(k+ " éléphants ...
    ↪ d'araignée"); // 100
    k++;          // 101: modif mem[0]
  }               // 102: sortie g

  public static void main(String[] args) {
    while (k <= 10 ) { // 1: saut cond.
      g();        // 2: appel à g
                  // 3: point de retour
    }             // 4: saut incond.
    System.out.println("C'était un jeu
    ↪ tellement\n"
        + "Tellement amusant\n"
        + "Que tout à coup\n"
        + "Ba da boum");  // 5: affichage
  } }             // 6: sortie
```

```java
import java.util.Stack;
public class AppelSimpleDeuxTrad {
  static int ic = 0; int[] mem = new int[1000];
  static Stack<Bloc> p = new Stack<Bloc>();
  public static void main(String[] args) {
   while(true) {
    switch(ic) {
    case 0: mem[0] = 1; ic++; break;   // init. de k
    case 1: if(mem[0] > 10){ ic += 4;} else {ic++;};
            break;                //saut cond.
    case 2: p.push(new Bloc(ic+1)); ic = 100; break;
            //empiler le  bloc d'act. + saut incond.
    case 3: p.pop(); ic++; break;// dépiler le bloc
    case 4: ic -= 3; break;       //saut incond.
    case 5: S.out.println("C'était ..."); ic++; break;
    case 6: System.exit(0);
    case 100: S.o.println(mem[0]+"éléphants...");
     ↪ ic++; break;
    case 101: mem[0]++; ic++; break;
    case 102: ic = p.peek().getAdresse(); break;
             //sortie de g: saut incondionnel vers
              ↪ l'adresse de retour
```

Bloc d'activation pour les fonctions avec un paramètre

Nous considérons maintenant une classe modélisant le bloc d'activation d'une fonction avec un paramètre, mais sans valeur de retour ni variables locales.

```
public class BlocAvecPar {
    private int adresseRetour;
    private int param;

    // constructeur: l'appelant doit spécifier l'adresse de retour et la valeur du paramètre
    public BlocAvecPar(int adr, int param) {
      this.adresseRetour = adr;
      this.param = param;
    }

    // accesseurs utilisés par le code appelé
    public int getAdresse() {
      return this.adresseRetour;
    }
    public int getParam() {
      return this.param;
    }
```

Exemple de traduction d'un appel de fonction avec un paramètre

```
public class AppelSimple3 {

  public static void g(int k) {
    System.out.println(k+ " éléphants ...");
  }

  public static void main(String[] args) {
    int k = 1;          // 0: init mem[0]
    while (k <= 10 ) { // 1: saut incondionnel vers 6
      g(k++);



    }
    System.out.println("... ba da boum");
  }
}
```

Exemple de traduction d'un appel de fonction avec un paramètre

```
public class AppelSimple3 {

  public static void g(int k) {
    System.out.println(k+ " éléphants ...");
  }

  public static void main(String[] args) {
    int k = 1;          // 0: init mem[0]
    while (k <= 10 ) { // 1: saut incondionnel vers 6
      g(k++);           // 2: appel à g
                        // 3: retour
                        // 4: incrémentation de k
    }                   //
    System.out.println("... ba da boum");
  }
}
```

Exemple de traduction d'un appel de fonction avec un paramètre

```
public class AppelSimple3 {

  public static void g(int k) {
    System.out.println(k+ " éléphants ...");
  }

  public static void main(String[] args) {
    int k = 1;          // 0: init mem[0]
    while (k <= 10 ) { // 1: saut incondionnel vers 6
      g(k++);           // 2: appel à g
                        // 3: retour
                        // 4: incrémentation de k
    }                   // 5: saut inconditionnel vers 1
    System.out.println("... ba da boum"); // 6
  } // 7: sortie du programme
}
```

Exemple de traduction d'un appel de fonction avec un paramètre

```
public class AppelSimple3 {

  public static void g(int k) {
    System.out.println(k+ " éléphants ..."); // 100
  } //101: sortie du g, saut incondionnel vers l'adresse de retour

  public static void main(String[] args) {
    int k = 1;          // 0: init mem[0]
    while (k <= 10 ) { // 1: saut incondionnel vers 6
      g(k++);           // 2: appel à g
                        // 3: retour
                        // 4: incrémentation de k
    }                   // 5: saut inconditionnel vers 1
    System.out.println("... ba da boum"); // 6
  } // 7: sortie du programme
}
```

```java
public class AppelSimple3 {

  public static void g(int k) {
    System.out.println(k+ " éléphants ...");
    ↪ // 100
  } //101: sortie du g, saut incondionnel
  ↪ vers l'adresse de retour

  public static void main(String[] args) {
    int k = 1;          // 0: init mem[0]
    while (k <= 10 ) { // 1: saut
    ↪ incondionnel vers 6
      g(k++);           // 2: appel à g
                        // 3: retour
                        // 4: incrémentation
                        ↪ de k
    }                   // 5: saut
    ↪ inconditionnel vers 1
    System.out.println("... ba da boum"); //
    ↪ 6
  } // 7: sortie du programme
}
```

```java
import java.util.Stack;
public class AppelSimple3Trad {
  static int ic = 0; int [] mem = new int[1000];
  static Stack<BlocAvecPar> p = new
  ↪ Stack<BlocAvecPar>();
  public static void main(String[] args) {
   while(true) {
    switch(ic) {
    case 0: mem[0] = 1; ic++; break;   // init. de k
    case 1:

    case 2:

    case 3:
    case 4:
    case 5:
    case 6:
    case 7:
    case 100:
    case 101:
 } } } }
```

```java
public class AppelSimple3 {

  public static void g(int k) {
    System.out.println(k+ " éléphants ...");
    // 100
  } //101: sortie du g, saut incondionnel
  vers l'adresse de retour

  public static void main(String[] args) {
    int k = 1;          // 0: init mem[0]
    while (k <= 10 ) { // 1: saut
    incondionnel vers 6
      g(k++);           // 2: appel à g
                        // 3: retour
                        // 4: incrémentation
                        de k
    }                   // 5: saut
    inconditionnel vers 1
    System.out.println("... ba da boum"); //
    6
  } // 7: sortie du programme
}
```

```java
import java.util.Stack;
public class AppelSimple3Trad {
  static int ic = 0; int [] mem = new int[1000];
  static Stack<BlocAvecPar> p = new
  Stack<BlocAvecPar>();
  public static void main(String[] args) {
   while(true) {
    switch(ic) {
    case 0: mem[0] = 1; ic++; break;   // init. de k
    case 1: if(mem[0]>10) {ic+= 5;} else {ic++;};
     break;
    case 2:
    case 3:
    case 4:
    case 5:
    case 6:
    case 7:
    case 100:
    case 101:
 } } } }
```

```java
public class AppelSimple3 {

  public static void g(int k) {
    System.out.println(k+ " éléphants ...");
    ↪ // 100
  } //101: sortie du g, saut incondionnel
  ↪ vers l'adresse de retour

  public static void main(String[] args) {
    int k = 1;          // 0: init mem[0]
    while (k <= 10 ) { // 1: saut
    ↪ incondionnel vers 6
      g(k++);           // 2: appel à g
                        // 3: retour
                        // 4: incrémentation
                        ↪ de k
    }                   // 5: saut
    ↪ inconditionnel vers 1
    System.out.println("... ba da boum"); //
    ↪ 6
  } // 7: sortie du programme
}
```

```java
import java.util.Stack;
public class AppelSimple3Trad {
  static int ic = 0; int [] mem = new int[1000];
  static Stack<BlocAvecPar> p = new
  ↪ Stack<BlocAvecPar>();
  public static void main(String[] args) {
   while(true) {
    switch(ic) {
    case 0: mem[0] = 1;  ic++; break;   // init. de k
    case 1: if(mem[0]>10) {ic+= 5;} else {ic++;};
            break;
    case 2: p.push(new BlocAvecPar(ic+1, mem[0])); //
    ↪ on transmets l'@ de retour et la val. du
    ↪ paramètre,
            ic = 100;  break;  // puis saut
            ↪ inconditionnel
    case 3:
    case 4:
    case 5:
    case 6:
    case 7:
    case 100:
```

```java
public class AppelSimple3 {

  public static void g(int k) {
    System.out.println(k+ " éléphants ...");
    ↪ // 100
  } //101: sortie du g, saut incondionnel
  ↪ vers l'adresse de retour

  public static void main(String[] args) {
    int k = 1;          // 0: init mem[0]
    while (k <= 10 ) { // 1: saut
    ↪ incondionnel vers 6
      g(k++);           // 2: appel à g
                        // 3: retour
                        // 4: incrémentation
                        ↪ de k
    }                   // 5: saut
    ↪ inconditionnel vers 1
    System.out.println("... ba da boum"); //
    ↪ 6
  } // 7: sortie du programme
}
```

```java
import java.util.Stack;
public class AppelSimple3Trad {
  static int ic = 0; int [] mem = new int[1000];
  static Stack<BlocAvecPar> p = new
  ↪ Stack<BlocAvecPar>();
  public static void main(String[] args) {
   while(true) {
    switch(ic) {
    case 0: mem[0] = 1;  ic++; break;   // init. de k
    case 1: if(mem[0]>10) {ic+= 5;} else {ic++;};
            break;
    case 2: p.push(new BlocAvecPar(ic+1, mem[0]));
            ic = 100;  break;
    case 3: p.pop(); ic++;  break;
    case 4:
    case 5:
    case 6:
    case 7:
    case 100:
    case 101:
 } } } }
```

```java
public class AppelSimple3 {

  public static void g(int k) {
    System.out.println(k+ " éléphants ...");
    // 100
  } //101: sortie du g, saut incondionnel
  vers l'adresse de retour

  public static void main(String[] args) {
    int k = 1;          // 0: init mem[0]
    while (k <= 10 ) { // 1: saut
    incondionnel vers 6
      g(k++);           // 2: appel à g
                        // 3: retour
                        // 4: incrémentation
                        de k
    }                   // 5: saut
    inconditionnel vers 1
    System.out.println("... ba da boum"); //
    6
  } // 7: sortie du programme
}
```

```java
import java.util.Stack;
public class AppelSimple3Trad {
  static int ic = 0; int [] mem = new int[1000];
  static Stack<BlocAvecPar> p = new
  Stack<BlocAvecPar>();
  public static void main(String[] args) {
   while(true) {
    switch(ic) {
    case 0: mem[0] = 1;  ic++; break;   // init. de k
    case 1: if(mem[0]>10) {ic+= 5;} else {ic++;};
            break;
    case 2: p.push(new BlocAvecPar(ic+1, mem[0]));
            ic = 100;  break;
    case 3: p.pop(); ic++;  break;
    case 4: mem[0]++;  ic++;  break;
    case 5:
    case 6:
    case 7:
    case 100:
    case 101:
 } } } }
```

```java
public class AppelSimple3 {

  public static void g(int k) {
    System.out.println(k+ " éléphants ...");
    // 100
  } //101: sortie du g, saut incondionnel
  vers l'adresse de retour

  public static void main(String[] args) {
    int k = 1;          // 0: init mem[0]
    while (k <= 10 ) { // 1: saut
    incondionnel vers 6
      g(k++);           // 2: appel à g
                        // 3: retour
                        // 4: incrémentation
                        de k
    }                   // 5: saut
    inconditionnel vers 1
    System.out.println("... ba da boum"); //
    6
  } // 7: sortie du programme
}
```

```java
import java.util.Stack;
public class AppelSimple3Trad {
  static int ic = 0; int [] mem = new int[1000];
  static Stack<BlocAvecPar> p = new
  Stack<BlocAvecPar>();
  public static void main(String[] args) {
   while(true) {
    switch(ic) {
    case 0: mem[0] = 1;  ic++; break;   // init. de k
    case 1: if(mem[0]>10) {ic+= 5;} else {ic++;};
            break;
    case 2: p.push(new BlocAvecPar(ic+1, mem[0]));
            ic = 100;  break;
    case 3: p.pop(); ic++;  break;
    case 4: mem[0]++;  ic++;  break;
    case 5: ic -= 4;  break;
    case 6:
    case 7:
    case 100:
    case 101:
 } } } }
```

```java
public class AppelSimple3 {

  public static void g(int k) {
    System.out.println(k+ " éléphants ...");
    ↪ // 100
  } //101: sortie du g, saut incondionnel
  ↪ vers l'adresse de retour

  public static void main(String[] args) {
    int k = 1;          // 0: init mem[0]
    while (k <= 10 ) { // 1: saut
    ↪ inconditionnel vers 6
      g(k++);           // 2: appel à g
                        // 3: retour
                        // 4: incrémentation
                        ↪ de k
    }                   // 5: saut
    ↪ inconditionnel vers 1
    System.out.println("... ba da boum"); //
    ↪ 6
  } // 7: sortie du programme
}
```

```java
import java.util.Stack;
public class AppelSimple3Trad {
  static int ic = 0; int [] mem = new int[1000];
  static Stack<BlocAvecPar> p = new
  ↪ Stack<BlocAvecPar>();
  public static void main(String[] args) {
   while(true) {
    switch(ic) {
    case 0: mem[0] = 1;  ic++; break;   // init. de k
    case 1: if(mem[0]>10) {ic+= 5;} else {ic++;};
            break;
    case 2: p.push(new BlocAvecPar(ic+1, mem[0]));
            ic = 100;  break;
    case 3: p.pop(); ic++;  break;
    case 4: mem[0]++;  ic++;  break;
    case 5: ic -= 4;  break;
    case 6: System.out.println("... ba da boum");
     ↪ ic++;  break;
    case 7: System.exit(0);
    case 100:
    case 101:
 } } } }
```

```java
public class AppelSimple3 {

  public static void g(int k) {
    System.out.println(k+ " éléphants ...");
    // 100
  } //101: sortie du g, saut incondionnel
  vers l'adresse de retour

  public static void main(String[] args) {
    int k = 1;          // 0: init mem[0]
    while (k <= 10 ) { // 1: saut
    incondionnel vers 6
      g(k++);           // 2: appel à g
                        // 3: retour
                        // 4: incrémentation
                        de k
    }                   // 5: saut
    inconditionnel vers 1
    System.out.println("... ba da boum"); //
    6
  } // 7: sortie du programme
}
```

```java
import java.util.Stack;
public class AppelSimple3Trad {
  static int ic = 0; int [] mem = new int[1000];
  static Stack<BlocAvecPar> p = new
  Stack<BlocAvecPar>();
  public static void main(String[] args) {
   while(true) {
    switch(ic) {
    case 0: mem[0] = 1;  ic++; break;   // init. de k
    case 1: if(mem[0]>10) {ic+= 5;} else {ic++;};
            break;
    case 2: p.push(new BlocAvecPar(ic+1, mem[0]));
            ic = 100;  break;
    case 3: p.pop(); ic++;  break;
    case 4: mem[0]++;  ic++;  break;
    case 5: ic -= 4;  break;
    case 6: System.out.println("... ba da boum");
     ic++;  break;
    case 7: System.exit(0);
    case 100: System.out.println(p.peek().getParam()+"
     éléphants ... "); ic++;break;
    case 101:
```

```java
public class AppelSimple3 {

  public static void g(int k) {
    System.out.println(k+ " éléphants ...");
    // 100
  } //101: sortie du g, saut incondionnel
  vers l'adresse de retour

  public static void main(String[] args) {
    int k = 1;          // 0: init mem[0]
    while (k <= 10 ) { // 1: saut
    incondionnel vers 6
      g(k++);           // 2: appel à g
                        // 3: retour
                        // 4: incrémentation
                        de k
    }                   // 5: saut
    inconditionnel vers 1
    System.out.println("... ba da boum"); //
    6
  } // 7: sortie du programme
}
```

```java
import java.util.Stack;
public class AppelSimple3Trad {
  static int ic = 0; int [] mem = new int[1000];
  static Stack<BlocAvecPar> p = new
  Stack<BlocAvecPar>();
  public static void main(String[] args) {
   while(true) {
    switch(ic) {
    case 0: mem[0] = 1; ic++; break;   // init. de k
    case 1: if(mem[0]>10) {ic+= 5;} else {ic++;};
            break;
    case 2: p.push(new BlocAvecPar(ic+1, mem[0]));
            ic = 100;  break;
    case 3: p.pop(); ic++;  break;
    case 4: mem[0]++;  ic++;  break;
    case 5: ic -= 4;  break;
    case 6: System.out.println("... ba da boum");
     ic++;  break;
    case 7: System.exit(0);
    case 100: System.out.println(p.peek().getParam()+"
     éléphants ... "); ic++;break;
    case 101: ic = p.peek().getAdresse(); break;
```