[CI2] Cours 6: Traduction de programmes III

Daniela Petrișan
Université de Paris, IRIF

Rappel ...

Nous poursuivons la traduction de programmes et nous nous intéressons aux appels de fonctions avec des paramètres, des variables locales et une valeur de retour.

Rappel ...

Nous poursuivons la traduction de programmes et nous nous intéressons aux appels de fonctions avec des paramètres, des variables locales et une valeur de retour.

Dans le cours précédent, on a fait la traduction d'un appel de fonction simple, pour une fonction avec un paramètre, mais sans variables locales et dont le type de retour est `void`.

```
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 ....
}
```

Rappel : 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

Rappel : 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.

Dans ce cours : Nous introduisons des classes modélisant des blocs plus généraux. Nous avons également besoin de quelques notions d'héritage.

Factorielle

```
class Facto{
  static int f(int n){
    int i=1, r=1;
    while(i<=n){
      r = r*i;
      i = i+1;
    }
    return r;
  }

  public static void main(String[] args){
    int x=4;
    System.out.println("resultat : "+f(x));

  }
}
```

Factorielle

```
// attention: on adopte  la convention où c'est l'appelant qui dépile
class Facto{
  static int f(int n){
    int i=1, r=1;        // 500 màj variable1
                         // 501 màj variable2
    while(i<=n){         // 502 saut conditionnel
      r = r*i;           // 503 màj de variable2
      i = i+1;           // 504 màj de variable1
    }                    // 505 saut inconditionnel
    return r;            // 506 màj valeur de retour
  }                      // 507 retour

  public static void main(String[] args){
    int x=4;                          // 0: màj de mem[0]
    System.out.println("resultat : "+f(x)); // 1: appel
                                              // 2: retour
  }                       //3: sortie du main/programme
}
```

Bloc spécifique à la fonction f

```
class BlocF{
  // attributs (environnement de f)
  private final int adresse_retour;
  private int valeur_retour;
  private int argument;
  private int variable1, variable2;
  // constructeurs
    public BlocF(int adr, int arg){
      /* l'appelant doit spécifier l'adresse de
       * retour et les valeurs des arguments
       */
      this.adresse_retour=adr;
      this.argument=arg;
    }

    // accesseur utilisé par le code appelant
    public int getVal(){
        return this.valeur_retour;
    }
    // getters et setters utilisés par le code
     ↪  appelé
    public int getVar1(){
        return this.variable1;
    }
    public int getVar2(){
        return this.variable2;
    }
    public int getArg(){
        return this.argument;
    }
    public int getAdr(){
        return this.adresse_retour;
    }
    public void setVar1(int v){
        this.variable1=v;
    }
    public void setVar2(int v){
        this.variable2=v;
    }
    public void setVal(int v){
        this.valeur_retour=v;
    }
```

Factorielle traduite

```
class Facto{
  static int f(int n){
    int i=1, r=1;          // 500 màj variable1
                           // 501 màj variable2
    while(i<=n){           // 502 saut conditionnel
      r = r*i;             // 503 màj de variable2
      i = i+1;             // 504 màj de variable1
    }                      // 505 saut
    ↪ inconditionnel
    return r;              // 506 màj valeur de
    ↪ retour
  }                        // 507 retour

  public static void main(String[] args){
    int x=4;               // 0: màj de mem[0]
    System.out.println("resultat : "+f(x)); // 1
                           // 2: retour
  }                        //3: sortie du
  ↪ main/programme
}
```

```
class FactoTraduit{
  public static void main(String[] args){
    int[] mem=new int[1000];
    int ic=0;
    Stack<BlocF> p=new Stack<BlocF>();  //pile
    ↪ d'appel
    while(true){
      switch(ic){
        case 0:
            mem[0]=4;
            ic++; break;
        case 1:


        case 2:


        case 3:
        //à écrire les cas 500 -- 507
```

Factorielle traduite

```
class Facto{
  static int f(int n){
    int i=1, r=1;          // 500 màj variable1
                           // 501 màj variable2
    while(i<=n){           // 502 saut conditionnel
      r = r*i;             // 503 màj de variable2
      i = i+1;             // 504 màj de variable1
    }                      // 505 saut
    ↪ inconditionnel
    return r;              // 506 màj valeur de
    ↪ retour
  }                        // 507 retour

  public static void main(String[] args){
    int x=4;               // 0: màj de mem[0]
    System.out.println("resultat : "+f(x)); // 1
                           // 2: retour
  }                        //3: sortie du
  ↪ main/programme
}
```

```
class FactoTraduit{
  public static void main(String[] args){
    int[] mem=new int[1000];
    int ic=0;
    Stack<BlocF> p=new Stack<BlocF>();  //pile
    ↪ d'appel
    while(true){
      switch(ic){
        case 0:
            mem[0]=4;
            ic++; break;
        case 1:
            p.push(new BlocF(ic+1,mem[0]));
            ic=500; break;
        case 2:




        case 3:
        //à écrire les cas 500 -- 507
```

Factorielle traduite

```
class Facto{
  static int f(int n){
    int i=1, r=1;          // 500 màj variable1
                           // 501 màj variable2
    while(i<=n){           // 502 saut conditionnel
      r = r*i;             // 503 màj de variable2
      i = i+1;             // 504 màj de variable1
    }                      // 505 saut
    ↪ inconditionnel
    return r;              // 506 màj valeur de
    ↪ retour
  }                        // 507 retour

  public static void main(String[] args){
    int x=4;               // 0: màj de mem[0]
    System.out.println("resultat : "+f(x)); // 1
                           // 2: retour
  }                        //3: sortie du
  ↪ main/programme
}
```

```
class FactoTraduit{
  public static void main(String[] args){
    int[] mem=new int[1000];
    int ic=0;
    Stack<BlocF> p=new Stack<BlocF>();  //pile
    ↪ d'appel
    while(true){
      switch(ic){
        case 0:
            mem[0]=4;
            ic++; break;
        case 1:
            p.push(new BlocF(ic+1,mem[0]));
            ic=500; break;
        case 2:
            System.out.println("resultat : " +
            ↪ p.pop().getVal());
            ic++; break;
        case 3:
        //à écrire les cas 500 -- 507
```

Factorielle traduite

```
class Facto{
  static int f(int n){
    int i=1, r=1;          // 500 màj variable1
                           // 501 màj variable2
    while(i<=n){           // 502 saut conditionnel
      r = r*i;             // 503 màj de variable2
      i = i+1;             // 504 màj de variable1
    }                      // 505 saut
    ↪ inconditionnel
    return r;              // 506 màj valeur de
    ↪ retour
  }                        // 507 retour

  public static void main(String[] args){
    int x=4;               // 0: màj de mem[0]
    System.out.println("resultat : "+f(x)); // 1
                           // 2: retour
  }                        //3: sortie du
  ↪ main/programme
}
```

```
class FactoTraduit{
  public static void main(String[] args){
    int[] mem=new int[1000];
    int ic=0;
    Stack<BlocF> p=new Stack<BlocF>();  //pile
    ↪ d'appel
    while(true){
      switch(ic){
        case 0:
            mem[0]=4;
            ic++; break;
        case 1:
            p.push(new BlocF(ic+1,mem[0]));
            ic=500; break;
        case 2:
            System.out.println("resultat : " +
            ↪ p.pop().getVal());
            ic++; break;
        case 3: System.exit(0);
        //à écrire les cas 500 -- 507
```

Factorielle traduite

```
class Facto{
  static int f(int n){
    int i=1, r=1;          // 500 màj variable1
                           // 501 màj variable2
    while(i<=n){           // 502 saut conditionnel
      r = r*i;             // 503 màj de variable2
      i = i+1;             // 504 màj de variable1
    }                      // 505 saut
    ↪ inconditionnel
    return r;              // 506 màj valeur de
    ↪ retour
  }                        // 507 retour

  public static void main(String[] args){
    int x=4;               // 0: màj de mem[0]
    System.out.println("resultat : "+f(x)); // 1
                           // 2: retour
  }                        //3: sortie du
  ↪ main/programme
}
```

```
class FactoTraduit{
  ...
  case 500: p.peek().setVar1(1); ic++; break;
  case 501:
  case 502:



  case 503:



  case 504:



  case 505:
  case 506:

  case 507:



  } } } }
```

Factorielle traduite

```
class Facto{
  static int f(int n){
    int i=1, r=1;          // 500 màj variable1
                           // 501 màj variable2
    while(i<=n){           // 502 saut conditionnel
      r = r*i;             // 503 màj de variable2
      i = i+1;             // 504 màj de variable1
    }                      // 505 saut
    ↪ inconditionnel
    return r;              // 506 màj valeur de
    ↪ retour
  }                        // 507 retour

  public static void main(String[] args){
    int x=4;               // 0: màj de mem[0]
    System.out.println("resultat : "+f(x)); // 1
                           // 2: retour
  }                        //3: sortie du
  ↪ main/programme
}
```

```
class FactoTraduit{
  ...
  case 500: p.peek().setVar1(1); ic++; break;
  case 501: p.peek().setVar2(1); ic++; break;
  case 502:



  case 503:



  case 504:



  case 505:
  case 506:

  case 507:



  } } } }
```

Factorielle traduite

```
class Facto{
  static int f(int n){
    int i=1, r=1;          // 500 màj variable1
                           // 501 màj variable2
    while(i<=n){           // 502 saut conditionnel
      r = r*i;             // 503 màj de variable2
      i = i+1;             // 504 màj de variable1
    }                      // 505 saut
    ↪ inconditionnel
    return r;              // 506 màj valeur de
    ↪ retour
  }                        // 507 retour

  public static void main(String[] args){
    int x=4;               // 0: màj de mem[0]
    System.out.println("resultat : "+f(x)); // 1
                           // 2: retour
  }                        //3: sortie du
  ↪ main/programme
}
```

```
class FactoTraduit{
  ...
  case 500: p.peek().setVar1(1); ic++; break;
  case 501: p.peek().setVar2(1); ic++; break;
  case 502:
      if(p.peek().getVar1()>p.peek().getArg()) {
      ↪ ic+=4; } else { ic++; } break;
  case 503:



  case 504:



  case 505:
  case 506:

  case 507:



  } } } }
```

Factorielle traduite

```
class Facto{
  static int f(int n){
    int i=1, r=1;          // 500 màj variable1
                           // 501 màj variable2
    while(i<=n){           // 502 saut conditionnel
      r = r*i;             // 503 màj de variable2
      i = i+1;             // 504 màj de variable1
    }                      // 505 saut
    ↪ inconditionnel
    return r;              // 506 màj valeur de
    ↪ retour
  }                        // 507 retour

  public static void main(String[] args){
    int x=4;               // 0: màj de mem[0]
    System.out.println("resultat : "+f(x)); // 1
                           // 2: retour
  }                        //3: sortie du
  ↪ main/programme
}
```

```
class FactoTraduit{
  ...
  case 500: p.peek().setVar1(1); ic++; break;
  case 501: p.peek().setVar2(1); ic++; break;
  case 502:
      if(p.peek().getVar1()>p.peek().getArg()) {
      ↪ ic+=4; } else { ic++; } break;
  case 503:
      p.peek().setVar2(p.peek().getVar2()*
        p.peek().getVar1()); ic++; break;
  case 504:



  case 505:
  case 506:

  case 507:



  } } } }
```

Factorielle traduite

```
class Facto{
  static int f(int n){
    int i=1, r=1;          // 500 màj variable1
                           // 501 màj variable2
    while(i<=n){           // 502 saut conditionnel
      r = r*i;             // 503 màj de variable2
      i = i+1;             // 504 màj de variable1
    }                      // 505 saut
    ↪ inconditionnel
    return r;              // 506 màj valeur de
    ↪ retour
  }                        // 507 retour

  public static void main(String[] args){
    int x=4;               // 0: màj de mem[0]
    System.out.println("resultat : "+f(x)); // 1
                           // 2: retour
  }                        //3: sortie du
  ↪ main/programme
}
```

```
class FactoTraduit{
  ...
  case 500: p.peek().setVar1(1); ic++; break;
  case 501: p.peek().setVar2(1); ic++; break;
  case 502:
      if(p.peek().getVar1()>p.peek().getArg()) {
      ↪ ic+=4; } else { ic++; } break;
  case 503:
      p.peek().setVar2(p.peek().getVar2()*
        p.peek().getVar1()); ic++; break;
  case 504:
      p.peek().setVar1(p.peek().getVar1()+1);
      ↪ ic++; break;
  case 505:
  case 506:

  case 507:



  } } } }
```

Factorielle traduite

```
class Facto{
  static int f(int n){
    int i=1, r=1;          // 500 màj variable1
                           // 501 màj variable2
    while(i<=n){           // 502 saut conditionnel
      r = r*i;             // 503 màj de variable2
      i = i+1;             // 504 màj de variable1
    }                      // 505 saut
    ↪ inconditionnel
    return r;              // 506 màj valeur de
    ↪ retour
  }                        // 507 retour

  public static void main(String[] args){
    int x=4;               // 0: màj de mem[0]
    System.out.println("resultat : "+f(x)); // 1
                           // 2: retour
  }                        //3: sortie du
  ↪ main/programme
}
```

```
class FactoTraduit{
  ...
  case 500: p.peek().setVar1(1); ic++; break;
  case 501: p.peek().setVar2(1); ic++; break;
  case 502:
      if(p.peek().getVar1()>p.peek().getArg()) {
      ↪ ic+=4; } else { ic++; } break;
  case 503:
      p.peek().setVar2(p.peek().getVar2()*
        p.peek().getVar1()); ic++; break;
  case 504:
      p.peek().setVar1(p.peek().getVar1()+1);
      ↪ ic++; break;
  case 505: ic-=3; break;
  case 506:

  case 507:



  } } } }
```

Factorielle traduite

```
class Facto{
  static int f(int n){
    int i=1, r=1;          // 500 màj variable1
                           // 501 màj variable2
    while(i<=n){           // 502 saut conditionnel
      r = r*i;             // 503 màj de variable2
      i = i+1;             // 504 màj de variable1
    }                      // 505 saut
    ↪ inconditionnel
    return r;              // 506 màj valeur de
    ↪ retour
  }                        // 507 retour

  public static void main(String[] args){
    int x=4;               // 0: màj de mem[0]
    System.out.println("resultat : "+f(x)); // 1
                           // 2: retour
  }                        //3: sortie du
  ↪ main/programme
}
```

```
class FactoTraduit{
  ...
  case 500: p.peek().setVar1(1); ic++; break;
  case 501: p.peek().setVar2(1); ic++; break;
  case 502:
      if(p.peek().getVar1()>p.peek().getArg()) {
      ↪ ic+=4; } else { ic++; } break;
  case 503:
      p.peek().setVar2(p.peek().getVar2()*
        p.peek().getVar1()); ic++; break;
  case 504:
      p.peek().setVar1(p.peek().getVar1()+1);
      ↪ ic++; break;
  case 505: ic-=3; break;
  case 506: p.peek().setVal(p.peek().getVar2());
  ↪ ic++; break;
  case 507:



  } } } }
```

Factorielle traduite

```
class Facto{
  static int f(int n){
    int i=1, r=1;          // 500 màj variable1
                           // 501 màj variable2
    while(i<=n){           // 502 saut conditionnel
      r = r*i;             // 503 màj de variable2
      i = i+1;             // 504 màj de variable1
    }                      // 505 saut
    ↪ inconditionnel
    return r;              // 506 màj valeur de
    ↪ retour
  }                        // 507 retour

  public static void main(String[] args){
    int x=4;               // 0: màj de mem[0]
    System.out.println("resultat : "+f(x)); // 1
                           // 2: retour
  }                        //3: sortie du
  ↪ main/programme
}
```

```
class FactoTraduit{
  ...
  case 500: p.peek().setVar1(1); ic++; break;
  case 501: p.peek().setVar2(1); ic++; break;
  case 502:
      if(p.peek().getVar1()>p.peek().getArg()) {
      ↪ ic+=4; } else { ic++; } break;
  case 503:
      p.peek().setVar2(p.peek().getVar2()*
        p.peek().getVar1()); ic++; break;
  case 504:
      p.peek().setVar1(p.peek().getVar1()+1);
      ↪ ic++; break;
  case 505: ic-=3; break;
  case 506: p.peek().setVal(p.peek().getVar2());
  ↪ ic++; break;
  case 507: ic=p.peek().getAdr(); break;
  //on utilise l'adresse de retour en maintenant
  ↪ le bloc sur la pile
  } } } }
```

Translations of general functions

voir les fichiers java....