Université Paris Cité

L1 informatique - Année 2022–2023

TD de Concepts Informatique n° 7

Dans l'Exercice 2 ci-dessous, nous utiliserons les classes `Bloc`, `BlocR`, `BlocF` et `BlocG`.

- La classe `Bloc` est utilisée pour représenter un bloc d'activation minimal pour une fonction `void foo()` avec type de retour `void`, sans paramètres et sans variables locales.
- La classe `BlocR` est utilisée pour modéliser un bloc d'activation pour une fonction de la forme `int foo()` sans variables locales. Rappelons que `BlocR` peut être vu comme un sous-type de `Bloc` dont il hérite l'adresse de retour.
- Remarquez que la classe `BlocF` est utilisé pour modéliser un bloc d'activation pour les appels de la fonction `f` de l'Exercice 2. En plus de l'adresse de retour (héritée de la classe `Bloc`), cette classe possède également des attributs pour un argument et une variable locale, tous deux de type `int`.
- De même, la classe `BlocG` est utilisée pour modéliser un bloc d'activation pour les appels de la fonction `g` de l'Exercice 2. En plus de la valeur de retour de type `int` et de l'adresse de retour (qui sont héritées de `BlocR`), cette classe possède également des attributs pour deux arguments de type `int`.

```
public class Bloc {
  private int adresseRetour;

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

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

```
public class BlocR extends Bloc{
  private int valeurDeRetour;

  public BlocR(int adresseDeRetour) {
    super(adresseDeRetour);
  }

  public int getVal() {
    return this.valeurDeRetour;
  }

  public void setVal(int valeurDeRetour) {
    this.valeurDeRetour = valeurDeRetour;
  }
}
```

```
public class BlocF extends Bloc{
  private int argument1;
  private int variable1;

  public BlocF(int adr, int arg1) {
    super(adr);
    this.argument1 = arg1;
  }

  public int getArg1() {
    return this.argument1;
  }

  public int getVar1() {
    return this.variable1;
  }

  public void setVar1(int variable1) {
    this.variable1 = variable1;
  }
}
```

```
public class BlocG extends BlocR{
  private int argument1;
  private int argument2;

  public BlocG(int adr, int arg1, int arg2) {
    super(adr);
    this.argument1 = arg1;
    this.argument2 = arg2;
  }

  public int getArg1() {
    return this.argument1;
  }

  public int getArg2() {
    return this.argument2;
  }
}
```

Exercice 1. *Blocs d'activation.*

1. Donner un diagramme pour représenter la relation de sous-typage entre ces classes.
2. Lesquelles des lignes suivantes compilent ?

   ```
   Bloc b1 = new BlocF(2, 7);
   BlocF b2 = new Bloc(2);
   BlocF b3 = (BlocF)b1;
   BlocR b4 = (BlocR)b1;
   ```

3. Nous allons modéliser la pile d'exécution avec un objet de type `Stack<Bloc>`. Justifier ce choix.
4. Pour chaque méthode dans les classes ci-dessus, précisez si elle est utilisée par le code appelant ou le code appelé et justifiez.

Exercice 2. *Appel général.*
On considére le programme suivant :

```
public class AppelFG {
  static int a = 2;
  static int b = 1;

  public static void f(int i) {
    int tmp = 0;
    if (i % 2 == 0) {
      a = 3 * a;
      b = 2 * b;
    } else {
      tmp = a;
      a = b;
      b = tmp;
    }
  }

  public static int g(int x, int y) {
    return x * a + y * b;
  }

  public static void main(String[] args) {
    int [] t = {2,3,3,4};
    for (int i = 0; i < 4; i++) {
      f(t[i]);
    }
    System.out.println(g(1,6));
  }
}
```

1. Que affiche le programme ?
2. Annoter le code.
3. Traduire le programme.
4. Décrire l'évolution de la pile d'appel (chaque `push`, chaque `pop`).