Double branchement

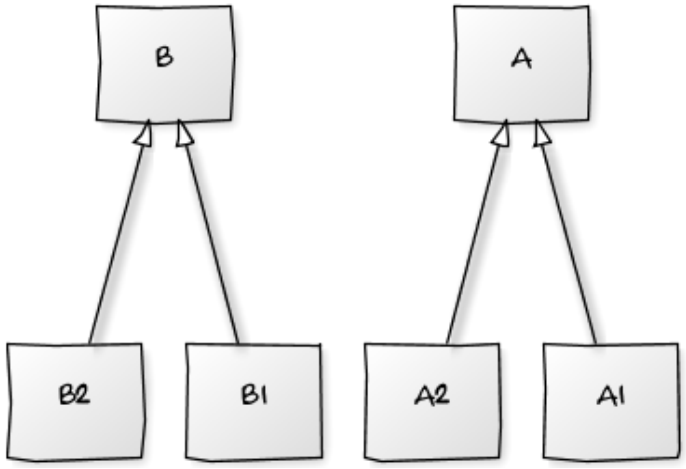
Le problème

On peut se trouver dans la situation suivante :
Une action doit être exécutée différemment suivant la valeur de deux arguments.
Exemples fréquents :

- animation/représentation d'une collision entre des objets de types différents ;
- représentation de recouvrements dans un logiciel de dessin
- gestion des évènements : plusieurs types d'évènements et plusieurs types de receveurs des évènements.

Attention, le type des arguments n'est connu qu'à l'exécution. Le traitement est donc forcément dynamique et ne peut pas reposer sur de la redéfinition de fonction.

Double branchement




```
class X {
   public static void main (String args) {
      X x = new X().
      x.faire (new A1(), new B2());
      x.faire (new A2(), new B2());
      x.faire (new A1(), new B1());
   }
   void faire (A a, B b) { ...
```

Double branchement

une solution : on peut questionner le type des arguments avec instanceof :

```
class X {
   void faire (A a, B b) {
      if (a instanceof A1 && b instanceof B1)    { System.out.println("A1 – B1"); }
      else if (a instanceof A1 && b instanceof B2)   { System.out.println("A1 – B2"); }
      else if (a instanceof A2 && b instanceof B1)   { System.out.println("A2 – B1"); }
      else if (a instanceof A2 && b instanceof B2)   { System.out.println("A2 – B2"); }
   }
}
```

Etape 1 : Début de solution

On élimine l'inconnue sur le 1er argument :

```
class A { abstract void faire (B b); }

class A1 extends A {
   void faire (B b) {
      if (b instanceof B1) { System.out.println("A1 – B1"); }
      else { System.out.println("A1 – B2"); }
   }
}
class A2 extends A {
   void faire (B b) {
      if (b instanceof B1) { System.out.println("A2 – B1"); }
      else { System.out.println("A2 – B2"); }
   }
}

class Y {
   void faire (A a, B b) {
      a.faire(b);
   }
}
```

Etape 2 : Solution complète

Puis, on élimine l'inconnue sur le 2ème argument :

```
class A { abstract void faire (B b); }
class B { abstract void faireA1 (A1 a); abstract void faireA2 (A2 a);}

class B1 extends B {
   void faireA1 (A1 a1) {
      { System.out.println("A1 – B1"); }
   }
   void faireA2 (A2 a1) {
      { System.out.println("A2 – B1"); }
   }
}

class B2 extends B {
   void faireA1 (A1 a1) {
      { System.out.println("A1 – B2"); }
   }
   void faireA2 (A2 a1) {
      { System.out.println("A2 – B2"); }
   }
}
```

Exemple concret : drag&drop sur un bureau

Sur un bureau virtuel :

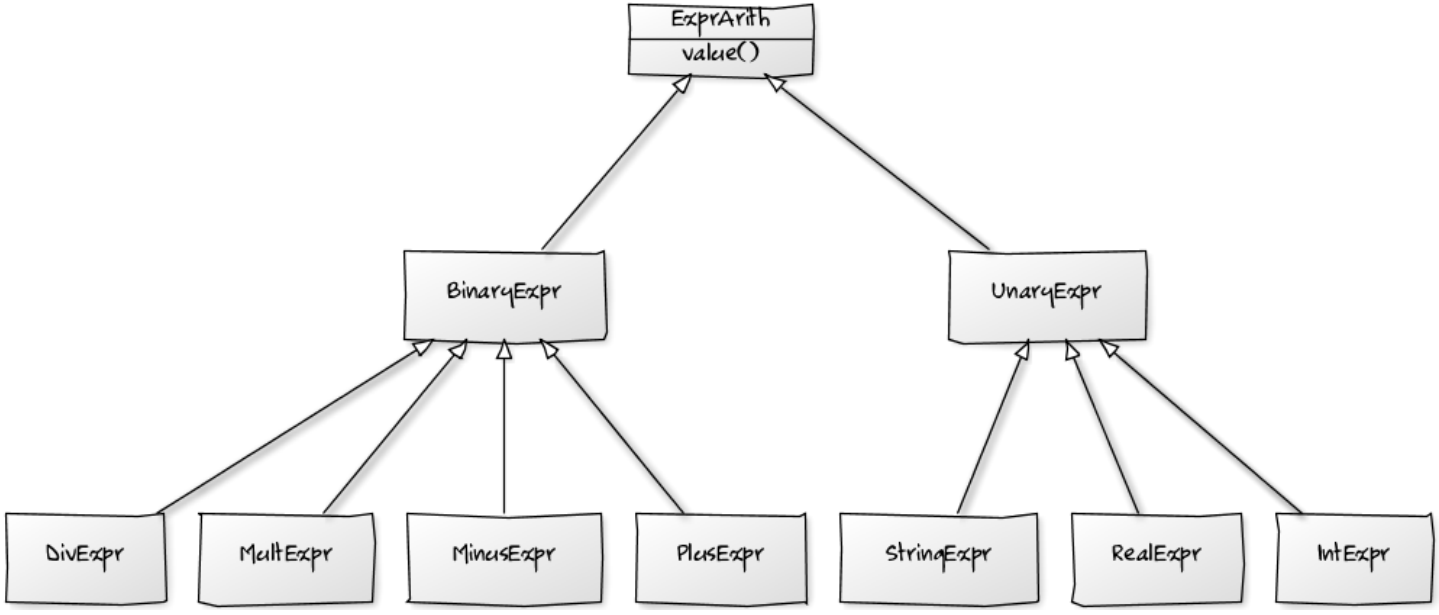
- On a toute sorte d'objet : fichier, répertoire, périphérique ...
- On a toute sorte de receveur pour un "drop" : application, répertoire, périphérique ...

On veut pouvoir glisser un fichier dans un répertoire, une imprimante, la poubelle ...
On ne veut surtout pas avoir à tester le type des objets (gros "switch"), on veut pouvoir ajouter des objets facilement

drag&drop sur un bureau

```
interface DropReceiver {
   void receiveFile(File f);
   void receiveDir(Directory d);
   void receiveDevice(Device d);
}
class Trash implements DropReceiver {
   void receiveFile(File f) { f.moveToTrash(this); }
   void receiveDir(Directory d) { d.moveToTrash(this); }
   void receiveDevice(Directory d) { /* Ignored */ }
}
class Printer implements DropReceiver {
   void receiveFile(File f) { f.printWithPrinter(this); }
   void receiveDir(Directory d) { /* Ignored */ }
   void receiveDevice(Directory d) { /* Ignored */ }
}
class File {
   void dropInto(DropReceiver dr) { dr.receiveFile(this); }
}
class Desktop {
   void dropInto(Dropable d, DropReceiver dr) { d.dropInto(dr); }
}
```

Calcul d'une expression arithmétique



- pour calculer la valeur d'une expression : envoi du message *value*
- quelle solution pour ?:
  - 1 + 'Bibi'
  - 'Bibi' + 1
  - 1.0 + 2
  - 2 + 1.0
  - 'bibi' + 2.0

Calcul d'une expression arithmétique

```
IntExpr>>value
   ^self
StringExpr>>value
   ^self
RealExpr>>value
   ^self

PlusExpr>>value
   ^self left value plus : self right value
IntExpr>>plus : arg
   ^arg plusInt : self
StringExpr>>plus : arg
   ^StringExpr newWith : (self str, arg asStringExpr str)
StringExpr>>plusInt : anIntExpr
   ^StringExpr newVith : anIntExpr asStringExpr str
```