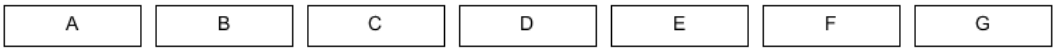
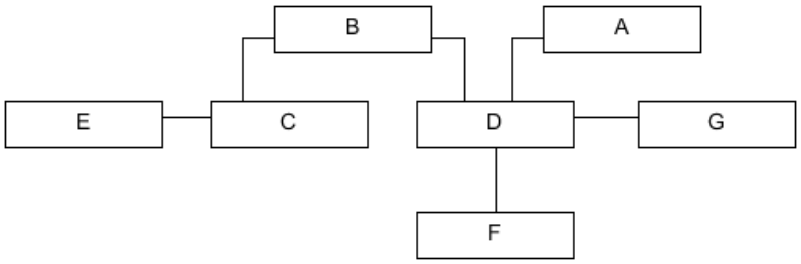
UML Flow – INFO-H-200 – ULB

Énoncé : B est un objet qui […], C et D sont des B. D possède un comportement A. C peut effectuer une action sur 3 E, et les E peuvent communiquer avec 0 à n C. D par contre possède autant de F qu'il en a besoin qui eux sont possédés par 3 D précisément. D est aussi composé par un G.

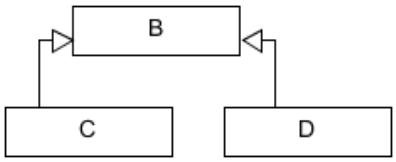
1) Déterminer les objets, en utilisant l'énoncé du problème



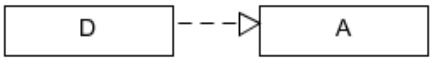
2) Déterminer ceux qui ont un lien, one dessine des lignes entre les objets qui possèdent un lien



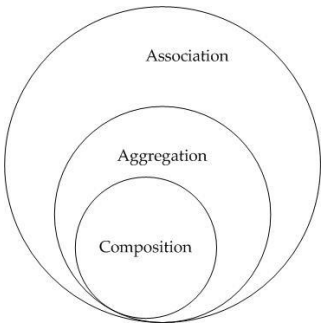
3) Déterminer le type de lien :
   a. Est-on dans une relation de parent enfant (héritage) ? ou de comportements (implémente) ?
      i. Relation **« C est un B »**, Héritage



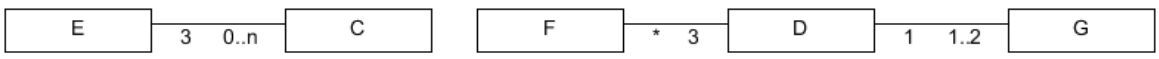
      ii. Relation **« D adopte un comportement de A »**, implémente



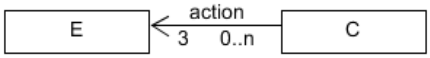
   b. Tous les autres objets qui ne sont pas dans la précédente catégorie sont simplement des objets qui communiquent entre eux, mais quel type d'objets ? Association, agrégation, composition ?
      Étant donné que l'association est la relation la plus permissive suivie par les agrégations suivies par les compositions :



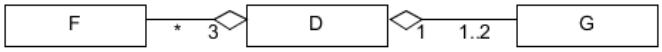
      i. On va supposer que tous les liens du schéma sont des associations **« C communique avec 0..n E »** (échange de message, tous les objets ont leurs cycle de vie) et on va déterminer la cardinalité.
         1. Si la communication est dans les deux sens, nous sommes dans une communication cyclique et nous avons donc :
            « C communique avec 0..n E » ET « E communique avec 3 C »



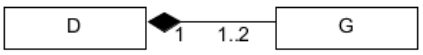
2. On peut accessoirement mettre du texte pour exprimer ce que l'objet passe comme message à l'autre au-dessus de la flèche.



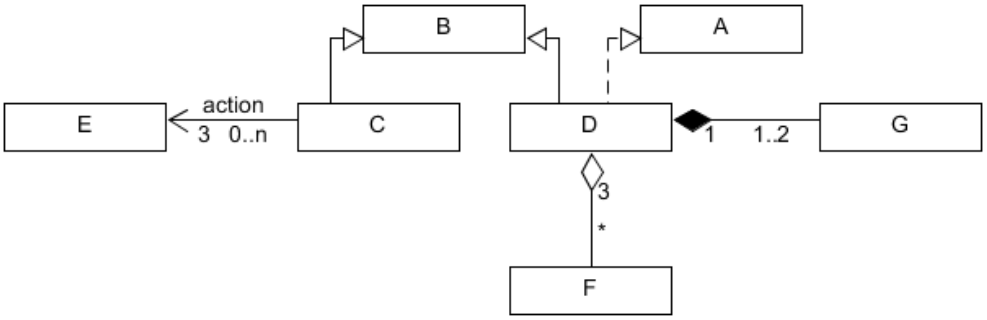
ii. On va inspecter toutes les relations qui ne sont pas cycliques (étant donné qu'une agrégation est non cyclique (et donc plus restrictive)), nous avons une relation du type <u>agrégation</u> **« D possède un F »**

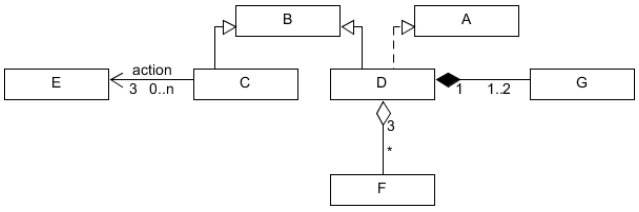


iii. Finalement, on va se concentrer sur les agrégations et on va regarder si les deux objets d'une agrégation peuvent exister indépendamment l'un de l'autre. S'ils ne peuvent pas, on a affaire à une <u>composition (agrégation forte)</u> **« D est composé de G »**, nous avons donc « possède » qui devient « est composé de »



1. Remarque : il faut bien faire attention que la cardinalité d'une composition du coté de l'agrégat ne peut être que 1 ou 0..1






```
public interface A {
  public void aBehaviour();
}
public class B {}
public class E {
  private ArrayList<C> cs;
  public void addend(C c) {
    cs.append(c);
  }
}
public class G {}
public class F {
  private D d1, d2, d3; // Like C
  public F(D d1, D d2, D d3) {
    this.d1 = d1;
    this.d2 = d2;
    this.d3 = d3;
  }
}
public class C extends B {
  private E e1, e2, e3;
  public setE1(E eNew) {e1 = eNew;}
  public action1() {
    if (!e1 || !e2 || !e3)
        throws new RuntimeException("No 3 E setted");
    e1.aMethod();
  }
  // other setters/actions
} // OR
public class C extends B {
  // if * cardinality for E (assoc)
  void action(E e) {
    e.aMethod(); // call a method on e
  }
}
public class D extends B implements A {
  private ArrayList<F> fs;
  private G g = new G();
  public void aBehaviour() {code;} // A
  public void append(F f) {
    Fs.append(f);
  }
  @Override
  public void finalize() // Destructor
    // Only to show the mechanics:
    g = null; // delete G - Garbage Coll.
  }
}
```

```
public class Main {
  public static void main(String [] args) {
        C c = new C();
    E e1 = new E();
    e1.append(c);
    e1.append(c);
    E e2 = new E();
    e2.append(c);
    E e3 = new E();
    e3.append(c);
    E e4 = new E();
    c.setE1(e1);
    c.setE2(e2);
    c.setE3(e3);
    try {
      c.doSomething1();
    } catch (RuntimeException e) {
      System.out.println(e.getMessage(););
    }
    // OR if * E in C
    c.action(e4);

    D d1 = new D();
    D d2 = new D();
    D d3 = new D();

    F f1 = new F(d1,d2,d3);
    F f2 = new F(d1,d2,d3);

    d1.append(f1);
    d1.append(f2);
    d2.append(f1);
    d3.append(f2);

    d.aBehaviour();
  }
}
```