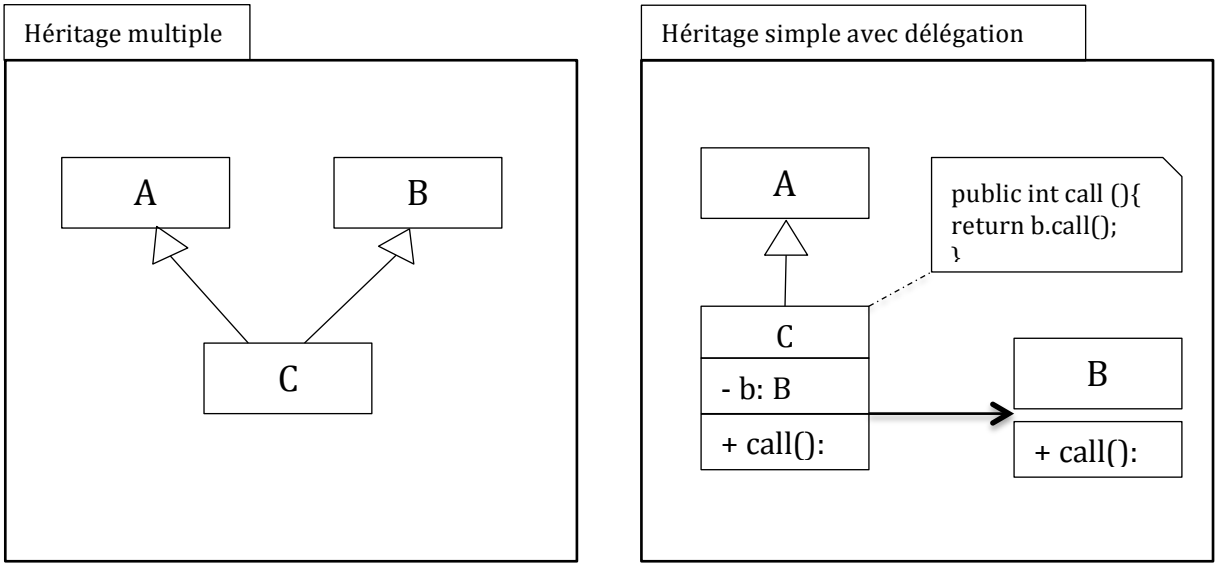
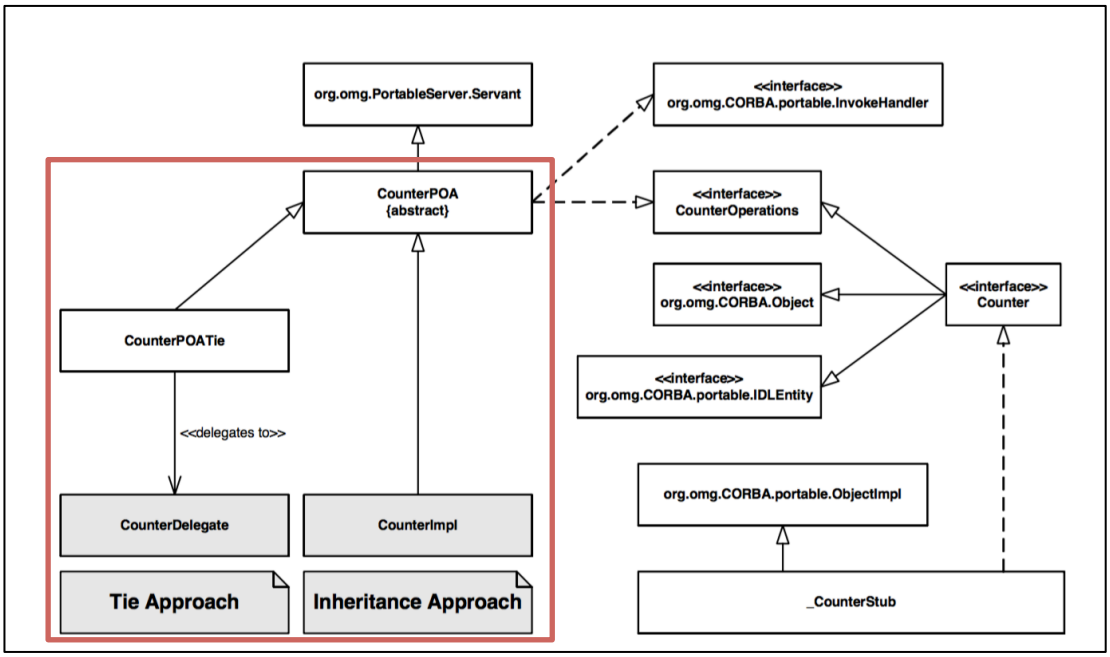
Séance 02 : TP – Approche par délégation

- Héritage vs Délégation : Rappel



- L'approche par délégation est utilisée lorsque la classe d'implémentation doit hériter d'une classe différente de la classe du servant. Dans le contexte Java où l'héritage multiple n'est pas autorisé, l'approche proposée lors du TP précédent n'est plus possible.



- Compilation : idlj –fallTIE Counter.idl

- Au niveau du serveur, un objet de la classe d'implémentation est instancié (« delegate ») et sera passé au constructeur de la classe tie.

CounterPOATie.java

```
1
2   /**
3   * CounterPOATie.java .
4   * Generated by the IDL-to-Java compiler (portable), version "3.2"
5   * from Counter.idl
6   * dimanche 20 mai 2018 17 h 17 WET
7   */
8
9
10  // Counter.idl
11  public class CounterPOATie extends CounterPOA
12  {
13
14    // Constructors
15
16    public CounterPOATie ( CounterOperations delegate ) {
17        this._impl = delegate;
18    }
19    public CounterPOATie ( CounterOperations delegate , org.omg.PortableServer.POA poa ) {
20        this._impl = delegate;
21        this._poa      = poa;
22    }
23    public CounterOperations _delegate() {
24        return this._impl;
25    }
26    public void _delegate (CounterOperations delegate ) {
27        this._impl = delegate;
28    }
29    public org.omg.PortableServer.POA _default_POA() {
30        if(_poa != null) {
31            return _poa;
32        }
33        else {
34            return super._default_POA();
35        }
36    }
37    public int value ()
38    {
39      return _impl.value();
40    } // value
41
42    public void inc ()
43    {
44      _impl.inc();
45    } // inc
46
47    public void dec ()
48    {
49      _impl.dec();
50    } // dec
51
52    private CounterOperations _impl;
53    private org.omg.PortableServer.POA _poa;
54
55  } // class CounterPOATie
56
```

2. Constructeur qui permet d'assigner _impl

3. Les 3 méthodes d'interface : `value()`, `inc()`, and `dec()` destinées à déléguer l'exécution à l'objet `CounterOperations`

1. Déclaration d'une variable _impl de type `CounterOperations`

- Création de la classe d'implémentation :
  Au niveau du constructeur le compteur (`count`) est initialisé et l'interface graphique créée.

  Les méthodes `inc()`, `dec()`, et `value()` implémentent les opérations d'interface et affichent la nouvelle valeur du compteur.

CounterDelegate.java

```
1  // CounterDelegate.java
2  import javax.swing.*;
3  public class CounterDelegate extends JPanel
4      implements CounterOperations {
5    private int count;
6    private JTextField value;
7    public CounterDelegate() {
8      count = 0;
9      add(new JLabel("Counter value: ", JLabel.RIGHT));
10     add(value =
11       new JTextField((String.valueOf(count)), 10));
12     value.setEditable(false);
13   }
14   public void inc() {
15     value.setText(String.valueOf(++count));
16   }
17   public void dec() {
18     value.setText(String.valueOf(--count));
19   }
20   public int value() {
21     return count;
22   }
23 }
```

- Implémentation du serveur : Approche par délégation
  Il est à noter deux aspects essentiels :
  - Usage d'une instance de « `CounterPOATie` » au lieu de « `CounterImpl` » (Cas héritage)
  - Une instance « `CounterDelegate` » est créée et passée au constructeur de « `CounterPOATie` »

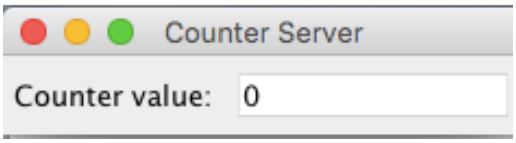
  → L'objet «`CounterPOATie`» pourra alors déléguer toutes les invocations à l'objet de délégation

Pour le contexte graphique, il y a création d'un JFrame auquel sera ajouté le panel (`CounterPOATie`).

Il sera possible de combiner plusieurs clients au serveur.

DelegationServer.java

```
// DelegationServer.java
import java.io.*;
import java.util.Properties;
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
import javax.swing.*;
import static java.lang.System.*;
public class DelegationServer {
  public static void main(String[] args) {
    try {
      CounterDelegate cd;
      JFrame f = new JFrame("Counter Server");
      f.getContentPane().add(cd = new CounterDelegate());
      f.pack();
      f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      f.setVisible(true);
      Properties props = getProperties();
      ORB orb = ORB.init(args, props);
      org.omg.CORBA.Object obj = null;
      POA rootPOA = null;
      try {
        obj = orb.resolve_initial_references("RootPOA");
        rootPOA = POAHelper.narrow(obj);
      } catch (org.omg.CORBA.ORBPackage.InvalidName e) { }
      CounterPOATie c_impl = new CounterPOATie(cd);
      Counter c = c_impl._this(orb);
      try {
        FileOutputStream file =
          new FileOutputStream("Counter.ref");
        PrintWriter writer = new PrintWriter(file);
        String ref = orb.object_to_string(c);
        writer.println(ref);
        writer.flush();
        file.close();
        out.println("Server started."
          + " Stop: Close-Button");
      } catch(IOException ex) {
        err.println("File error: " + ex.getMessage());
exit(2); }
      rootPOA.the_POAManager().activate();
      orb.run();
    } catch(Exception ex) {
      out.println("Exception: " + ex.getMessage());
exit(1); }
} }
```



- Implémentation du client : Approche par délégation

L'implémentation suivante consiste principalement en trois méthodes essentielles : `initializeORB()`, `getRef()`, et `createGUI()`.

Les deux premières méthodes comportent des actions spécifiques CORBA :

```
  private void initializeORB(String[] args) {
    Properties props = getProperties();
    orb = ORB.init(args, props);
  }
```

```
  private org.omg.CORBA.Object getRef(String refFile) {
    String ref = null;
    try {
      Scanner reader = new Scanner(new File(refFile));
      ref = reader.nextLine();
    } catch (IOException ex) {
      out.println("File error: "   + ex.getMessage());
    exit(2);
  }
  org.omg.CORBA.Object obj = orb.string_to_object(ref);
  if (obj == null) {
    out.println("Invalid IOR");
exit(4); }
return obj; }
```

- « `createGUI()` » contient les spécificités graphiques : un label représentant la valeur du compteur, des boutons « Increment » et « Decrement » ainsi que les écouteurs associés qui invoquent respectivement les appels « inc () » et « dec() ».

```
  private void createGUI() {
    setLayout(new GridLayout(2, 1));
    JPanel p = new JPanel();
    final JLabel value;
    p.add(new JLabel("Counter value: ", JLabel.RIGHT));
    p.add(value = new JLabel(String.valueOf(c.value())));
    add(p);
    p = new JPanel();
    JButton inc, dec;
    p.add(inc = new JButton("Increment"));
    p.add(dec = new JButton("Decrement"));
    add(p);
    inc.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        c.inc();
        value.setText(String.valueOf(c.value()));
      }
    });
    dec.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        c.dec();
        value.setText(String.valueOf(c.value()));
      }
  }); }
```

- Le constructeur initialise le middleware, ensuite récupère la référence de l'objet Serveur, ensuite crée l'interface graphique.

```
public GUIClient(String[] args, String refFile) {
  initializeORB(args);
  org.omg.CORBA.Object obj = getRef(refFile);
  try {
    c = CounterHelper.narrow(obj);
  } catch (BAD_PARAM ex) {
    out.println("Narrowing failed");
exit(3); }
  createGUI();
}
```

- La méthode main() crée un JFrame et y ajoute un nouveau GUIClient .

```
public static void main(String[] args) {
  try {
    String refFile = "Counter.ref";
    JFrame f = new JFrame("Counter Client");
    f.getContentPane().add(
      new GUIClient(args, refFile));
    f.pack();
    f.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    f.setVisible(true);
    } catch (Exception ex) {
        out.println("Exception: " + ex.getMessage());
        exit(1);
} }
```

Une ou plusieurs instances de « `GUIClient` » peuvent être lancées ensemble avec une ou plusieurs instances en mode console.