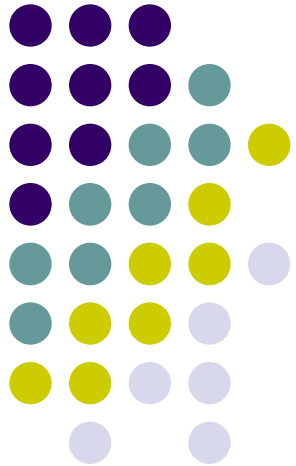
Αλεξάνδρειο Τεχνολογικό Εκπαιδευτικό Ίδρυμα Θεσσαλονίκης

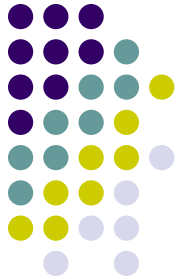
Τεκμηρίωση Προτύπων Σχεδίασης της UML



Της φοιτήτριας
Μαργαρίτη Θεοδώρα

Επιβλέπων καθηγητής
Αμπατζόγλου Απόστολος

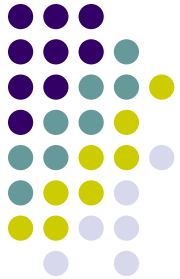
Πρότυπα Σχεδίασης



Κάθε πρότυπο περιγράφει ένα πρόβλημα που εμφανίζεται ξανά και ξανά στο περιβάλλον μας, και στη συνέχεια περιγράφει τον πυρήνα της λύσης του προβλήματος αυτού, με τέτοιο τρόπο ώστε να μπορεί κανείς να χρησιμοποιήσει αυτή τη λύση ένα εκατομμύριο φορές, χωρίς ποτέ να το κάνει δύο φορές με τον ίδιο τρόπο .
Ορισμένα πρότυπα είναι τα εξής:

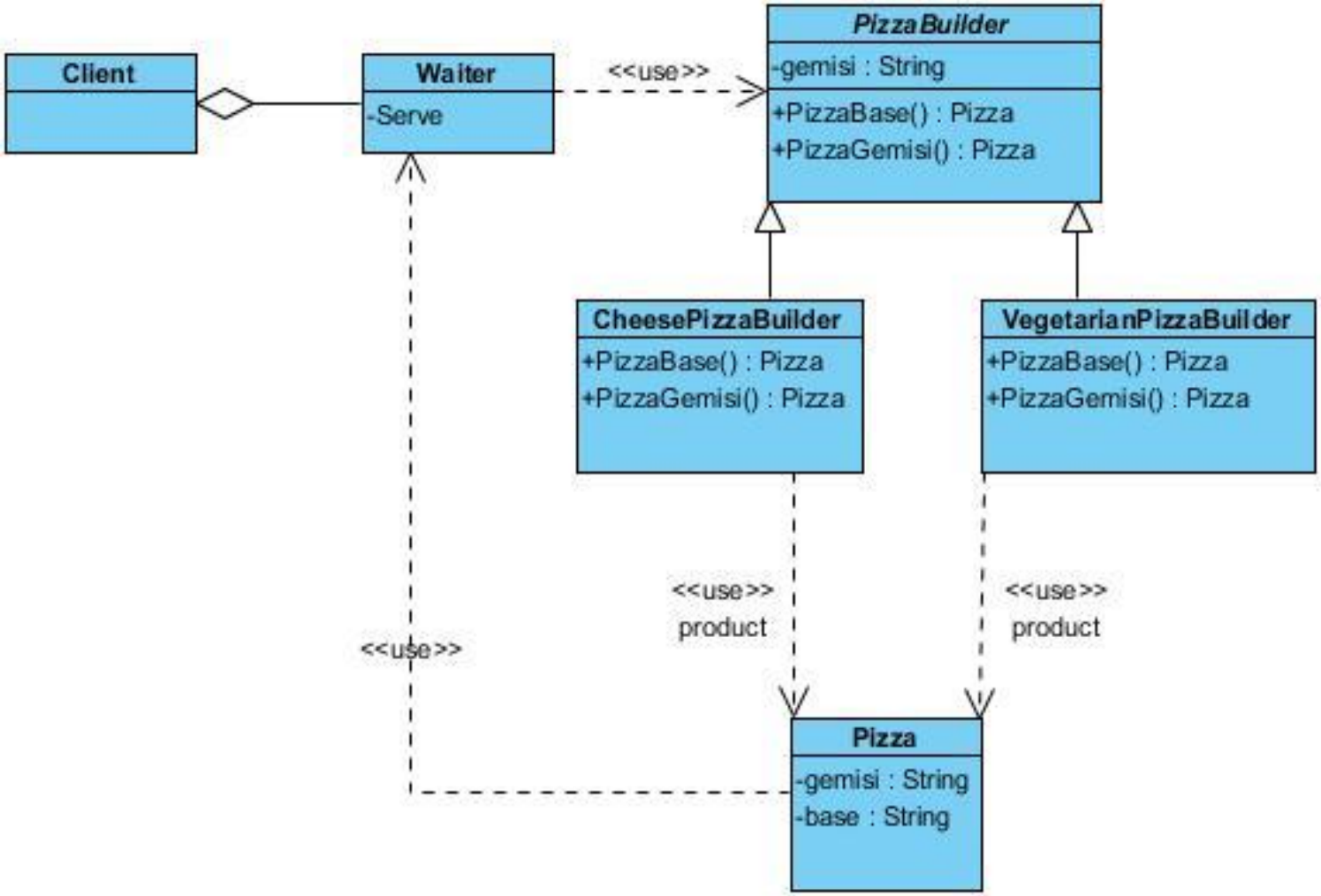
- Builder
- Composite
- Factory Method
- Template Method

Builder

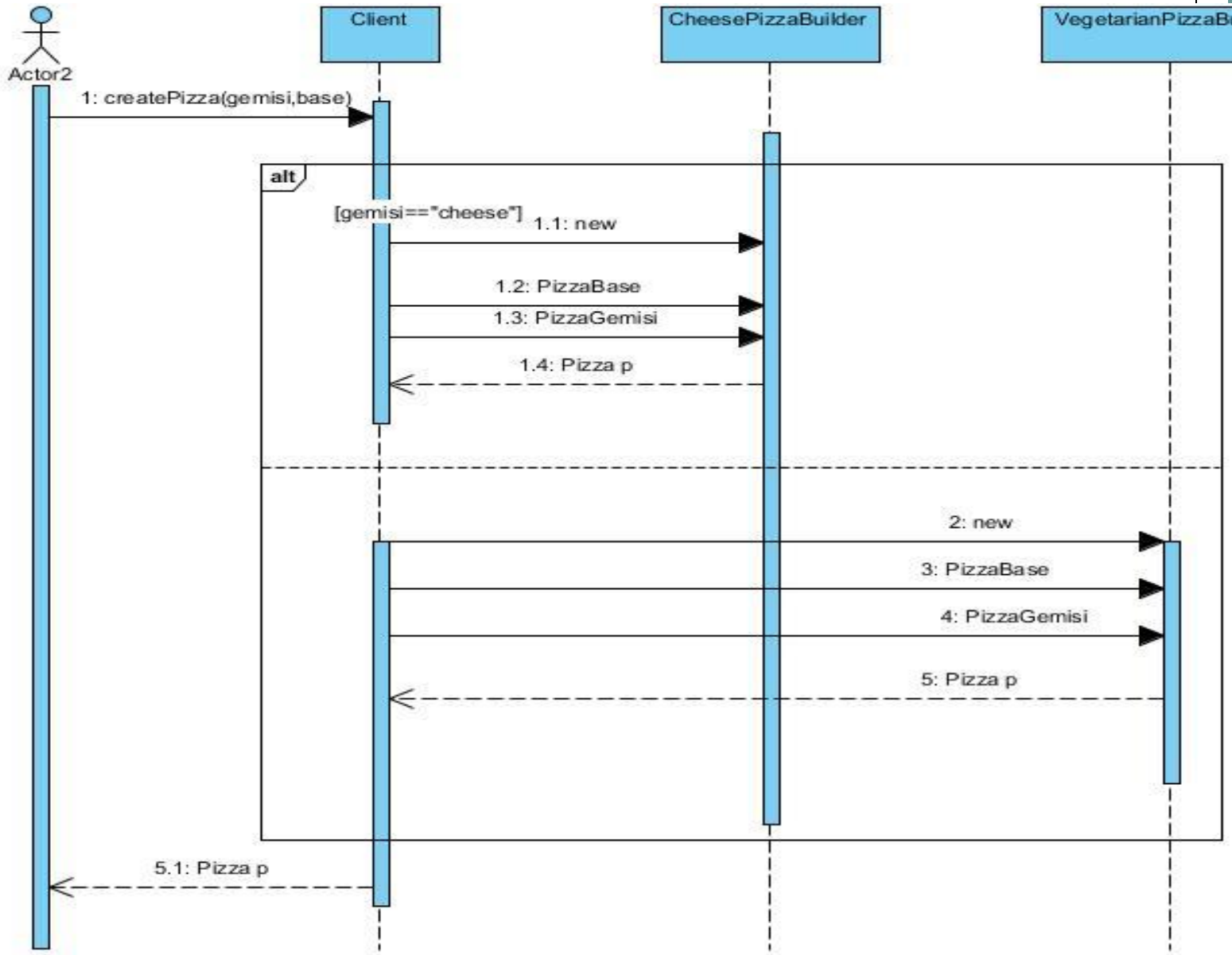


- Η πρόθεση του προτύπου σχεδίου δόμησης είναι να διαχωριστεί η κατασκευή ενός σύνθετου αντικειμένου από την αναπαράσταση του. Με αυτόν τον τρόπο, η ίδια διαδικασία κατασκευής μπορεί να δημιουργήσει διαφορετικές αναπαραστάσεις.
- Το Builder χτίζει συχνά σύνθετα αντικείμενα
- Ένα παράδειγμα αυτού του προτύπου είναι η κατασκευή της πίτσας.

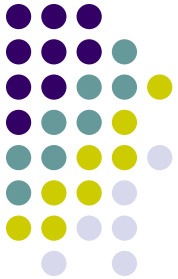
Builder Class Diagram



Builder Sequence Diagram

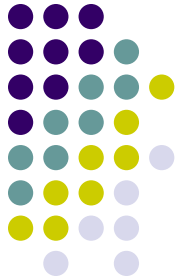


Builder java code part 1



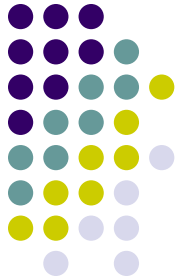
```
public class Waiter {
    public void servePizza(Client cl){

}
}
abstract public class PizzaBuilder {
    public String base;
    public String gemisi;
    abstract String PizzaBase(String Base);
    abstract String PizzaGemisi(String Gemisi);
}
```

Builder java code part 2



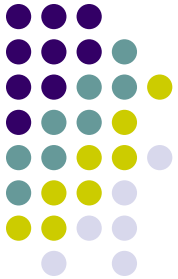
```
public class CheesePizzaBuilder extends PizzaBuilder {
    CheesePizzaBuilder(String arg){
        System.out.print("cheese"+arg);
    }
    public String PizzaBase(String base){
       System.out.print("oil");
       return base;
    }
    public String PizzaGemisi(String gemisi){
         System.out.print("cheese");
    return gemisi;
    }

}
```

Builder java code part 3



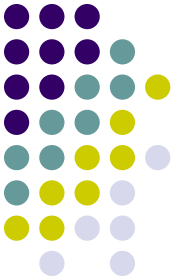
```
public class VegetarianPizzaBuilder extends PizzaBuilder {
    VegetarianPizzaBuilder(String arg){
        System.out.print("vegetarian"+arg);
    }
    public String PizzaBase(String base){
         System.out.print("oil");
      return base;
    }
    public String PizzaGemisi(String gemisi){
         System.out.print("sogia cheese");
    return gemisi;
    }
```

Builder java code part 4



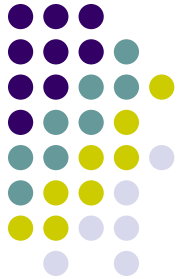
```
public class VegetarianPizzaBuilder extends PizzaBuilder {
    VegetarianPizzaBuilder(String arg){
        System.out.print("vegetarian"+arg);
    }
    public String PizzaBase(String base){
         System.out.print("oil");
      return base;
    }
    public String PizzaGemisi(String gemisi){
         System.out.print("sogia cheese");
    return gemisi;
    }
```

Builder java code part 5



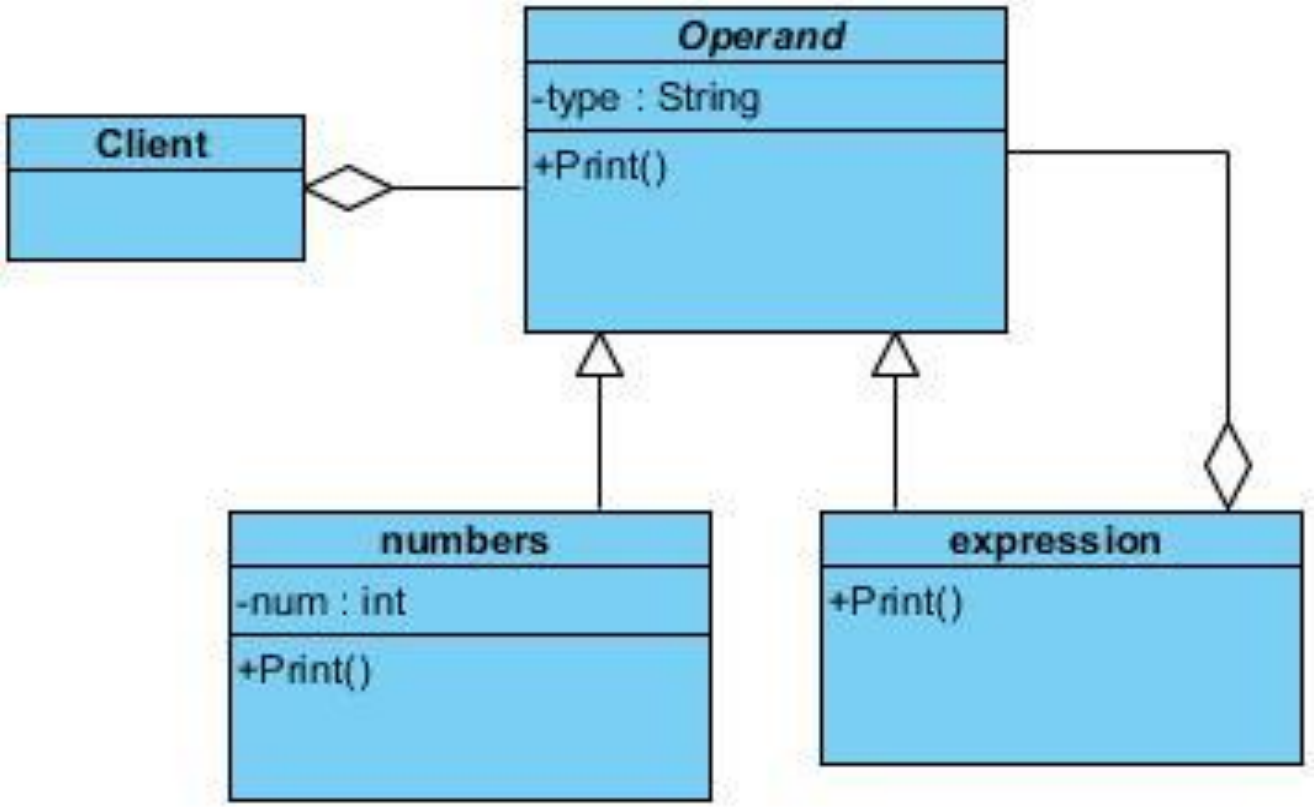
```
public class Client {

    public static void main(String args[])
    {
        Client cl=new Client();
        Waiter w=new Waiter();

        VegetarianPizzaBuilder vg=new VegetarianPizzaBuilder(" sogia
     cheese \n");
        CheesePizzaBuilder ch=new CheesePizzaBuilder(" milk cheese \n");
      w.servePizza(cl);
      System.out.println(vg.PizzaBase("")+" is the base");
        System.out.println(ch.PizzaBase("")+" is the base");

    }
```

Composite

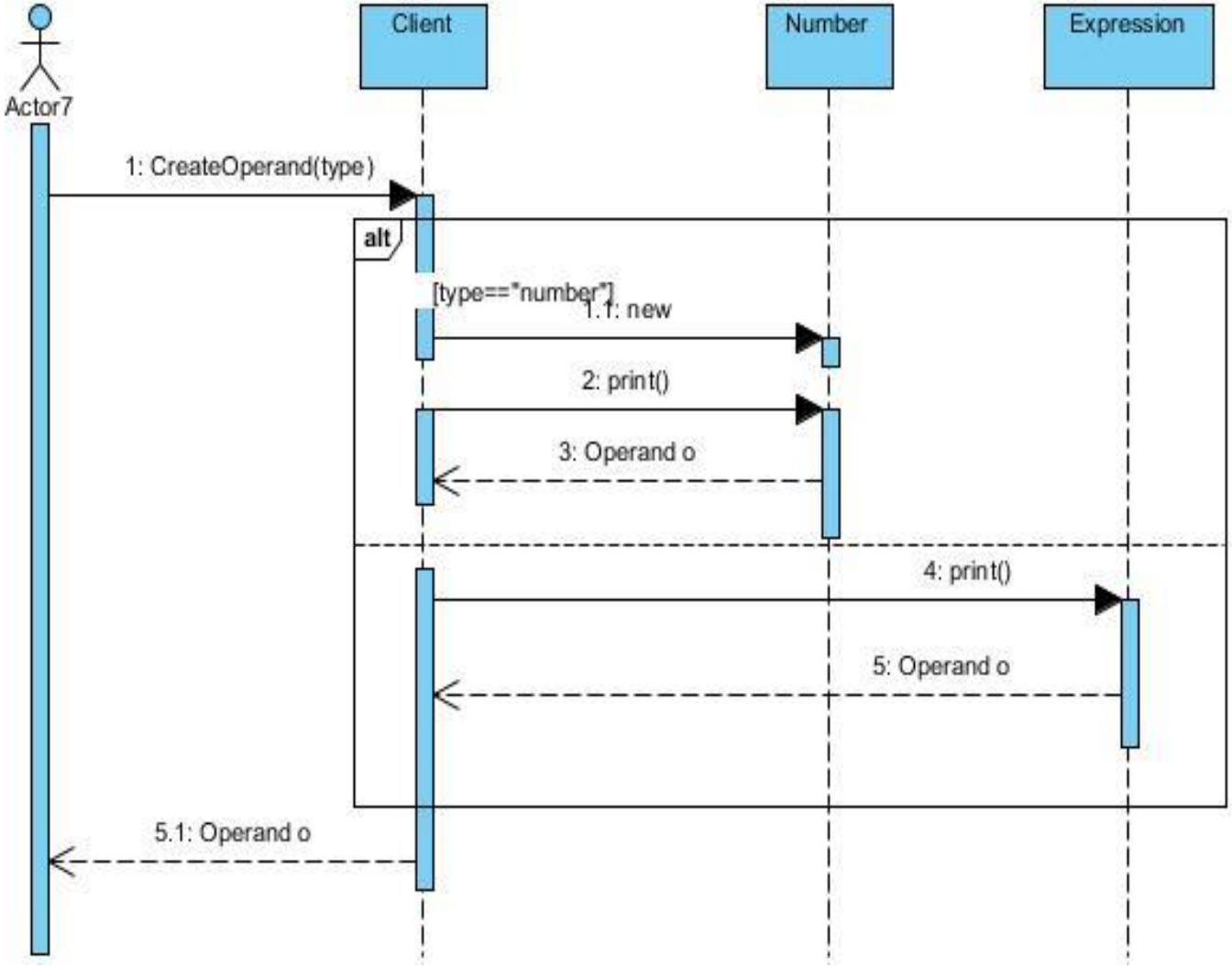


- Χρησιμοποιείται όταν οι χρήστες θα πρέπει να διαχωρίσουν σύνθετα από μεμονωμένα αντικείμενα
- Παράδειγμα οι αριθμητικές εκφράσεις(μεμονωμένες και σύνθετες)

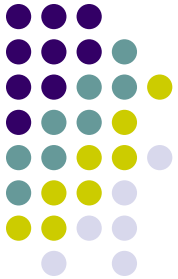
Composite Class Diagram



Composite Sequence Diagram

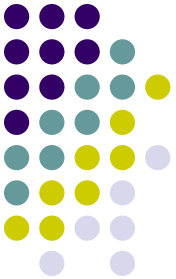


Composite java code part 1



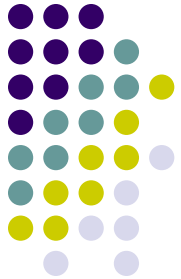
```
abstract public class Operand {
    protected String type;

    public Operand(String s) {type=s;}
    public abstract void print();
}
public class numbers extends Operand {
    private int a;
    private int b;

    public numbers(String s, int a0, int b0) {
        super(s);
        a=a0;
        b=b0;
    }
    public void print() {
        System.out.print(a + " " + type+ " " + b);
    }
}
```

Composite java code part 2



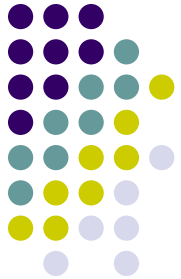
```
import java.util.ArrayList;
public class Expression extends Operand {
    private ArrayList<Operand> opList = new ArrayList<Operand>();

    public Expression(String arg){
        super(arg);
    }
    public void addExpression(Operand a) {
        opList.add(a);
    }

    public void print(){
        for (int i=0;i<opList.size();i++) {
            opList.get(i).print();
            if (i!=opList.size()-1) System.out.print(" " + type + " ");
        }
    }
}
```

Composite java code part 3



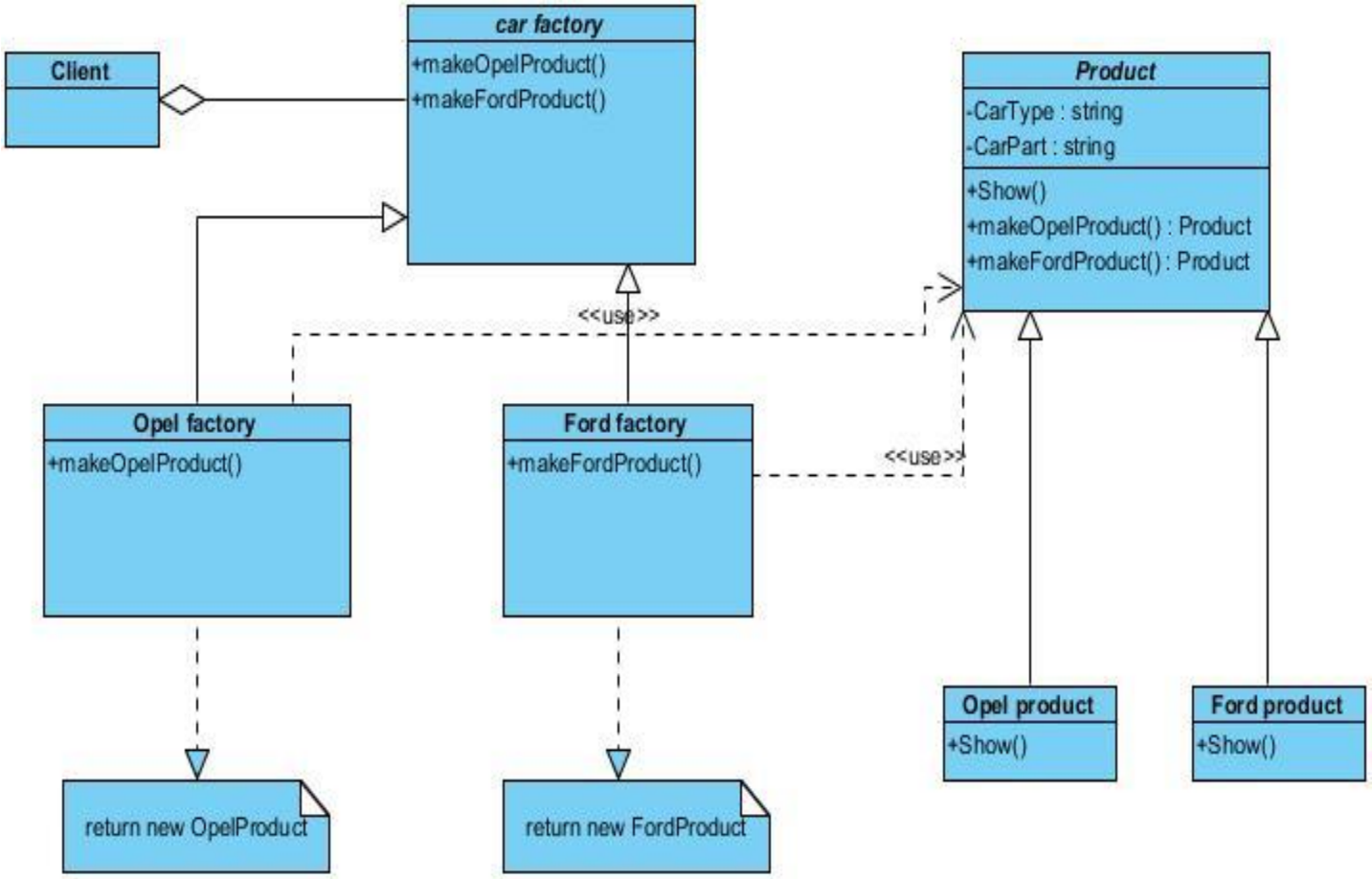
```
public class Client {
    public static void main(String args[])
    {

        Operand o = new numbers("+", 2, 8);
        o.print();
        System.out.println(" ");
        Expression o1 = new Expression("+");
        Operand o2 = new numbers("+", 4, 11);
        Operand o3 = new numbers("+", 1, 6);

        o1.addExpression(o2);
        o1.addExpression(o3);
        o1.print();
        System.out.println(" ");

    }
}
```

Factory Method

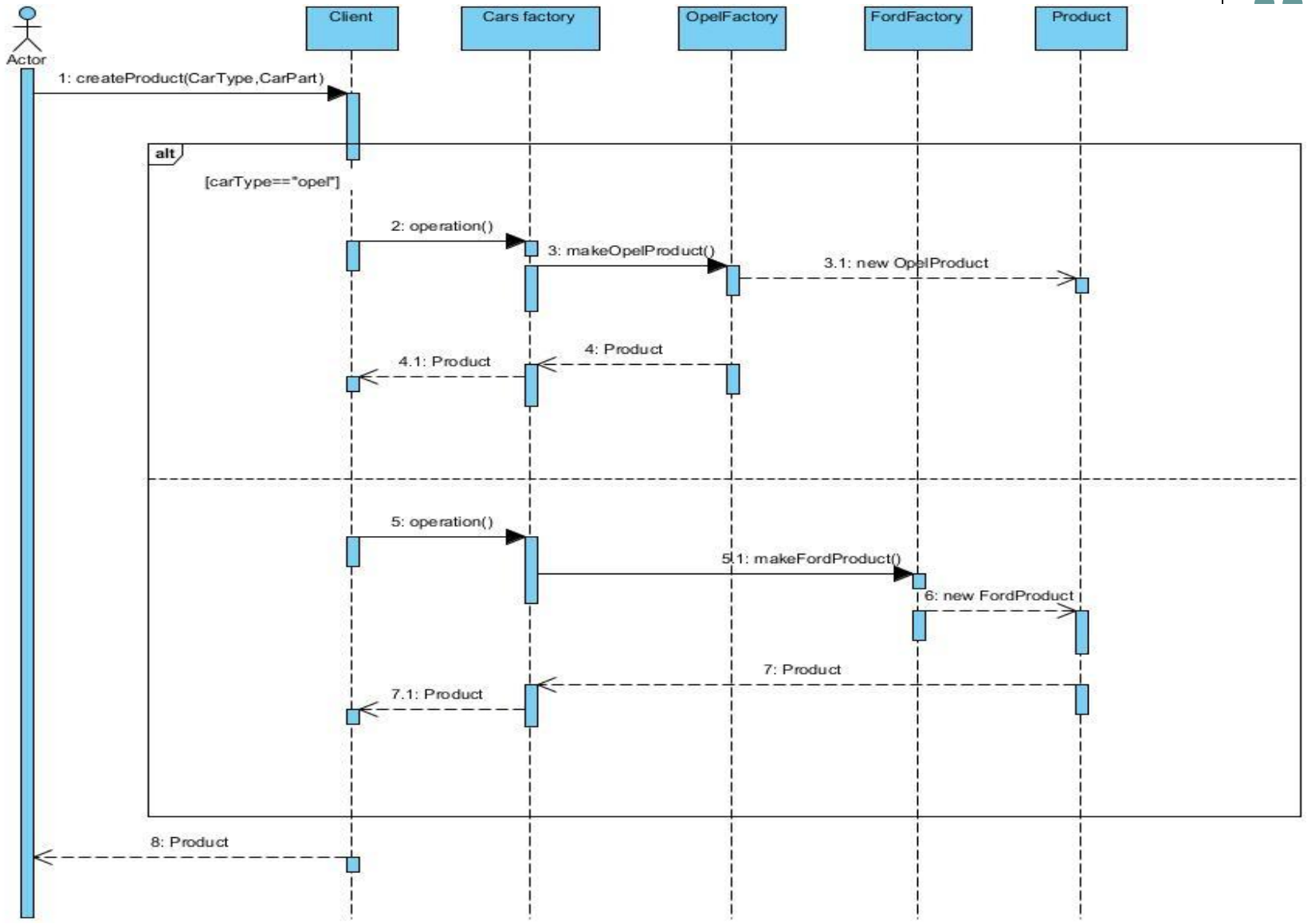


- ασχολείται με το πρόβλημα της δημιουργίας αντικειμένων(προϊόντων), χωρίς να προσδιορίζει την ακριβή κλάση του αντικειμένου που θα δημιουργηθεί.
- Ένα παράδειγμα η κατασκευή αυτοκινήτων

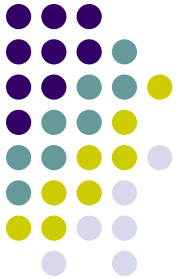
Factory Method Class Diagram



Factory Method Sequence Diagram

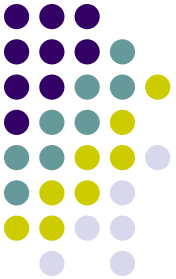


Factory Method java code part 1



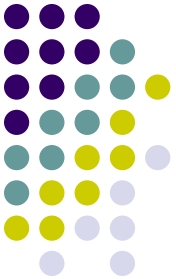
```
public class Product {
}
abstract public class Cars_factory {
    abstract FordProduct makeFordProduct(String msg);
    abstract OpelProduct makeOpelProduct(String msg);
}
public class FordFactory extends Cars_factory {
     FordProduct makeFordProduct(String msg)
   {
       return new FordProduct("Ford product "+msg);
   }
    @Override
    OpelProduct makeOpelProduct(String msg) {
}

}
```

Factory Method java code part 2



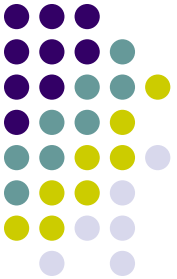
```
public class OpelFactory extends Cars_factory{
   OpelProduct makeOpelProduct(String msg)
   {
       return new OpelProduct("Opel product "+msg);
   }
    @Override
    FordProduct makeFordProduct(String msg) {
}
}
public class FordProduct {
   public FordProduct(String msg)
   {
       System.out.println(msg);
   }
    public void Show(String arg)
        {
            System.out.println(arg);
        }
}
```

Factory Method java code part 3



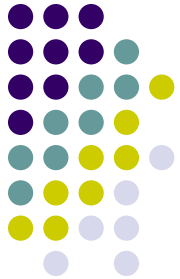
```
public class OpelProduct {
    public OpelProduct(String msg)
    {
        System.out.println(msg);
    }

    public void Show(String arg)
        {
            System.out.println(arg);
        }
}
```

Factory Method java code part 4



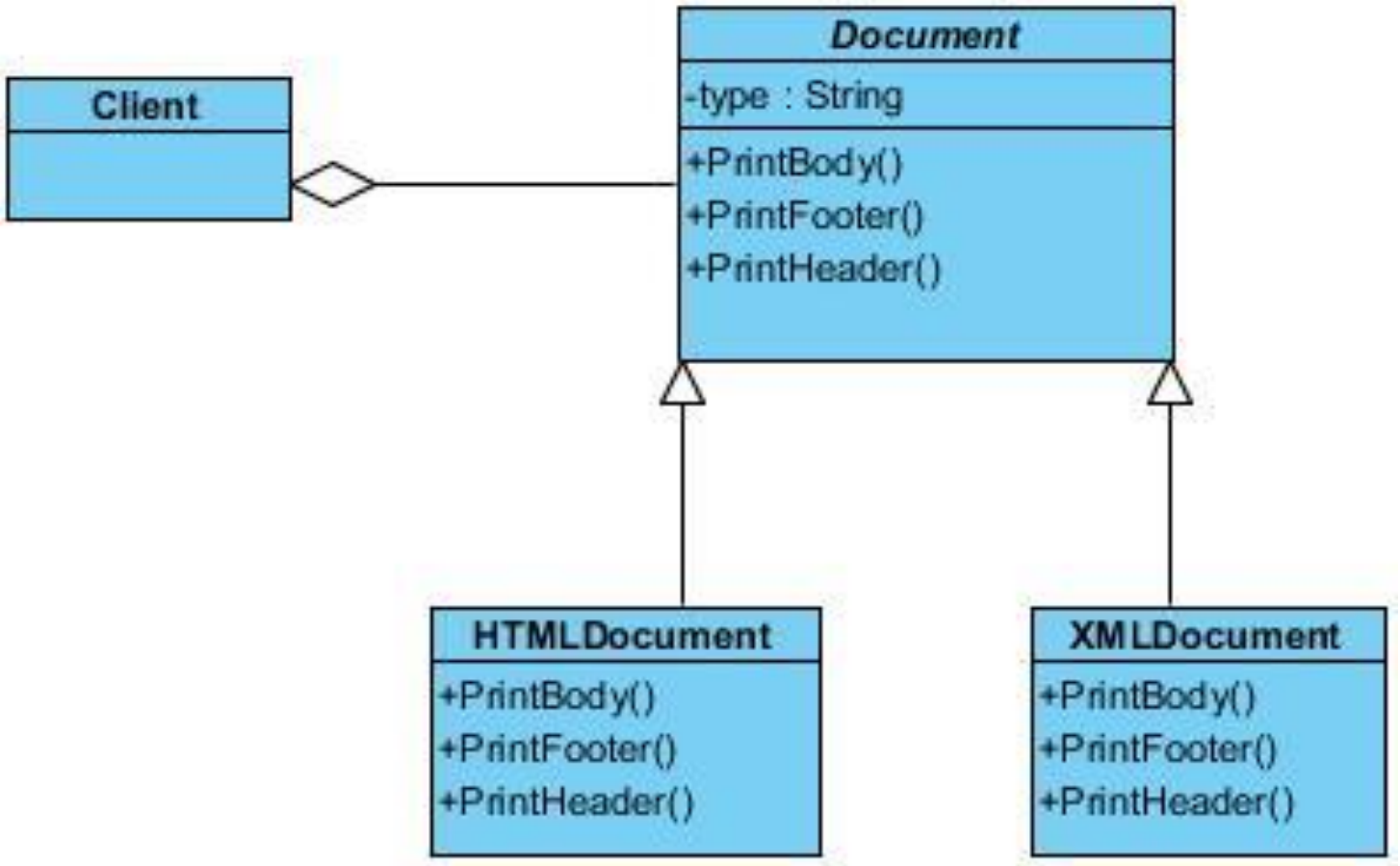
```
public class Client {
      private static Cars_factory pf=null;
       static Cars_factory getFactory(String string){
                if(string.equals("a")){
                            pf=new OpelFactory();
                }else if(string.equals("b")){
                                        pf=new FordFactory();
                            } return pf;
      }
    public static void main(String args[])
       {
                Cars_factory pfo=Client.getFactory("a");
                OpelProduct product=pfo.makeOpelProduct("astra");
               product.Show("opel astra");

               Cars_factory pff=Client.getFactory("b");
                FordProduct productf=pff.makeFordProduct("focus");
               product.Show("ford focus");
}
}
```

Template Method

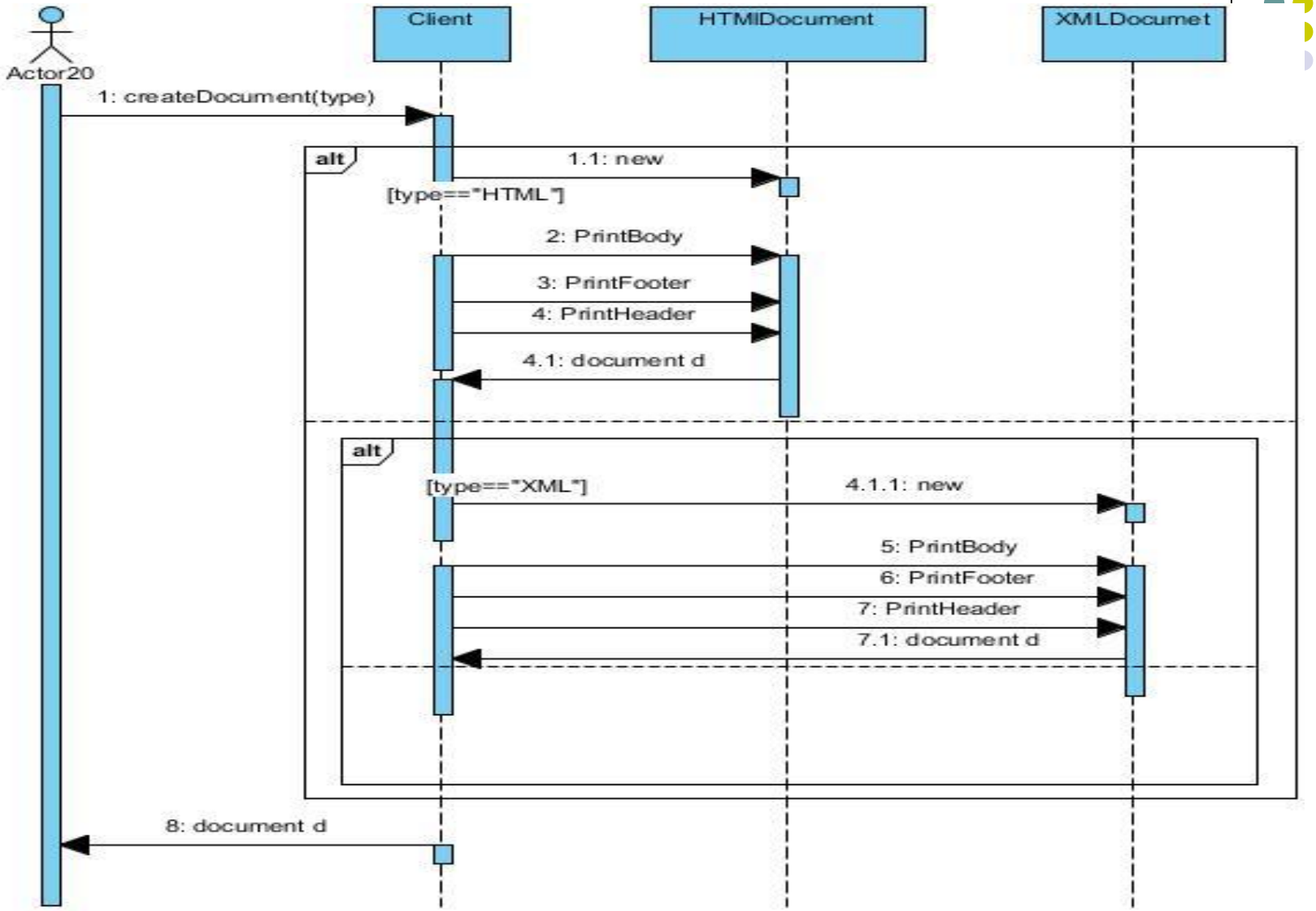


- Μέθοδος πρότυπο που η αρχική κλάση παρέχει τα βασικά βήματα ενός σχεδιασμού αλγορίθμων
- Οι υποκλάσεις εφαρμόζουν αυτά τα βήματα σε πραγματικό χρόνο
- Παράδειγμα αλγόριθμος εκτύπωσης για διαφορετικούς τύπους εγγράφων

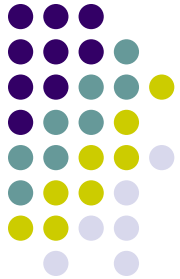
Template Method Class Diagram



Template Method Sequence Diagram

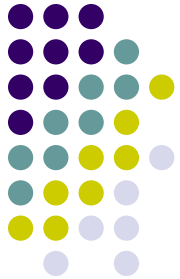


Template Method java code part 1



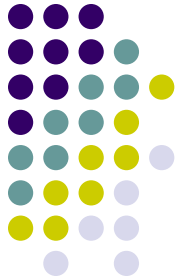
```
abstract public class Document {
        public abstract void PrintBody();
        public abstract void PrintFooter();
        public abstract void PrintHeader();
        public void drawDocument() {
            PrintHeader();
            PrintBody();
            PrintFooter();
        }
}
```

Template Method java code part 2



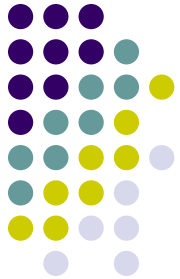
```
public class HTMLDocument extends Document {
    public HTMLDocument(){}
    public void PrintBody() {
            System.out.println("<Body>");
        }
        public void PrintFooter() {
            System.out.println("<Foot>");
        }
        public void PrintHeader() {
            System.out.println("<Head>");
        }

}
```

Template Method java code part 3



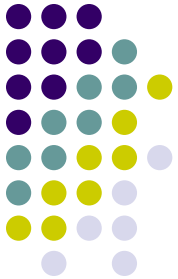
```
public class XMLDocument extends Document {
    public XMLDocument(){}
    public void PrintBody() {
            System.out.println("<Body>");
        }
        public void PrintFooter() {
            System.out.println("<Foot>");
        }
        public void PrintHeader() {
            System.out.println("<?xml...>");
        }
}
```

Template Method java code part 4



```
public class Client {
    public static void main(String args[]) {
        HTMLDocument html=new HTMLDocument();
        html.drawDocument();
        XMLDocument xml=new XMLDocument();
        xml.drawDocument();
    }
}
```

Συμπέρασμα



Υπάρχουν πολλά και διάφορα πρότυπα τα οποία ανάλογα με το πρόβλημα χρησιμοποιούνται.Μπορεί να χρησιμοποιηθεί το ίδιο πρότυπο σε διαφορετικές καταστάσεις.Από τα συγκεκριμένα πρότυπα διαπιστώθηκε ότι δίνουν λύσεις σε πραγματικό χρόνο .Ο κάθε χρήστης μπορεί να δει τη λειτουργία μιας πραγματικής κατάστασης.