Creational Design Patterns

Creational Patterns

- Abstract Factory
  - Provide an interface for creating families of related or dependent objects without specifying their concrete classes .
  - يمرر وسيطاً عند البناء و الذي يحدد ما نرغب ببناءه
- Builder
  - Separate the construction of a complex object from its representation so that the same construction process can create different representations .
  - نمرر كوسيط غرض و الذي يعرف بناء الغرض انطلاقاً من توصيفه.
- Factory Method
  - Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses
  - الصف ينادي مناهج مجردة. يكفينا الاشتقاق من هذا الصف.
- Prototype
  - Specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype
  - يوجد عدة نماذج أولية مختلفة و التي يتم نسخهم و تجميعهم.
- Singleton
  - Ensure a class only has one instance, and provide a global point of access to it.

Creational Patterns

- Abstract Factory
  - Provide an interface for creating families of related or dependent objects without specifying their concrete classes .
  - يمرر وسيطاً عند البناء و الذي يحدد ما نرغب ببناءه
- Builder
  - Separate the construction of a complex object from its representation so that the same construction process can create different representations .
  - نمرر كوسيط غرض و الذي يعرف بناء الغرض انطلاقاً من توصيفه.
- Factory Method
  - Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses
  - الصف ينادي مناهج مجردة. يكفينا الاشتقاق من هذا الصف.
- Prototype
  - Specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype
  - يوجد عدة نماذج أولية مختلفة و التي يتم نسخهم و تجميعهم.
- Singleton
  - Ensure a class only has one instance, and provide a global point of access to it.

Singleton

- Ensure a class only has one instance, and provide a global point of access to it.
- Implementation
  - Make the Constructor private
  - Construct an instance of the class in the class
  - Make an access method to this instance

```
public class SingletonClass {
    private SingletonClass() {}
    private SingletonClass uniqueInstance = new SingletonClass();
    public static SingletonClass getTheInstance() {
            return uniqueInstance;
    }
}
```

- utilisation : the enumerated types in Java < 1.5
  - Ex : the days of weeks

Example 1

```
class PrintSpooler{
        static boolean instance_flag=false;
        public PrintSpooler() throws SingletonException {
                if (instance_flag)
                                throw new SingletonException("Only one spooler allowed");
                else
                {
                                instance_flag = true;
                                System.out.println("spooler opened");
                }
        }
        public void finalize()                {
                instance_flag = false; //clear if destroyed
        }
}
```

Enumerated types as constant values

```
public class DaysOfWeek {
        public static final int LUNDI = 0;
        public static final int MARDI = 1;
        public static final int MERCREDI = 2;
        public static final int JEUDI = 3;
        public static final int VENDREDI = 4;
        public static final int SAMEDI = 5;
        public static final int DIMANCHE = 6;
}
```

```
Void methodProcessingDays(int day){…}
```

```
Obj. methodProcessingDays(DaysOfWeek.LUNDI);
Obj. methodProcessingDays(7+5);
```

Desadvantages : no type for the days of week but integer values

The correct solution must : typed and constant values & nomber of these values is fix

Enumerated types as Singleton pattern

```
public class DaysOfWeek {
        private DaysOfWeek() {}
        public static final DaysOfWeek LUNDI = new DaysOfWeek();
        public static final DaysOfWeek MARDI = new DaysOfWeek();
        public static final DaysOfWeek MERCREDI = new DaysOfWeek();
        public static final DaysOfWeek JEUDI = new DaysOfWeek();
        public static final DaysOfWeek VENDREDI = new DaysOfWeek();
        public static final DaysOfWeek SAMEDI = new DaysOfWeek();
        public static final DaysOfWeek DIMANCHE = new DaysOfWeek();

}
```

```
Void methodProcessingDays(DaysOfWeek day){…}
```

Advantages :

1) Days are only objects of the class DaysOfWeek
2) We can compare the days using " == " because each day has unique referece during all the life of the program
3) Days are only the instances created in the class DaysOfWeek

Enumerated types as Singleton pattern +

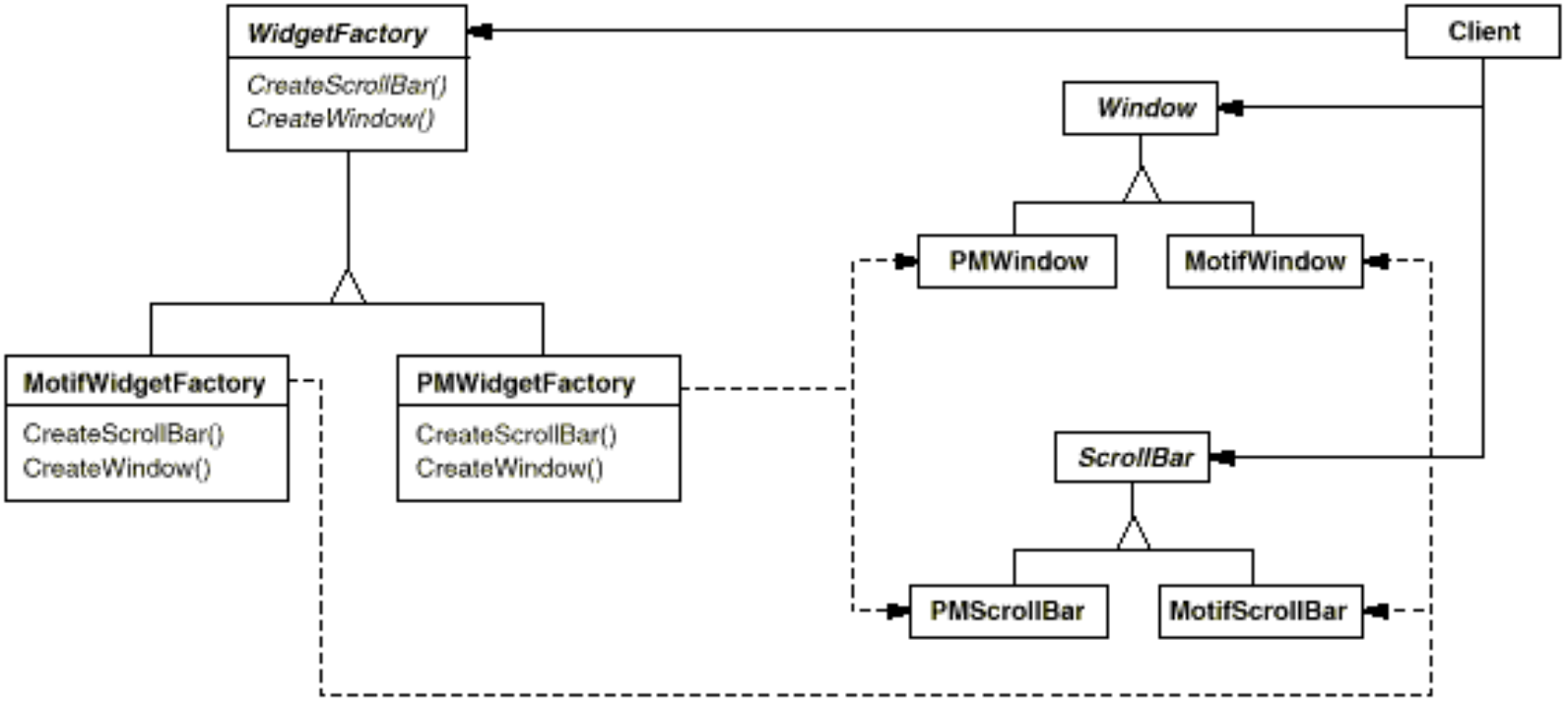
```
public class DaysOfWeek  implements Comparable {
        private String name;
        private static int counter = 0;
        public final int index; // final so is not modifiable
        private DaysOfWeek(String name) {
                this.name = name;
                index = counter++;
        }
        public int compareTo(Object o)
        {       DaysOfWeek j = (DaysOfWeek) o;
                return this.index – j.index;
        }
        public static final DaysOfWeek LUNDI = new DaysOfWeek("Lundi");
        …..
        public String toString() {return this.name; }
    }
```

Creational Patterns

- Abstract Factory
  - Provide an interface for creating families of related or dependent objects without specifying their concrete classes .
  - يمرر وسيطاً عند البناء و الذي يحدد ما نرغب ببناءه
- Builder
  - Separate the construction of a complex object from its representation so that the same construction process can create different representations .
  - نمرر كوسيط غرض و الذي يعرف بناء الغرض انطلاقاً من توصيفه.
- Factory Method
  - Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses
  - الصف ينادي مناهج مجردة. يكفينا الاشتقاق من هذا الصف.
- Prototype
  - Specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype
  - يوجد عدة نماذج أولية مختلفة و التي يتم نسخهم و تجميعهم.
- Singleton
  - Ensure a class only has one instance, and provide a global point of access to it.

Abstract Factory (1)

- Provide an interface for creating families of related or dependent objects without specifying their concrete classes.

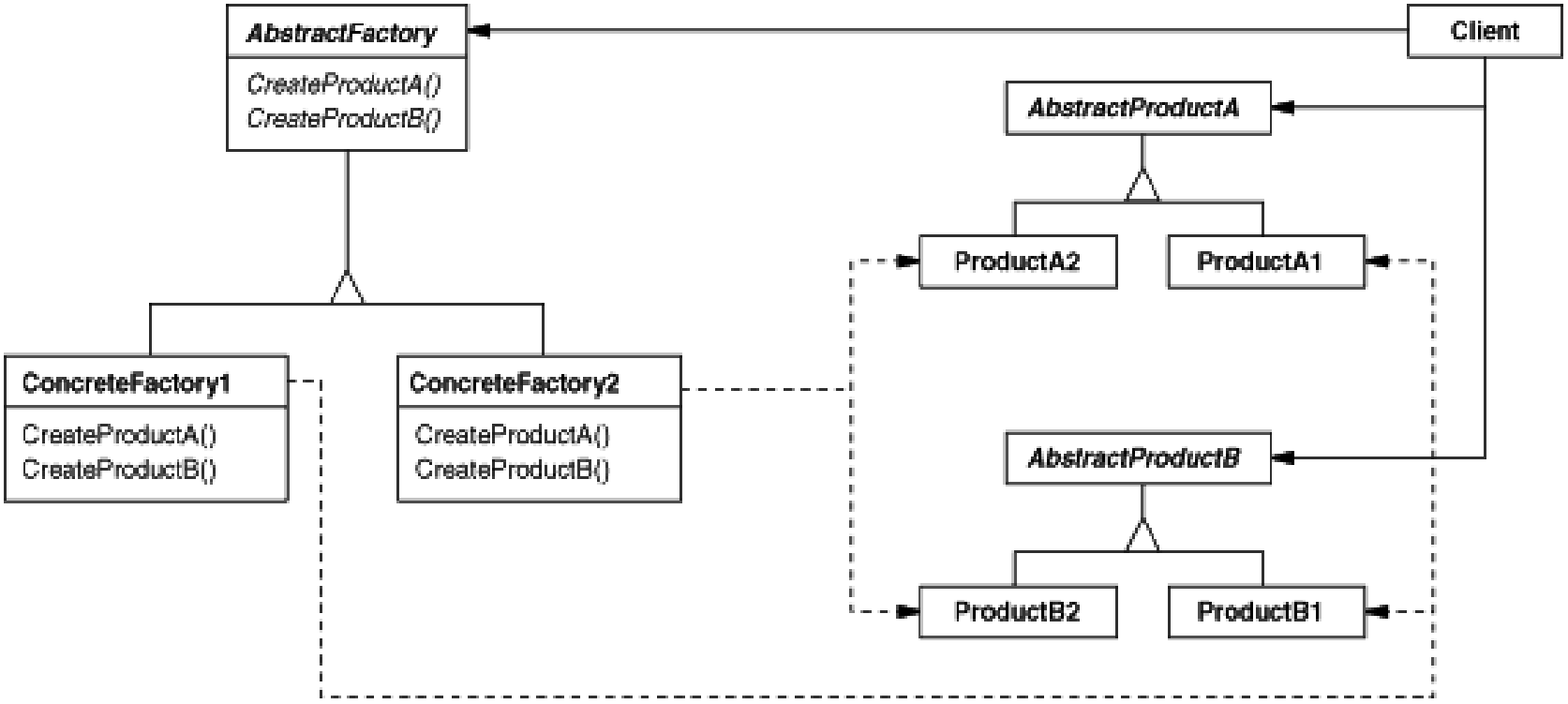


Abstract Factory (2)

- Applicability (usage)
  - a system should be independent of how its products are created, composed, and represented.
  - a system should be configured with one of multiple families of products.
  - a family of related product objects is designed to be used together, and you need to enforce this constraint.
  - we want to provide a class library of products, and you want to reveal just their interfaces, not their implementations.

Abstract Factory (3)

- Structure



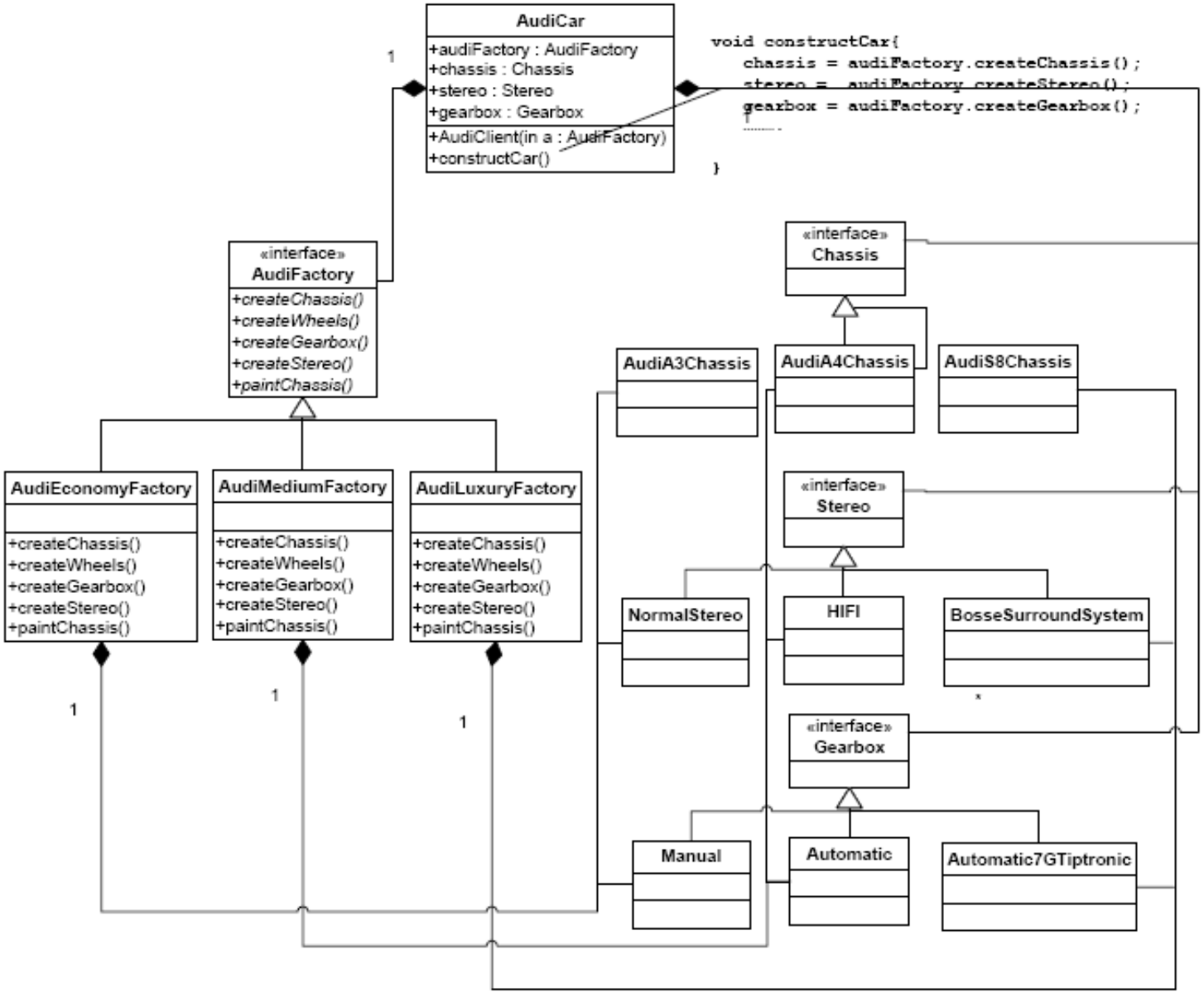
Abstract Factory (4) : Example

```
interface MazeFactory {
        Maze MakeMaze();
        Wall MakeWall();
        Room MakeRoom(int n);
        Door MakeDoor(Room r1, Room r2) ;
}
```

```
Class MazeGame {
        Maze CreateMaze (MazeFactory factory) {
                Maze aMaze = factory.MakeMaze();
                Room r1 = factory.MakeRoom(1);
                Room r2 = factory.MakeRoom(2);
                Door aDoor = factory.MakeDoor(r1, r2);
                aMaze.AddRoom(r1);
                aMaze.AddRoom(r2);
                ........
                return aMaze;
        }
}
```

```
class OrdinaryMazeFactory implements MazeFactory {
        public MazeFactory();
        Maze MakeMaze() { return new Maze(); }
        Wall MakeWall() { return new Wall(); }
        Room MakeRoom(int n) { return new Room(n); }
        Door MakeDoor(Room r1, Room r2)
        { return new Door(r1, r2); }
}
```

```
class BombedMazeFactory implements MazeFactory {
        Wall MakeWall() { return new BombedWall(); }
        Room MakeRoom(int n) { return new BombedRoom(n);
        …..
        }
}
```


AudiCar
+audiFactory : AudiFactory
+chassis : Chassis
+stereo : Stereo
+gearbox : Gearbox
+AudiClient(in a : AudiFactory)
+constructCar()
void constructCar{
chassis = audiFactory.createChassis();
gearbox = audiFactory.createGearbox();
}
1
«interface»
AudiFactory
+createChassis()
+createWheels()
+createGearbox()
+createStereo()
+paintChassis()
«interface»
Chassis
AudiA3Chassis
AudiA4Chassis
AudiS8Chassis
AudiEconomyFactory
+createChassis()
+createWheels()
+createGearbox()
+createStereo()
+paintChassis()
AudiMediumFactory
+createChassis()
+createWheels()
+createGearbox()
+createStereo()
+paintChassis()
AudiLuxuryFactory
+createChassis()
+createWheels()
+createGearbox()
+createStereo()
+paintChassis()
«interface»
Stereo
NormalStereo
HIFI
BosseSurroundSystem
«interface»
Gearbox
Manual
Automatic
Automatic7GTiptronic

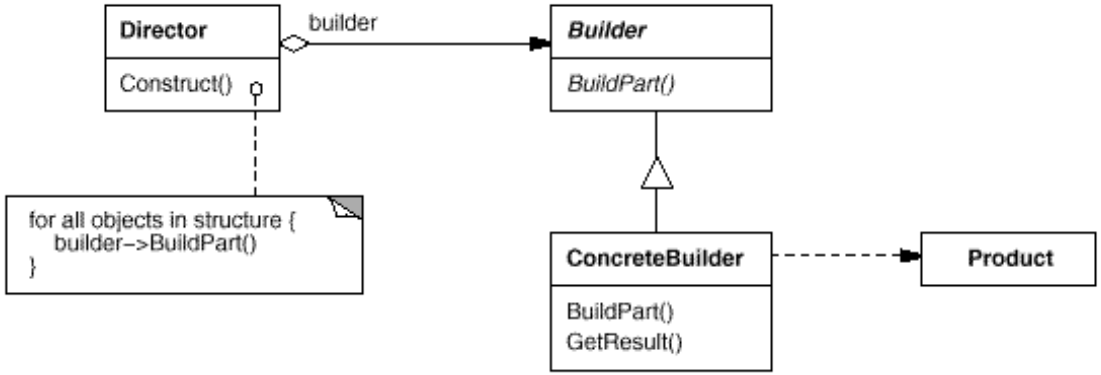
```
public class AudiTestDrive {
 public static void main(String[] args) {

    AudiCar myAudi = null;

    //create and economy car
    AudiFactory factory1 = new AudiEconomyFactory
    myAudi = new AudiCar(factory1);

    //create and medium car
    AudiFactory factory2 = new AudiMediumFactory
    myAudi = new AudiCar(factory2);

    //create and luxury car
    AudiFactory factory3 = new AudiLuxuryFactory
    myAudi = new AudiCar(factory3);

}
```

Creational Patterns

- Abstract Factory
  - Provide an interface for creating families of related or dependent objects without specifying their concrete classes .
  - يمرر وسيطاً عند البناء و الذي يحدد ما نرغب ببناءه
- Builder
  - Separate the construction of a complex object from its representation so that the same construction process can create different representations .
  - نمرر كوسيط غرض و الذي يعرف بناء الغرض انطلاقاً من توصيفه.
- Factory Method
  - Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses
  - الصف ينادي مناهج مجردة. يكفينا الاشتقاق من هذا الصف.
- Prototype
  - Specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype
  - يوجد عدة نماذج أولية مختلفة و التي يتم نسخهم و تجميعهم.
- Singleton
  - Ensure a class only has one instance, and provide a global point of access to it.

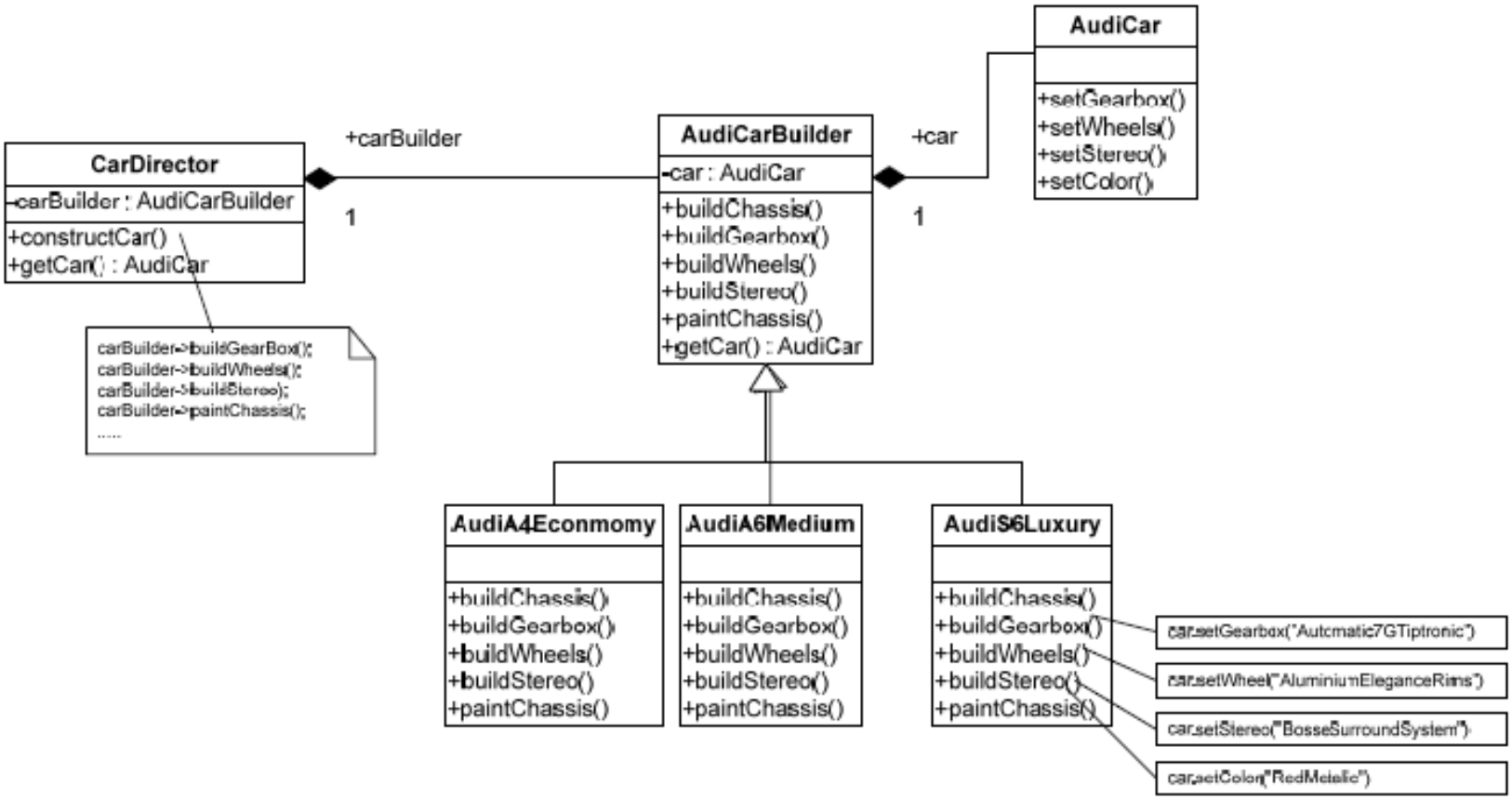
Builder (1)

- Separate the construction of a complex object from its representation so that the same construction process can create different representations.



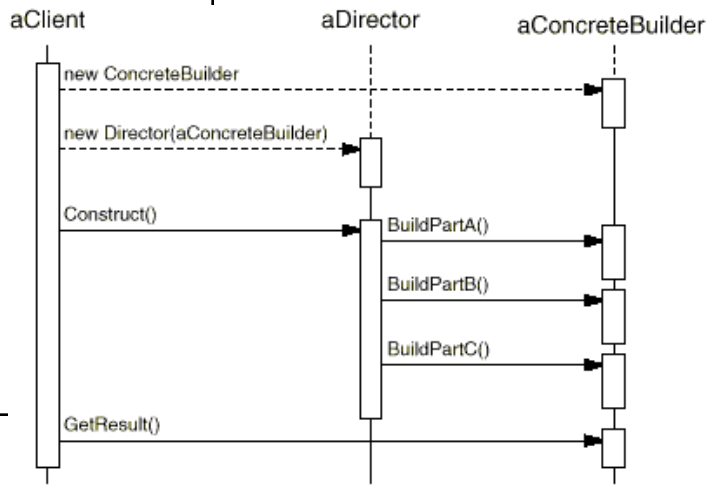
- Utilization :
  - Algorithm to create an object must be independent of
    - the parts that compose the object
    - the assembly mannar of these parts
  - Construction process permits differents representations of the object

Builder (2)


AudiCar
+setGearbox()
+setWheels()
+setStereo()
+setColor()
+carBuilder
+car
CarDirector
-carBuilder : AudiCarBuilder
+constructCar()
+getCar() : AudiCar
1
1
AudiCarBuilder
-car : AudiCar
+buildChassis()
+buildGearbox()
+buildWheels()
+buildStereo()
+paintChassis()
+getCar() : AudiCar
carBuilder->buildGearBox();
carBuilder->buildWheels();
carBuilder->buildStereo();
carBuilder->paintChassis();
.....
AudiA4Econmomy
+buildChassis()
+buildGearbox()
+buildWheels()
+buildStereo()
+paintChassis()
AudiA6Medium
+buildChassis()
+buildGearbox()
+buildWheels()
+buildStereo()
+paintChassis()
AudiS6Luxury
+buildChassis()
+buildGearbox()
+buildWheels()
+buildStereo()
+paintChassis()
car.setGearbox("Autcmatic7GTiptronic")
car.setWheel("AluminiumEleganceRims")
car.setStereo("BosseSurroundSystem")
car.setColor("RedMetalic")

Builder (3)

```
public class AudiCarBuilderExample {
    public static void main(String[] args) {
        CarDirector cdirector = new CarDirector();
        AudiCarBuilder a4EconomyBuilder = new AudiA4Economy();
        AudiCarBuilder a6MediumBuilder = new AudiA6Medium();
        AudiCarBuilder s6LuxuryBuilder = new AudisS6Luxury();

        cdirector .setCarBuilder(a4EconomyBuilder );
        cdirector.constructCar();
        AudiCar car = cdirector.getCar();

       cdirector .setCarBuilder(a6MediumBuilder );
        cdirector.constructCar();
        car = cdirector.getCar();

        cdirector .setCarBuilder(s6LuxuryBuilder );
        cdirector.constructCar();
        car = cdirector.getCar();
    }
}
```

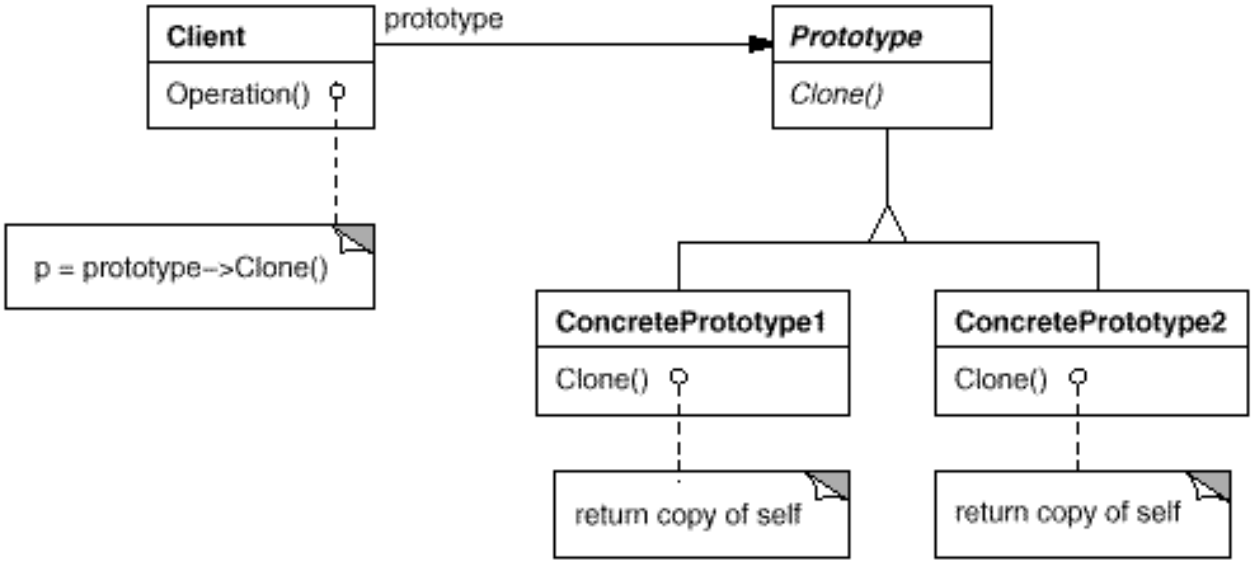


Creational Patterns

- Abstract Factory
  - Provide an interface for creating families of related or dependent objects without specifying their concrete classes .
  - يمرر وسيطاً عند البناء و الذي يحدد ما نرغب ببناءه
- Builder
  - Separate the construction of a complex object from its representation so that the same construction process can create different representations .
  - نمرر كوسيط غرض و الذي يعرف بناء الغرض انطلاقاً من توصيفه.
- Factory Method
  - Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses
  - الصف ينادي مناهج مجردة. يكفينا الاشتقاق من هذا الصف.
- Prototype
  - Specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype
  - يوجد عدة نماذج أولية مختلفة و التي يتم نسخهم و تجميعهم.
- Singleton
  - Ensure a class only has one instance, and provide a global point of access to it.

Prototype (1)

- Specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype.
- Structure :



Prototype (2)

```
class MazePrototypeFactory extends MazeFactory {
        private Maze _prototypeMaze;
        private Room _prototypeRoom;
        private Wall _prototypeWall;
        private Door _prototypeDoor;

        public MazePrototypeFactory(Maze m, Wall w, Room r, Door d) {
                _prototypeMaze = m; _prototypeRoom = r; _prototypeWall = w; _prototypeDoor = d;
        }

        Wall makeWall() {return (Wall) _prototypeWall.clone();}
        Door makeDoor(Room r1, Room r2) {
                Door d = (Door) _prototypeDoor.clone();
                d.initialize(r1, r2);
                return d;
        }
        Maze makeMaze() { …… }
        Room makeRoom(int) { …… }
        ‘‘‘‘‘
        }
```

Prototype (3)

- **ملاحظة** : عند تنفيذ المنهج clone() و الموروث من الصف Object في الجافا سيكون هناك مشكلة حتماً عندما يحوي الصف على أعضاء غير بدائيين (مراجع على أغراض أخرى) و لذلك >>>

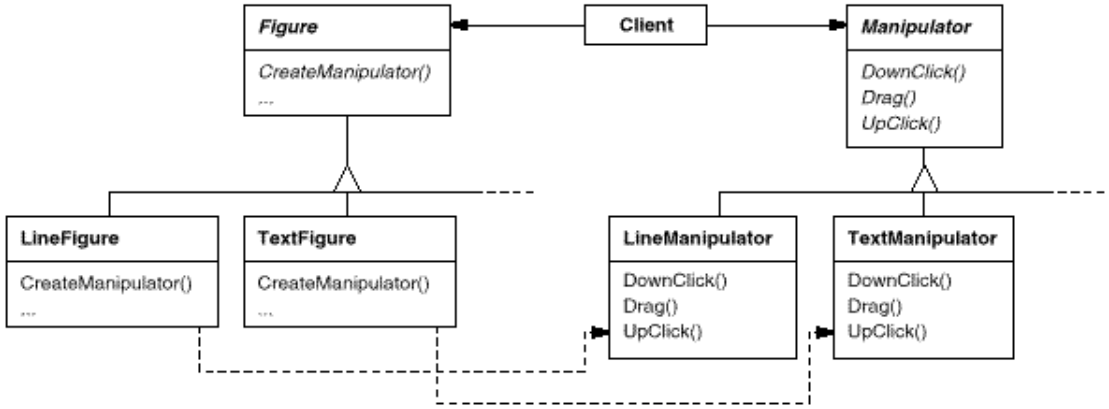
```
public Object deepClone(){
        try{
                ByteArrayOutputStream b = new ByteArrayOutputStream();
                ObjectOutputStream out = new ObjectOutputStream(b);
                out.writeObject(this);
                ByteArrayInputStream bIn = new    ByteArrayInputStream(b.toByteArray());
                ObjectInputStream oi = new ObjectInputStream(bIn);
                return (oi.readObject());
        }
        catch (Exception e)
        {       System.out.println("exception:"+e.getMessage());
                return null;
        }
}
```

Creational Patterns

- Abstract Factory
  - Provide an interface for creating families of related or dependent objects without specifying their concrete classes .
  - يمرر وسيطاً عند البناء و الذي يحدد ما نرغب ببناءه
- Builder
  - Separate the construction of a complex object from its representation so that the same construction process can create different representations .
  - نمرر كوسيط غرض و الذي يعرف بناء الغرض انطلاقاً من توصيفه.
- Factory Method
  - Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses
  - الصف ينادي مناهج مجردة. يكفينا الاشتقاق من هذا الصف.
- Prototype
  - Specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype
  - يوجد عدة نماذج أولية مختلفة و التي يتم نسخهم و تجميعهم.
- Singleton
  - Ensure a class only has one instance, and provide a global point of access to it.

Factory Method (1)

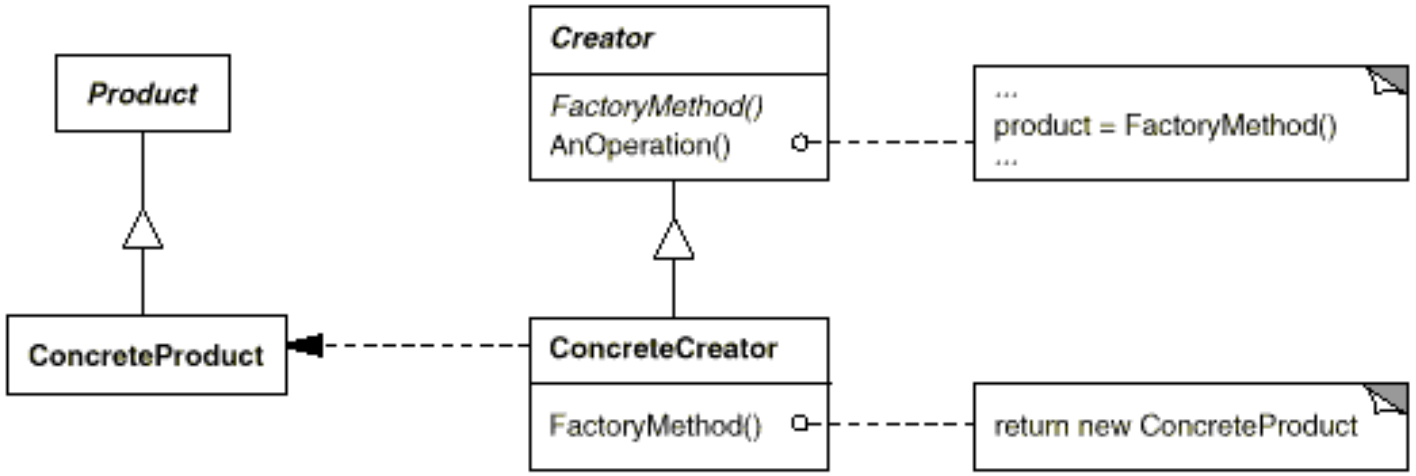
- Define an interface for creating an object, but let subclasses decide which class to instantiate.
- Factory Method lets a class defer instantiation to subclasses.
  - Addition to new class is made by polymorphism
  - It still the instance creation
  - .....



- To use when :
  - A class can not anticipate the class of objets that it must create
  - A class want that its subclasses specifie the objects that it create
- Exemples : the method *iterator()* in the *Collection*

Factory Method (2)

- Structure



Factory Method (3)

```
Class MazeGame {
        Maze MakeMaze() { return new Maze(); }
        Wall MakeWall() { return new Wall(); }
        Room MakeRoom(int n) { return new Room(n);}
        Door MakeDoor(Room r1, Room r2)
        { return new Door(r1, r2); }

        Maze CreateMaze () {
                Maze aMaze = MakeMaze();
                Room r1 = MakeRoom(1);
                Room r2 = MakeRoom(2);
                Door aDoor = MakeDoor(r1, r2);
                aMaze.AddRoom(r1);
                aMaze.AddRoom(r2);
                ........
                return aMaze;
        }
}
```

```
class BombedMazeGame extends MazeGame {
        Wall MakeWall() { return new BombedWall(); }
        Room MakeRoom(int n)
        { return new RoomWithBombe(n);}
}
```

```
class EnchantedMazeGame extends MazeGame {
        Room MakeRoom(int n)
        { return new EnchantedRoom(n);}
         Door MakeDoor(Room r1, Room r2)
        { return new DoorNeedingSpell(r1, r2); }
}
```

Factory Method (4) : Example (1/4)

```
public abstract class Shape {
      public abstract void draw();
      public abstract void erase();
      public static Shape factory(String type) throws BadShapeCreation {
            if(type.equals("Circle")) return new Circle();
            if(type.equals("Square")) return new Square();
            throw new BadShapeCreation(type);
      }
}

public class Circle extends Shape {
      public Circle() {}
      public void draw() { System.out.println("Circle.draw");}
      public void erase() { System.out.println("Circle.erase");}
}
```

OCP?

Factory Method (6) : Example (2/4)

```
public interface ShapeFactory {
        public Shape createShape();
}
public class AllShapeFactories {
        public static Map<String,ShapeFactory> allFactories = new
                HashMap<String,ShapeFactory>();
        public static final Shape createShape(String id) {
                // "throw BadShapeCreation"
                return allFactories.get(id).createShape();
        }
}
public class Circle extends Shape {
        static { AllShapeFactories.allFactories.put("Circle", new CircleFactory()); }
        (...)
}
public class CircleFactory implements ShapeFactory {
        public Shape createShape() { return new Circle(); }
}
public class Rectangle extends Shape {
        static { AllShapeFactories.allFactories.put("Rectangle", new RectangleFactory()); }
        (...)
}
public class RectangleFactory implements ShapeFactory {
        public Shape createShape() { return new Rectangle(); }
}
```

Factory Method (7) : Example (3/4)

```
//creation starting from the name of class
Shape shape = AllShapeFactories.createShape("Circle");
...
// random creation
int alea = new random.nextInt(ShapeFactory.factories.size());
Iterator<String> it = AllShapeFactories.factories.keySet().iterator();
for (int i = 0 ; i < alea; i++) {
        it.next(); }
shape = AllShapeFactories.createShape(it.next());
```

To avoid the pollution of classes : use intern classes

```
public class Circle extends Shape {
        static { AllShapeFactories.allFactories.put("Circle", new Circle.MaFactory()); }
        (...)
        class MaFactory implements ShapeFactory {
                // classe interne
                public Shape createShape() { return new Circle(); }
        }
}
public class Rectangle extends Shape {
        static { AllShapeFactories.allFactories.put("Rectangle", new Rectangle.MaFactory()); }
        class MaFactory implements ShapeFactory {
                public Shape createShape() { return new Rectangle(); }
        }
        (...)
}
```

Factory Method (8) : Example (4/4) with dynamic loading

```
public interface Shape {
        public void draw();
        public void erase();
}
public abstract class ShapeFactory {
        protected abstract Shape create();
        static Map<String,ShapeFactory> factories = new HashMap<String,ShapeFactory>();
        // A Template Method:
        public static final Shape createShape(String id) throws BadShapeCreation {
                if(!factories.containsKey(id)) {
                                try { Class.forName(id);}
                                catch(ClassNotFoundException e) { throw new BadShapeCreation(id); }
                                if(!factories.containsKey(id)) throw new BadShapeCreation(id);
                }
                return factories.get(id).create();
        }
}
public class Circle implements Shape {
        private Circle() {}
        public void draw() { System.out.println("Circle.draw"); }
        public void erase() { System.out.println("Circle.erase"); }
        private static class Factory extends ShapeFactory {
                protected Shape create() { return new Circle(); }
        }
        static { ShapeFactory.factories.put("Circle", new Circle.Factory()); }
}
```

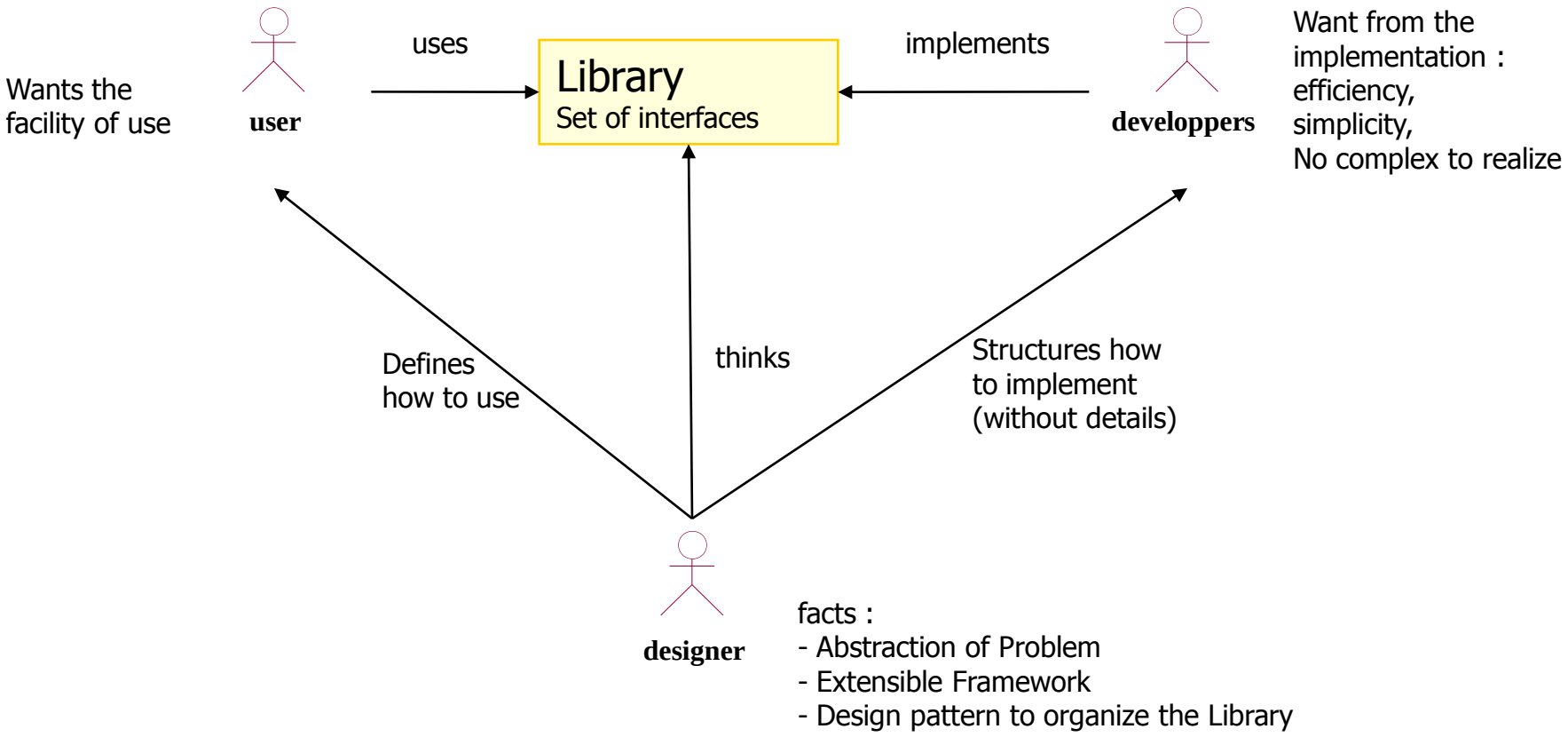
Summary of Creational Patterns

- **The Factory Method Pattern** is used to choose and return an instance of a class from a number of similar classes based on data you provide to the factory.
- **The Abstract Factory Pattern** is used to return one of several groups of classes. In some cases it actually returns a Factory for that group of classes.
- **The Builder Pattern** assembles a number of objects to make a new object, based on the data with which it is presented. Frequently, the choice of which way the objects are assembled is achieved using a Factory.
- **The Prototype Pattern** copies or clones an existing class rather than creating a new instance when creating new instances is more expensive.
- **The Singleton Pattern** is a pattern that insures there is one and only one instance of an object, and that it is possible to obtain global access to that one instance.

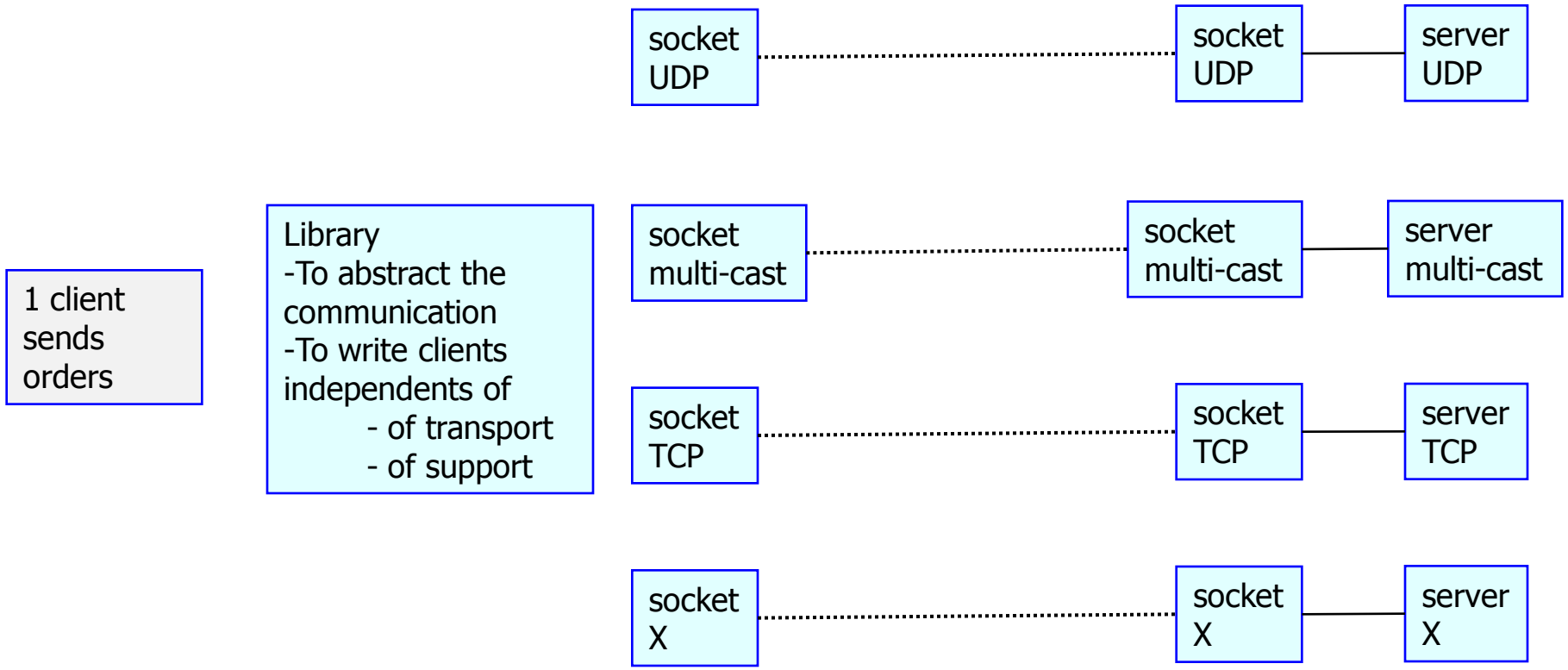
مسألة (١) :
Factory & Singleton patterns

- نريد بناء مكتبة للاتصالات عبر البرتوكولات (TCP, UDP, ...)
- هذه المكتبة ستسمح للمستخدم بالاتصال بمخدم من خلال إعطاء عنوانه
- المستخدم يفتح قناة و يتصل عبرها
  - المستخدم يحدد عنوان المخدم و البرتوكول الواجب استخدامه
  - القناة تستخدم البرتوكول المحدد
  - عدة برتوكولات متاحة (TCP, UDP, MultiCast, ...)

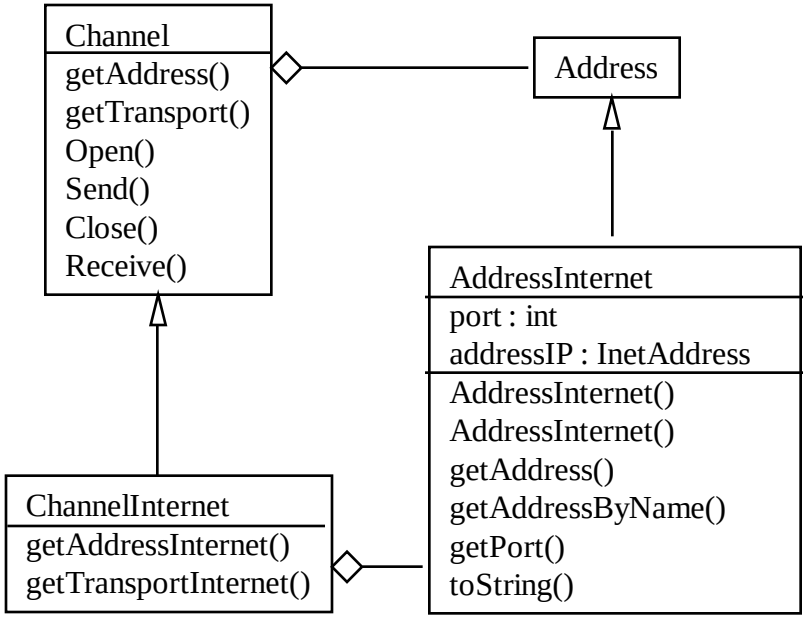
مسألة (٢) : توزيع الأدوار



مسألة (٣)

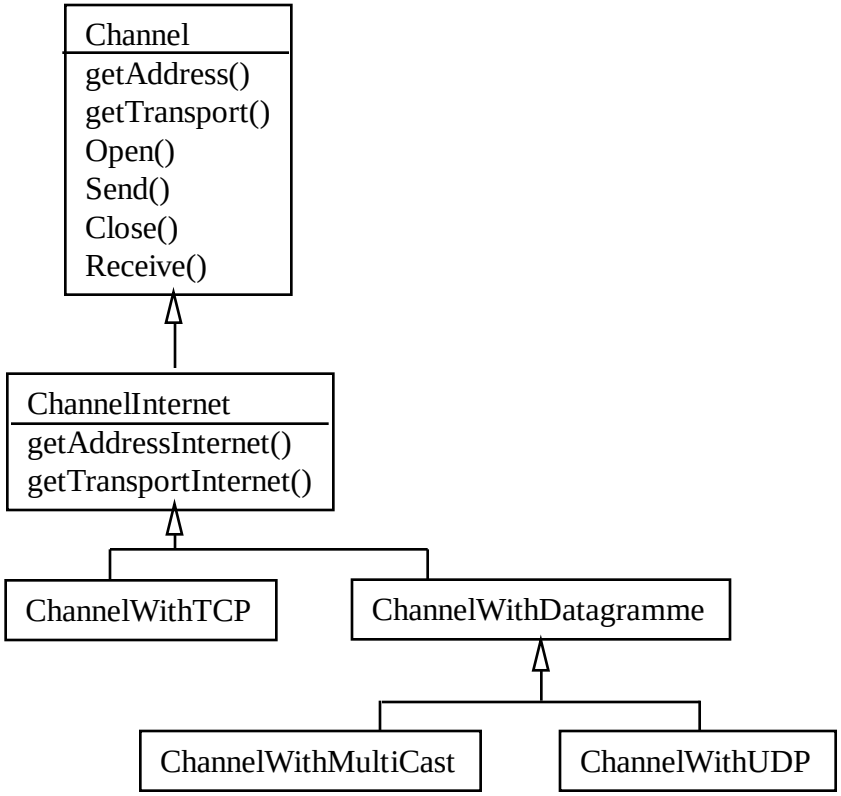

1 client sends orders
Library
-To abstract the communication
-To write clients independents of
- of transport
- of support
socket UDP
socket UDP
server UDP
socket multi-cast
socket multi-cast
server multi-cast
socket TCP
socket TCP
server TCP
socket X
socket X
server X

مسألة (4): تجريد المشكلة



- مفهوم عنوان المستقبل
  - Abstract Version = Address
  - Concrete Version = InternetAddress
- مفهوم قناة الاتصال
  - Channel = address + transport
  - Abstract Version = Channel
  - Concrete Version = ChannelInternet, ChannelWithUDP, ChannelWithTCP, …

مسألة (5): تنفيذ القناة




```
Channel ch = …; //How to create a channel
String repsonse1 = ch.send(" order1 ");
String response2 = ch.send(" order2 ");
…
…
ch.close();
```

```
String transport = …; //IHM
if( transport.equals("UDP ") )
        ch= new ChannelWithUDP(addr);
else if( transport.equals("multicast ") )
        ch= new ChannelWithMulticastIP(addr);
else if( transport.equals("TCP ") )
        ch= new ChannelWithTCP(addr);
```

- أسلوب غير قابل لإعادة الاستخدام و للتوسع
- <<< لذلك نستخدم الـ Factory pattern

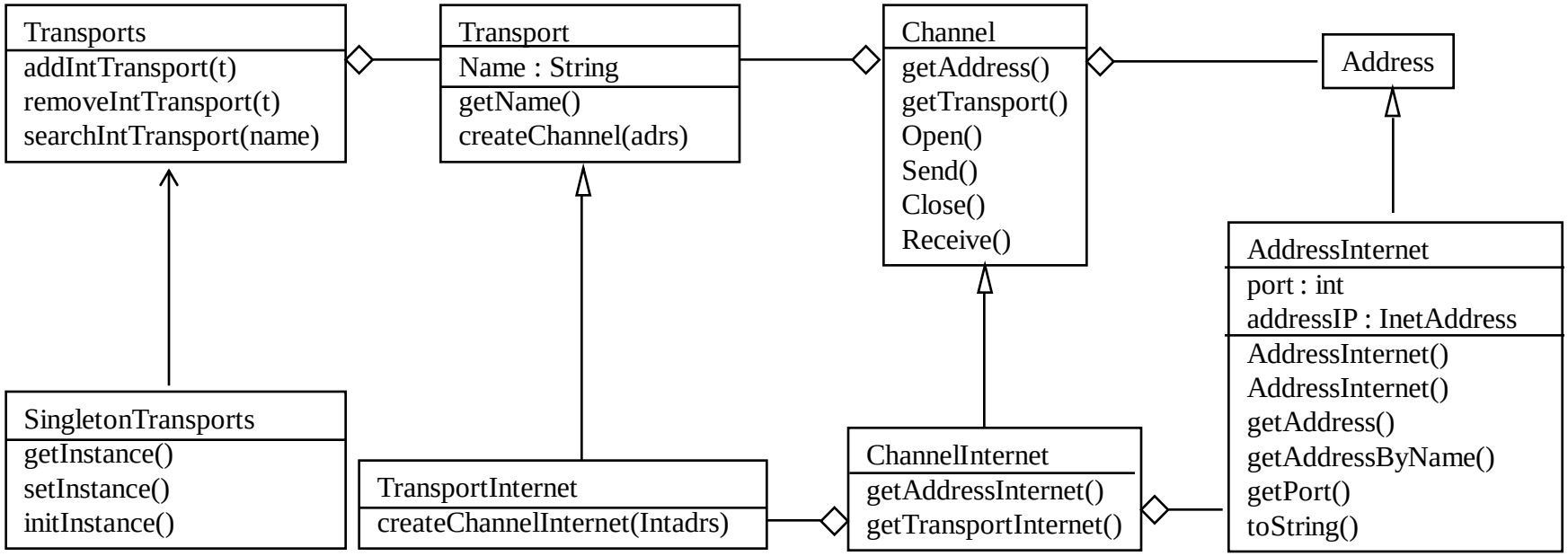
مسألة (٦) :

```
public class FactoryInternetChannels {
        static public ChannelInternet createChannel(
                String transport, AdresseInternet addr)
        {
                ChannelInternet ch=null;
                if( transport.equals("UDP ") )
                        ch= new ChannelWithUDP(addr);
                else if( transport.equals("multicast ") )
                        ch= new ChannelWithMulticastIP(addr);
                else if( transport.equals("TCP ") )
                        ch= new ChannelWithTCP(addr);
                return ch;
        }
}
```

```
Channel ch = FactoryInternetChannel.createChannel(
                transport, new AdresseInternet(…) );
```

- أسلوب قابل لإعادة الاستخدام
- أسلوب غير قابل للتوسع بسهولة
- إن إضافة transport جديد يتطلب تغيير createChannel ()

مسألة (٧) :


Transports
addIntTransport(t)
removeIntTransport(t)
searchIntTransport(name)
Transport
Name : String
getName()
createChannel(adrs)
Channel
getAddress()
getTransport()
Open()
Send()
Close()
Receive()
Address
AddressInternet
port : int
addressIP : InetAddress
AddressInternet()
AddressInternet()
getAddress()
getAddressByName()
getPort()
toString()
SingletonTransports
getInstance()
setInstance()
initInstance()
TransportInternet
createChannelInternet(Intadrs)
ChannelInternet
getAddressInternet()
getTransportInternet()

مسألة (8):

- الهدف من الصف TransportInternetBase هو الربط بين منهجي إنشاء القنوات

```
public interface Transport {
   public String getName();
   public Channel createChannel(Address dest);
}
```

```
public interface TransportInternet extends Transport {
    public ChannelInternet createInternetChannel(InterneAddress dst);
}
```

```
public class TransportInternetBase implements TransportInternet {
    public String getName();
    public Channel createChannel(Address dst) {
         createInternetChannel((InterneAddress) dst);
    }
    public ChannelInternet createInternetChannel(InterneAddress dst);
}
```

```
public class TransportWithTCP extends TransportInternetBase {
   public String getName() {return "TCP";}
   public ChannelInternet createChannelInternet(AddressInternet d)
   { return new ChannelWithTCP(d); }
}
```

```
public class TransportWithUDP
extends TransportInternetBase
{ ... ..}
```

مسألة (9):

```
public class SingletonTransports {
    protected static Transports instance;
    public static Transports getInstance() {
        if(instance == null)
            initInstance();
        return instance;
    }

  public static void setInstance(Transports transports) {
      instance = transports;
  }

  public static void initInstance() {
     instance = new TransportsImpl();
     instance.addTransport(new TransportWithUDP());
     instance.addTransport(new TransportWithMulticastIP());
     instance.addTransport(new TransportWithTCP());
  }
}
```

- يمكننا إضافة برتوكول جديد
  - من خلال initInstace()
  - أو من خلال addTransport() على الغرض transports