Paradigma Modelelor de Proiectare

Cursul nr. 8

Mihai Zaharia

Design Pattern - Definiții

- **Conform dicţionarului** Merriam-Webster termenul de pattern înseamnă:
  - 1. o formă sau model propus pentru imitare
  - 2. ceva proiectat sau folosit ca model pentru a face lucruri (calapodul croitorului)
  - 3. o formă sau un proiect
  - 4. o configuraţie de evenimente
  - 5. ruta prestabilită a unui avion
  - 6. model comportamental
- Are ca sinonim indicat termenul de **model**

Cum a apărut termenul?

Au preluat anagajatorii din IT modelul de interviu prezentat mai jos

Istoria evoluției conceptului în IT

- 1987 Cunningham şi Beck - limbaj
- 1990 “Gaşca celor patru” – G4 - catalog
- 1995 GoF - carte

Apoi...

- Riehle şi Zullighoven menţionează trei tipuri de modele software

Model conceptual

Model de proiectare

Model de programare

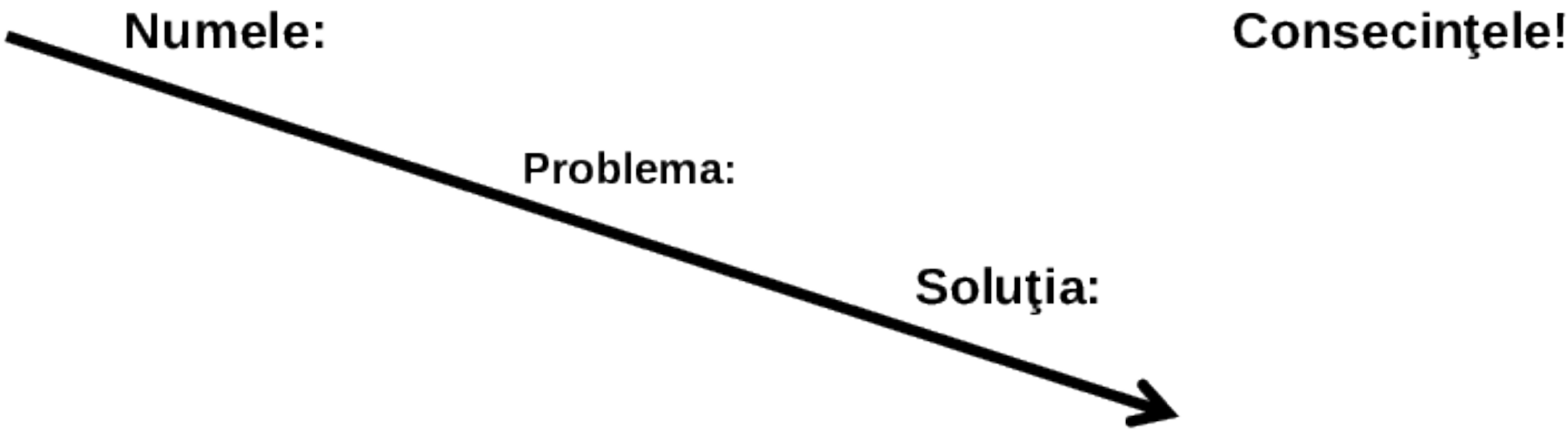
Unde sunt utile?

OOP+ADT

modelul aplicației

GoF

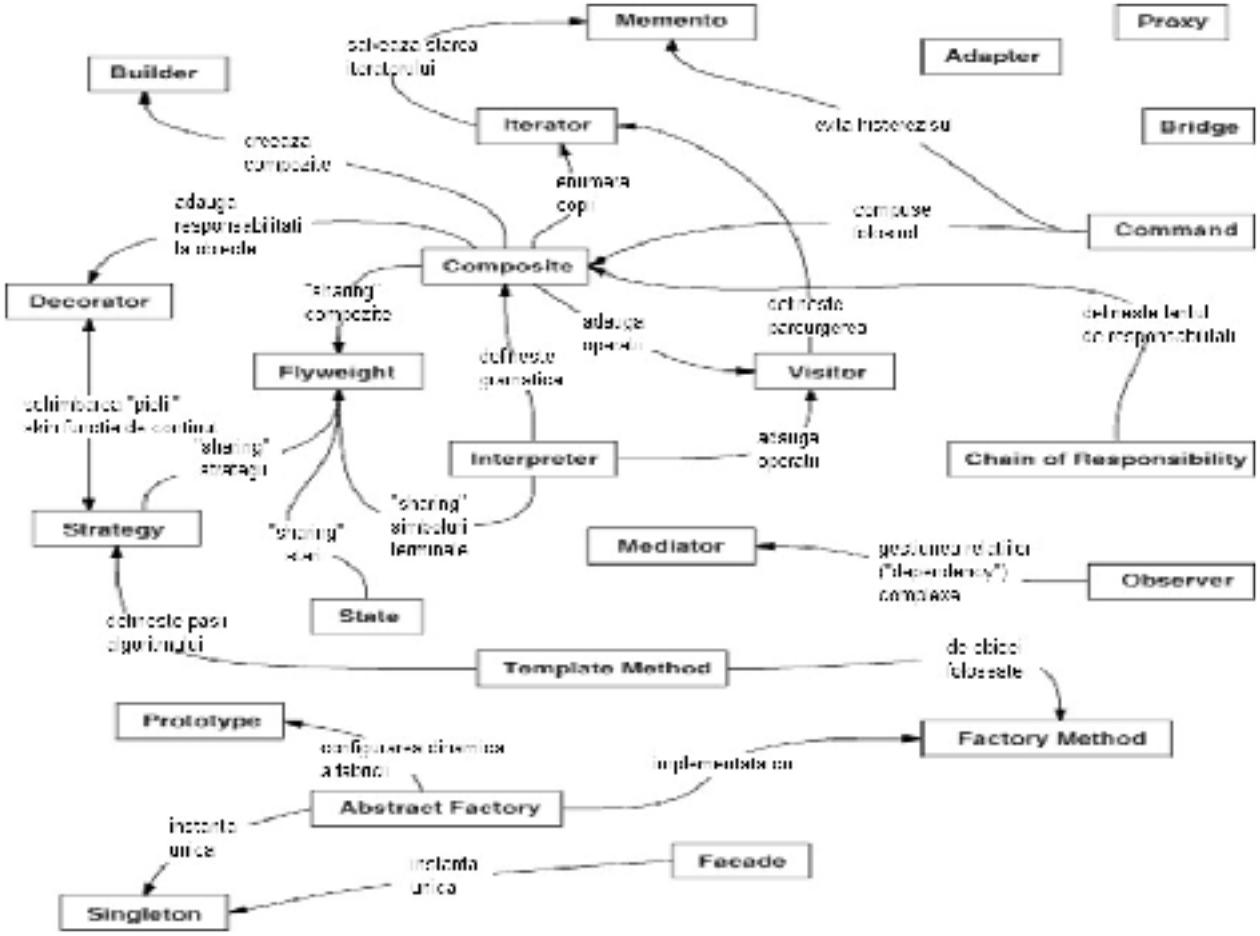
Elementele unui model



G4

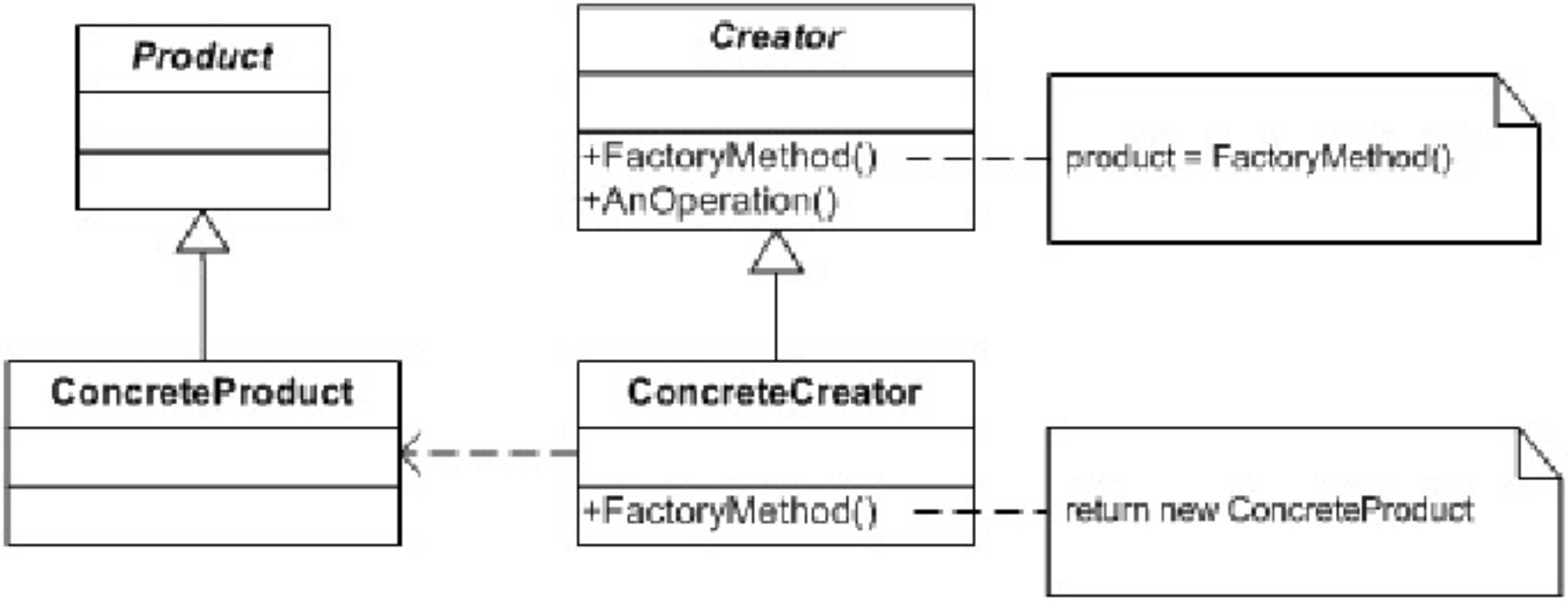
		Scop		
		Creaţional	Structural	**Comportamental**
Domeniu	Clasă	Fabric Method	Adapter (clasă)	**Interpreter** **Template Method**
	Obiect	**Abstract Fabrica** **Builder** **Prototype** **Singleton**	**Adapter (obiect)** **Bridge** **Composite** **Decorator** **Facade** **Flyweight** **Proxy**	**Chain of Responsibility** **Command** **Iterator** **Mediator** **Memento** **Observer** **State** **Strategy** **Visitor**

Relații între modelele GoF

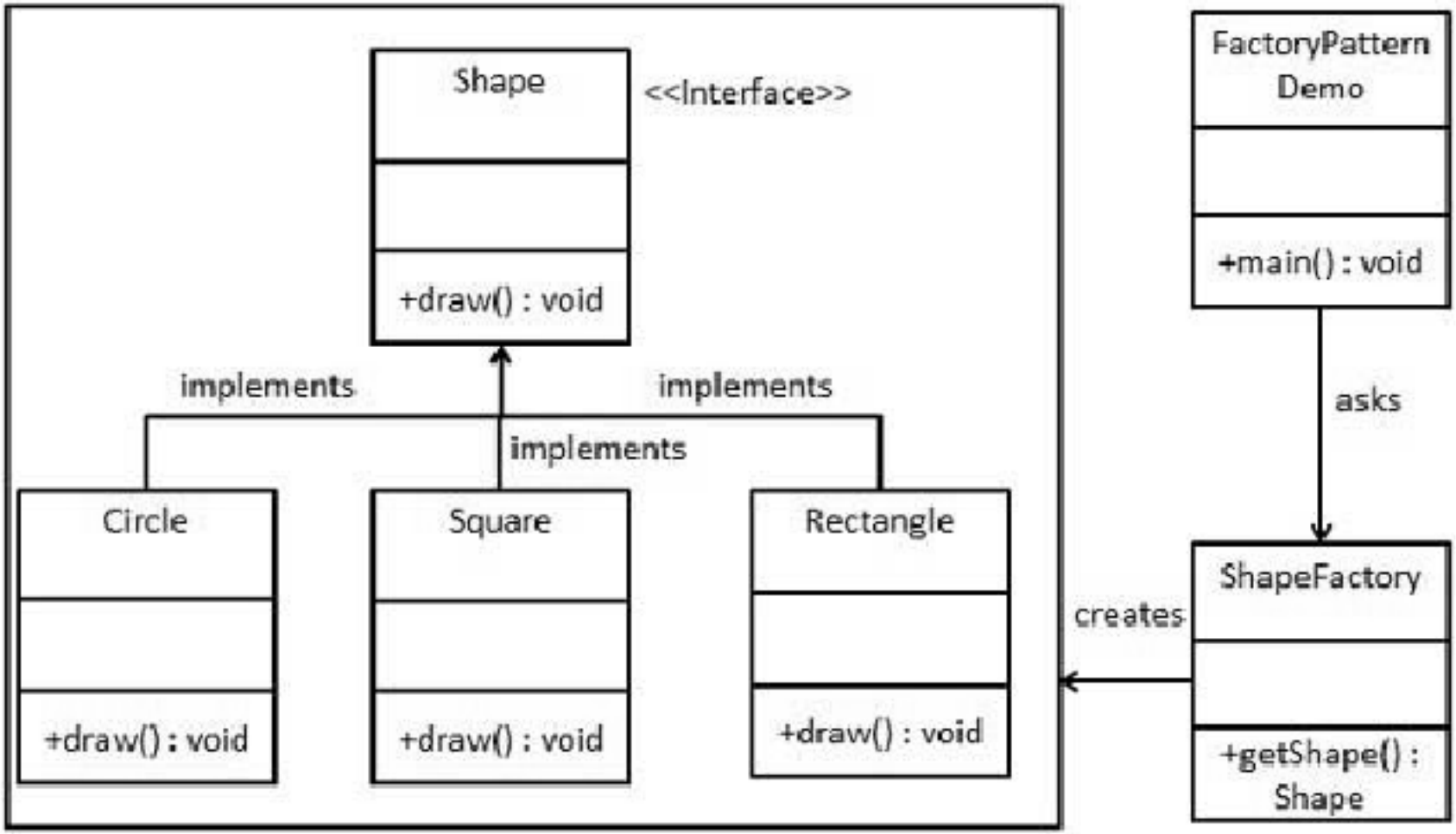


Modele creaționale

Modelul Fabrică de obiecte



Modelul Fabrică de obiecte - caz de utilizare

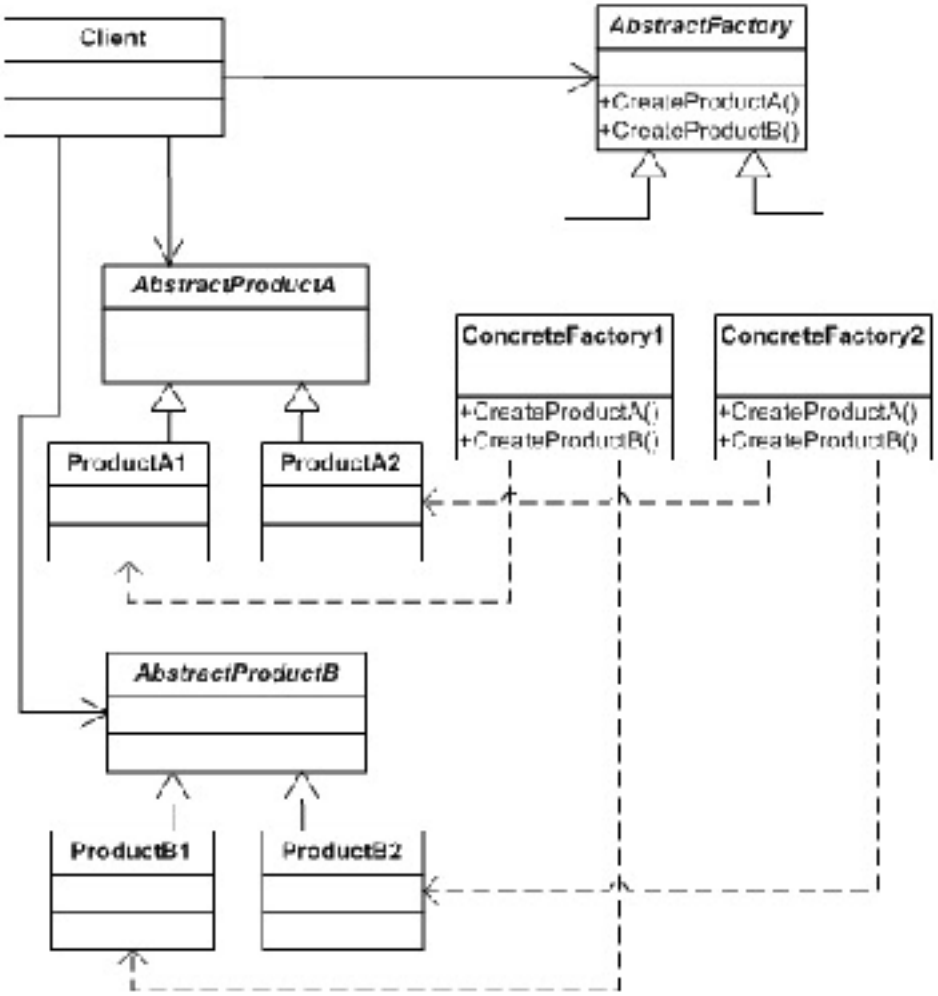


Modelul Fabrică de obiecte - caz de utilizare - implementare

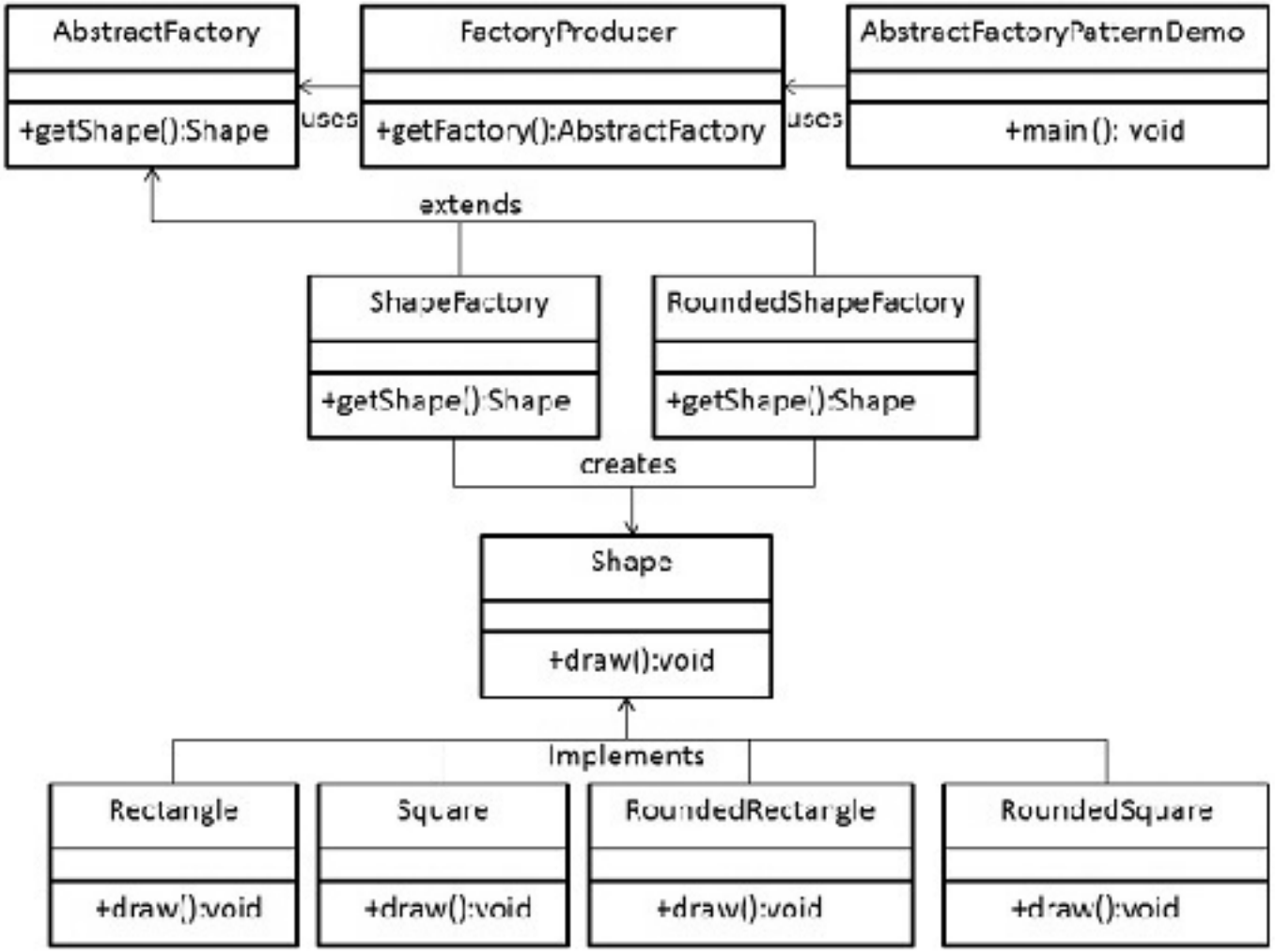
```
interface Shape
{ fun draw() }
class ShapeFactory {
  fun getShape(shapeType: String?): Shape?
   if (shapeType.equals("CIRCLE", true))
       return Circle()
   if (shapeType.equals("RECTANGLE", true))
       return Rectangle()
   if (shapeType.equals("SQUARE", true))
       return Square()
   return null  }
}

fun main(args: Array<String>)
{
    val shapeFactory = ShapeFactory()
    shapeFactory.getShape("CIRCLE")?.draw()
    shapeFactory.getShape("RECTANGLE")?.draw()
    shapeFactory.getShape("SQUARE")?.draw()
}
```

```
class Circle : Shape
{
    override fun draw()
    { println("Inside Circle::draw() method.") }
}

class Rectangle : Shape
{
    override fun draw()
    { println("Inside Rectangle::draw() method.") }
}
class Square : Shape
{
    override fun draw()
    { println("Inside Square::draw() method.") }
}
```

Model Fabrica abstractă



Modelul Fabrica abstractă - caz de utilizare



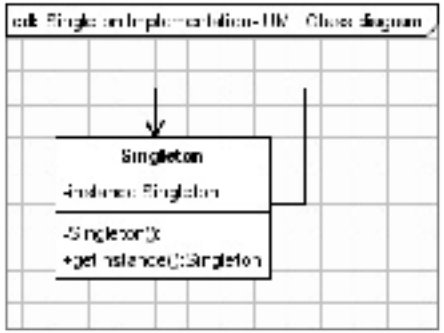
Modelul Fabrica abstractă - implementare

```
interface Shape
{    fun draw() }
interface Color
{ fun fill() }
abstract class AbstractFactory {
    abstract fun getColor(color: String): Color?
    abstract fun getShape(shape: String): Shape? }
class ShapeFactory : AbstractFactory() {
    override fun getShape(shape: String): Shape?
       { if (shape.equals("CIRCLE", true)) return Circle()
        if (shape.equals("RECTANGLE", true)) return Rectangle()
        if (shape.equals("SQUARE", true)) return Square()
        return null }
    override fun getColor(color: String): Color? = null }
class ColorFactory : AbstractFactory() {
    override fun getShape(shape: String): Shape? = null
    override fun getColor(color: String): Color?
       { if (color.equals("RED", true)) return Red()
        if (color.equals("GREEN", true)) return Green()
        if (color.equals("BLUE", true)) return Blue()
        return null } }
object FactoryProducer {
    fun getFactory(choice: String): AbstractFactory?
       { if (choice.equals("SHAPE", true)) return ShapeFactory()
        if (choice.equals("COLOR", true)) return ColorFactory()
        return null } }
```

```
class Circle : Shape {
    override fun draw()
    { println("Inside Circle::draw() method.") } }
class Square : Shape {
    override fun draw()
    { println("Inside Square::draw() method.") } }
class Rectangle : Shape {
    override fun draw()
    { println("Inside Rectangle::draw() method.") } }
class Red : Color {
    override fun fill()
    { println("Inside Red::fill() method.") } }
class Green : Color {
    override fun fill()
    { println("Inside Green::fill() method.") } }
class Blue : Color {
    override fun fill()
    { println("Inside Blue::fill() method.") } }
fun main(args: Array<String>)
{ val shapeFactory = FactoryProducer.getFactory("SHAPE")
    shapeFactory?.getShape("CIRCLE")?.draw()
    shapeFactory?.getShape("RECTANGLE")?.draw()
    shapeFactory?.getShape("SQUARE")?.draw()

    val colorFactory = FactoryProducer.getFactory("COLOR")
    colorFactory?.getColor("RED")?.fill()
    colorFactory?.getColor("GREEN")?.fill()
    colorFactory?.getColor("BLUE")?.fill()  }
```

Modelul burlacului

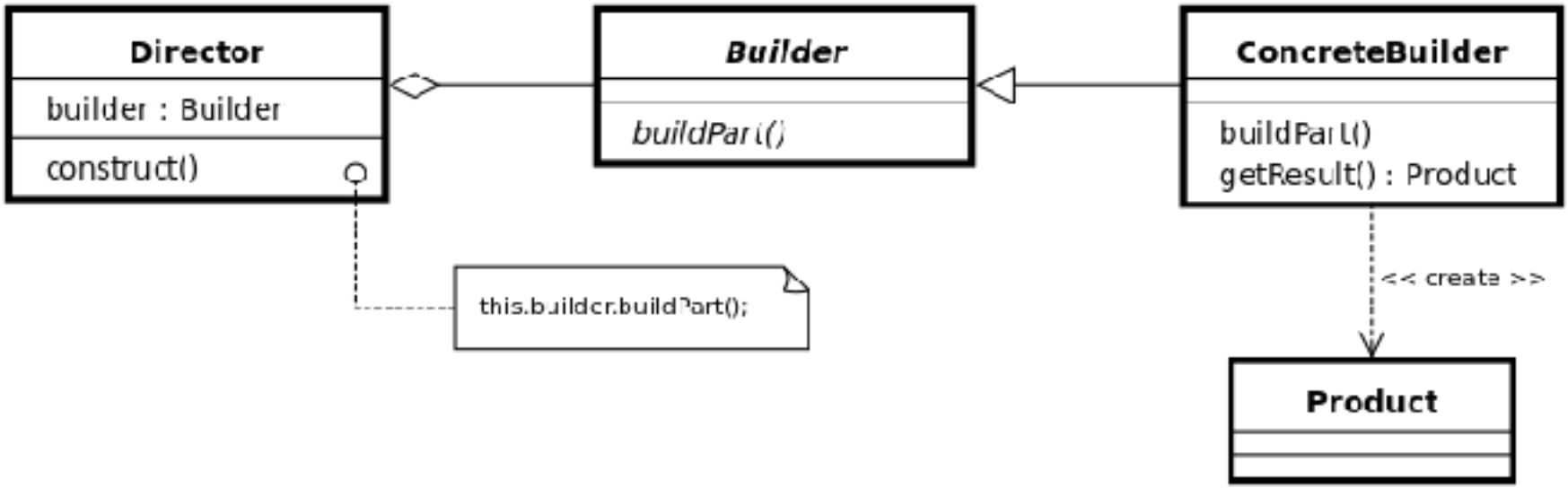
object burlac




```
object Payroll
{
val allEmployees = arrayListOf<Person>()
fun calculateSalary()
  {
  for (person in allEmployees)
     {
       ...
     }
  }
}
```

Modelul constructor

Modelul constructor



Model constructor - implementare concretă

```
data class Mail(val to: String,
    val title: String = "",
    val message: String = "",
    val cc: List<String> = listOf(),
    val bcc: List<String> = listOf(),
    val attachments: List<java.io.File> = listOf())

class MailBuilder(val to: String)
{
    private var mail: Mail = Mail(to)
    fun title(title: String): MailBuilder
    {
        mail.title = title
        return this
    }
    // acesta se repeta pentru alte variatii
    fun build(): Mail
    {  return mail  }
}
```

- și utilizare imediată:

```
val mail = Mail("one@recepient.org",
"Hi", "How are you")
```

- sau utilizare de obiect construit particularizat:

```
val email = MailBuilder("hello@hello.com").title
("What's up?").build()
```

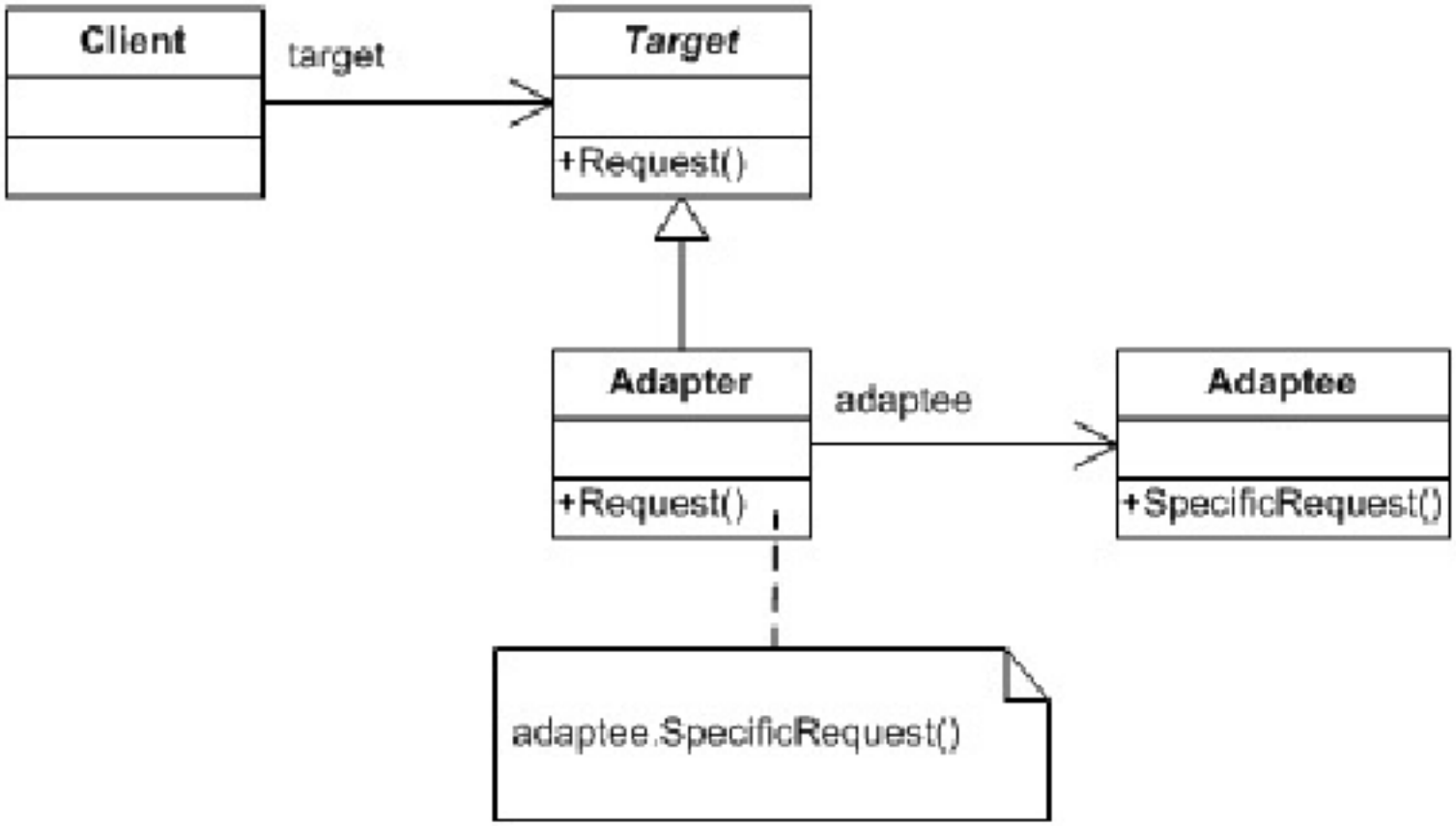
Modelul prototip

Model protitip - implemntare de caz

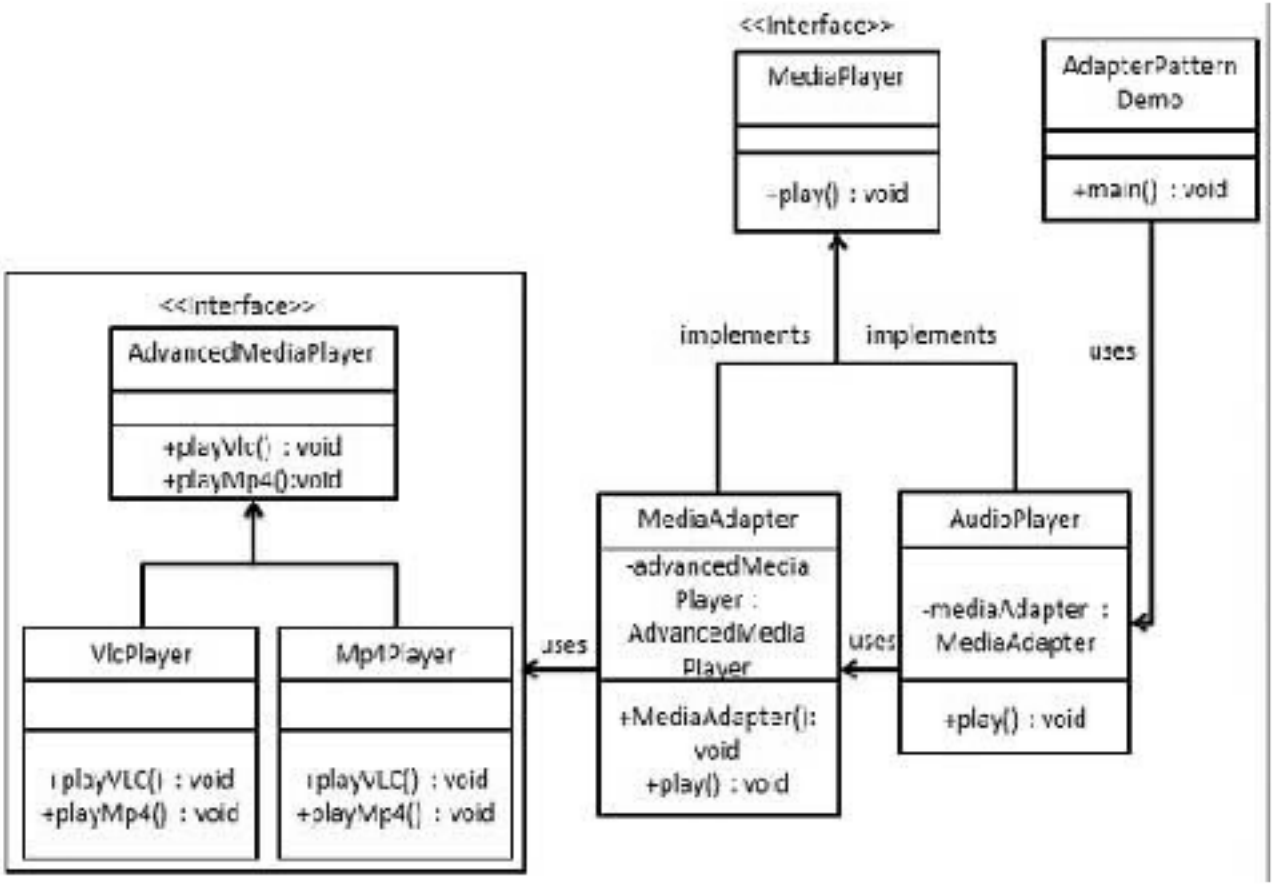
```
open class Bike : Cloneable
{
    private var gears: Int = 0
    private var bikeType: String? = null
    var model: String? = null
        private set

    init
    {
        bikeType = "Standard"
        model = "Carpati"
        gears = 4
    }

    public override fun clone(): Bike {
        return Bike()
    }

    fun makeAdvanced()
{
        bikeType = "Advanced"
        model = "Jaguar"
        gears = 6
    }
}

fun makeJaguar(basicBike: Bike): Bike
{
    basicBike.makeAdvanced()
    return basicBike
}

fun main(args: Array<String>)
{
    val bike = Bike()
    val basicBike = bike.clone()
    val advancedBike = makeJaguar(basicBike)
    println("Bicicleta mai buna: " + advancedBike.model!!)
}
```

Modele structurale

Modelul Adaptor



Model Adaptor - caz de utilizare

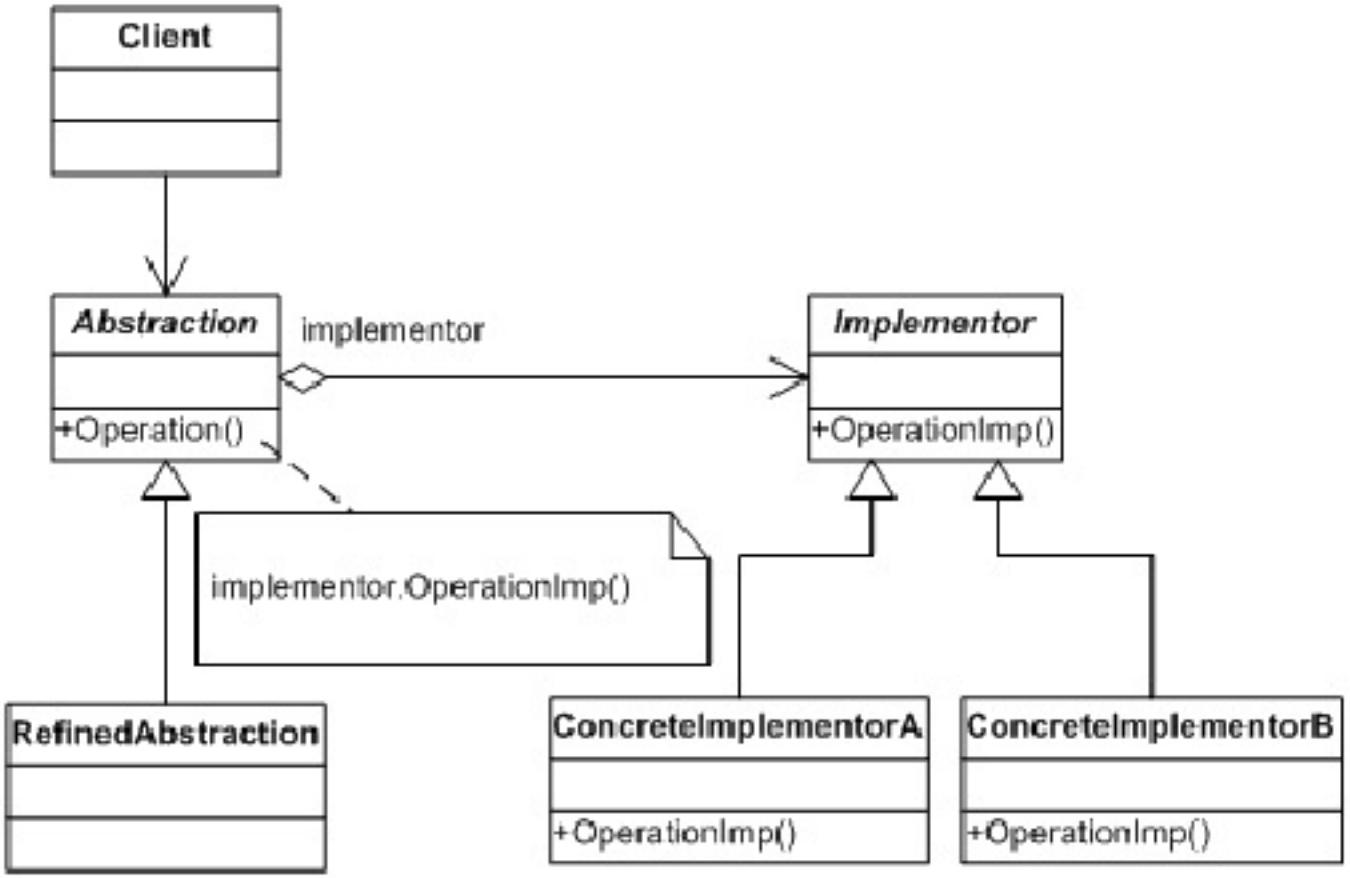


Model Adaptor - implementare

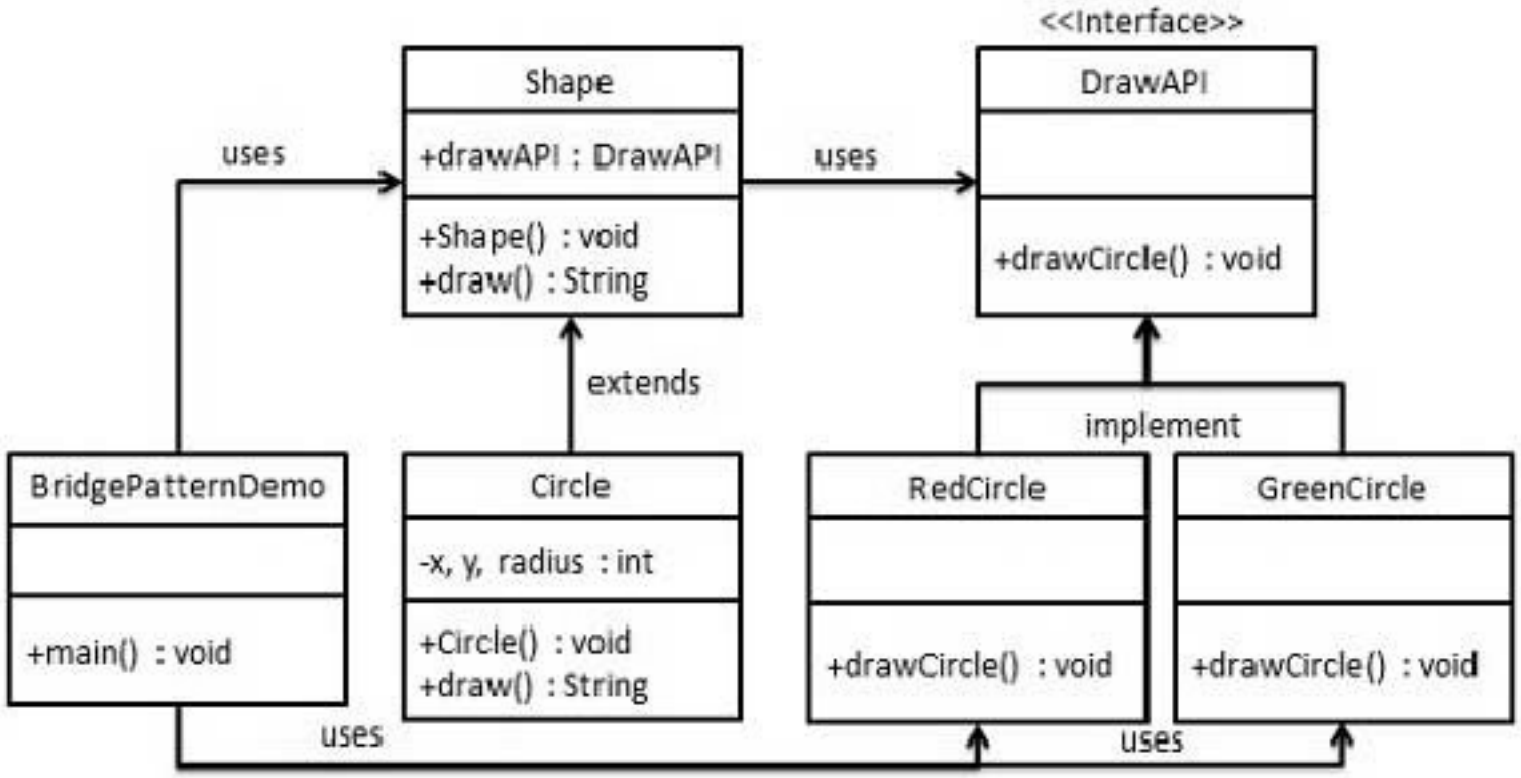
```
interface AdvanceMediaPlayer
  {fun playVlc(fileName: String)
   fun playMp4(fileName: String) }
interface MediaPlayer
  { fun play(audioType: String, fileName: String) }
open class MediaAdapter : MediaPlayer
  { private var advancedMusicPlayer: AdvanceMediaPlayer? = null
   override fun play(audioType: String, fileName: String)
   { if (audioType.equals("vlc", true))
        {if (advancedMusicPlayer == null)
             { advancedMusicPlayer = VlcPlayer() }
             advancedMusicPlayer?.playVlc(fileName) }
      else if (audioType.equals("mp4", true))
       {if (advancedMusicPlayer == null)
            { advancedMusicPlayer = Mp4Player()  }
            advancedMusicPlayer?.playMp4(fileName) }   }
}
class AudioPlayer : MediaAdapter()
{    override fun play(audioType: String, fileName: String)
    {  if (audioType.equals("mp3", true))
                { println("Playing mp3 file. Name: $fileName ") }
      else if (audioType.equals("vlc", true) || audioType.equals("mp4", true))
                { MediaAdapter().play(audioType, fileName) }
           else { println("Invalid media. $audioType format not supported") } }
    }
```

```
class Mp4Player : AdvanceMediaPlayer{
    override fun playMp4(fileName: String) {
        println("Playing mp4 file. Name: $fileName")
    }
    override fun playVlc(fileName: String) {
        println("Only support mp4 type")
    }
}
class VlcPlayer : AdvanceMediaPlayer{
    override fun playMp4(fileName: String) {
        println("Only support vlc type")
    }
    override fun playVlc(fileName: String) {
        println("Playing vlc file. Name: $fileName")
    }
}
fun main(args: Array<String>) {
    val audioPlayer = AudioPlayer()
    audioPlayer.play("mp3", "beyond the horizon.mp3")
    audioPlayer.play("mp4", "alone.mp4")
    audioPlayer.play("vlc", "far far away.vlc")
    audioPlayer.play("avi", "mind me.avi")
}
```

Modelul Pod


Client
Abstraction
implementor
Implementor
+Operation()
+OperationImp()
implementor.OperationImp()
RefinedAbstraction
ConcreteImplementorA
+OperationImp()
ConcreteImplementorB
+OperationImp()

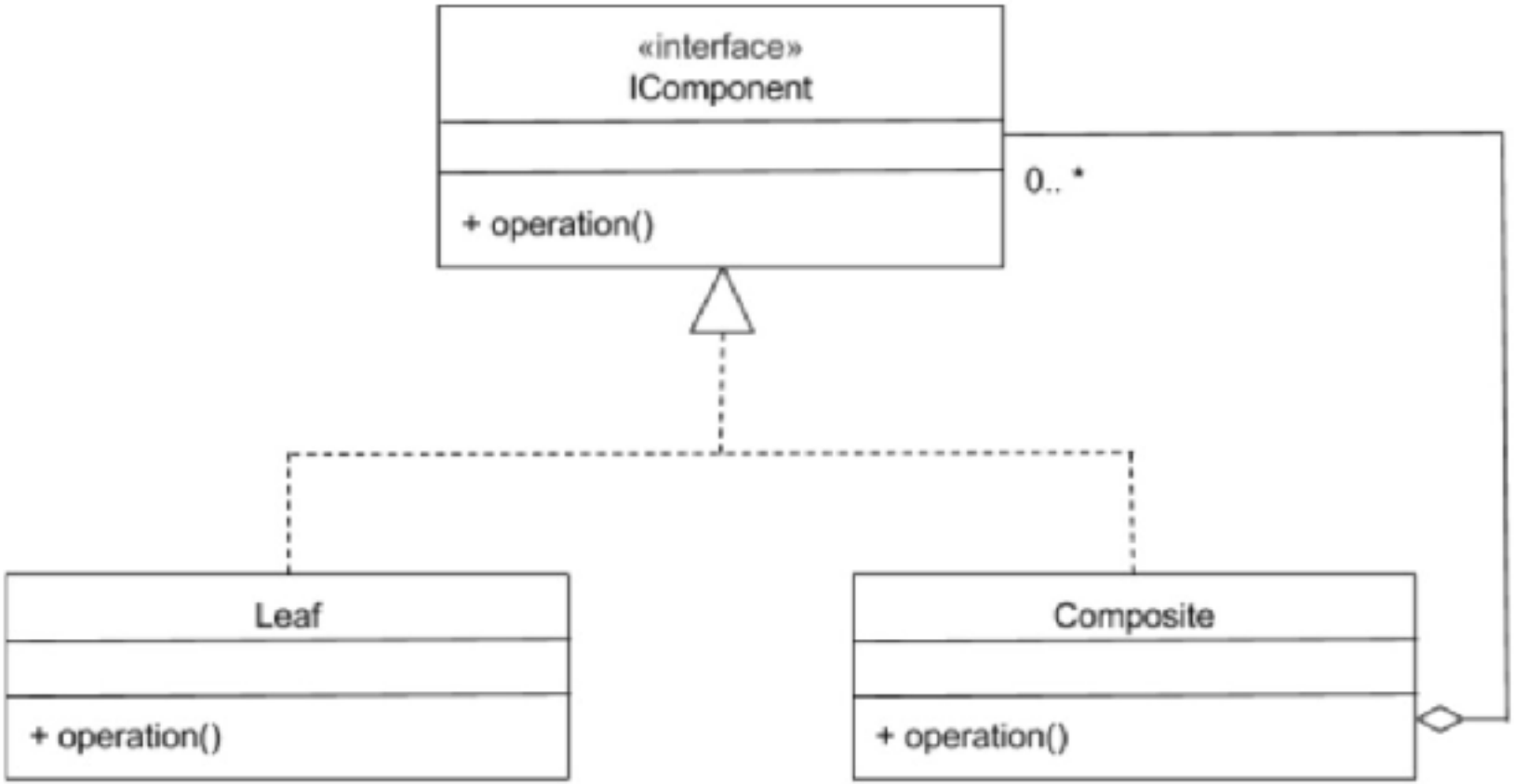
Modelul Pod - caz de utilizare



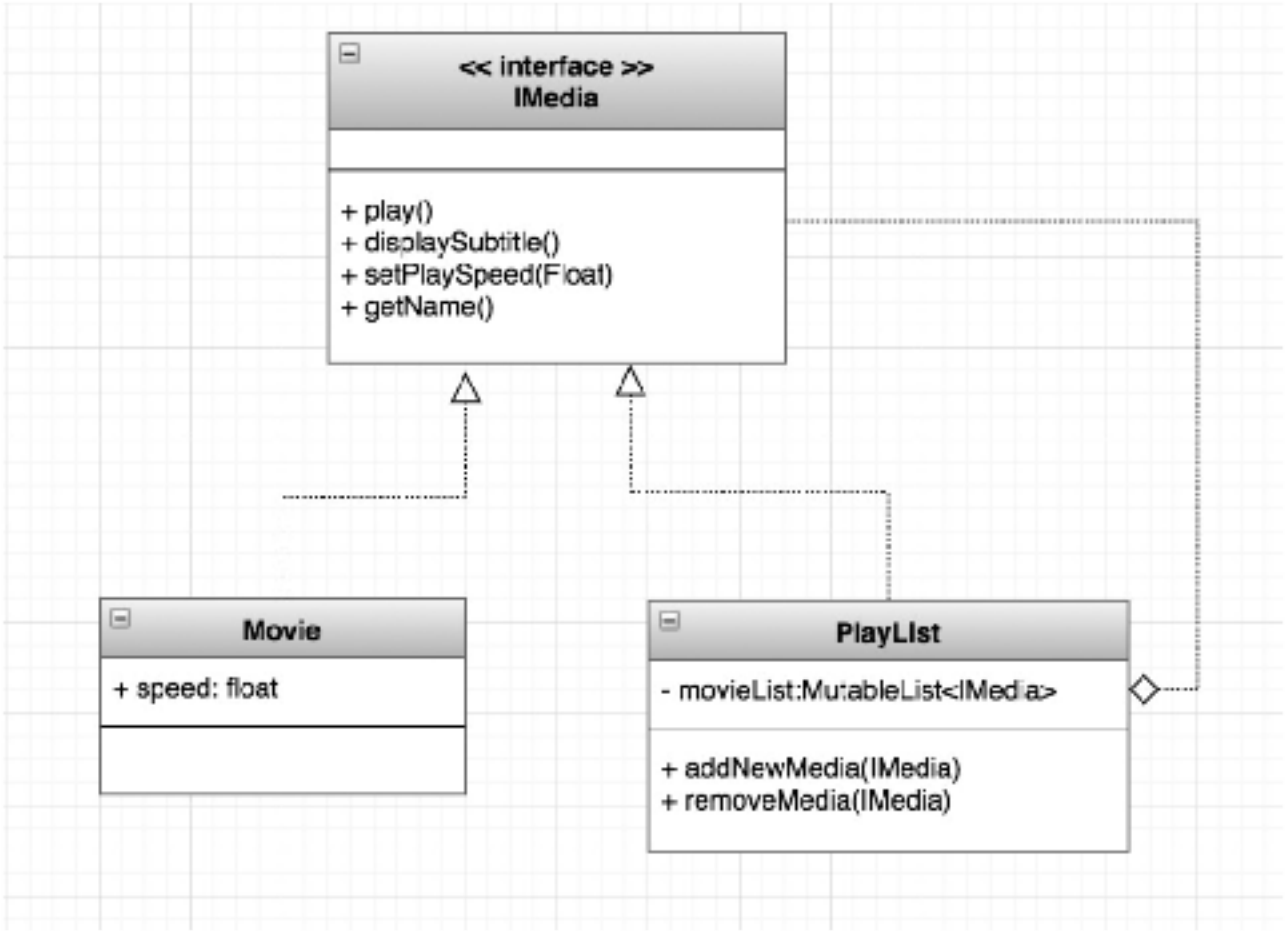
Modelul Pod - implementare

```
interface DrawAPI
{  fun drawCircle(radius: Int, x: Int, y: Int); }
abstract class Shape(protected val drawAPI: DrawAPI)
{ abstract fun draw()  }
class Circle(val x: Int, val y: Int, val radius: Int, drawAPI: DrawAPI) : Shape(drawAPI)
{  override fun draw()
   {  drawAPI.drawCircle(radius, x, y) }
}
class GreenCircle : DrawAPI
{  override fun drawCircle(radius: Int, x: Int, y: Int)
   {  println("Drawing Circle[ color: green, radius: $radius, x: $x, y: $y]")  }
}
class RedCircle : DrawAPI
{  override fun drawCircle(radius: Int, x: Int, y: Int)
   {  println("Drawing Circle[ color: red, radius: $radius, x: $x, y: $y]")  }
}
fun main(args: Array<String>)
{   val redCircle = Circle(100, 100, 10, RedCircle())
   val greenCircle = Circle(100, 100, 10, GreenCircle())
   redCircle.draw()
   greenCircle.draw()
}
```

Model Compus - forma generală



Model Compus - caz de utilizare

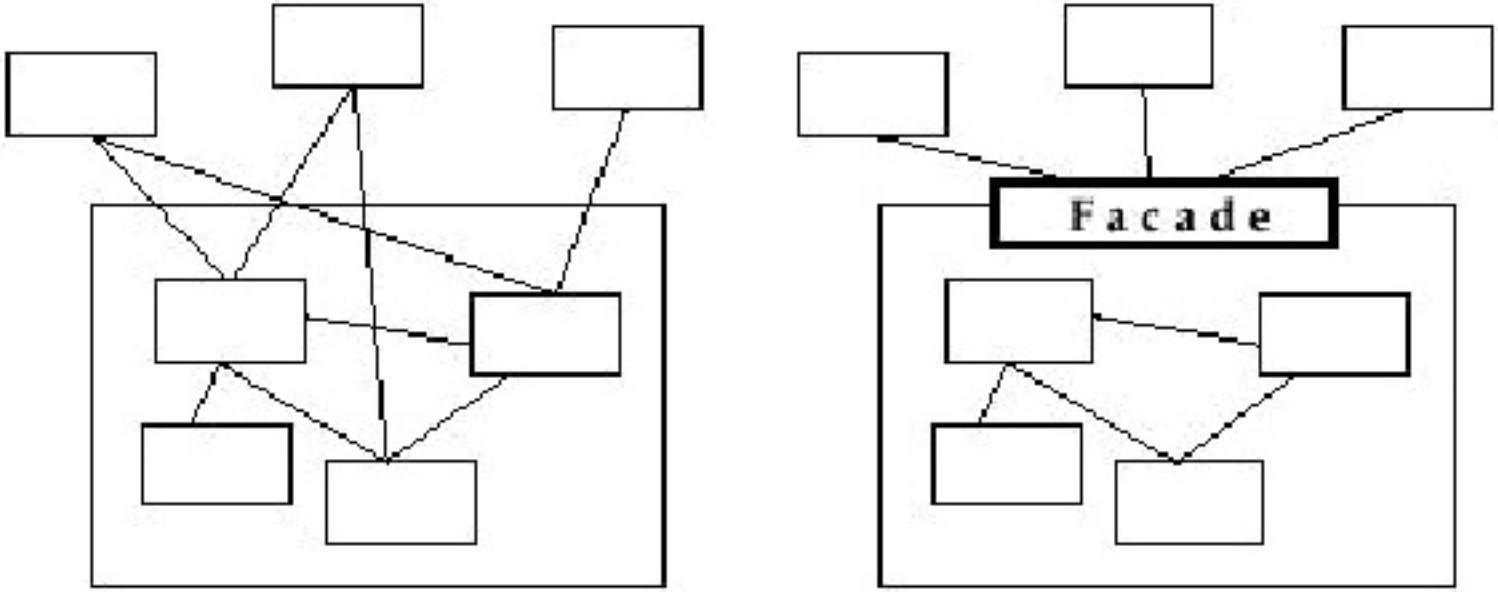


Compus - caz de utilizare - implementare

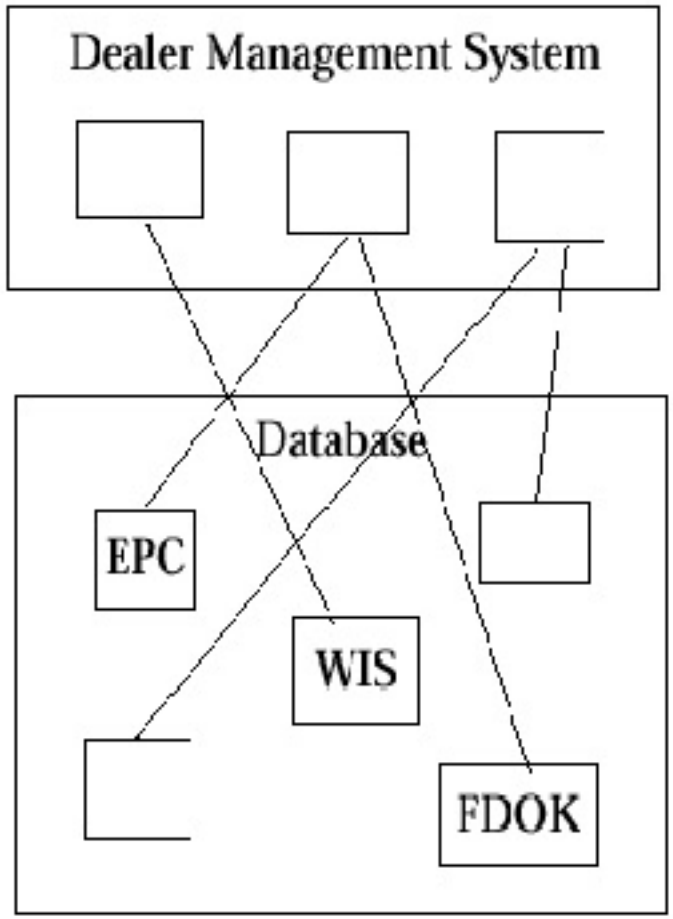
```
interface IMedia
  { fun play()
    fun displaySubtitle()
    fun setPlaySpeed(speed:Float)
    fun getName() :String }
class Movie(val title:String):IMedia
{   private var speed = 1f
    override fun play()
    {  println("Now playing: ${title}...")  }
    override fun displaySubtitle()
    {  println("display subtitle")  }
    override fun setPlaySpeed(speed:Float)
    {    this.speed = speed
         println("current play speed set to: $speed")   }
    override fun getName(): String
    {  return title  }
}
class PlayList(val title:String):IMedia
 {var movieList:MutableList<IMedia> = mutableListOf()
    fun addNewMedia(media:IMedia) = movieList.add(media)
    fun removeMedia(media: IMedia)
    { movieList = movieList.filter{ it.getName() !=
                        media.getName() }.toMutableList()  }
    override fun play()
    {  movieList.forEach { it.play() }
    override fun displaySubtitle()
    {  println("display certain subtitle") }
    override fun setPlaySpeed(speed: Float)
    { movieList.forEach { it.setPlaySpeed(speed) } }
    override fun getName(): String
    { return title }
}
```

```
fun main(args:Array<String>)
{
    val actionMoviePlayList:PlayList = PlayList("Action Movies")
    val movieB:IMedia = Movie("The Dark Knight")
    val movieC:IMedia = Movie("Inception")
    val movieD:IMedia = Movie("The Matrix")

    actionMoviePlayList.apply
     {
       addNewMedia(movieB)
       addNewMedia(movieC)
       addNewMedia(movieD)
     }

    val dramaPlayList:PlayList = PlayList("Drama Play List")
    val movie1:IMedia = Movie("The Godfather")
    val movie2:IMedia = Movie("The Shawshank Redemption")

    dramaPlayList.apply
      { addNewMedia(movie1);
        addNewMedia(movie2) }

    val myPlayList:PlayList = PlayList("My Play List")
    myPlayList.apply
      { addNewMedia(actionMoviePlayList)
        addNewMedia(dramaPlayList)  }
    myPlayList.play()
}
```

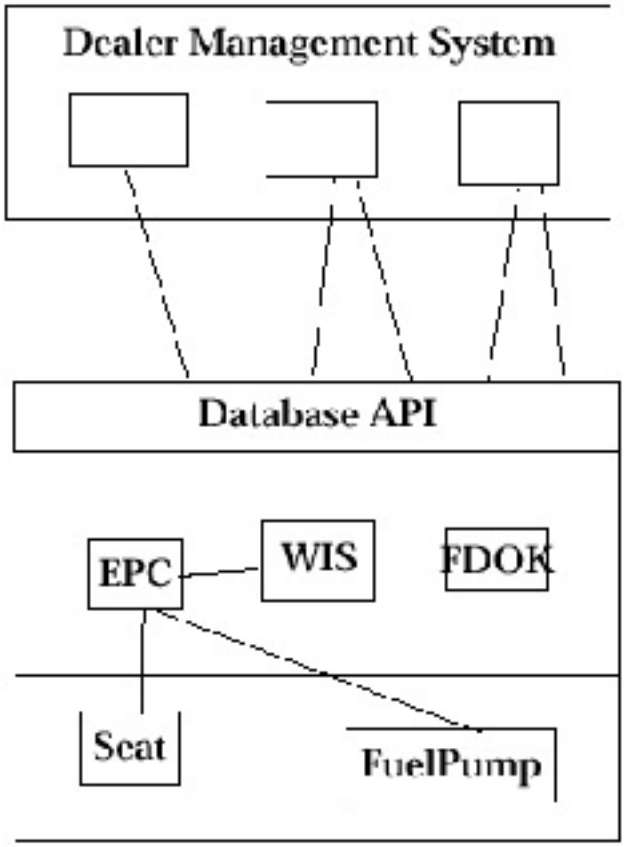
Model Fațadă

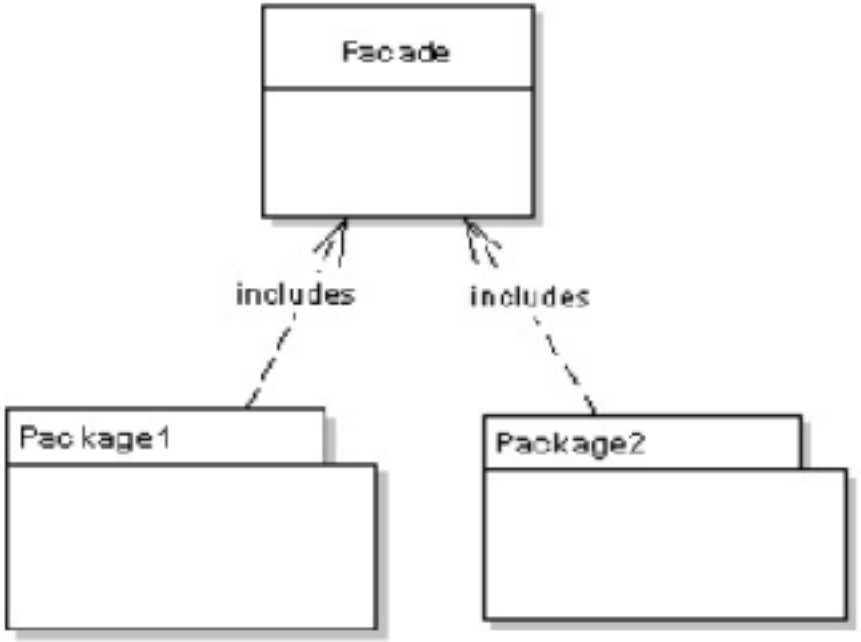


Arhitectură deschisă



Arhitectură închisă



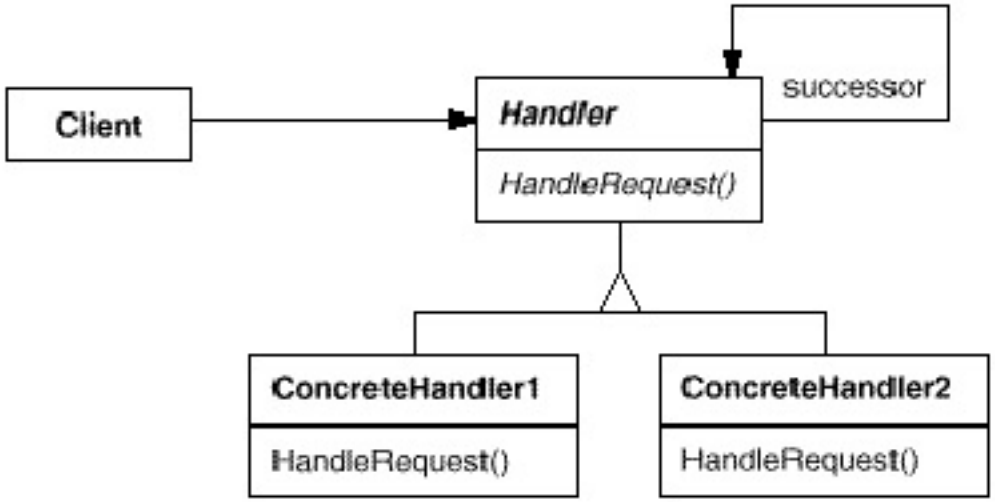


Model Fațadă - implementare

```
class CPU
{   fun freeze() = println("Freezing.")
    fun jump(position: Long) = println("Jump to $position.")
    fun execute() = println("Executing.")  }
class HardDrive
{    fun read(lba: Long, size: Int): ByteArray = byteArrayOf()   }
class Memory
{    fun load(position: Long, data: ByteArray) = println("Loading from memory position: $position") }

/* Fatada */
class Computer(val processor: CPU = CPU(), val ram: Memory = Memory(), val hd: HardDrive = HardDrive())
{  companion object
    {  val BOOT_ADDRESS = 0L
        val BOOT_SECTOR = 0L
        val SECTOR_SIZE = 0 }
    fun start()
    {  processor.freeze()
        ram.load(BOOT_ADDRESS, hd.read(BOOT_SECTOR, SECTOR_SIZE))
        processor.jump(BOOT_ADDRESS)
        processor.execute()  }
}
fun main(args: Array<String>)
{   val computer = Computer()
    computer.start() }
```

Modele comportamentale

Modelul lanț de responsabilități



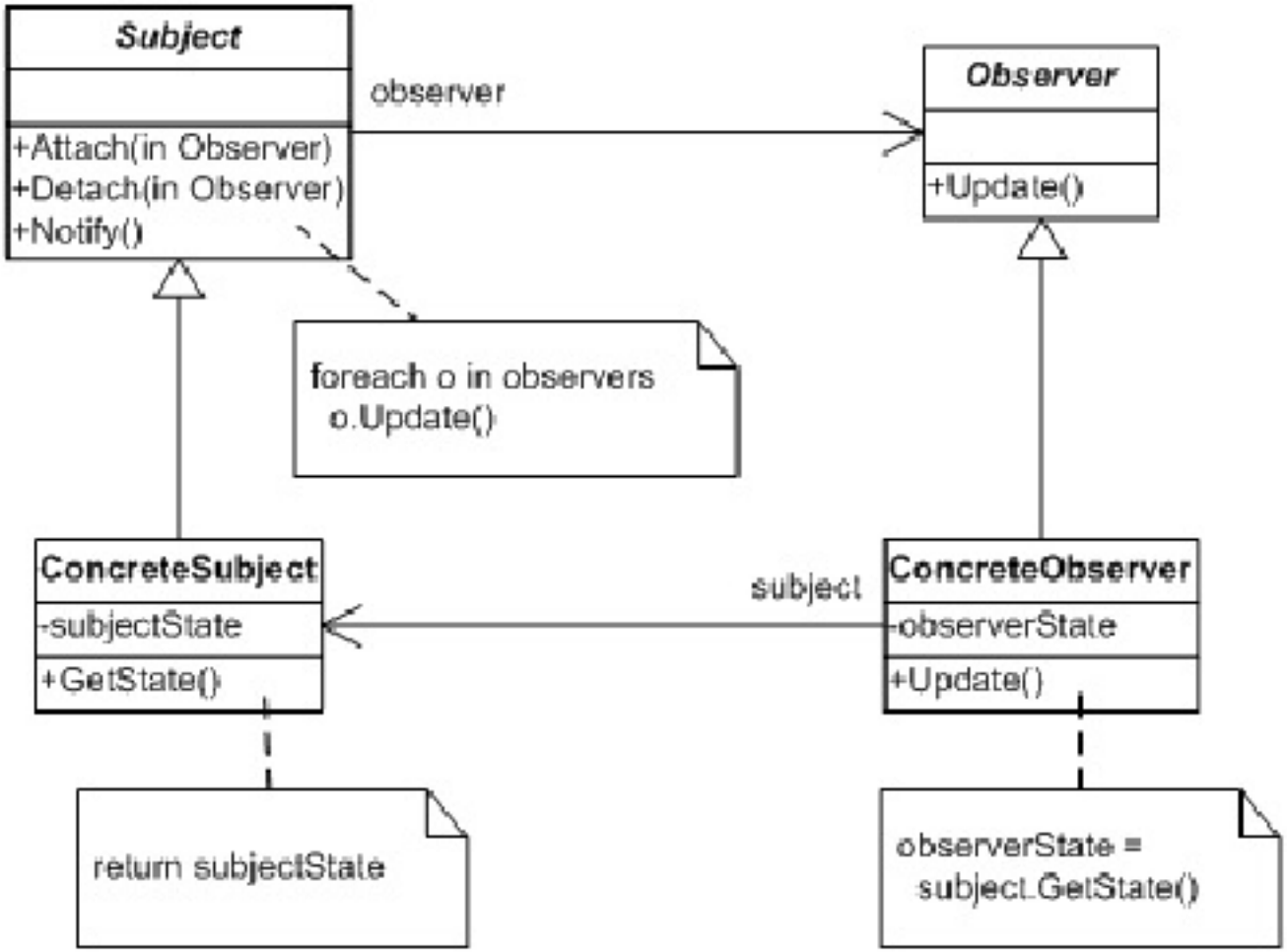
Unde o structură tipică de înlănțuire de obiecte ar fi

Modelul lanț de responsabilități - implementare

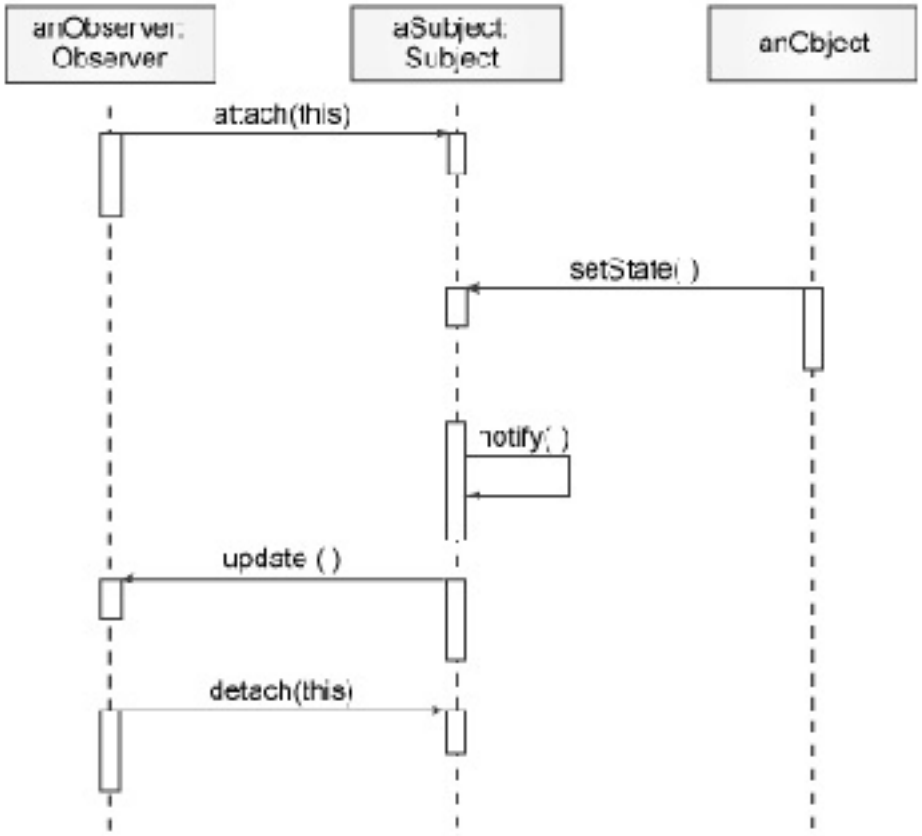
```
import org.assertj.core.api.Assertions.assertThat
import org.junit.jupiter.api.Test
interface HeadersChain
{    fun addHeader(inputHeader: String): String   }
class AuthenticationHeader(val token: String?, var next: HeadersChain? =
null) : HeadersChain
{    override fun addHeader(inputHeader: String): String
     { token ?: throw IllegalStateException("Token should be not null")
       return inputHeader + "Authorization: Bearer $token\n"
           .let{ next?.addHeader(it) ?: it } }     }
class ContentTypeHeader(val contentType: String, var next: HeadersChain?
= null) : HeadersChain
{  override fun addHeader(inputHeader: String): String =
       inputHeader + "ContentType: $contentType\n"
           .let{ next?.addHeader(it) ?: it } }
class BodyPayload(val body: String, var next: HeadersChain? = null) :
HeadersChain
{    override fun addHeader(inputHeader: String): String =
       inputHeader + "$body"
           .let{ next?.addHeader(it) ?: it }   }
class ChainOfResponsibilityTest
{    @Test
    fun `Chain Of Responsibility`()
     { //crearea elemnteloru lantuli
       val authenticationHeader = AuthenticationHeader("123456")
       val contentTypeHeader = ContentTypeHeader("json")
       val messageBody =
BodyPayload("Body:\n{\n\"username\"=\"dbacinski\"\n}")
//se construieste lantul
authenticationHeader.next = contentTypeHeader
contentTypeHeader.next = messageBody
//se executa lantul
val messageWithAuthentication =
   authenticationHeader.addHeader("Headers with Authentication:\n")
println(messageWithAuthentication)
val messageWithoutAuth =
   contentTypeHeader.addHeader("Headers:\n")
println(messageWithoutAuth)
assertThat(messageWithAuthentication).isEqualTo
(     """
       Headers with Authentication:
       Authorization: Bearer 123456
       ContentType: json
       Body:
       { "username"="bonjovi2987" }
       """.trimIndent() )
assertThat(messageWithoutAuth).isEqualTo
(     """
       Headers:
       ContentType: json
       Body:
       { "username"="dbacinski" }
   """.trimIndent() )
 }
}
```

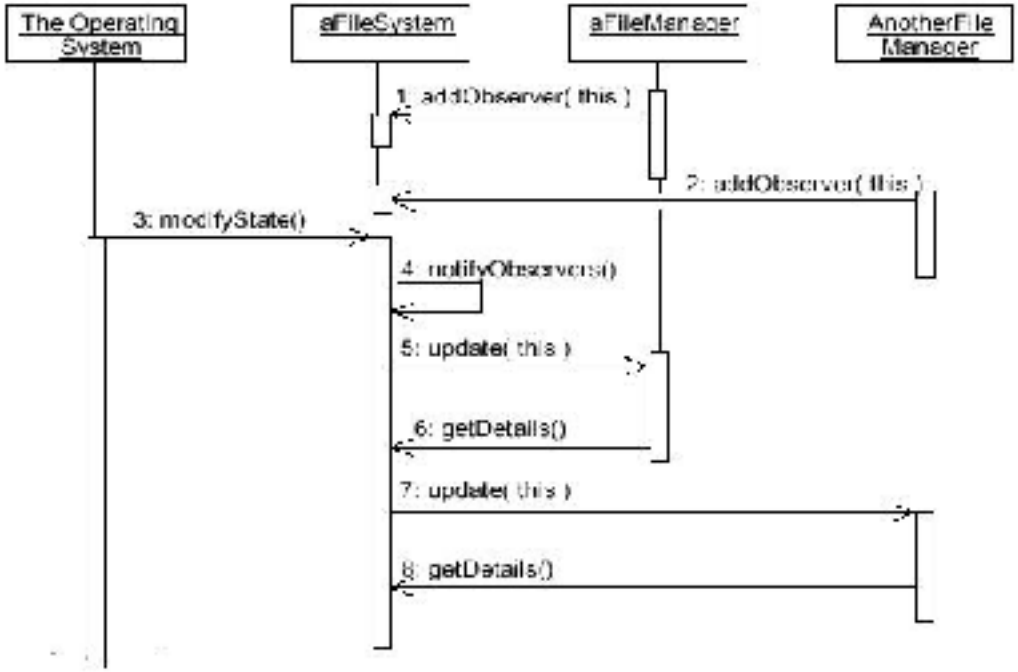
Modelul Observator

Modelul Observator

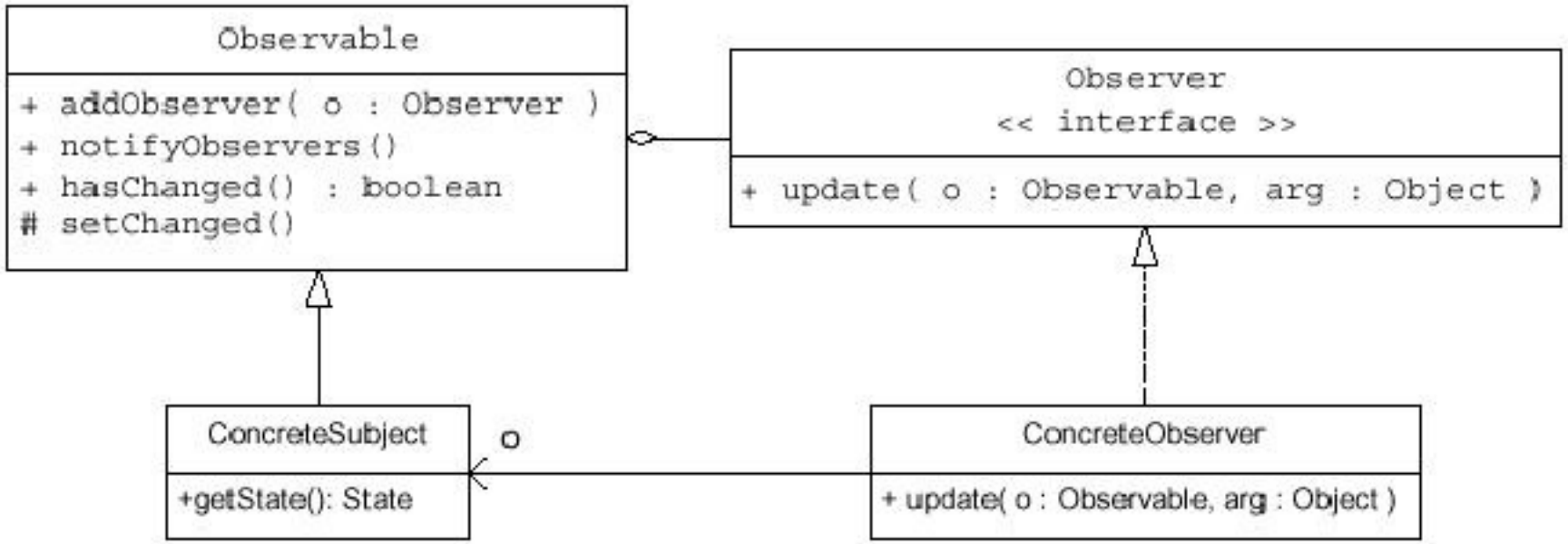


Model Observator - Secvențiere temporală





Suportul apelat din spate de la Java



```
interface ValueChangeListener {
    fun onValueChanged(newValue:String)
}

class PrintingTextChangedListener : ValueChangeListener {
    override fun onValueChanged(newValue: String) =
            println("Text is changed to: $newValue")
}

class ObservableObject(listener:ValueChangeListener) {
    var text: String by Delegates.observable(
            initialValue = "",
            onChange = {
                prop, old, new ->
                listener.onValueChanged(new)
            })
}

fun main(args: Array<String>) {
    val observableObject = ObservableObject(PrintingTextChangedListener())
    observableObject.text = "Hello"
    observableObject.text = "There"
}
```

Model Observator - Caz de utilizare

```
import kotlin.properties.ObservableProperty
//wrapper peste suportul - Java Observer
import kotlin.properties.ReadWriteProperty
import kotlin.reflect.KProperty

inline fun <T> observable(initialValue: T,
crossinline beforeChange:
        (property: KProperty<*>, oldValue: T, newValue: T) -> Boolean,
crossinline afterChange:
        (property: KProperty<*>, oldValue: T, newValue: T) ->Unit):
             ReadWriteProperty<Any?, T> = object :
  ObservableProperty<T>(initialValue)
   { override fun afterChange(property: KProperty<*>, oldValue: T,
newValue: T) = afterChange(property, oldValue, newValue)
     override fun beforeChange(property: KProperty<*>, oldValue: T,
newValue: T) = beforeChange(property, oldValue, newValue)
    }
interface PropertyObserver
{   fun willChange(propertyName: String, newPropertyValue: Any?)
    fun didChange(propertyName: String, oldPropertyValue: Any?)   }
```

```
class Observer : PropertyObserver
{  override fun willChange(propertyName: String, newPropertyValue: Any?)
    { if (newPropertyValue is String && newPropertyValue == "test")
        { println("Okay. Look. ,,,,,,,,") }
  }
override fun didChange(propertyName: String, oldPropertyValue: Any?)
{ if (oldPropertyValue is String && oldPropertyValue == "<no name>")
     { println("Sorry about the mess..") }
 }
}
class User(val propertyObserver: PropertyObserver?)
{   var name: String by observable("<no name>",
    { prop, old, new ->
      println("Before change: $old -> $new")
      propertyObserver?.willChange(name, new)
      return@observable true},
     { prop, old, new ->
      propertyObserver?.didChange(name, old)
      println("After change: $old -> $new")
     })
}
fun main(args: Array<String>)
{   val observer = Observer()
    val user = User(observer)
    user.name = "test" }
```

Modelul automatului finit State ????

Modelul automatului finit - caz de utilizare

```
class CoffeeMachine
{   var state: CoffeeMachineState
    val MAX_BEANS_QUANTITY = 100
    val MAX_WATER_QUANTITY = 100
    var beansQuantity = 0
    var waterQuantity = 0
    val offState = Off(this)
    val noIngredients = NoIngredients(this)
    val ready = Ready(this)
    init  {   state = offState  }
    fun turnOn() = state.turnOn()
    fun fillInBeans(quantity: Int) = state.fillInBeans(quantity)
    fun fillInWater(quantity: Int) = state.fillInWater(quantity)
    fun makeCoffee() = state.makeCoffee()
    fun turnOff() = state.turnOff()
    override fun toString(): String
    {   return """COFFEE MACHINE → ${state::class.simpleName}
        |  water quantity : $waterQuantity
        |  beans quantity : $beansQuantity
        |""".trimMargin() } }
abstract class CoffeeMachineState(val coffeeMachine: CoffeeMachine)
{    open fun makeCoffee(): Unit = throw UnsupportedOperationException("Op. not supported")
    open fun fillInBeans(quantity: Int): Unit = throw UnsupportedOperationException("Operation
not supported")
    open fun fillInWater(quantity: Int): Unit = throw UnsupportedOperationException("Operation
not supported")
    open fun turnOn(): Unit = throw UnsupportedOperationException("Operation not supported")
    fun turnOff(): Unit
    {   coffeeMachine.state = coffeeMachine.offState }   }
class Off(coffeeMachine: CoffeeMachine) : CoffeeMachineState(coffeeMachine)
{    override fun turnOn()
    {  coffeeMachine.state = coffeeMachine.noIngredients
       println("Coffee machine turned on") } }
class NoIngredients(coffeeMachine: CoffeeMachine) : CoffeeMachineState(coffeeMachine)
{   override fun fillInBeans(quantity: Int)
    {  if ((coffeeMachine.beansQuantity + quantity) <=
coffeeMachine.MAX_BEANS_QUANTITY)
          {  coffeeMachine.beansQuantity += quantity
             println("Beans filled in")
             if (coffeeMachine.waterQuantity > 0)
             {  coffeeMachine.state = coffeeMachine.ready  }  }     }
   override fun fillInWater(quantity: Int)
    { if ((coffeeMachine.waterQuantity + quantity) <=
coffeeMachine.MAX_WATER_QUANTITY)
         { coffeeMachine.waterQuantity += quantity
           println("Water filled in")
           if (coffeeMachine.beansQuantity > 0)
             {   coffeeMachine.state = coffeeMachine.ready } }  }  }
class Ready(coffeeMachine: CoffeeMachine) : CoffeeMachineState(coffeeMachine)
{    override fun makeCoffee()
     { coffeeMachine.beansQuantity--
       coffeeMachine.waterQuantity--
       println("Making coffee ... DONE")
       if (coffeeMachine.beansQuantity == 0 || coffeeMachine.waterQuantity == 0)
          {  coffeeMachine.state = coffeeMachine.noIngredients  }   }   }
fun main(args: Array<String>) {
   val coffeeMachine = CoffeeMachine()
   coffeeMachine.turnOn()
   println(coffeeMachine)
   coffeeMachine.fillInBeans(2)
   println(coffeeMachine)
   coffeeMachine.fillInWater(2)
   println(coffeeMachine)
   coffeeMachine.makeCoffee()
   println(coffeeMachine)
   coffeeMachine.makeCoffee()
   println(coffeeMachine)
   coffeeMachine.turnOff()
   println(coffeeMachine)        }
```

Modelul Vizitator? pa...pa...

Exemplu de tratare

```
abstract class ResultOrErrorVisitor<T>
{
abstract T visit(Result<T> result);
abstract T visit(Error<T> error);
}
interface ResultOrError<T>
{  T accept(ResultOrErrorVisitor<T> visitor);  }

class Result<T> implements ResultOrError<T>
{
   public final T result;
   Result(T result)
       {    this.result = result;  }
  @Override
  public T accept(ResultOrErrorVisitor<T> visitor)
       {   return visitor.visit(this);  }  }

class Error<T> implements ResultOrError<T>
{
   public final Throwable error;
   Error(Throwable error) { this.error = error; }
  @Override
  public T accept(ResultOrErrorVisitor<T> visitor)
   {   return visitor.visit(this); }
}
```

În Kotlin

```
sealed class ResultOrError<out T>
{
  data class Result<out T>(val result: T) : ResultOrError<T>()
  data class Error<out T>(val error: Throwable) : ResultOrError<T>()
}
```

Modelul comandă

```
data class User(val name: String) {
}

interface Command
{
    fun execute(user: User)
}

class AddUser : Command
{
    override fun execute(user: User)
    { println("Adding a new user with name: "+
user.name)  }
}

class DeleteUser : Command
{
    override fun execute(user: User)
   { println("Deleting user with name: "+user.name) }
}
```

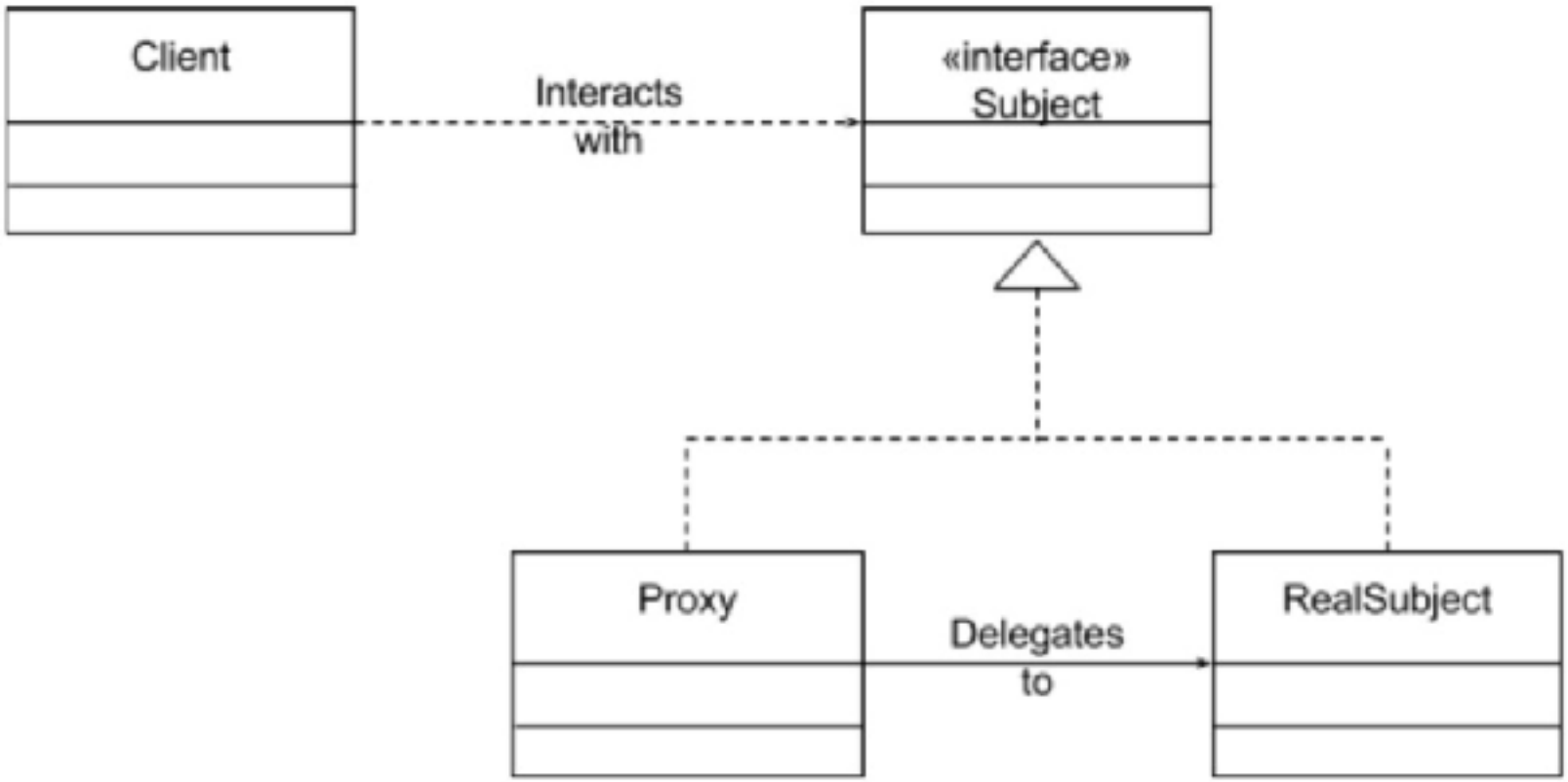
```
class EditUser : Command
{
    override fun execute(user: User)
    {  println("Editing user with name: "+user.name) }
}

fun main(args: Array<String>)
{
    var user = User("Kotlin")

    var add = AddUser()
    add.execute(user)

    var edit = EditUser();
    edit.execute(user)

    var delete = DeleteUser()
    delete.execute(user)

}
```

modelul restaurare/reamintire (latină: memento)

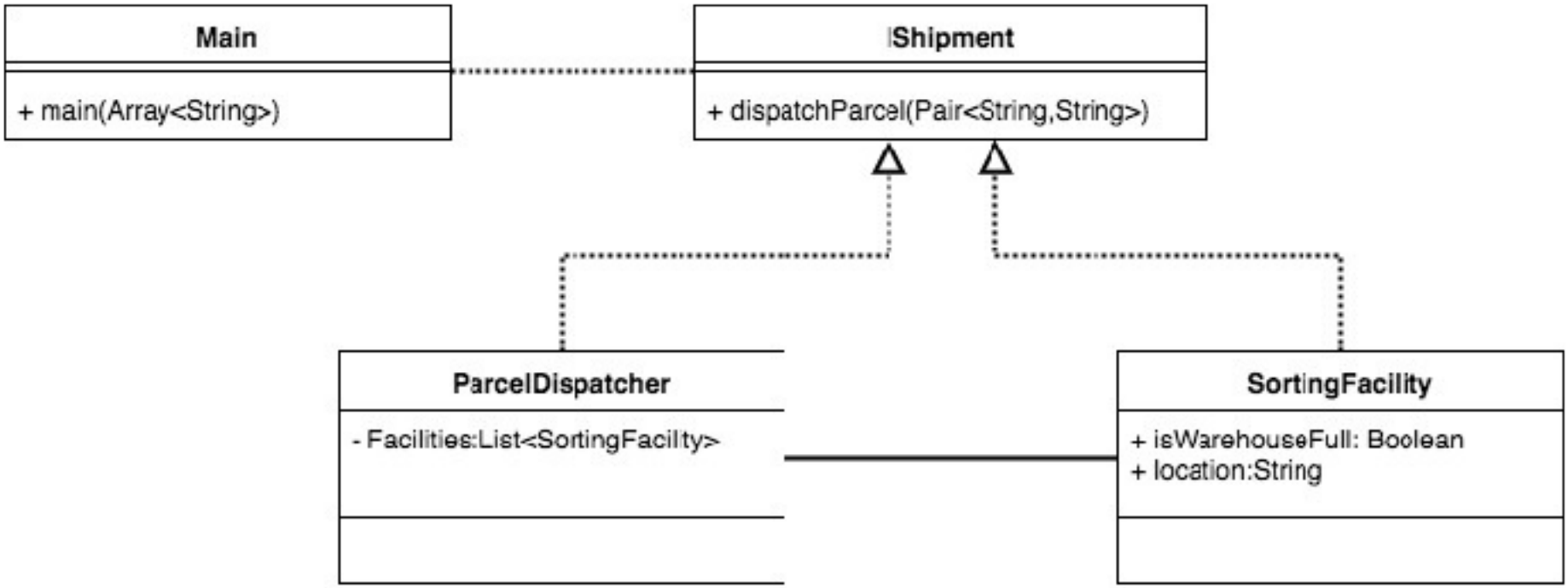
```
import kotlin.collections.ArrayList
data class Memento(val state: String)
class Originator
{ //acest String este doar pentru exemplu in realitate
//acesta ar fi inlocuit de obiectul al carui stare
//trebuie salvata si apoi restaurata
    var state: String? = null
    fun createMemento(): Memento
    {  return Memento(state!!) }
    fun setMemento(memento: Memento)
    {  state = memento.state }
}

class Caretaker
{  private val statesList = ArrayList<Memento>()
    fun addMemento(m: Memento)
    {  statesList.add(m) }
    fun getMemento(index: Int): Memento
    {  return statesList.get(index)  }
}
```

```
fun main(args: Array<String>)
{  val originator = Originator()
    originator.state = "Ironman"
    var memento = originator.createMemento()
    val caretaker = Caretaker()
    caretaker.addMemento(memento)
    originator.state = "Captain America"
    originator.state = "Hulk"
    memento = originator.createMemento()
    caretaker.addMemento(memento)
    originator.state = "Thor"
    println("Stare curenta Origine:" + originator.state!!)
    println("Restaurare Origine...")
    memento = caretaker.getMemento(1)
    originator.setMemento(memento)
    println("Stare curenta Origine: " + originator.state!!)
    println("Din nou Restaurare...")
    memento = caretaker.getMemento(0)
    originator.setMemento(memento)
    println("Stare curenta Origine:" + originator.state!!)
}
```

Intermediar - forma generală



Intermediar - caz de utilizare



Intermediar - caz de utilizare - implementare

```
interface IShipment
{   // pachetul este reprezentat de o pereche
    // unde primul String - continutul pachetului iar al doilea locația
    // (conținut to locație )
    fun dispatchParcel(parcel:Pair<String,String>)
}
class SortingFacility(val location:String,var
isWarehouseFull:Boolean) : IShipment
    {override fun dispatchParcel(parcel: Pair<String, String>)
       { println("${location} facility doing dispatching business...") }
    }

class ParcelDispatcher : IShipment
{ // locația a fost aleasă ca string din motive academice!
    private var facility = listOf<SortingFacility>
    (
            SortingFacility("North",true),
            SortingFacility("North West",false),
            SortingFacility("South",false),
            SortingFacility("West",true),
            SortingFacility("East",false)
    )
}
    override fun dispatchParcel(parcel: Pair<String, String>)
       {
       val facilityNearTpParcelLocation  = facility.filter
                        { it.location.contains(parcel.second,true)
                                        && !it.isWarehouseFull }.first()
       facilityNearTpParcelLocation.dispatchParcel(parcel)
    }


}

fun main(args:Array<String>)
{
    var pachet = "SmartPhone" to "Nord"
    var parcelDispatcher = ParcelDispatcher()
    parcelDispatcher.dispatchParcel(pachet)
}
```

Modelul iterator

```
class Schita(val name: String)
class Schite(val Schite: MutableList<Schita> = mutableListOf()) : Iterable<Schita>
{
    override fun iterator(): Iterator<Schita> = SchiteIterator(Schite)
}
class SchiteIterator(val Schite: MutableList<Schita> = mutableListOf(), var current: Int = 0) :
Iterator<Schita>
   {
    override fun hasNext(): Boolean = Schite.size > current
    override fun next(): Schita
          {
           val Schita = Schite[current]
           current++
           return Schita
           }
    }
fun main(args: Array<String>) {
    val Schite = Schite(mutableListOf(Schita("Test1"), Schita("Test2")))
    Schite.forEach { println(it.name) }
}
```

Modelul Strategie

```
interface BookingStrategy
{
   val fare: Double
}

class CarBookingStrategy : BookingStrategy
{
   override val fare: Double = 12.5

   override fun toString(): String
     { return "CarBookingStrategy"  }
}

class TrainBookingStrategy : BookingStrategy
{
   override val fare: Double = 8.5

   override fun toString(): String
     { return "TrainBookingStrategy"  }
}
```

```
class Customer(var bookingStrategy: BookingStrategy)
{
    fun calculateFare(numOfPassangeres: Int): Double
   {
        val fare = numOfPassangeres *
bookingStrategy.fare
        println("Calculating fares using " +
bookingStrategy)
        return fare
    }
}

fun main(args: Array<String>)
{
    //strategia de rezolvare a rezervarii pentru o masina
    val cust = Customer(CarBookingStrategy())
    var fare = cust.calculateFare(5)
    println(fare)

     //strategia de rezolvare a rezervarii pentru tren
    cust.bookingStrategy = TrainBookingStrategy()
    fare = cust.calculateFare(5)
    println(fare)
}
```

Modelul Mediator

•

Model Mediator - caz de utilizare

```
interface Command
  {  fun land()  }
class Flight(private val atcMediator: IATCMediator) : Command
{    override fun land()
       { if (atcMediator.isLandingOk)
              {  println("Landing done....")
           atcMediator.setLandingStatus(true)
        } else
           println("Will wait to land....")   }
    fun getReady()
     {  println("Getting ready...")  }   }
class Runway(private val atcMediator: IATCMediator) :
Command
{    init  { atcMediator.setLandingStatus(true)  }
     override fun land()
       {  println("Landing permission granted...")
          atcMediator.setLandingStatus(true)  }   }
interface IATCMediator
  { val isLandingOk: Boolean
     fun registerRunway(runway: Runway)
     fun registerFlight(flight: Flight)
     fun setLandingStatus(status: Boolean)  }
```

```
class ATCMediator : IATCMediator
{   private var flight: Flight? = null
    private var runway: Runway? = null
    override var isLandingOk: Boolean = false
    override fun registerRunway(runway: Runway)
       {   this.runway = runway  }
    override fun registerFlight(flight: Flight)
       {   this.flight = flight  }
    override fun setLandingStatus(status: Boolean)
       {   isLandingOk = status  }
}

fun main(args: Array<String>)
{
   val atcMediator = ATCMediator()
    val sparrow101 = Flight(atcMediator)
    val mainRunway = Runway(atcMediator)
    atcMediator.registerFlight(sparrow101)
    atcMediator.registerRunway(mainRunway)
    sparrow101.getReady()
    mainRunway.land()
    sparrow101.land()
}
```