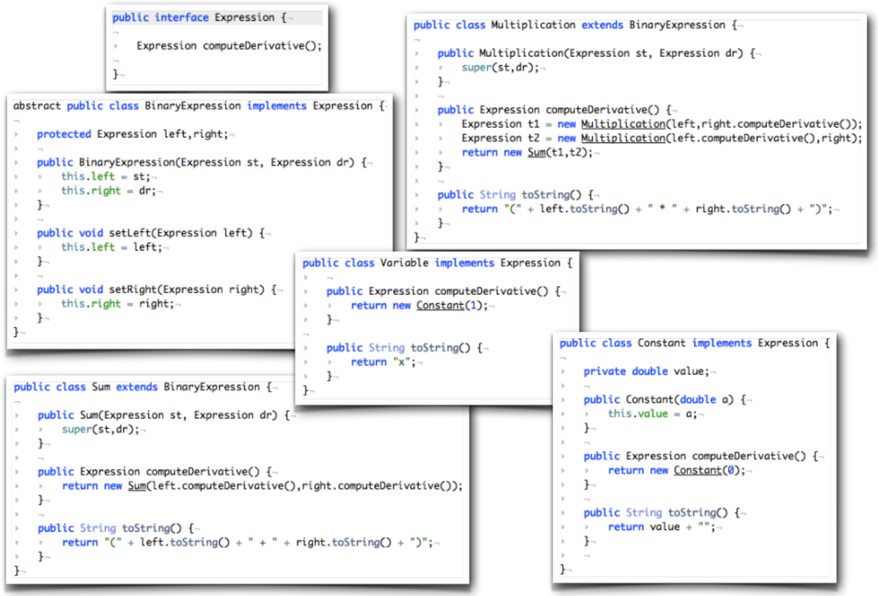
Foundations of Software Engineering

Introduction to Unified Modeling Language

Dr. Petru Florin Mihancea

Tell me about ...



```
public interface Expression {

    Expression computeDerivative();

}
```

```
abstract public class BinaryExpression implements Expression {

    protected Expression left,right;

    public BinaryExpression(Expression st, Expression dr) {
        this.left = st;
        this.right = dr;
    }

    public void setLeft(Expression left) {
        this.left = left;
    }

    public void setRight(Expression right) {
        this.right = right;
    }
}
```

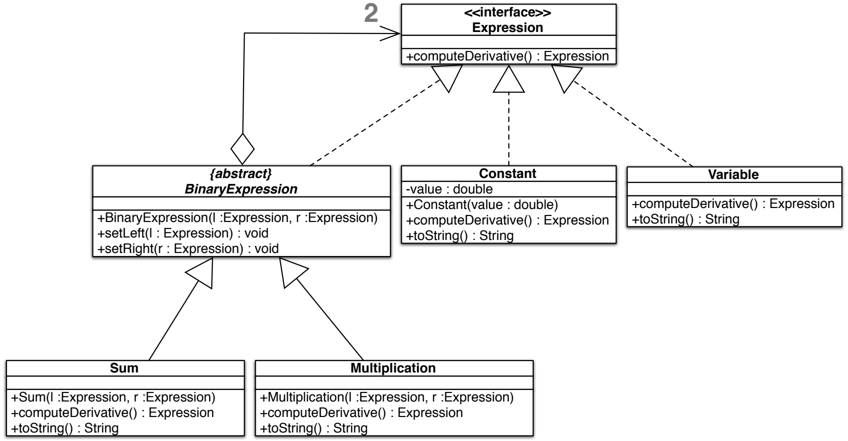
```
public class Multiplication extends BinaryExpression {

    public Multiplication(Expression st, Expression dr) {
        super(st,dr);
    }

    public Expression computeDerivative() {
        Expression t1 = new Multiplication(left,right.computeDerivative());
        Expression t2 = new Multiplication(left.computeDerivative(),right);
        return new Sum(t1,t2);
    }

    public String toString() {
        return "(" + left.toString() + " * " + right.toString() + ")";
    }
}
```

```
public class Variable implements Expression {

    public Expression computeDerivative() {
        return new Constant(1);
    }

    public String toString() {
        return "x";
    }
}
```

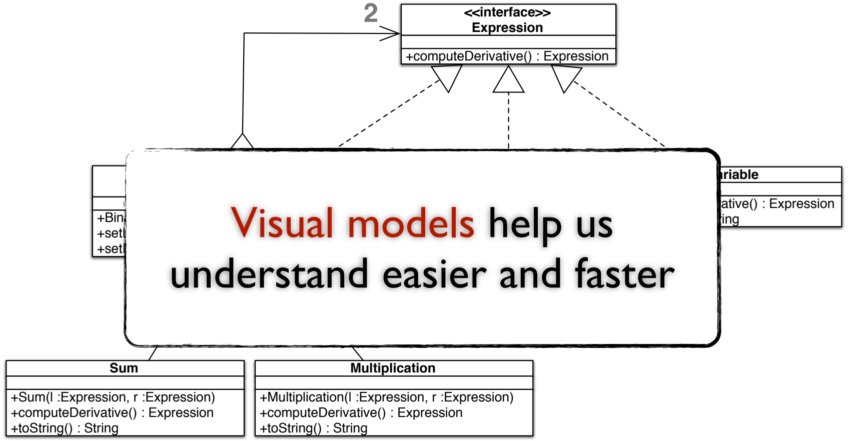
```
public class Sum extends BinaryExpression {

    public Sum(Expression st, Expression dr) {
        super(st,dr);
    }

    public Expression computeDerivative() {
        return new Sum(left.computeDerivative(),right.computeDerivative());
    }

    public String toString() {
        return "(" + left.toString() + " + " + right.toString() + ")";
    }
}
```

```
public class Constant implements Expression {

    private double value;

    public Constant(double a) {
        this.value = a;
    }

    public Expression computeDerivative() {
        return new Constant(0);
    }

    public String toString() {
        return value + "";
    }

}
```

Tell me about ...


2
<<interface>>
Expression
+computeDerivative() : Expression
{abstract}
BinaryExpression
+BinaryExpression(l :Expression, r :Expression)
+setLeft(l : Expression) : void
+setRight(r : Expression) : void
Constant
-value : double
+Constant(value : double)
+computeDerivative() : Expression
+toString() : String
Variable
+computeDerivative() : Expression
+toString() : String
Sum
+Sum(l :Expression, r :Expression)
+computeDerivative() : Expression
+toString() : String
Multiplication
+Multiplication(l :Expression, r :Expression)
+computeDerivative() : Expression
+toString() : String

Tell me about ...


2
<<interface>>
Expression
+computeDerivative() : Expression
+Bin
+set
+set
riable
ative() : Expression
ing
Sum
+Sum(l :Expression, r :Expression)
+computeDerivative() : Expression
+toString() : String
Multiplication
+Multiplication(l :Expression, r :Expression)
+computeDerivative() : Expression
+toString() : String

Visual models help us understand easier and faster

Unified Modeling Language

Family of **graphical** notations

for **modeling** an (OO) system

Goal

Learn the notations

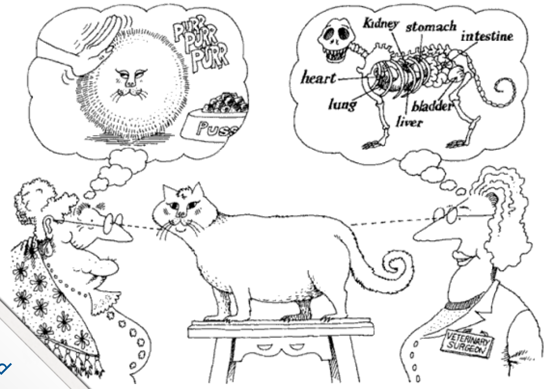
Types of UML models

Behavioral

e.g. Sequence diagram (SD)

Structural

e.g. Class diagram (CD)

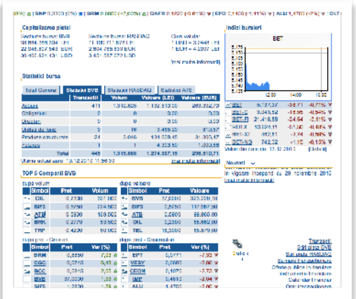


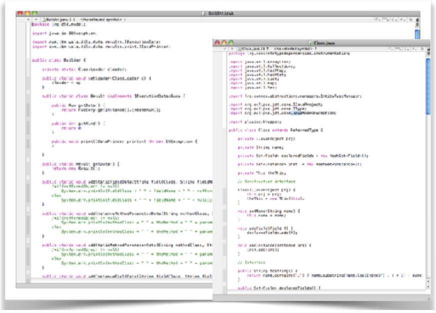
Booch - OO Analysis and Design

Goal Refined

Learn the notations for **CD and SD**

UML usage perspectives





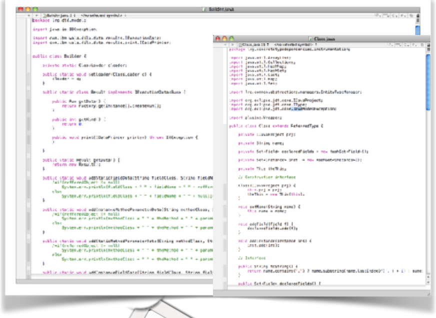
Conceptual

i.e. model a domain

Software

i.e. model a program

UML usage perspectives



Software

i.e. model a program

Goal Refined

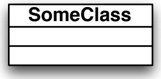
Learn notations for **CD and SD**

and their usual meaning in Java source code

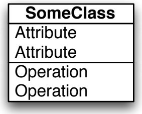
1

Class diagram

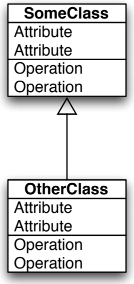
Structural model



Classes



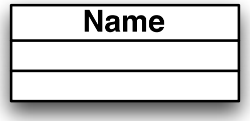
Features



Relations

1

Class diagram



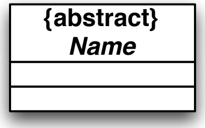
UML sketch

```
class Name {

}
```

Java sketch

1

Class diagram



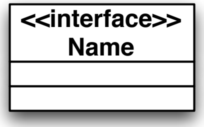
UML sketch

```
abstract class Name {

}
```

Java sketch

1

Class diagram



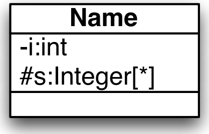
UML sketch

```
interface Name {

}
```

Java sketch

Class diagram

Attributes

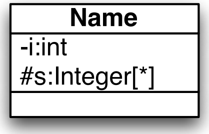


visibility name : type multiplicity
= implicitValue

+ public
- private
# protected
~ package

Class diagram

Attributes



visibility name : type multiplicity
= implicitValue

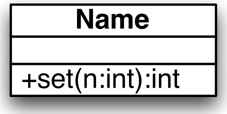
1 - exactly one
0..1 - zero or at most one
0..* or * - zero or more but
NO upper limit

```
class Name {
   private int i;
   protected List<Integer> s;
   //s must be somehow initialized / created
}
```

```
class Name {
   private int i;
   protected Integer[] s;
   //s must be somehow initialized / created
   //an index may be required + you must
   //guarantee NO upper limit if necessary
   //(e.g. re-create & copy the array)
}
```

Class diagram

Operations



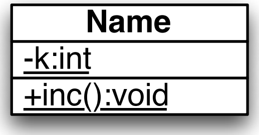
```
class Name {
   public int set(int n) {
      ...
   }
}
```

visibility name(param_list) : ret_type

direction name : type = default

Class diagram

Scope

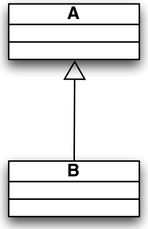


```
class Name {
  private static int k;
  public static void inc() {
     ...
  }
}
```

1

Class diagram

Generalization & Realization



```
class A {
}

class B extends A {
}
```

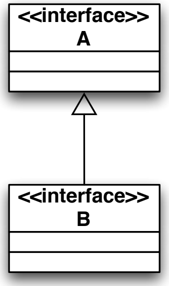
UML sketch

Java sketch

1

Class diagram

Generalization & Realization



```
interface A {
}

interface B extends A {
}
```

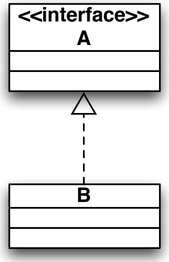
UML sketch

Java sketch

1

Class diagram

Generalization & Realization



```
interface A {
}

class B implements A {
}
```

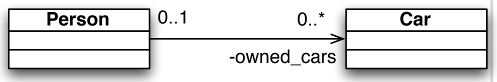
UML sketch

Java sketch

1

Class diagram

Association



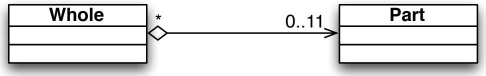
```
class Person {

    //list must be somehow initialized / created
    private List<Car> owned_car;

    //add, remove methods usually exist

}
```

UML sketch

Java sketch

1

Class diagram
Aggregation



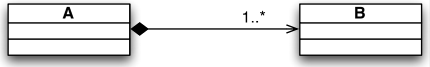
Similar to association

UML sketch

Java sketch

1

Class diagram
Composition



I. No-sharing
II. B objects cannot exists without their A object

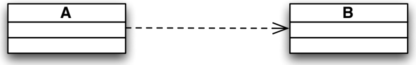
```
class  A {

   private List<B> my_list =
      new ...

   public A(...) {
     my_list.add(new B(...));
   }
   public void add(...) {
     my_list.add(new B(...));
   }
```

UML sketch

Java sketch

1

Class diagram
Dependency



```
class A {

   public void m(B x) {
     x.doS();
   }

}
```

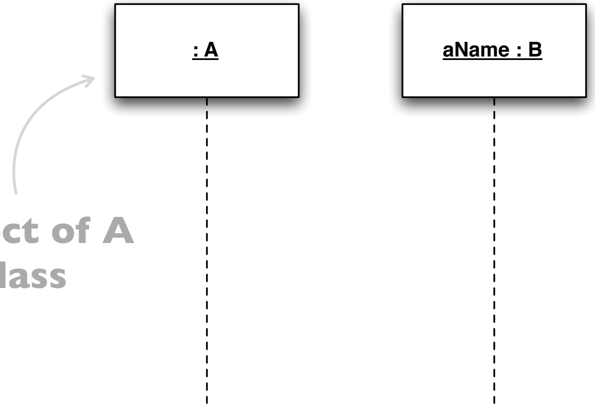
And many other cases ...

UML sketch

Java sketch

2

Sequence diagram
Behavioral & Interaction model

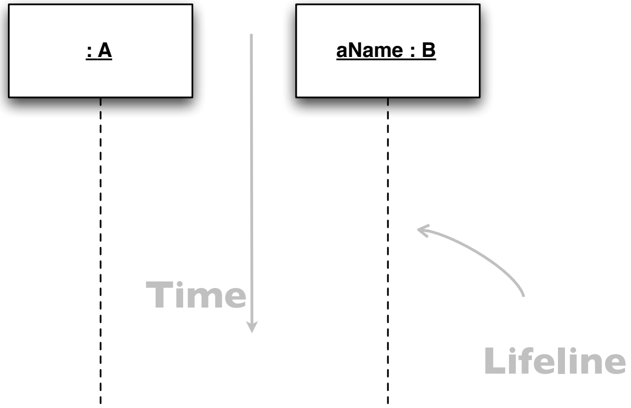


Object of A class

2

Sequence diagram

Behavioral & Interaction model



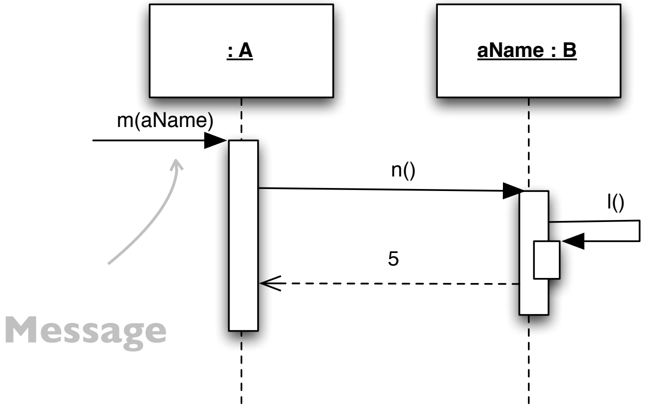
Time

Lifeline

2

Sequence diagram

Behavioral & Interaction model

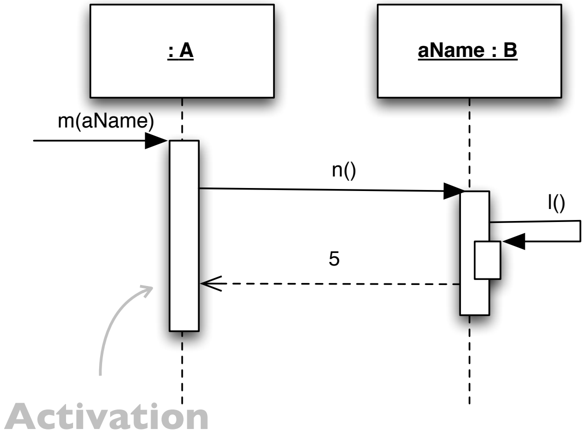


Message

2

Sequence diagram

Behavioral & Interaction model

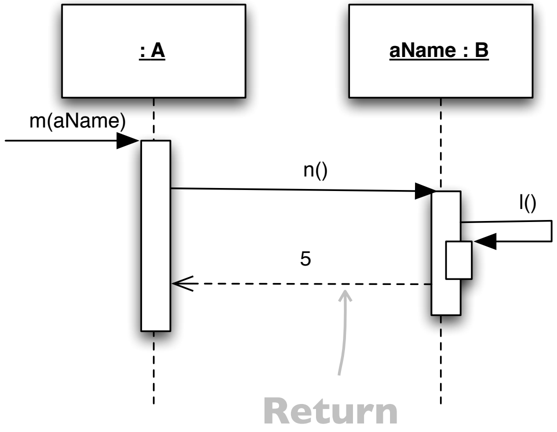


Activation

2

Sequence diagram

Behavioral & Interaction model

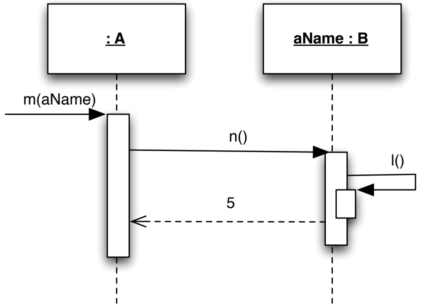


Return

2

Sequence diagram

Behavioral & Interaction model



```
class A {
   public void m(B x) {
      x.n();
   }
}
class B {
   public int n() {
      this.l();
      return 5;
   }
   public void l() {}
```

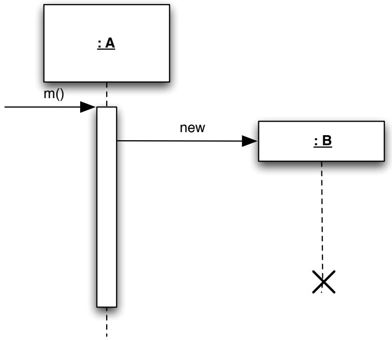
UML sketch

Java sketch

2

Sequence diagram

Object creation & deletion



```
class A {
   public void m() {
      ...
      new B();
      ...
      //the object is
      //no more accessible
   }
}
```

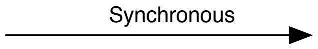
UML sketch

Java sketch

2

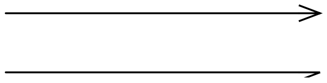
Sequence diagram

Other notations



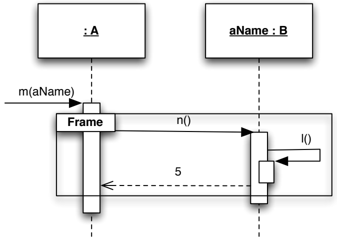
Synchronous

e.g. method invocation

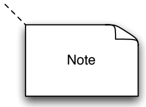


Async Message

e.g. start an execution thread



e.g. loops, conditions



Comments

2

Sequence diagram

Let us see an example ...