

Foundations of Software Engineering

Introduction to **U**nified **M**odeling **L**anguage

Dr. Petru Florin Mihancea

Tell me about ...

Tell me about ...

```
public interface Expression {  
    ~  
    > Expression computeDerivative();  
    ~  
}
```

```
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() + ")";  
    > }  
}
```

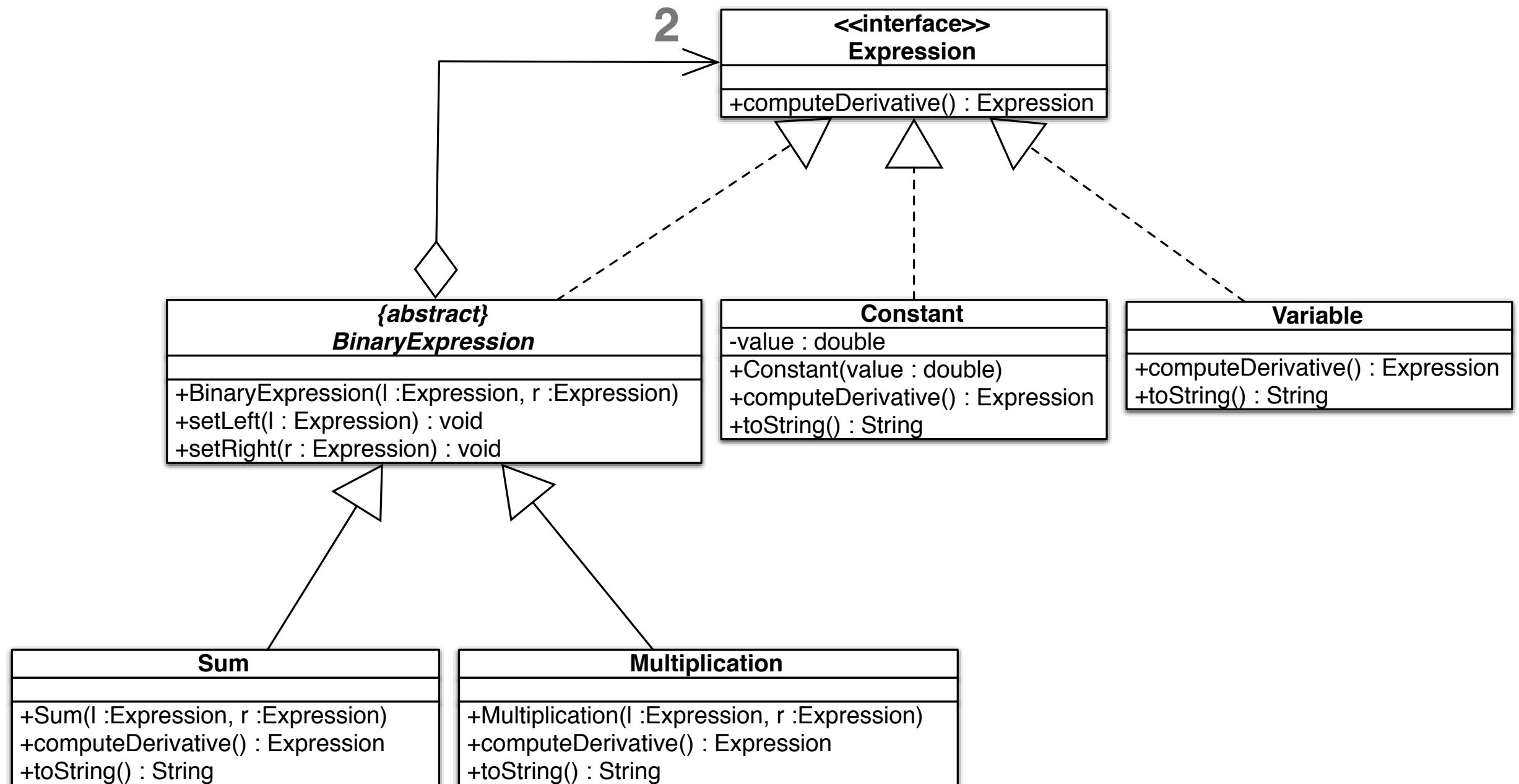
```
abstract public class BinaryExpression implements Expression {  
    ~  
    > protected Expression left,right;  
    ~  
    > public BinaryExpression(Expression st, Expression dr) {  
    >     > this.left = st;  
    >     > this.right = dr;  
    > }  
    ~  
    > public void setLeft(Expression left) {  
    >     > this.left = left;  
    > }  
    ~  
    > public void setRight(Expression right) {  
    >     > this.right = right;  
    > }  
}
```

```
public class Variable implements Expression {  
    ~  
    > public Expression computeDerivative() {  
    >     > return new Constant(1);  
    > }  
    ~  
    > public String toString() {  
    >     > return "x";  
    > }  
}
```

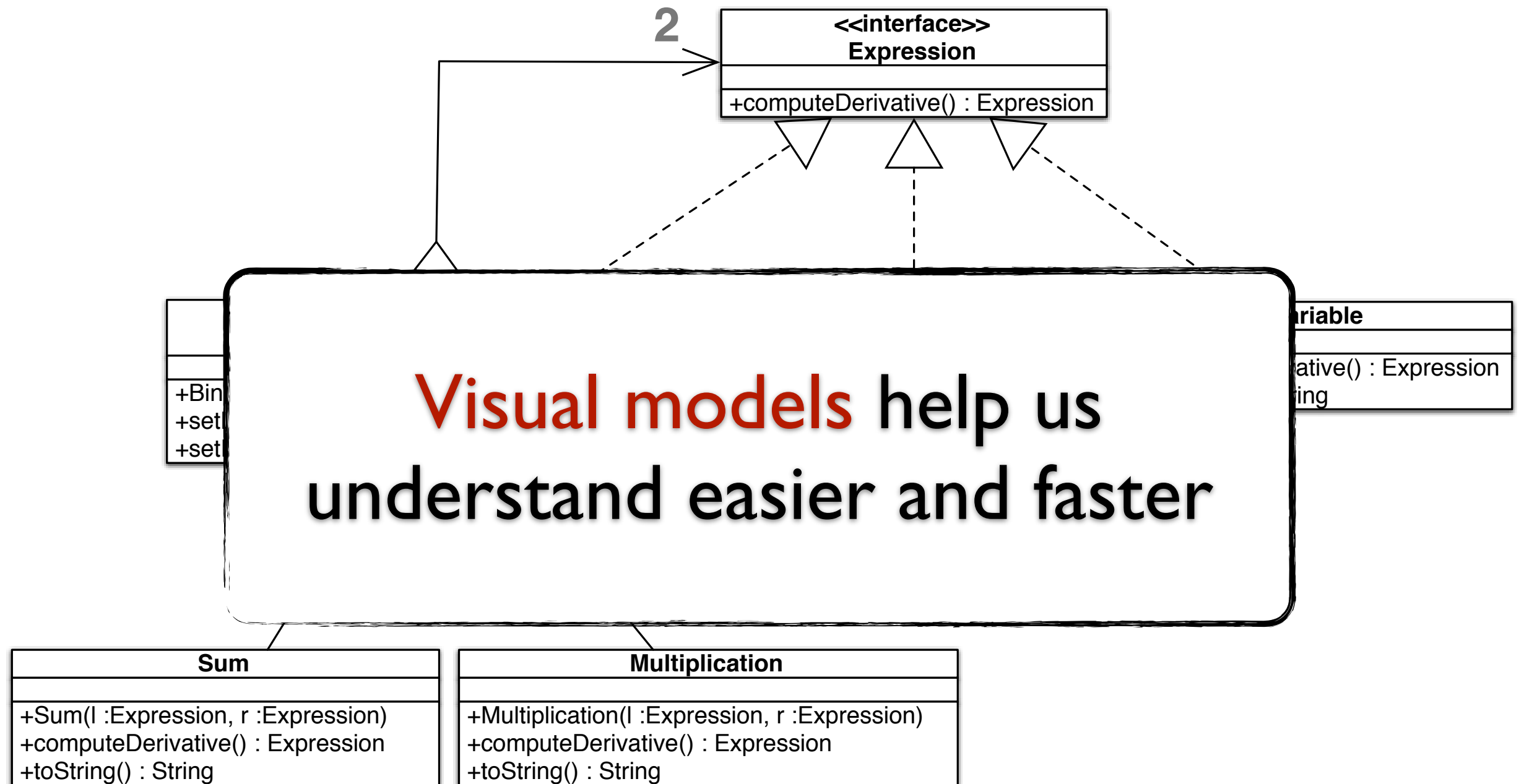
```
public class Sum extends BinaryExpression {  
    ~  
    > public Sum(Expression st, Expression dr) {  
    >     > super(st,dr);  
    > }  
    ~  
    > public Expression computeDerivative() {  
    >     > return new Sum(left.computeDerivative(),right.computeDerivative());  
    > }  
    ~  
    > 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 ...



Tell me about ...



Unified Modeling Language

Unified Modeling Language

Family of **graphical** notations

for **modeling** an (OO) system

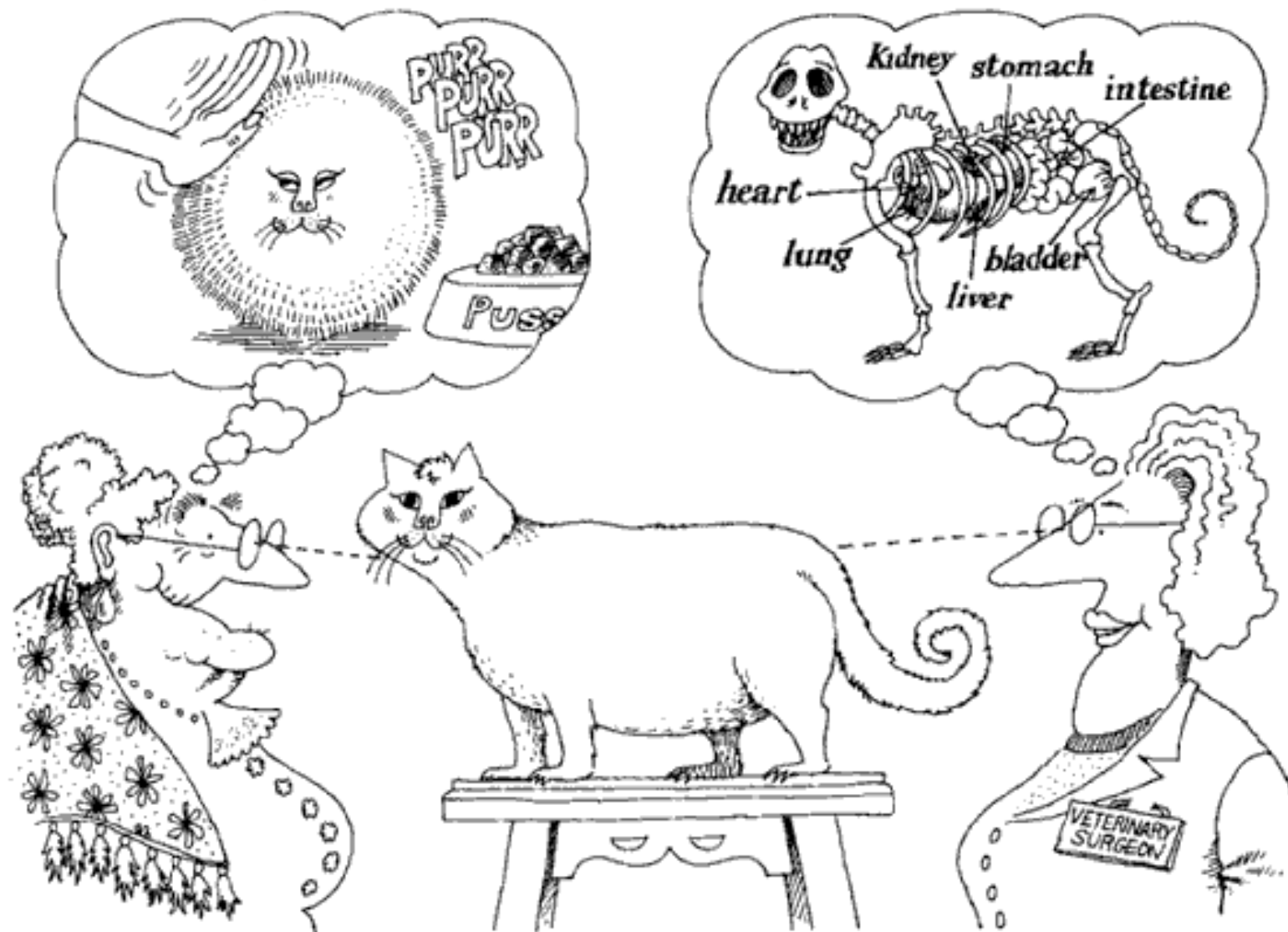
Unified Modeling Language

Family of **graphical** notations

for **modeling** an (OO) system



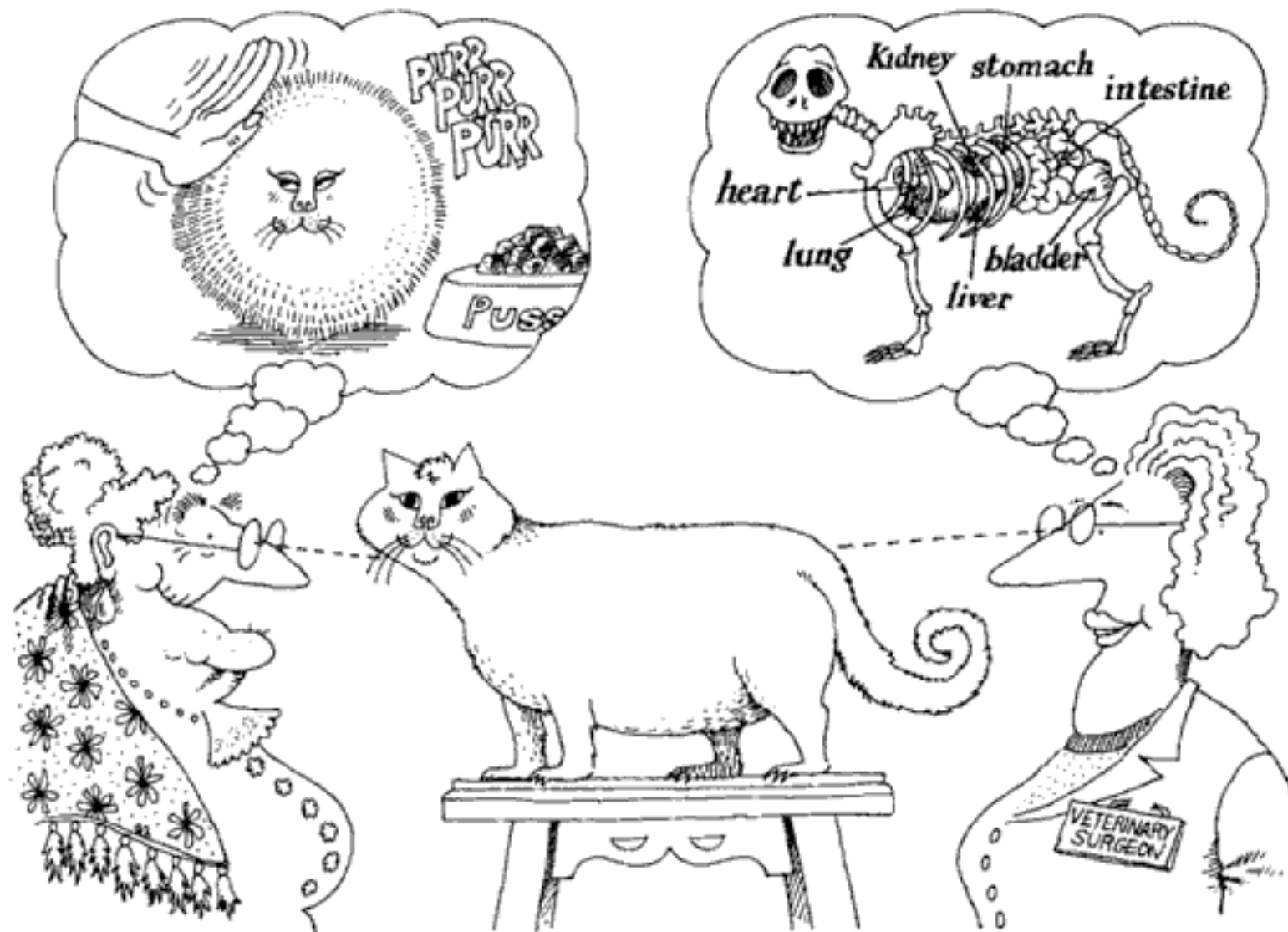
Types of UML models



Types of UML models

Structural

e.g. Class diagram (CD)



Types of UML models

Behavioral

e.g. Sequence diagram (SD)



Structural

e.g. Class diagram (CD)



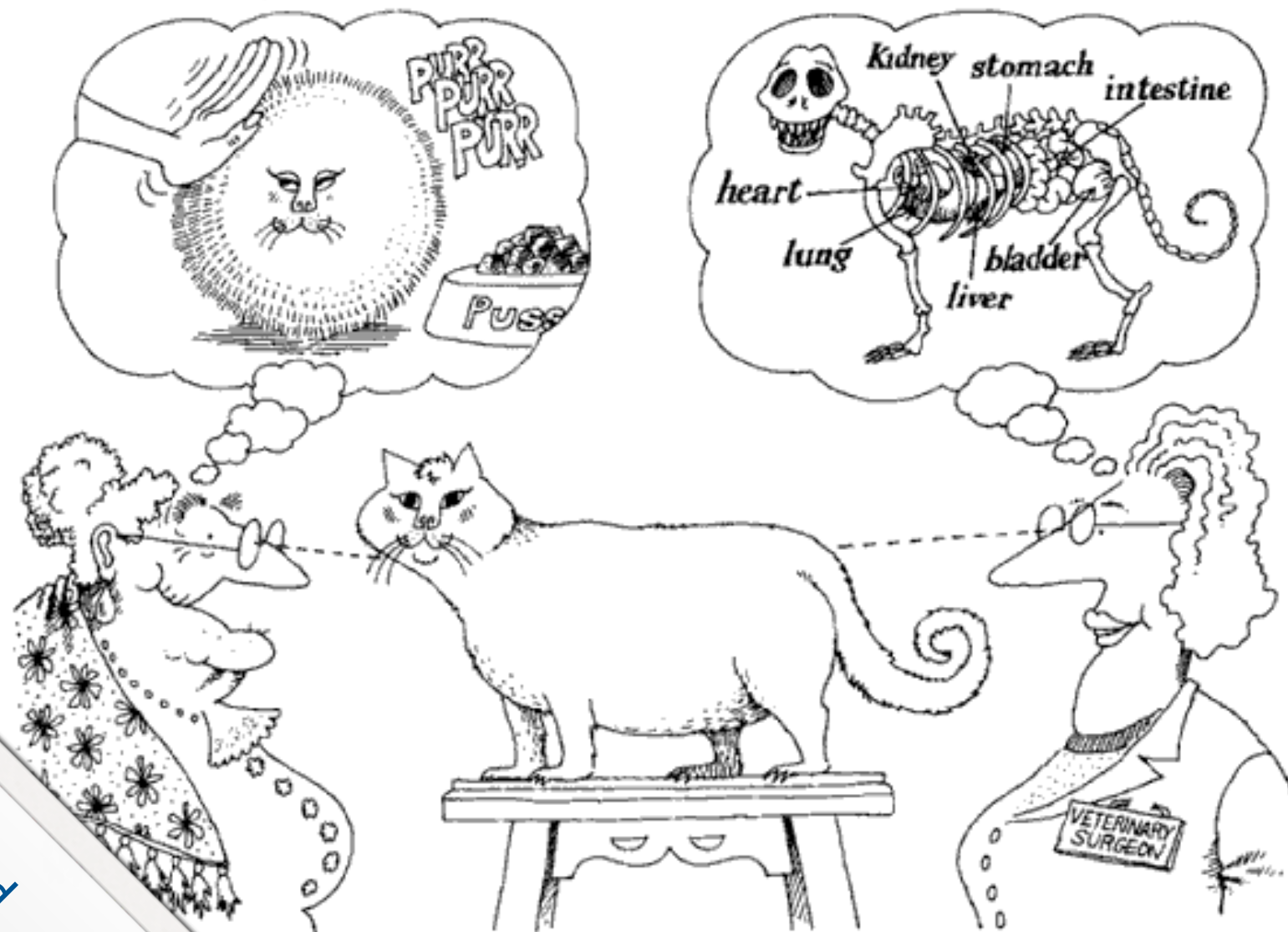
Types of UML models

Behavioral

e.g. Sequence diagram (SD)

Structural

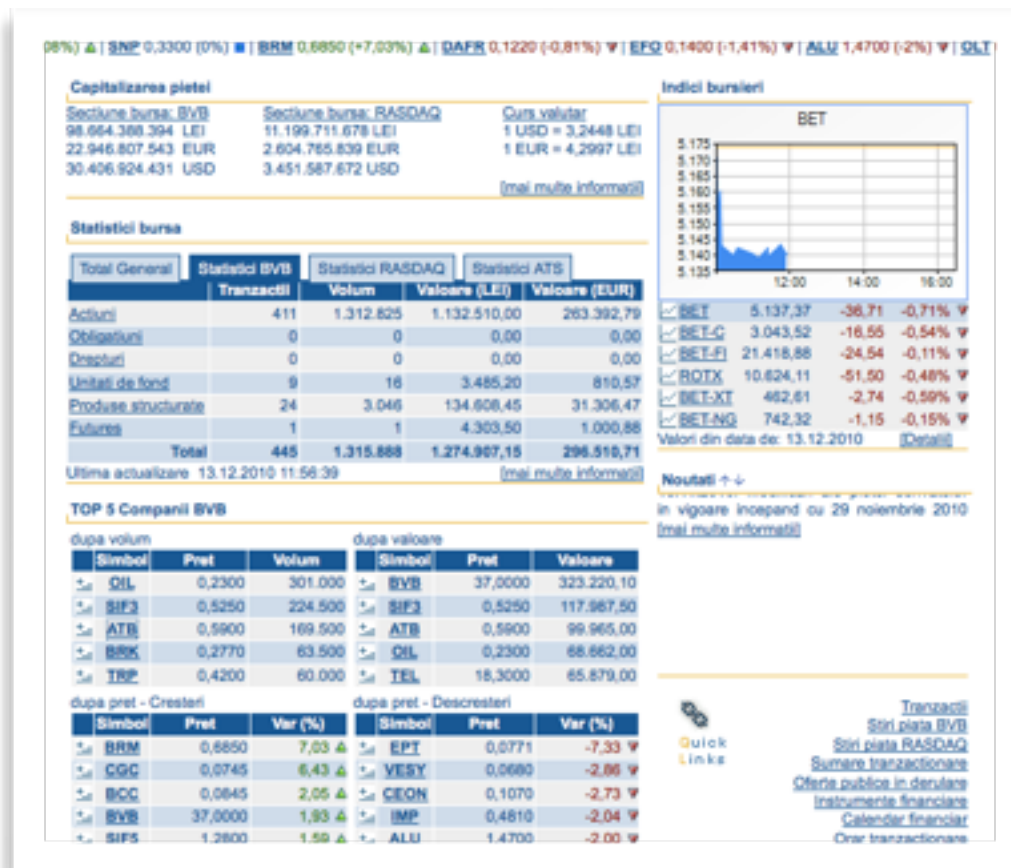
e.g. Class diagram (CD)



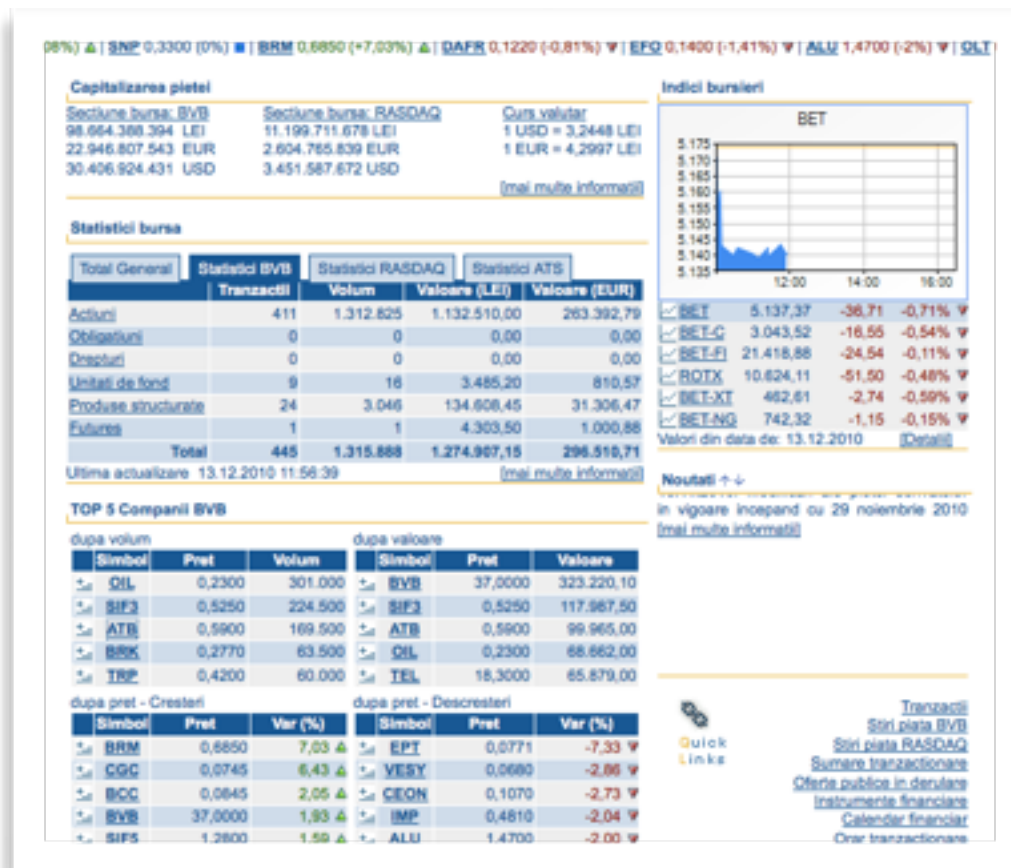
Goal Refined
Learn the notations for
CD and **SD**

UML usage **perspectives**

UML usage perspectives



UML usage perspectives



Conceptual
i.e. model a domain

UML usage perspectives



Conceptual
i.e. model a domain

```
Builder.java
package org.dtd.model;

import java.io.IOException;
import com.ibm.wala.dla.data.results.IExecutionData;
import com.ibm.wala.dla.data.results.IDataPrinter;

public class Builder {
    private static ClassLoader classLoader;

    public static void setLoader(ClassLoader cl) {
        classLoader = cl;
    }

    public static class Result implements IExecutionData {
        public Run getData() {
            return Factory.getInstance().createRun();
        }

        public int getKind() {
            return 0;
        }

        public void print(IDataPrinter printer) throws IOException {
        }

        public static Result getData() {
            return new Result();
        }

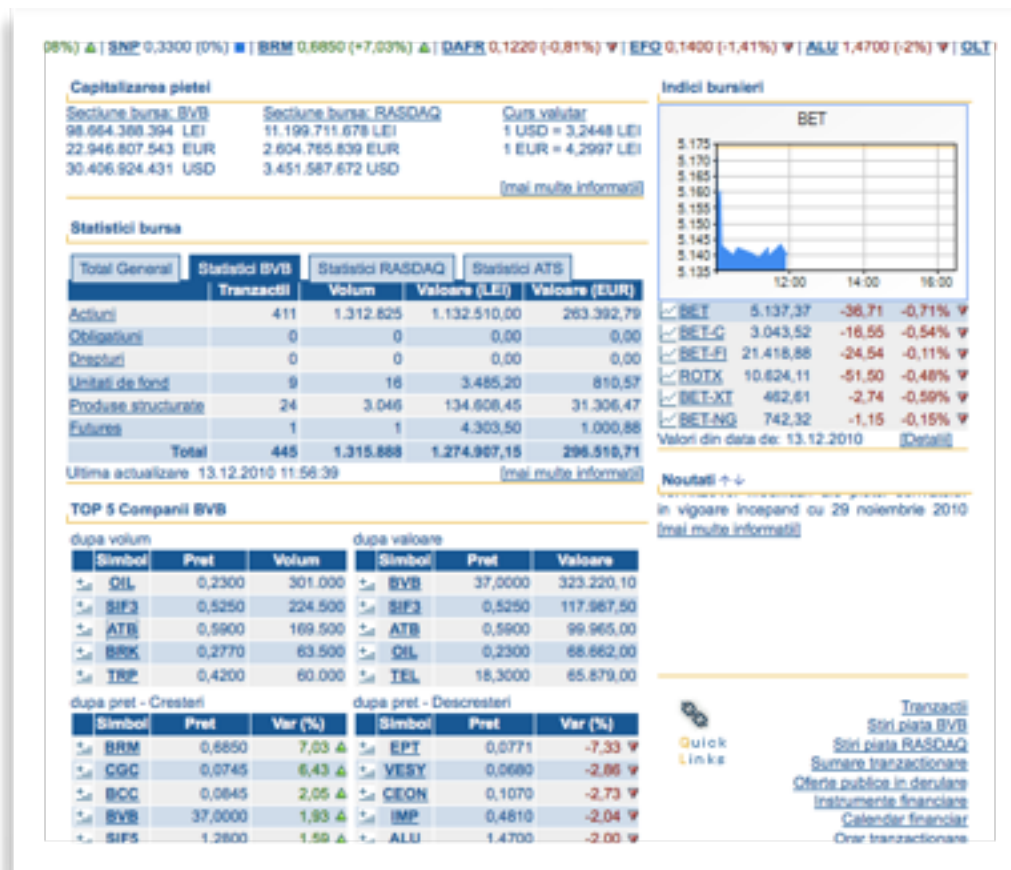
        public static void addStaticFieldData(String fieldClass, String fieldName) {
            if (referencedObject != null) {
                System.err.println(fieldClass + " " + fieldName + " " + referen
            } else {
                System.err.println(fieldClass + " " + fieldName + " " + null);
            }
        }

        public static void addInstanceMethodParameterData(String methodClass, String method, String param) {
            if (referencedObject != null) {
                System.err.println(methodClass + " " + method + " " + param
            } else {
                System.err.println(methodClass + " " + method + " " + param
            }
        }

        public static void addStaticMethodParameterData(String methodClass, String method, String param) {
            if (referencedObject != null) {
                System.err.println(methodClass + " " + method + " " + param
            } else {
                System.err.println(methodClass + " " + method + " " + param
            }
        }

        public static void addInstanceFieldData(String fieldClass, String field)
    }
}
```


UML usage perspectives



Conceptual
i.e. model a domain

The screenshot shows two Java code files. The left file is 'Builder.java' and the right file is 'Class.java'. Both files contain Java code for a domain model.

```
package org.dtd.model;

import java.io.IOException;
import com.ibm.wala.dla.data.results.IExecutionData;
import com.ibm.wala.dla.data.results.IDataPrinter;

public class Builder {
    private static ClassLoader classLoader;

    public static void setLoader(ClassLoader cl) {
        classLoader = cl;
    }

    public static class Result implements IExecutionData {
        public Run getData() {
            return Factory.getInstance().createRun();
        }

        public int getKind() {
            return 0;
        }

        public void print(IDataPrinter printer) throws IOException {
        }

        public static Result getData() {
            return new Result();
        }

        public static void addStaticFieldData(String fieldClass, String fieldName) {
            if (fieldClass != null) {
                System.err.println(fieldClass + " * " + fieldName + " * " + null;
            } else {
                System.err.println(fieldClass + " * " + fieldName + " * " + null;
            }
        }

        public static void addInstanceMethodParameterData(String methodClass, String methodName, String param) {
            if (methodClass != null) {
                System.err.println(methodClass + " * " + methodName + " * " + param;
            } else {
                System.err.println(methodClass + " * " + methodName + " * " + param;
            }
        }

        public static void addStaticMethodParameterData(String methodClass, String methodName, String param) {
            if (methodClass != null) {
                System.err.println(methodClass + " * " + methodName + " * " + param;
            } else {
                System.err.println(methodClass + " * " + methodName + " * " + param;
            }
        }

        public static void addInstanceFieldData(String fieldClass, String fieldName) {
        }
    }
}
```

```
package org.concretejpedependency.instrumentation;

import java.util.ArrayList;
import java.util.Collections;
import java.util.HashMap;
import java.util.HashSet;
import java.util.List;
import java.util.Map;
import java.util.Set;

import org.common.abstractions.managers.EntityTypeManager;
import org.eclipse.jdt.core.IJavaProject;
import org.eclipse.jdt.core.IType;
import org.eclipse.jdt.core.JavaModelException;

import plugins.Wrappers;

public class Class extends ReferencedType {
    private IJavaProject prj;
    private String name;
    private Set<Field> declaredFields = new HashSet<Field>();
    private Set<Instance> inst = new HashSet<Instance>();
    private This theThis;

    // Construction interface
    Class(IJavaProject prj) {
        this.prj = prj;
        this.theThis = new This(this);
    }

    void setName(String name) {
        this.name = name;
    }

    void addField(Field f) {
        declaredFields.add(f);
    }

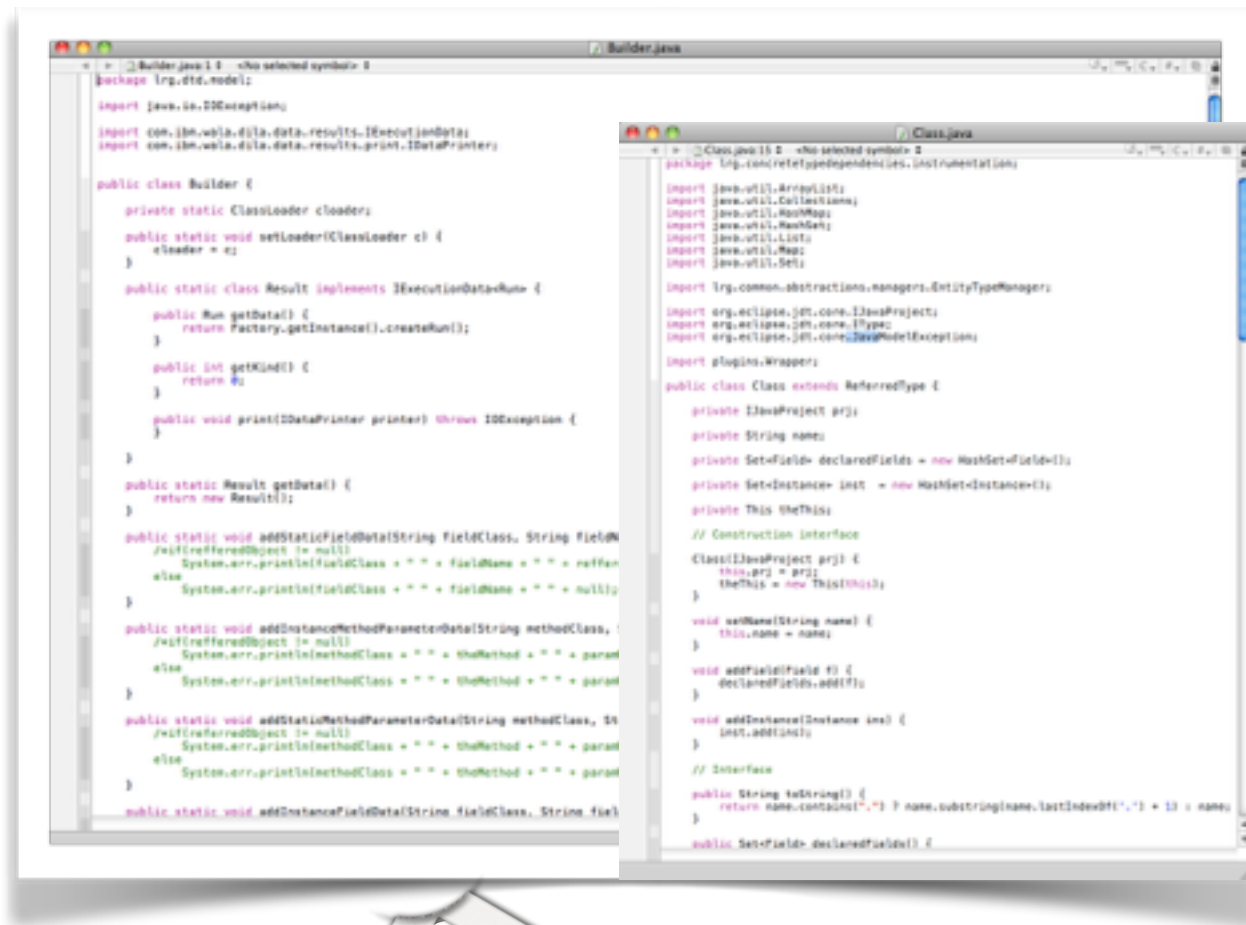
    void addInstance(Instance inst) {
        inst.add(this);
    }

    // Interface
    public String toString() {
        return name.contains(".") ? name.substring(name.lastIndexOf("."), name.length()) : name;
    }

    public Set<Field> declaredFields() {
    }
}
```

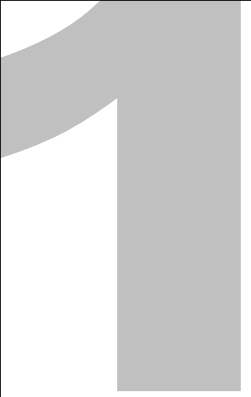
Software
i.e. model a program

UML usage perspectives



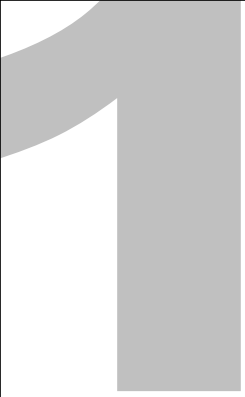
Software
i.e. model a program

Goal Refined
Learn **CD** and **SD**
notations for
and their usual meaning in
Java source code



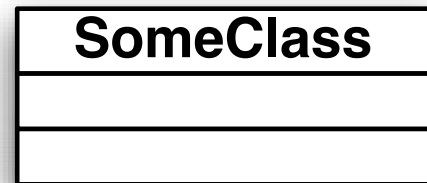
Class diagram

Structural model

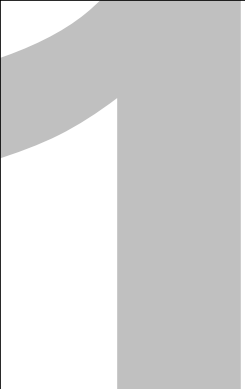


Class diagram

Structural model



Classes



Class diagram

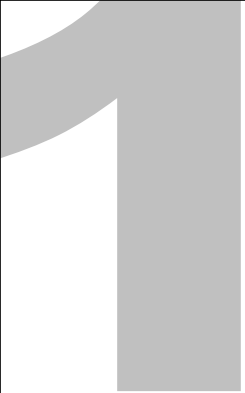
Structural model

SomeClass

Classes

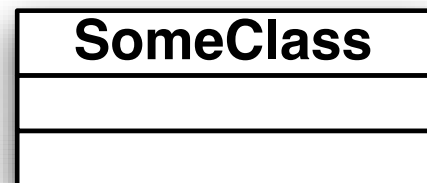
SomeClass
Attribute
Attribute
Operation
Operation

Features

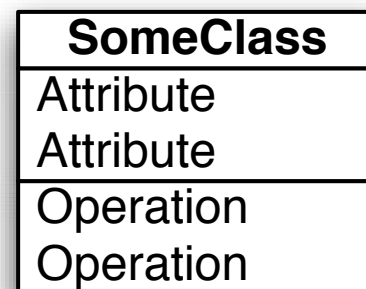


Class diagram

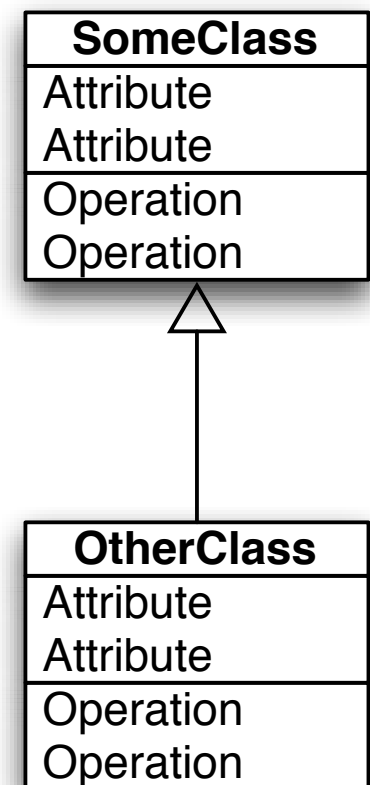
Structural model



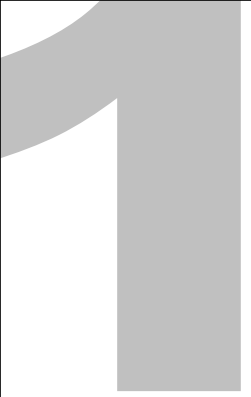
Classes



Features



Relations

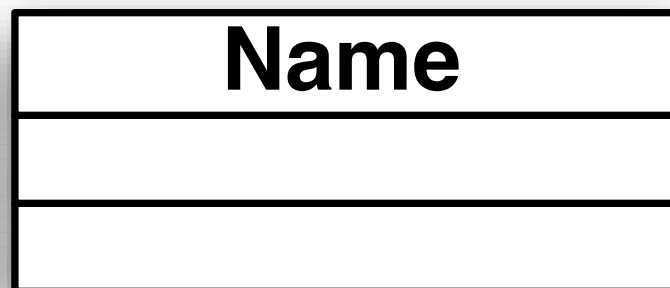


Class diagram

UML sketch

Java sketch

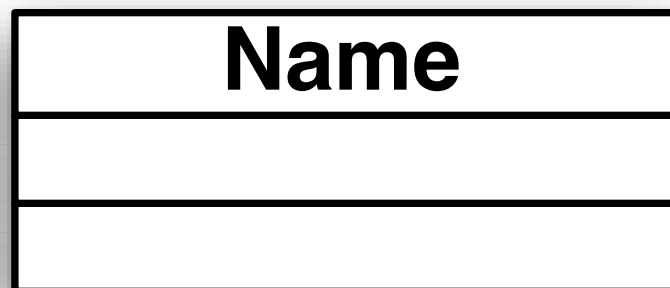
Class diagram



UML sketch

Java sketch

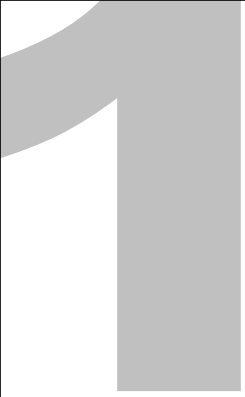
Class diagram



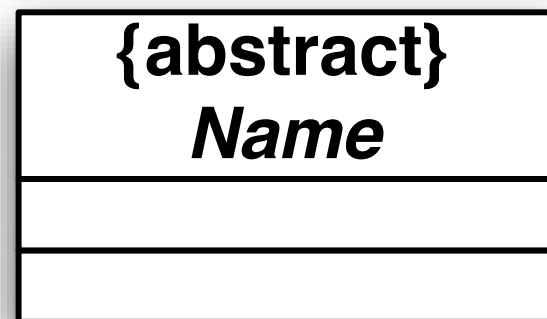
UML sketch

```
class Name {  
  
}
```

Java sketch



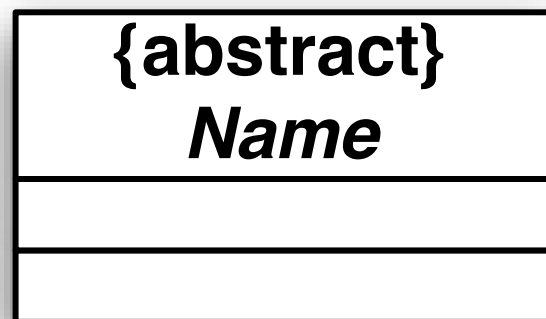
Class diagram



UML sketch

Java sketch

Class diagram

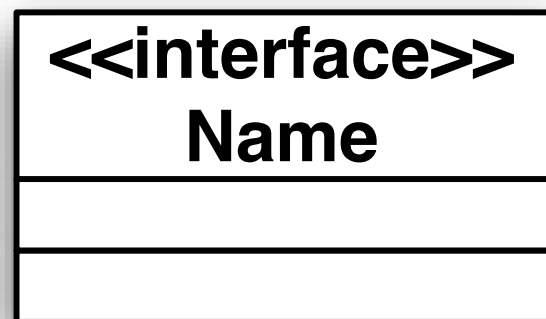


UML sketch

```
abstract class Name {  
  
}
```

Java sketch

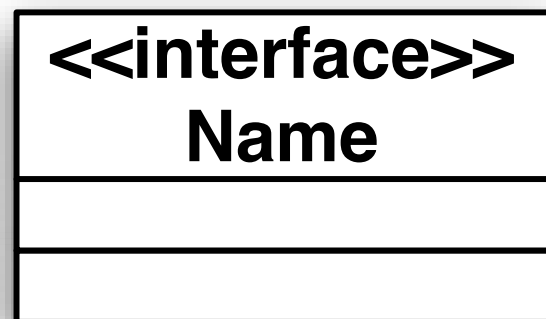
Class diagram



UML sketch

Java sketch

Class diagram



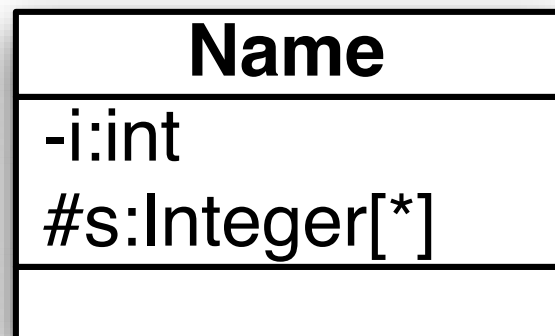
UML sketch

```
interface Name {  
  
}
```

Java sketch

Class diagram

Attributes



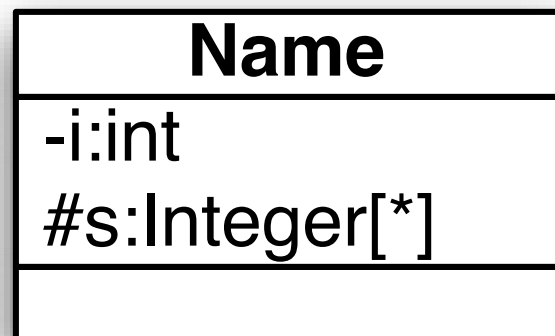
visibility name : type multiplicity
= implicitValue

UML sketch

Java sketch

Class diagram

Attributes



visibility name : type multiplicity
= implicitValue

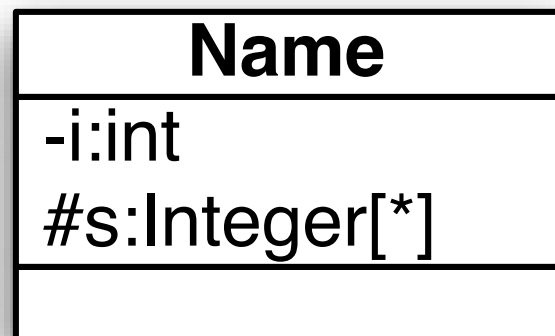
+ public
- private
protected
~ package

UML sketch

Java sketch

Class diagram

Attributes



visibility name : type multiplicity
= implicitValue

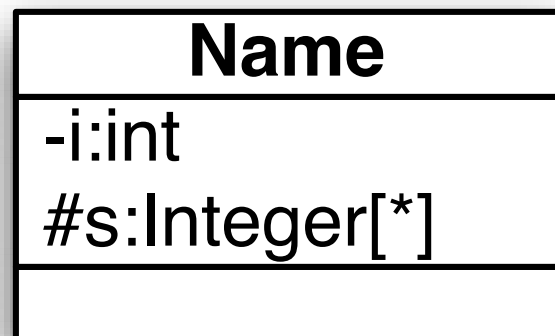
1 - exactly one
0..1 - zero or at most one
0..* or * - zero or more but
NO upper limit

UML sketch

Java sketch

Class diagram

Attributes



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

visibility name : type multiplicity
= implicitValue

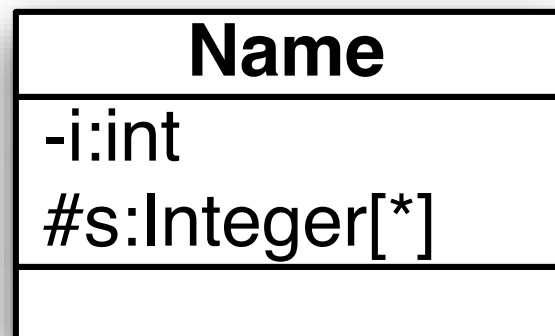
1 - exactly one
0..1 - zero or at most one
0..* or * - zero or more but
NO upper limit

UML sketch

Java sketch

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

UML sketch

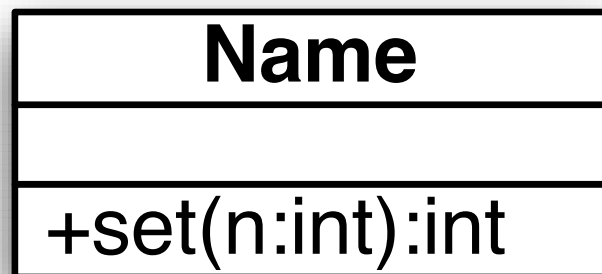
```
class Name {  
    private int i;  
    protected List<Integer> s;  
    //s must be somehow initialized / created  
}
```

```
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)  
}
```

Java sketch

Class diagram

Operations



visibility name(param_list) : ret_type

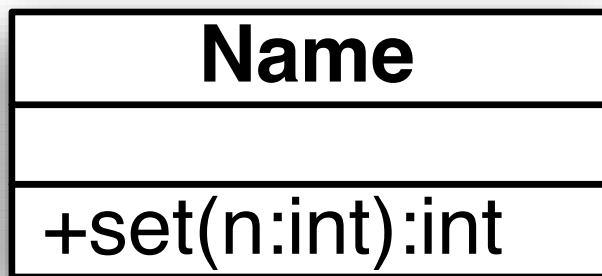
direction name : type = default

UML sketch

Java sketch

Class diagram

Operations



visibility name(param_list) : ret_type

direction name : type = default

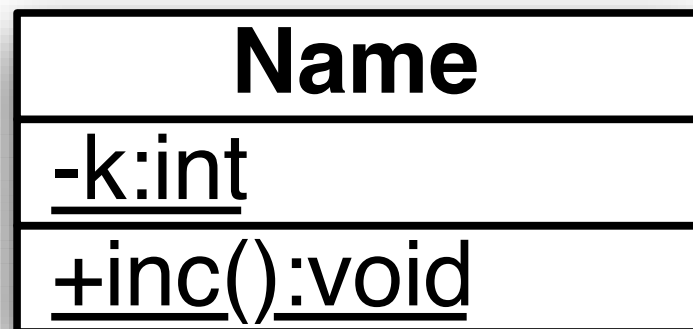
UML sketch

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

Java sketch

Class diagram

Scope

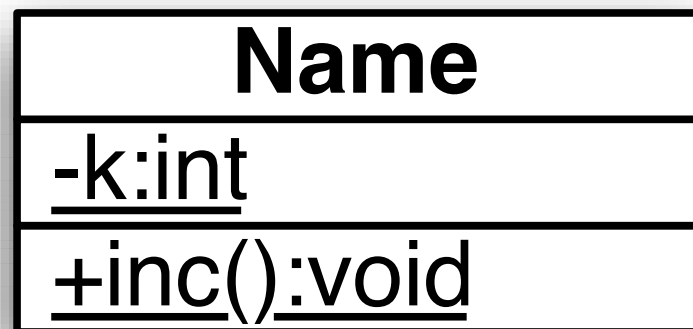


UML sketch

Java sketch

Class diagram

Scope



UML sketch

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

Java sketch

Class diagram

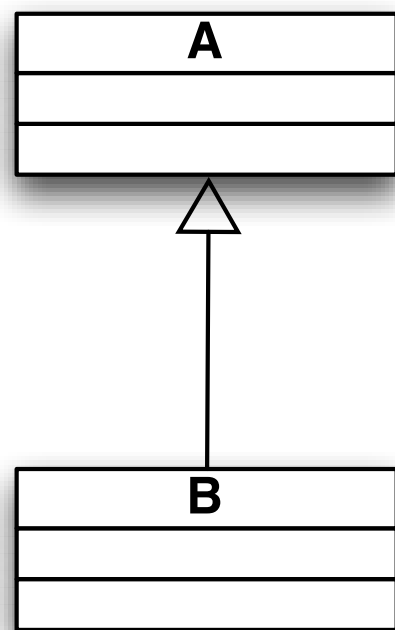
Generalization & Realization

UML sketch

Java sketch

Class diagram

Generalization & Realization

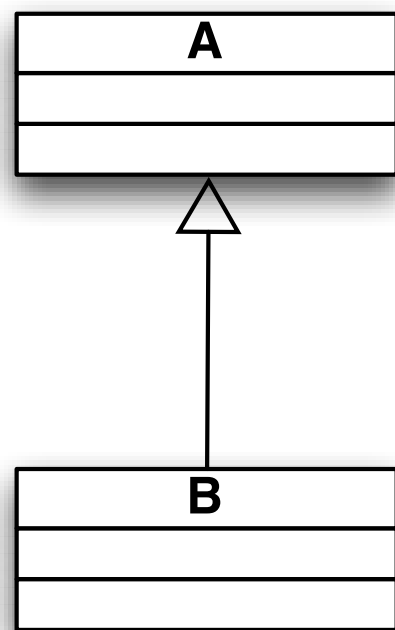


UML sketch

Java sketch

Class diagram

Generalization & Realization



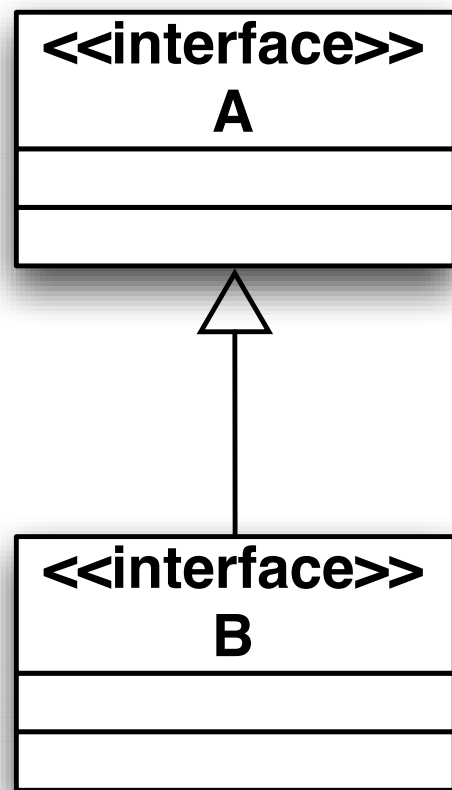
UML sketch

```
class A {  
}  
  
class B extends A {  
}
```

Java sketch

Class diagram

Generalization & Realization

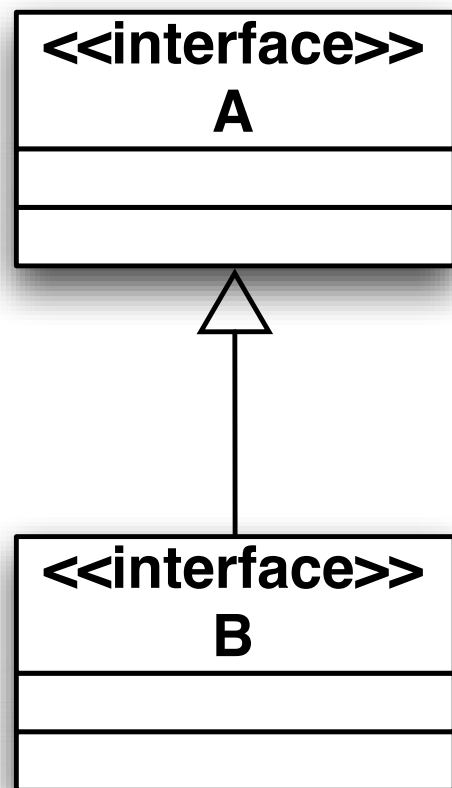


UML sketch

Java sketch

Class diagram

Generalization & Realization



UML sketch

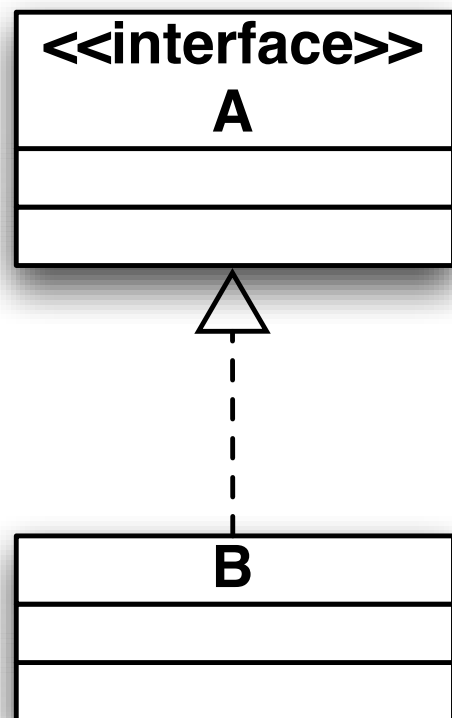
```
interface A {  
}
```

```
interface B extends A {  
}
```

Java sketch

Class diagram

Generalization & Realization

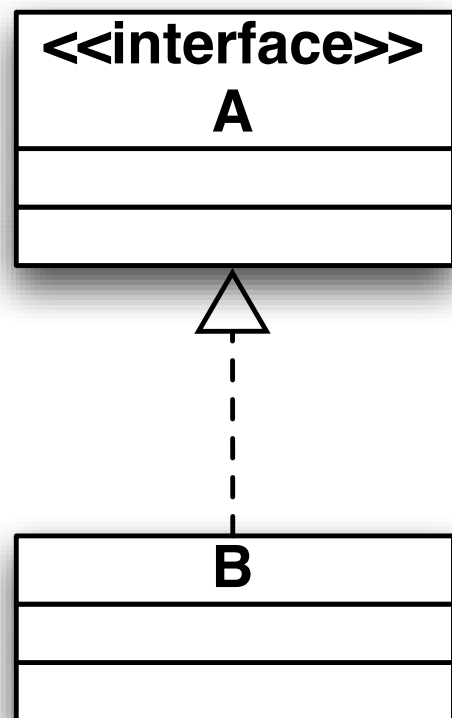


UML sketch

Java sketch

Class diagram

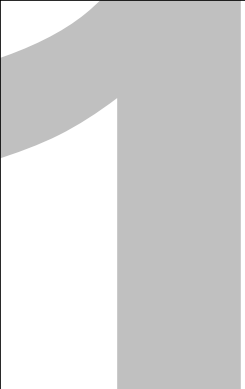
Generalization & Realization



UML sketch

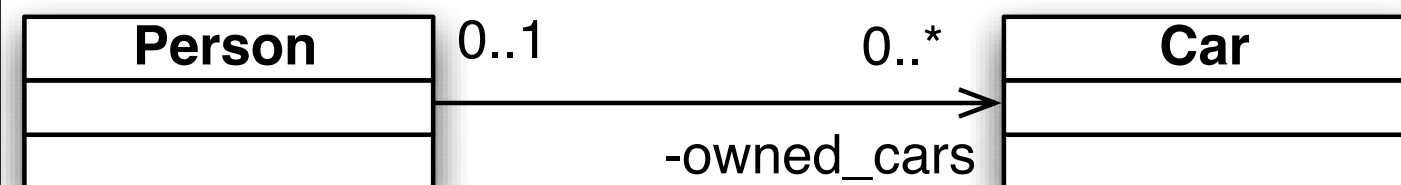
```
interface A {  
}  
  
class B implements A {  
}
```

Java sketch



Class diagram

Association

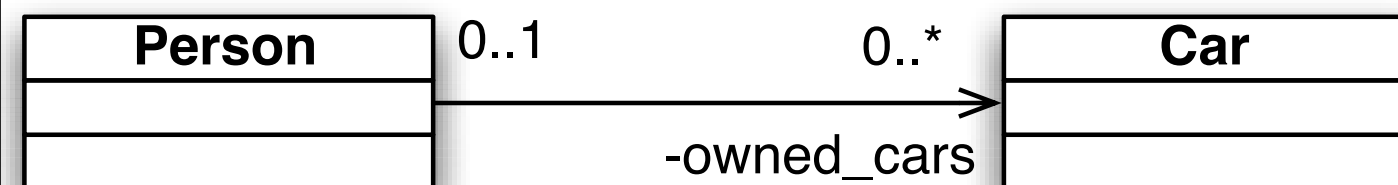


UML sketch

Java sketch

Class diagram

Association



UML sketch

```
class Person {
```

```
//list must be somehow initialized / created
```

```
private List<Car> owned_car;
```

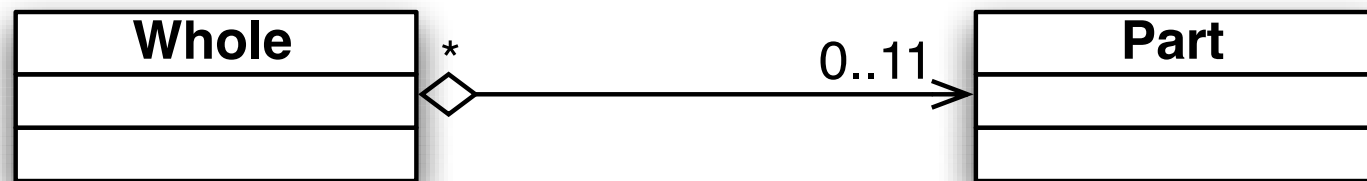
```
//add, remove methods usually exist
```

```
}
```

Java sketch

Class diagram

Aggregation

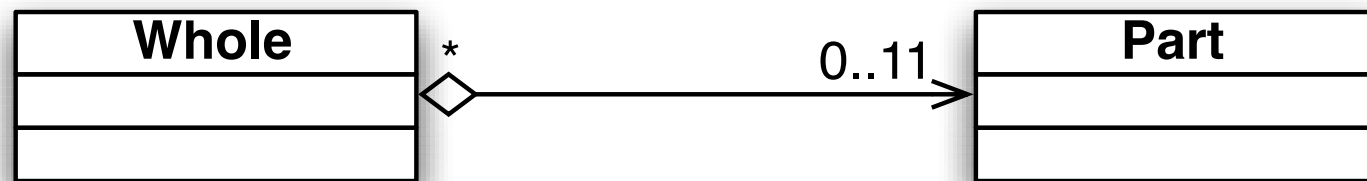


UML sketch

Java sketch

Class diagram

Aggregation



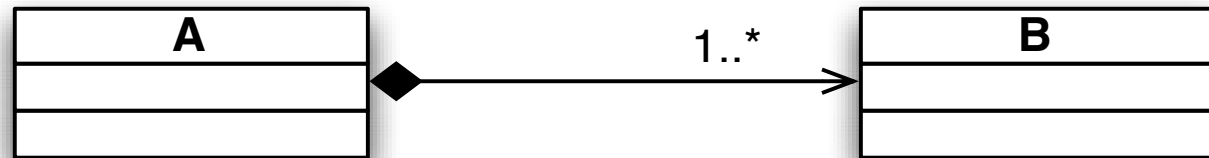
**Similar to
association**

UML sketch

Java sketch

Class diagram

Composition



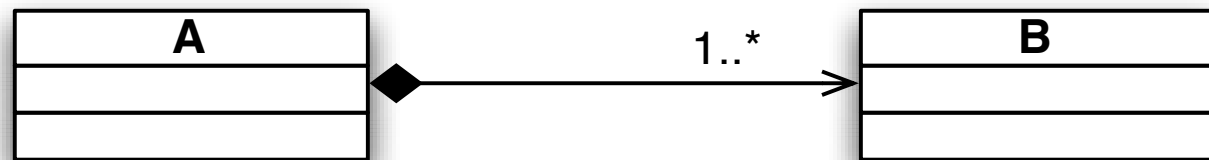
- I. No-sharing
- II. B objects cannot exist without their A object

UML sketch

Java sketch

Class diagram

Composition



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

UML sketch

```
class A {
```

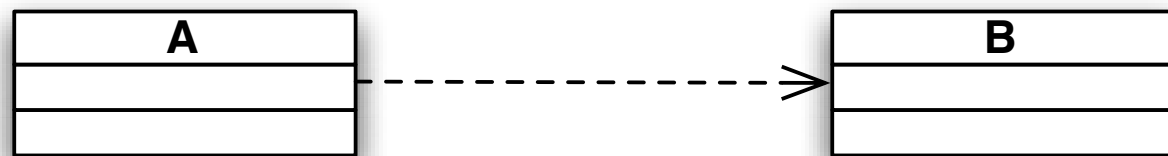
```
    private List<B> my_list =  
        new ...
```

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

Java sketch

Class diagram

Dependency

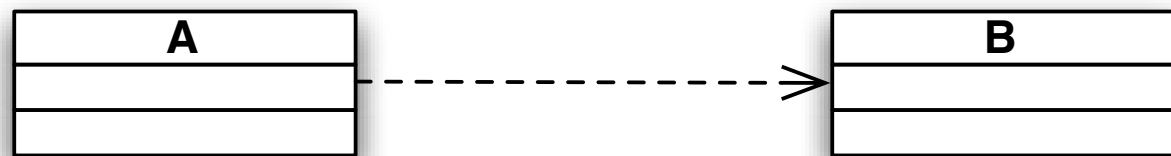


UML sketch

Java sketch

Class diagram

Dependency



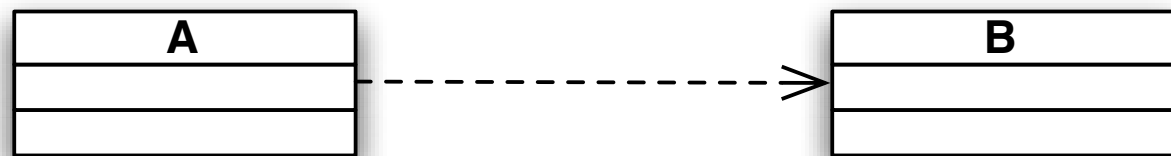
UML sketch

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

Java sketch

Class diagram

Dependency



UML sketch

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

And many other
cases ...

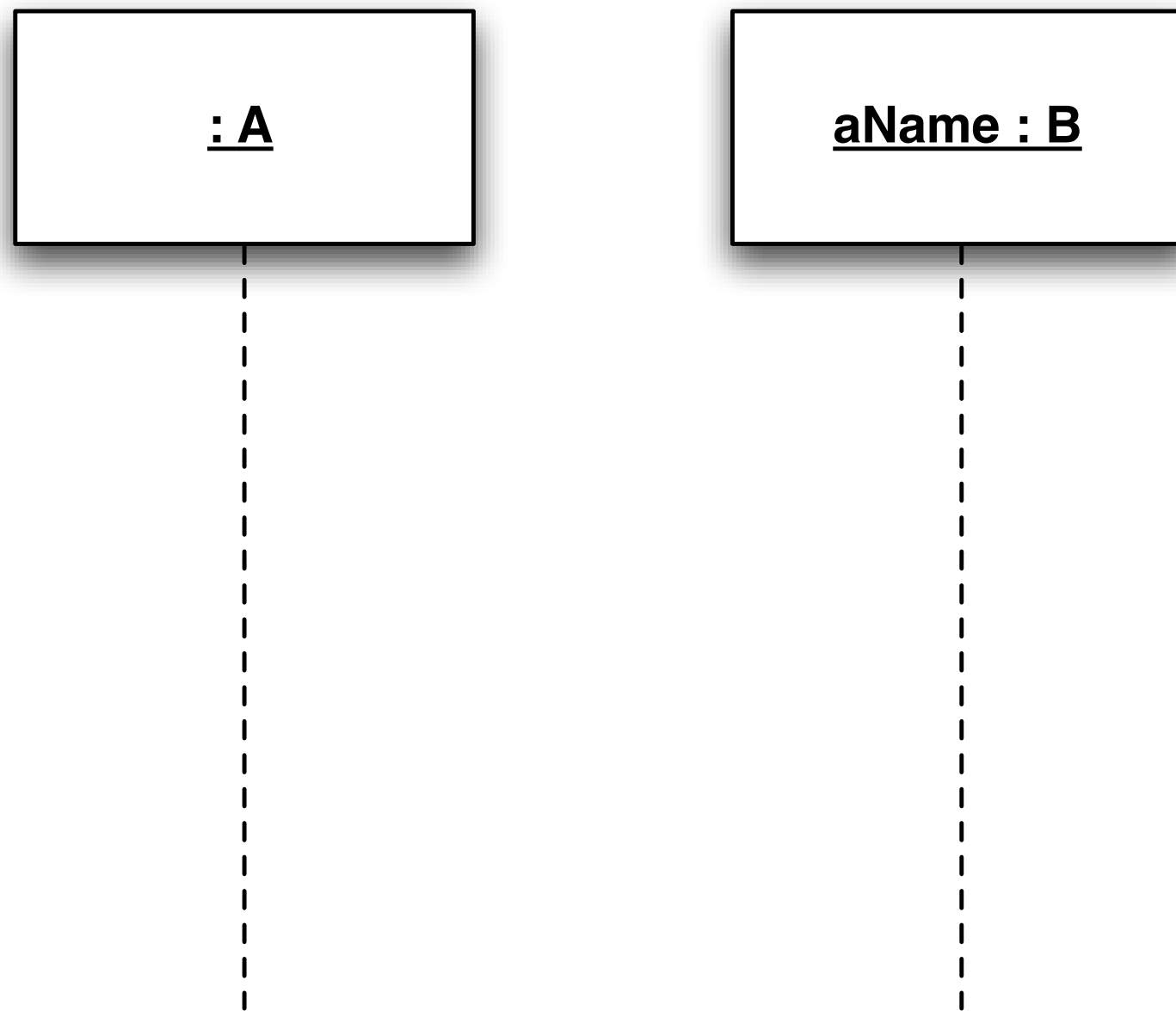
Java sketch

Sequence diagram

Behavioral & Interaction model

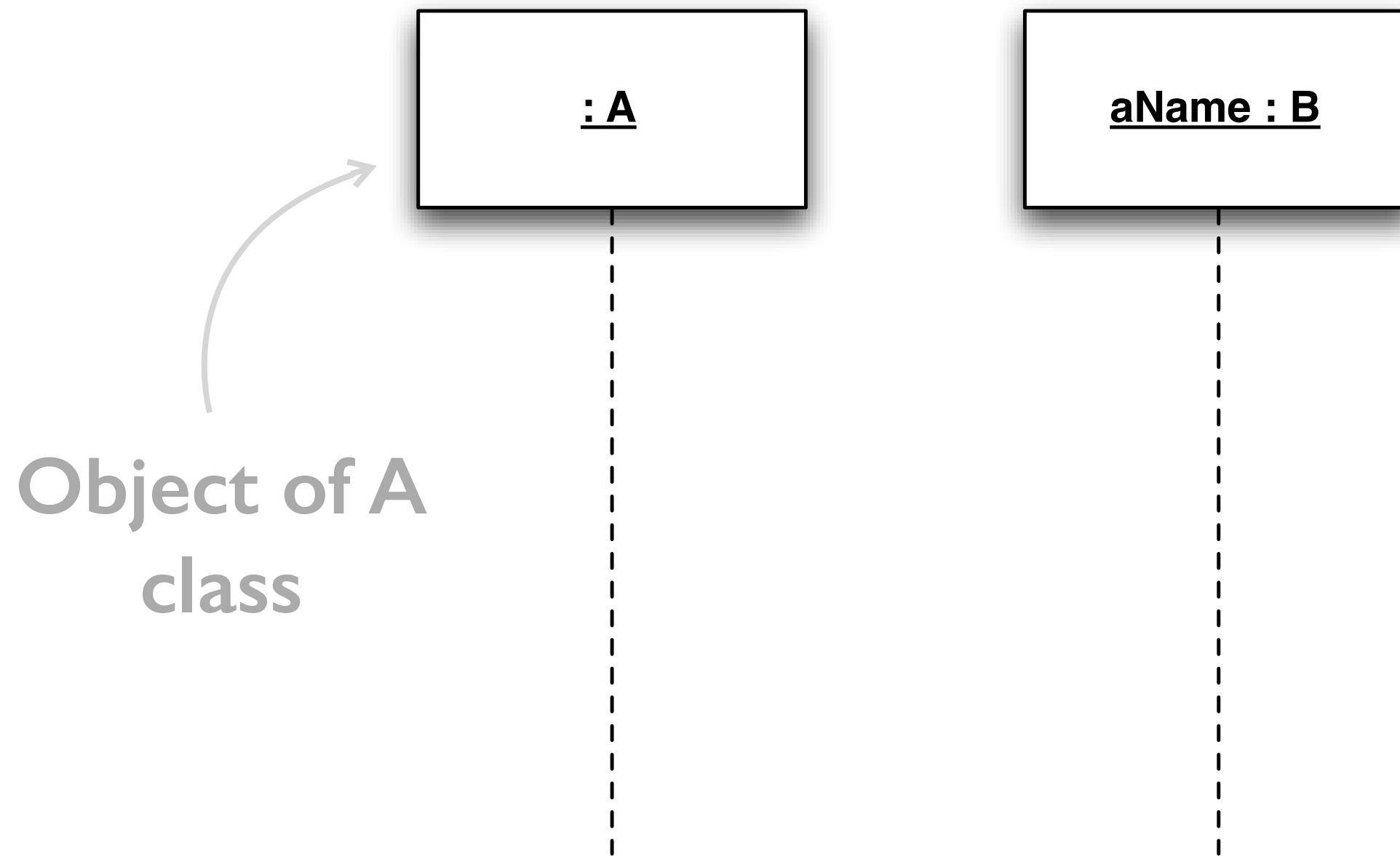
Sequence diagram

Behavioral & Interaction model



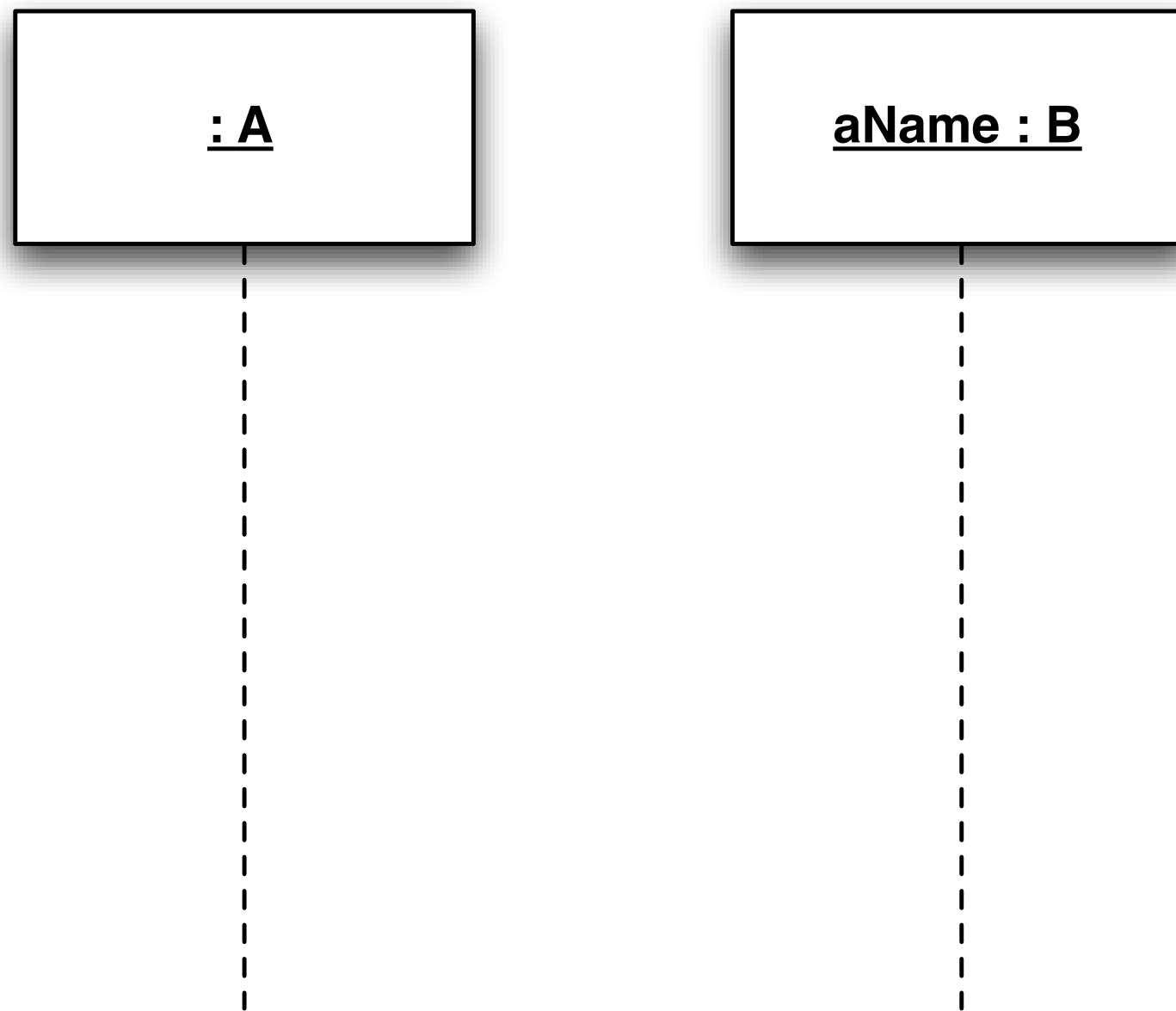
Sequence diagram

Behavioral & Interaction model



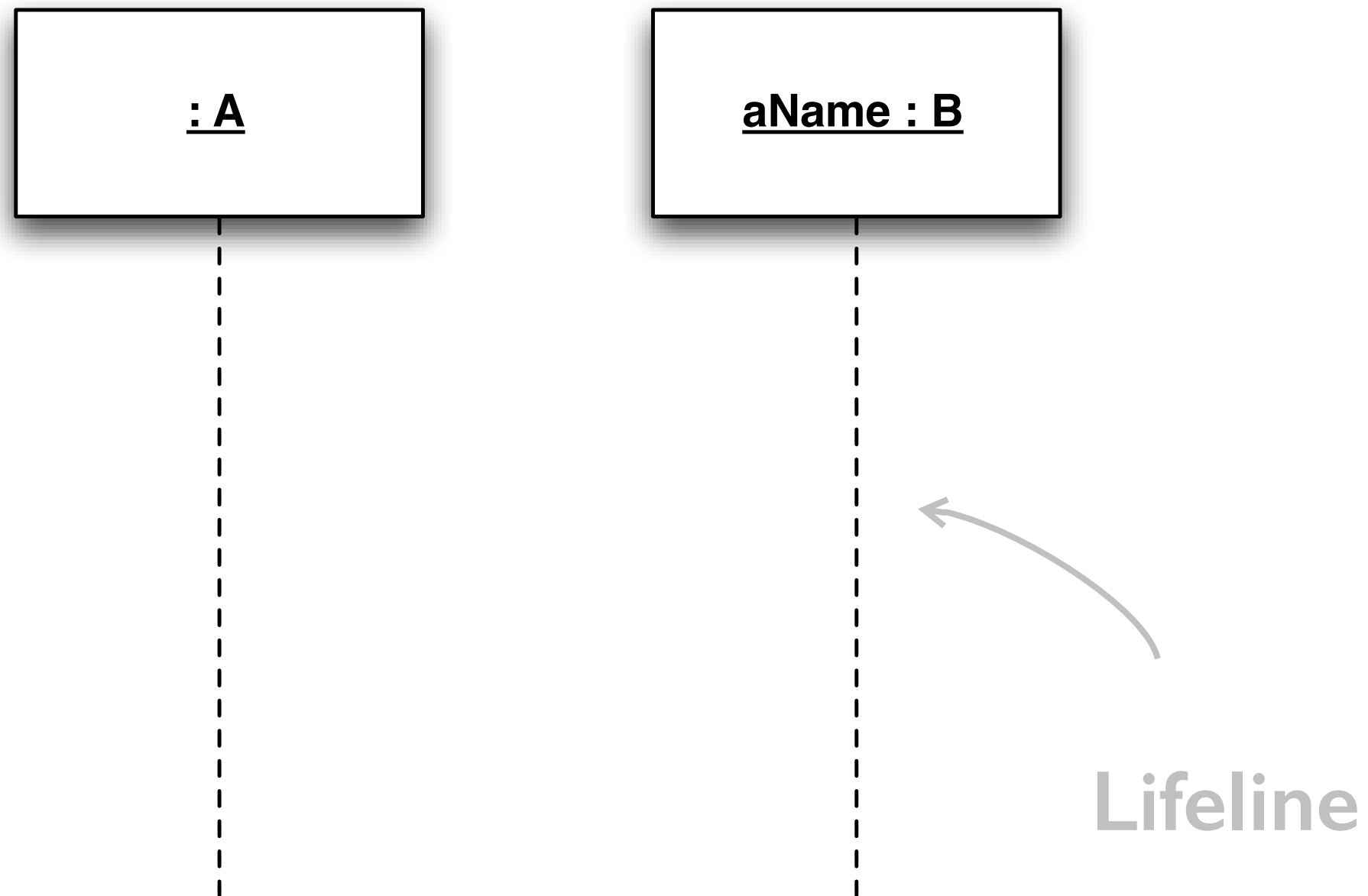
Sequence diagram

Behavioral & Interaction model



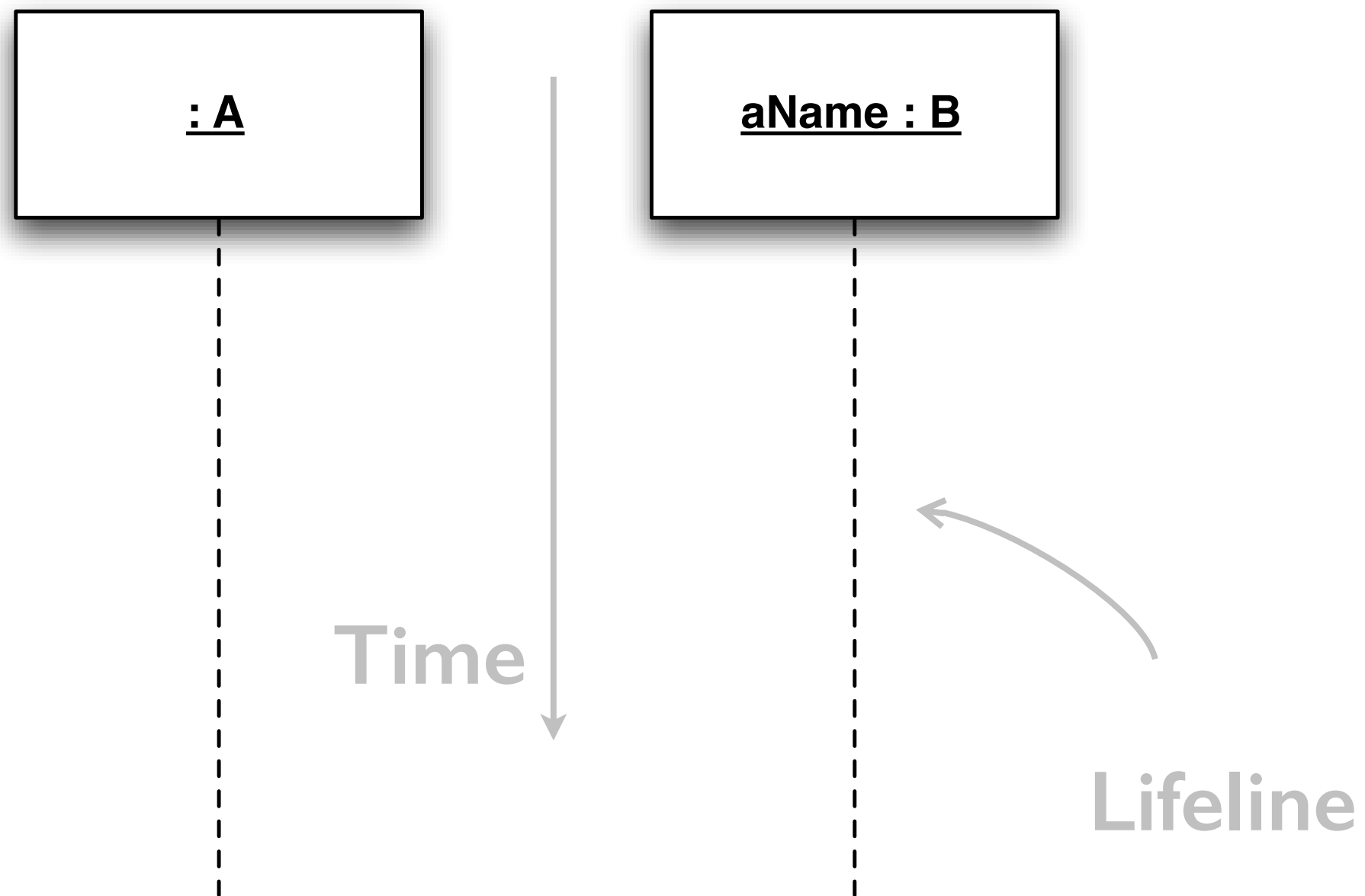
Sequence diagram

Behavioral & Interaction model



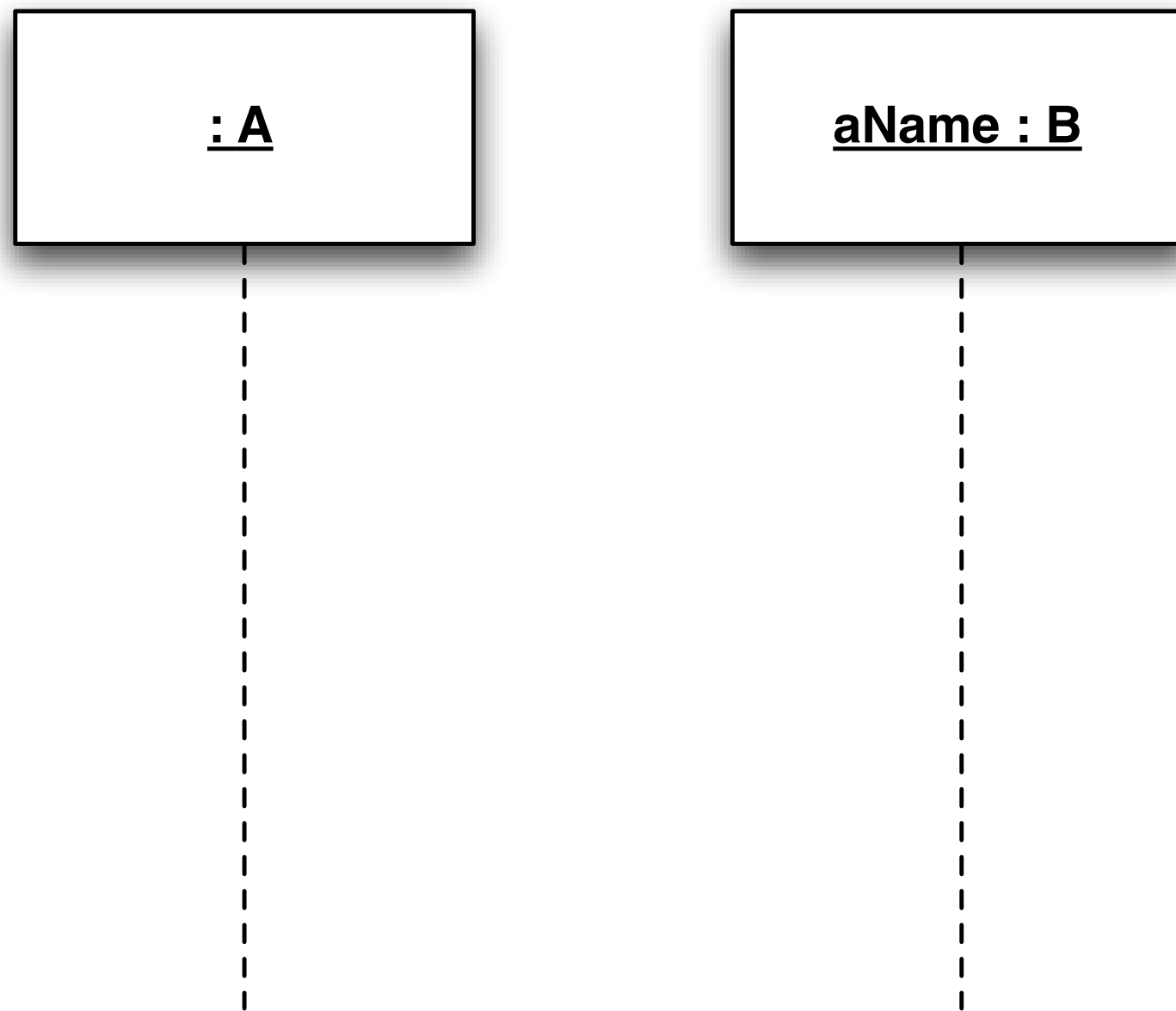
Sequence diagram

Behavioral & Interaction model



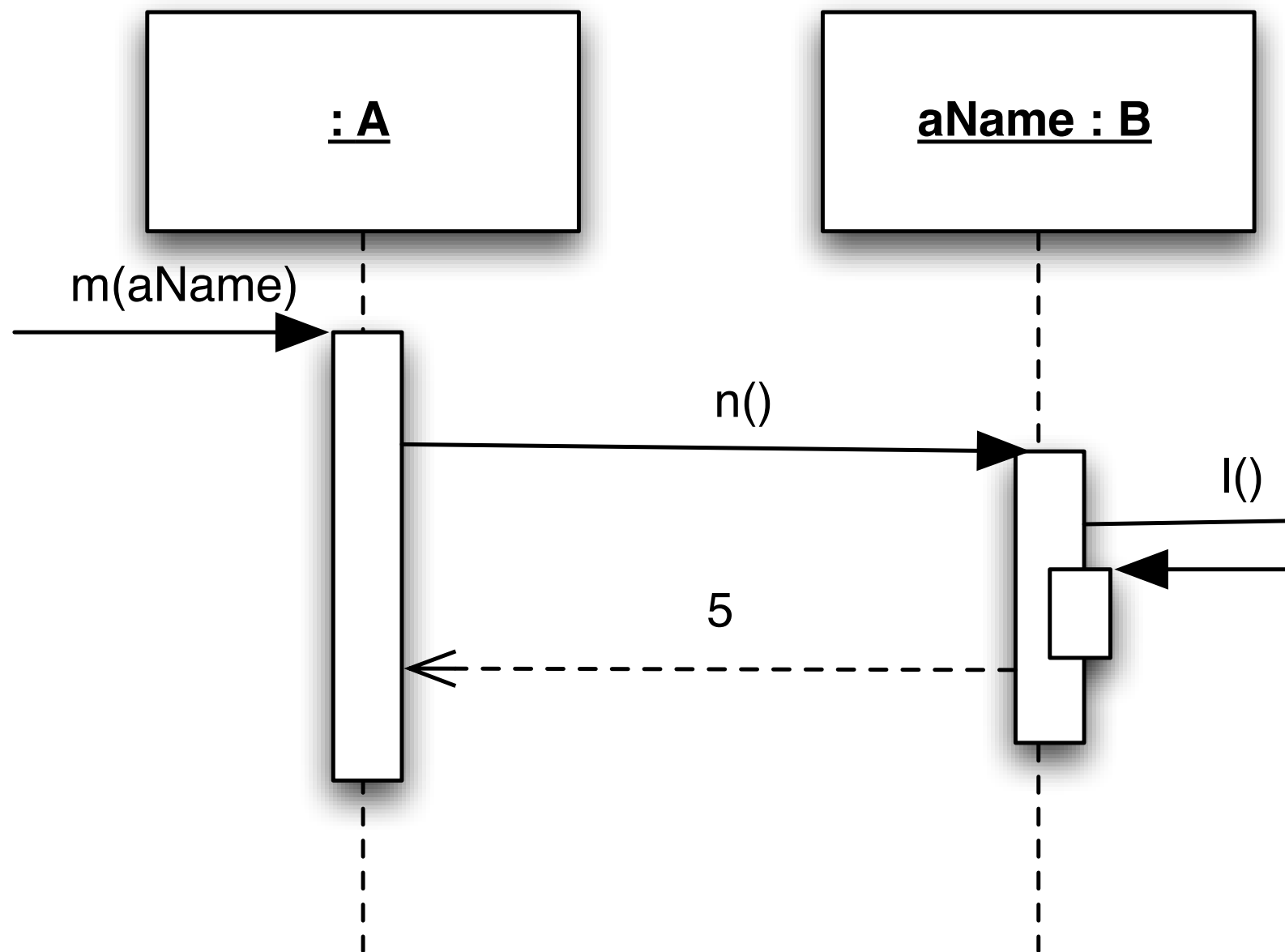
Sequence diagram

Behavioral & Interaction model



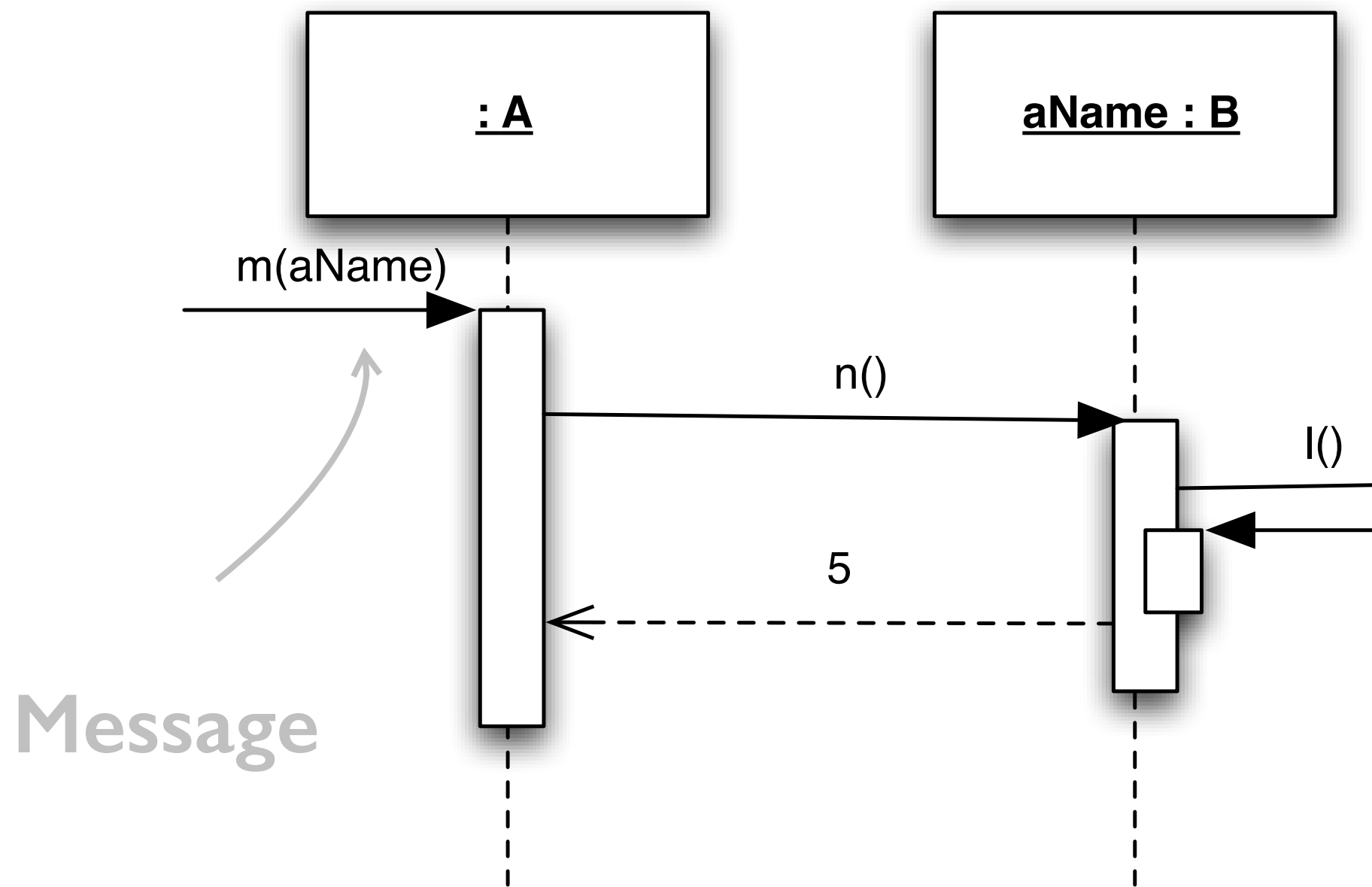
Sequence diagram

Behavioral & Interaction model



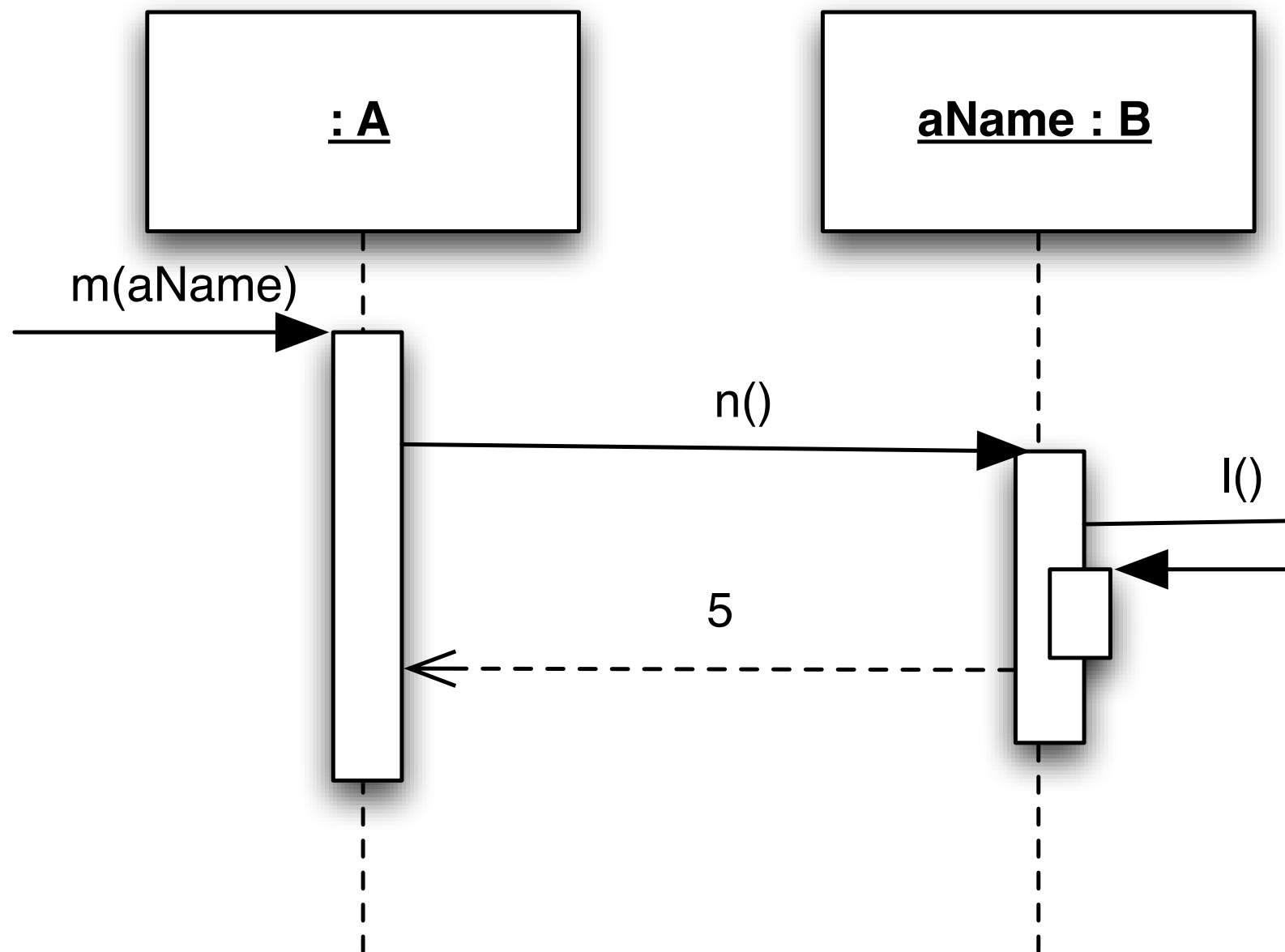
Sequence diagram

Behavioral & Interaction model



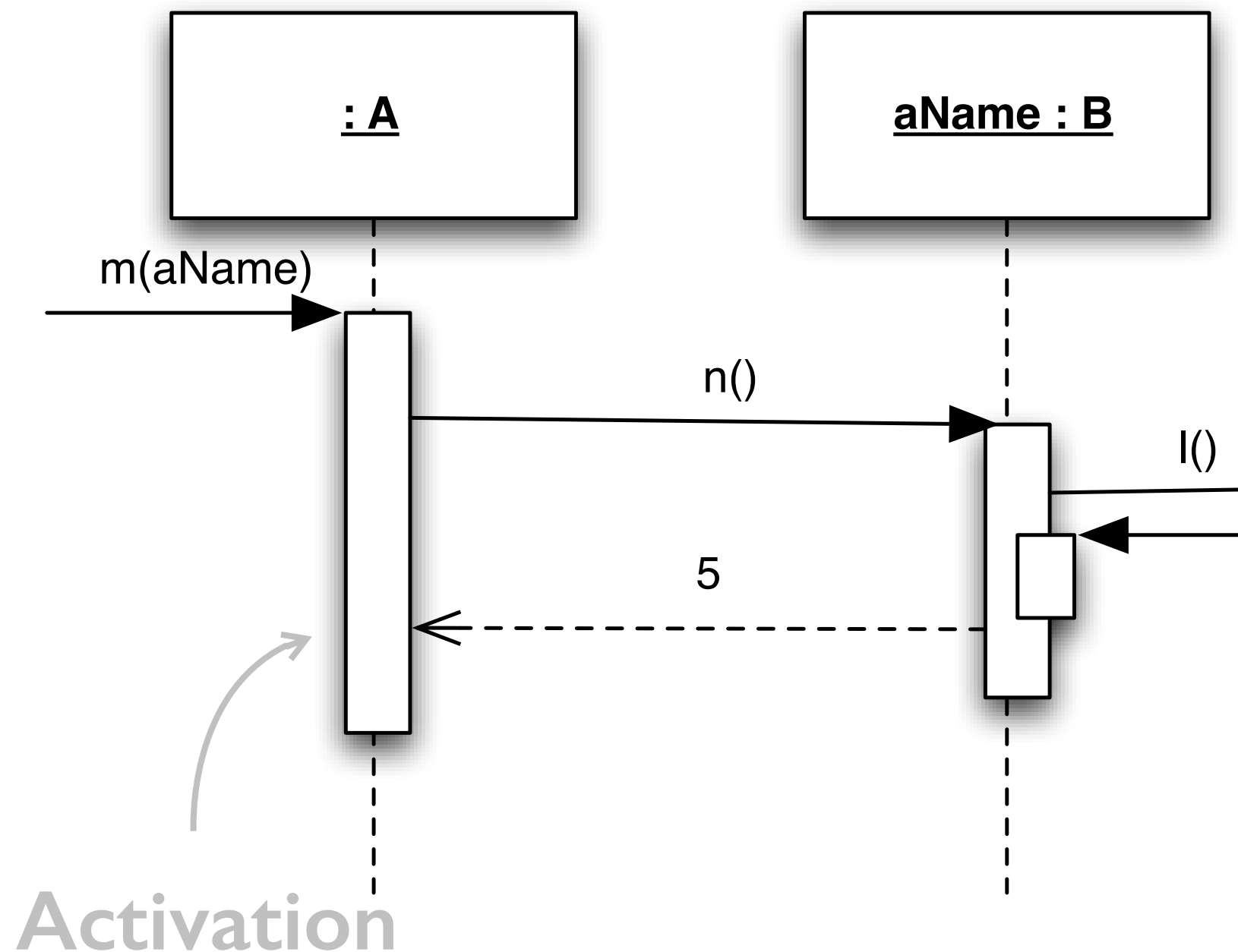
Sequence diagram

Behavioral & Interaction model



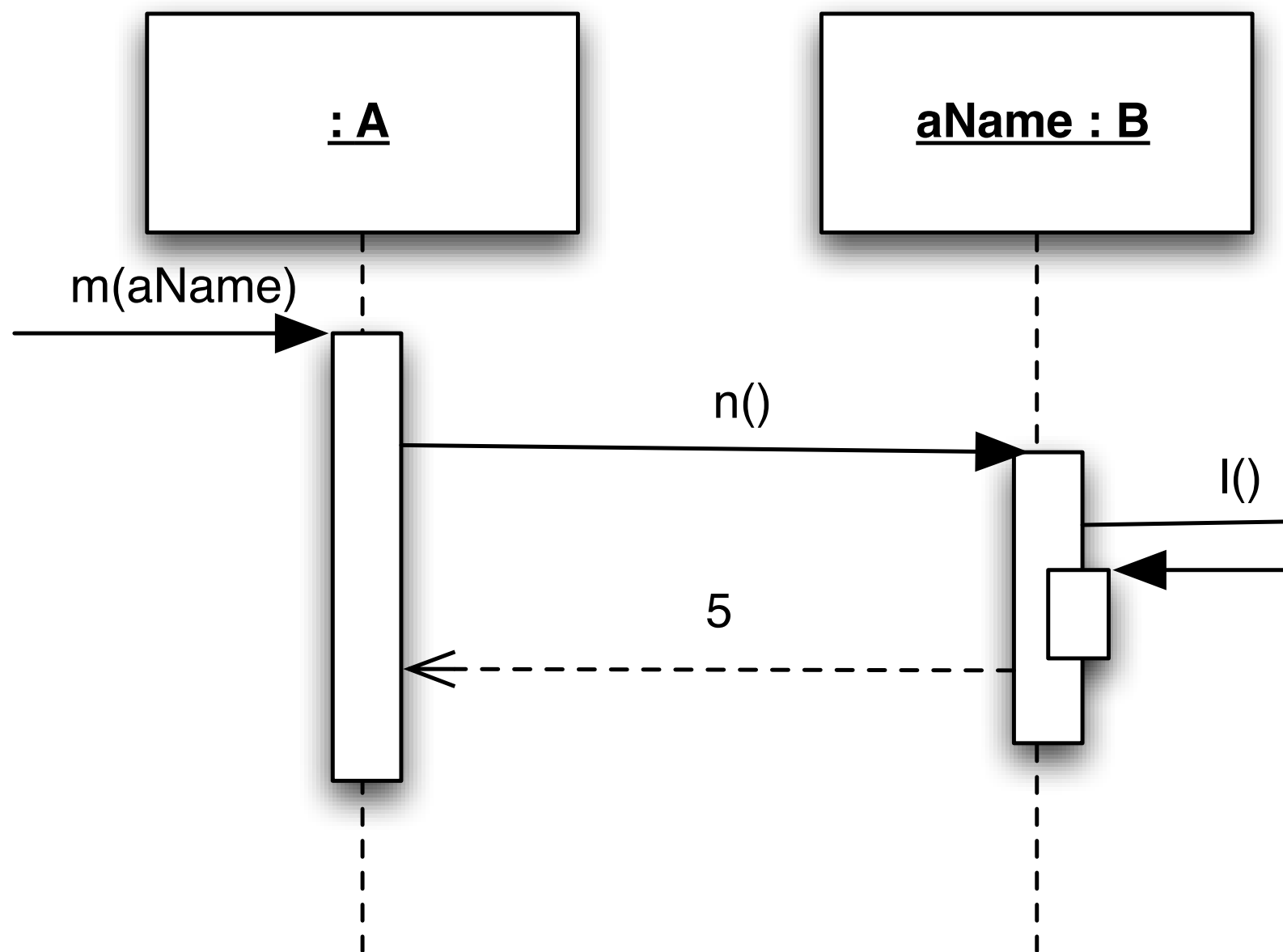
Sequence diagram

Behavioral & Interaction model



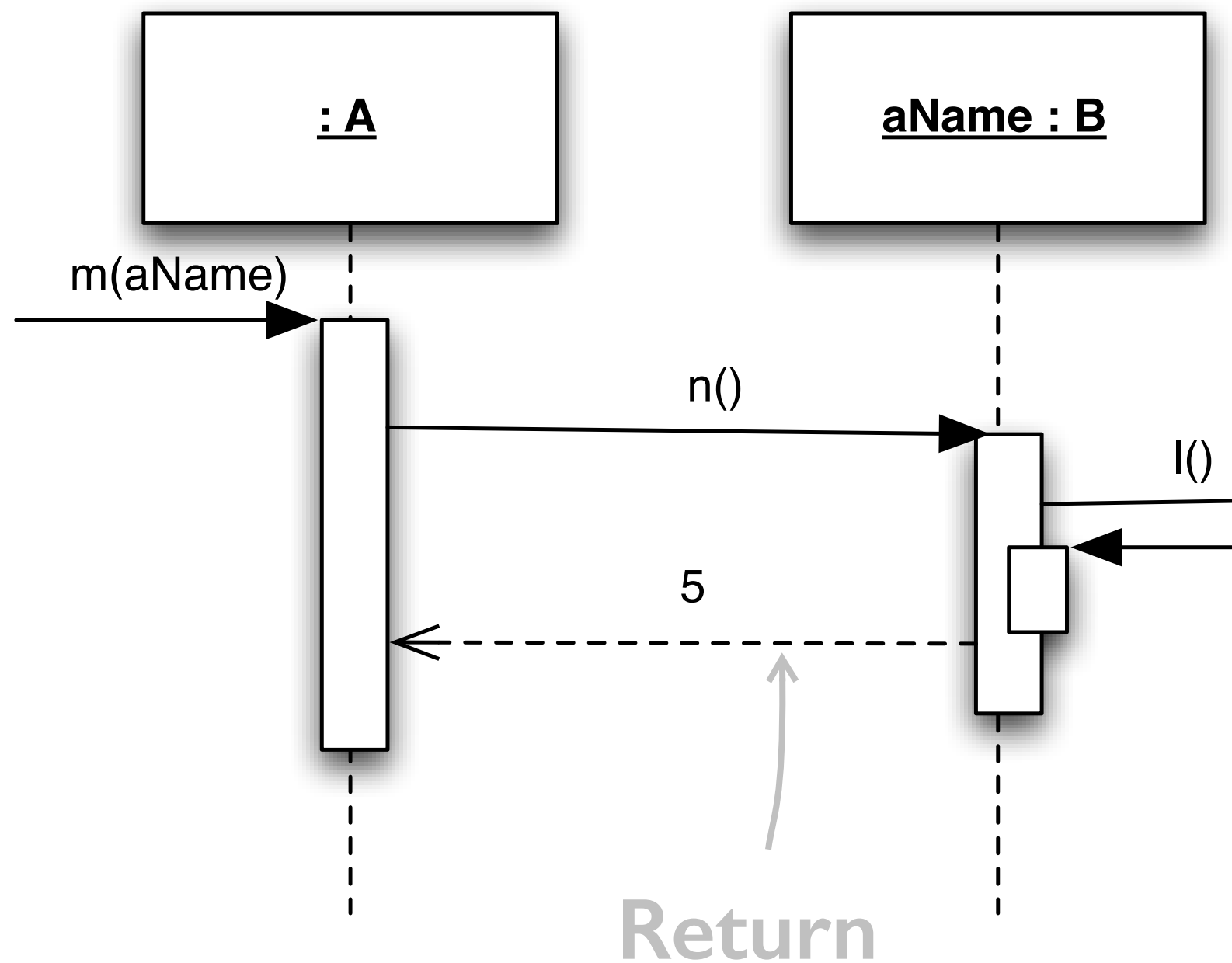
Sequence diagram

Behavioral & Interaction model



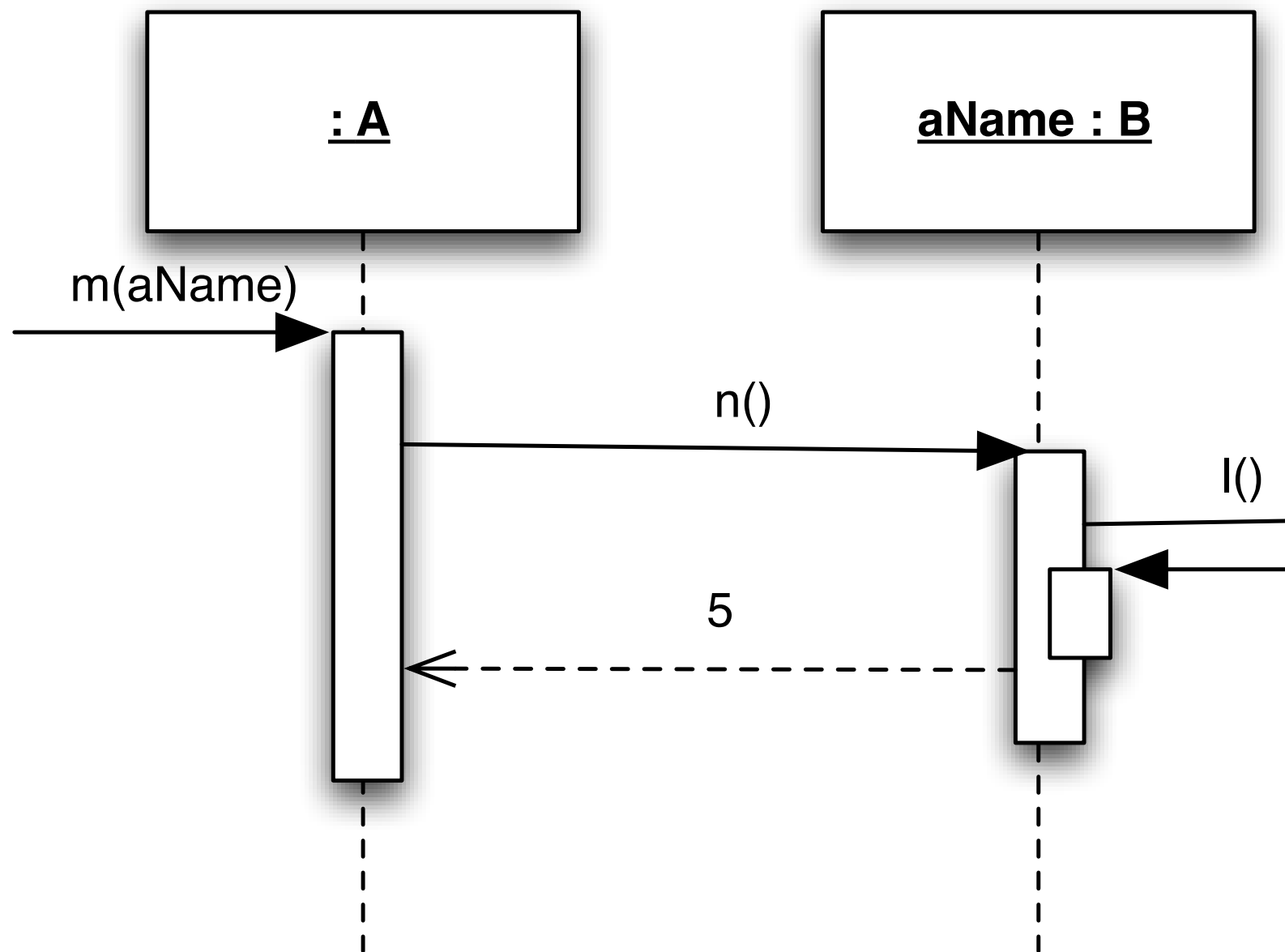
Sequence diagram

Behavioral & Interaction model



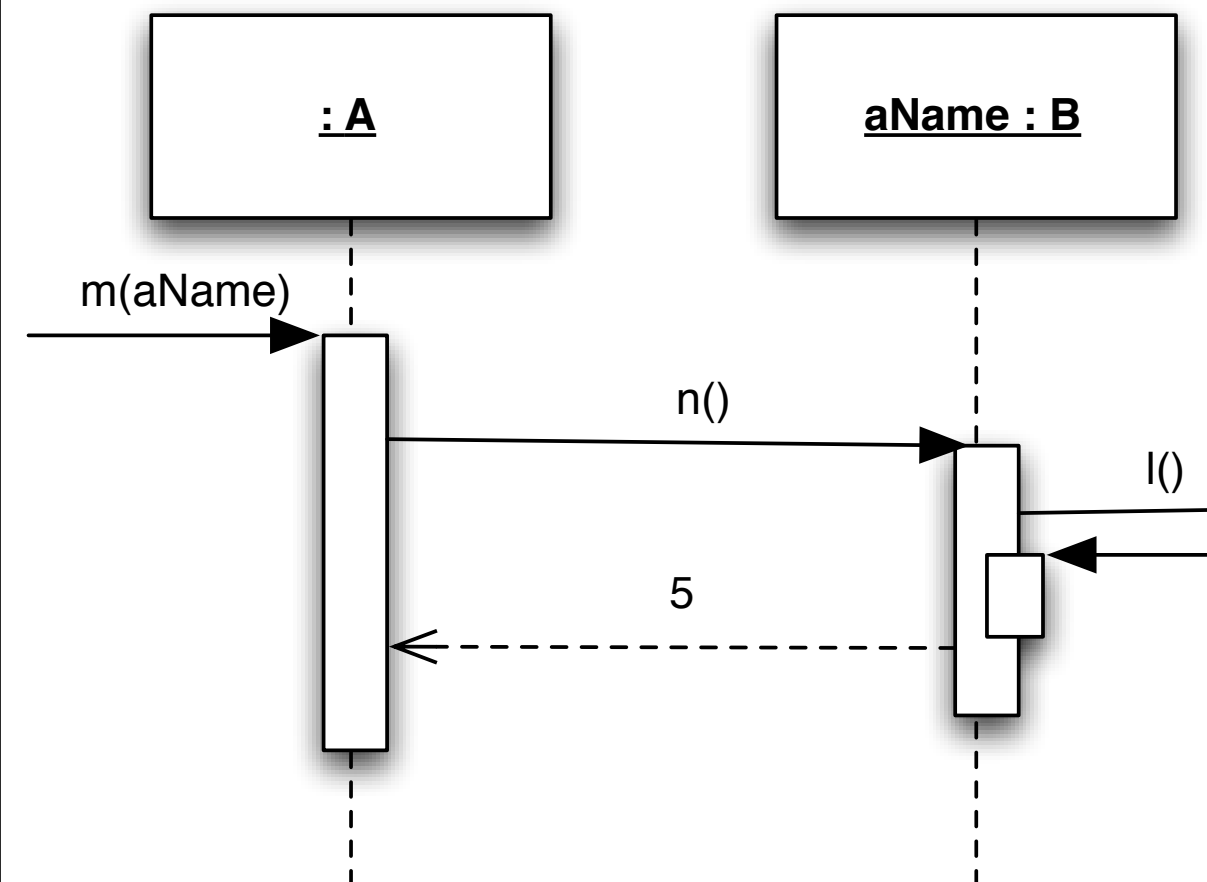
Sequence diagram

Behavioral & Interaction model



Sequence diagram

Behavioral & Interaction model



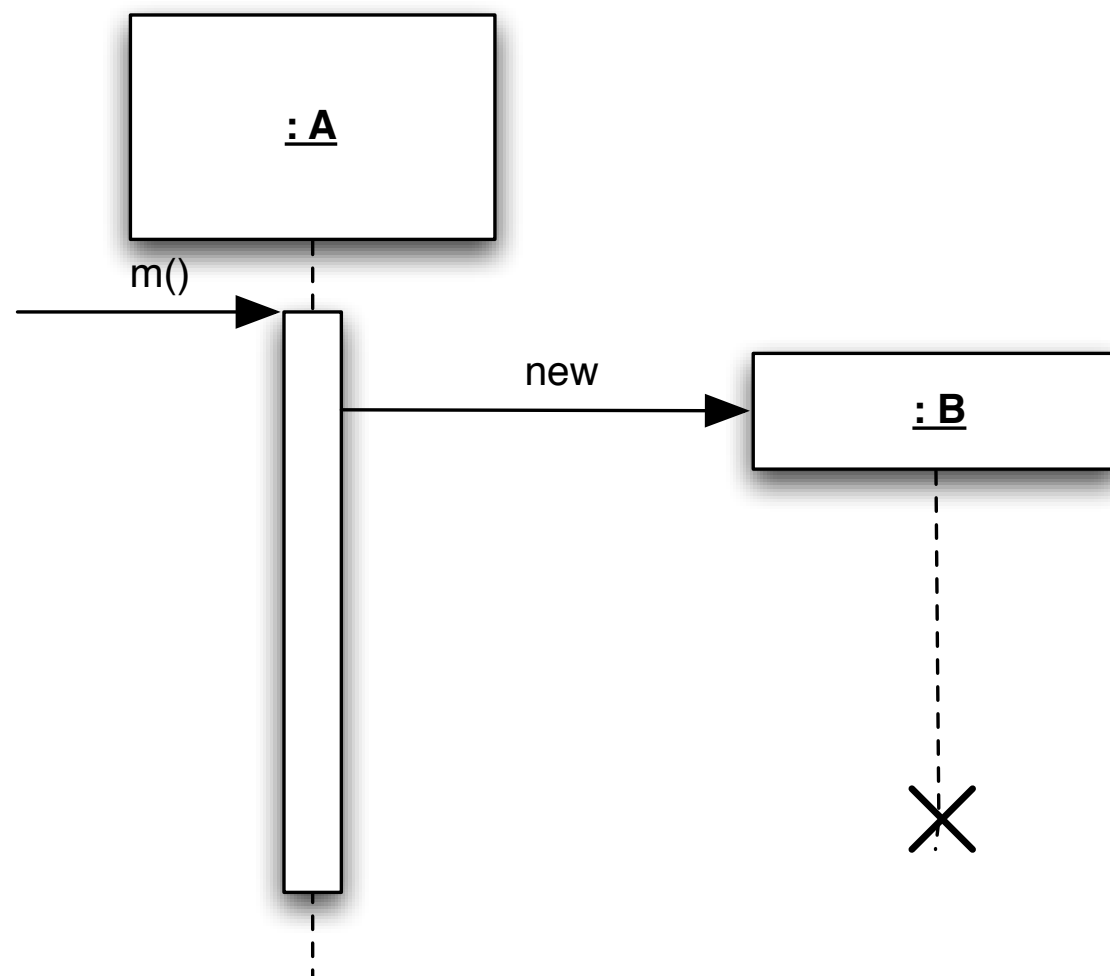
UML sketch

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

Java sketch

Sequence diagram

Object creation & deletion

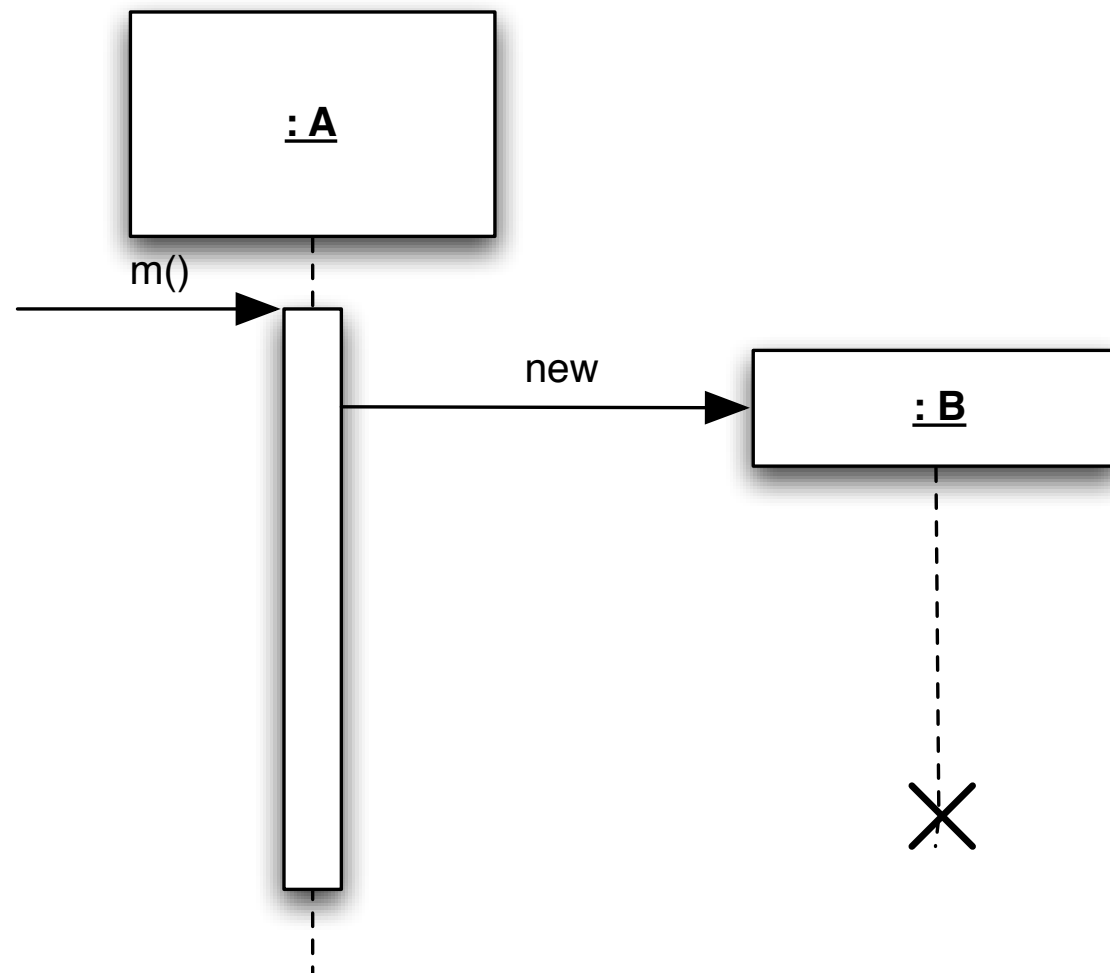


UML sketch

Java sketch

Sequence diagram

Object creation & deletion



UML sketch

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

Java sketch

Sequence diagram

Other notations



e.g. method invocation

Sequence diagram

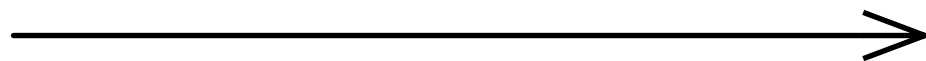
Other notations

Synchronous



e.g. method invocation

Async Message



e.g. start an execution thread

Sequence diagram

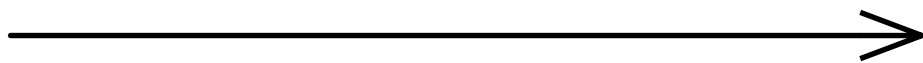
Other notations

Synchronous

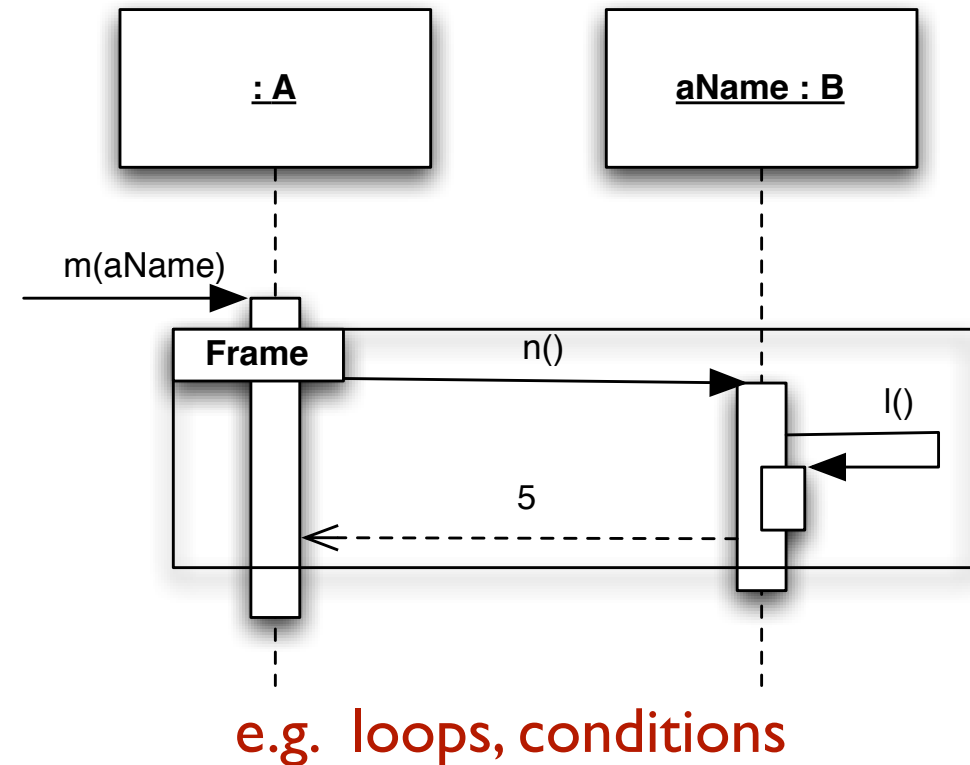


e.g. method invocation

Async Message



e.g. start an execution thread



Sequence diagram

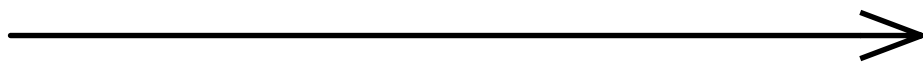
Other notations

Synchronous

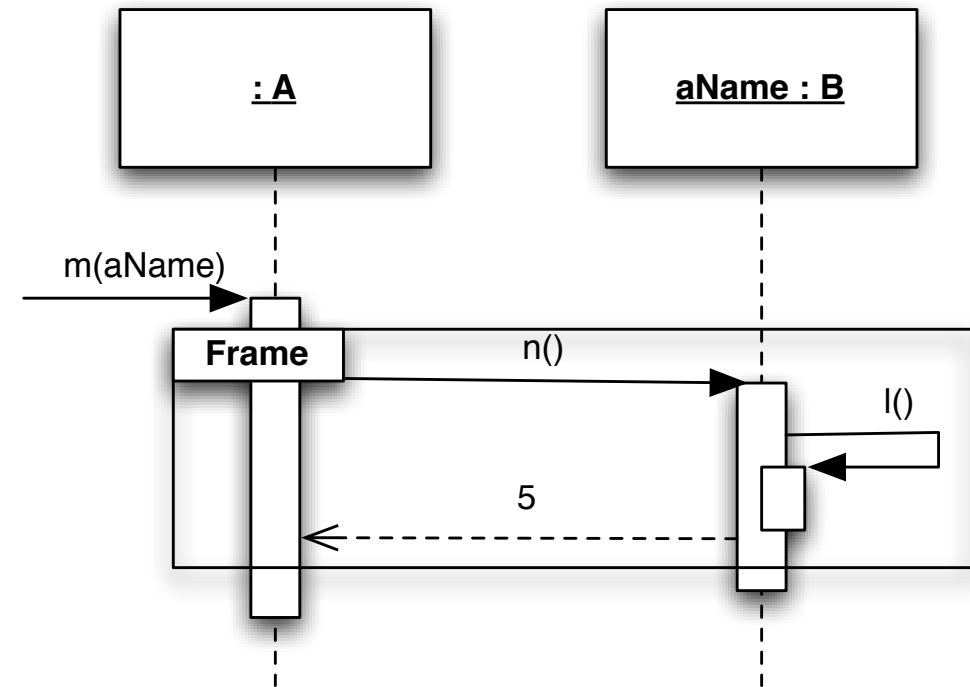


e.g. method invocation

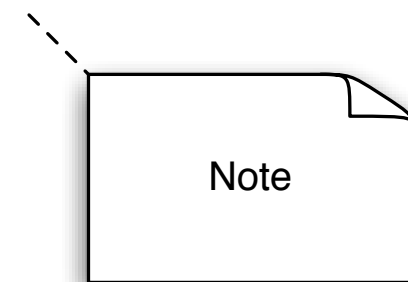
Async Message



e.g. start an execution thread



e.g. loops, conditions



Comments

Sequence diagram

Let us see an example ...