

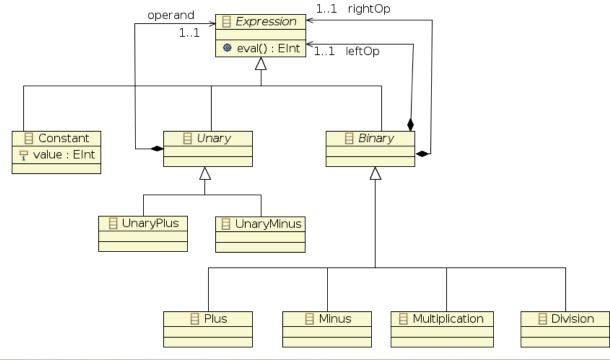
A (Basic) C++ Course

10 - SumUp

Julien Deantoni



- A class diagram gives
 - structural aspects
 - relational aspects



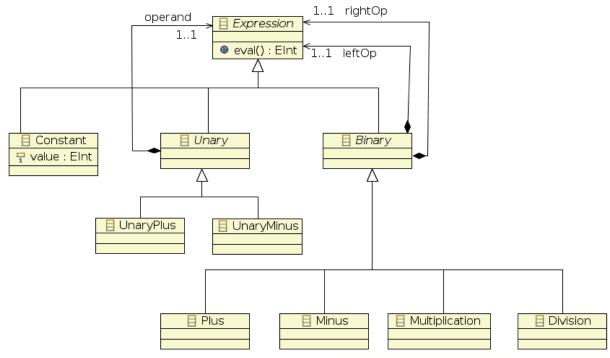




- A class diagram gives
 - structural aspects
 - relational aspects

No description of the behaviours

How does eval() behave?







- A class diagram gives
 - structural aspects
 - relational aspects

};

```
class Expression
{
public:
```

No description of the behaviours

How does eval() behave?

```
class ZeroDivide : public exception
{
public:
    const char* what() const throw () {return "Division
    by 0";}
};

virtual ~Expr() {}
virtual int eval() const = 0;
```

Expression.h



- A class diagram gives
 - structural aspects
 - relational aspects

No description of the behaviours

How does eval() behave?

```
class Expression
public:
    class ZeroDivide : public exception
    public:
        const char* what() const throw () {return "Division
        by 0";}
    virtual ~Expr() {}
    virtual int eval() const = 0;
};
```



- A class diagram gives
 - structural aspects
 - relational aspects

};

```
class Expression
{
public:
```

No description of the behaviours

How does eval() behave?

```
class ZeroDivide : public exception
{
public:
    const char* what() const throw () {return "Division by 0";}
};

virtual ~Expr() {}
virtual int eval() const = 0;
```



- A class diagram gives
 - structural aspects
 - relational aspects

};

class Expression

```
No description of
  the behaviours

Class ZeroDivide : public exception
  {
    public:
        const char* what() const throw () {return "Division
        by 0";}
    };

virtual ~Expr() {}
    virtual int eval() const = 0;
```





- A class diagram gives
 - structural aspects
 - relational aspects

No description of the behaviours

```
class UnaryPlus : public Unary
{
protected:
    Expr* operand;
public:
    Unary_Plus(Expr& pe);
    virtual int eval() const;
};
```

```
operand
1..1 rightOp

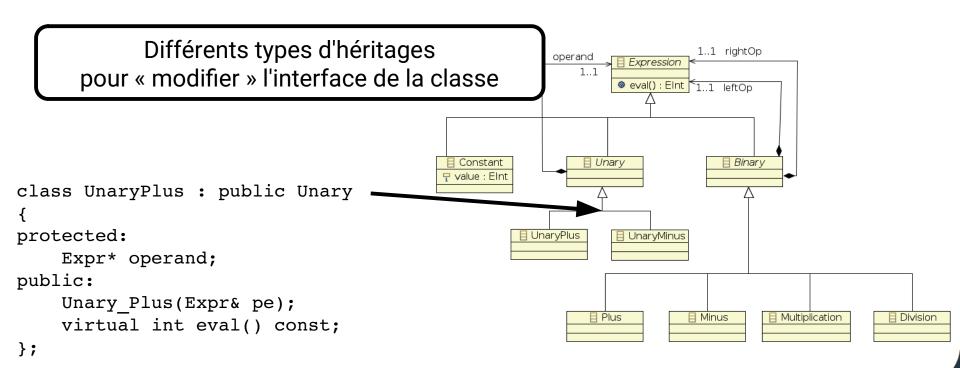
1..1 leftOp

| Constant | Unary | Binary
| value : Eint | UnaryMinus | Multiplication | Division
```



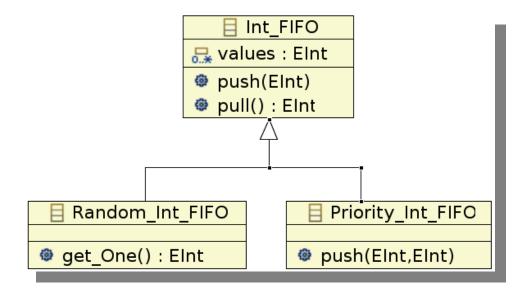


- A class diagram gives
 - structural aspects
 - relational aspects









What if we declare Random_Int_FIFO like that?

```
class Random_Int_FIFO : private Int_FIFO
{
  public:
   int get_One();
}
```



- The private derivation
 - All members of derived class become private
 - The "interface" of the derived class is lost...

```
class Random_Int_FIFO : private Int_FIFO
{
  public:
  int get_One();
}
```



- The private derivation
 - All members of derived class become private
 - The "interface" of the derived class is lost...

```
No more
possible to
push()
integers in
the FIFO
```

```
class Random Int_FIFO : private Int_FIFO
{
  public: o
  int get_One();
}
```

- The private derivation
 - All members of derived class become private
 - The "interface" of the derived class is lost...
 - But some parts of the interface can be set public again

```
class Random_Int_FIFO : private Int_FIFO
{
  public:
   int get_One();
   using Int_FIFO::push;
}
```





- A class diagram gives
 - structural aspects
 - relational aspects

```
1..1 rightOp
                                                                            Expression
       No description of
                                                                            eval(): EInt 1...1 leftOp
         the behaviours
                                                                                         ∃ Binar∨
                                                                        ☐ Unary
                                                       value : EInt
class Unary: public Expression
protected:
                                                                ☐ UnaryPlus
                                                                            ∃ UnarvMinus
     Expr* operand;
public:
     Unary(Expr& pe);
                                                                        目 Plus
                                                                                     ■ Minus

☐ Multiplication

                                                                                                              Division
     virtual int eval() const;
};
```





- A class diagram gives
 - structural aspects
 - relational aspects

```
1..1 rightOp
                                                                             Expression
       No description of
                                                                            eval(): EInt 1...1 leftOp
         the behaviours
                                                                                         ∃ Binar∨
                                                                        ☐ Unary
                                                       value : EInt
class Unary: public Expression
protected:
                                                                ☐ UnaryPlus
                                                                            ∃ UnarvMinus
     Expr* operand;
public:
     Unary(Expr& pe);
                                                                        目 Plus
                                                                                     ■ Minus

☐ Multiplication

                                                                                                              Division
     virtual int eval() const;
};
```





- A class diagram gives
 - structural aspects
 - relational aspects

```
1..1 rightOp
                                                                           Expression
       No description of
                                                                           eval(): EInt 1...1 leftOp
         the behaviours
                                                                                        ∃ Binar∨
                                                                       ☐ Unary
                                                       value : EInt
class Unary: public Expression
protected:
                                                               ☐ UnaryPlus
                                                                           ∃ UnarvMinus
     Expr* operand;
public:
     Unary(Expr& pe);
                                                                       目 Plus
                                                                                    ■ Minus

☐ Multiplication

                                                                                                             ☐ Division
     virtual int eval() const;
};
                       Réflexe associé?
```





1..1 rightOp

Object-Oriented concepts

- A class diagram gives
 - structural aspects
 - relational aspects

```
Expression
       No description of
                                                                         eval(): EInt 1...1 leftOp
         the behaviours
                                                                                     ∃ Binar∨
                                                                    ☐ Unary
class Unary: public Expression
                                                    7 value : EInt
protected:
                                                             ☐ UnaryPlus
                                                                         ∃ UnarvMinus
     Expr* operand;
public:
     Unary(Expr& pe);
     virtual int eval() const;
                                                                     目 Plus
                                                                                           Multiplication
                                                                                 ■ Minus
                                                                                                          Division
     virtual ~Unary();
```



};



- A class diagram gives
 - structural aspects
 - relational aspects

No description of the behaviours

```
class Unary : public Expression
{
protected:
    Expr* operand;
public:
    Unary(Expr& pe);
    virtual int eval() const;
    virtual ~Unary();
};
```

Remember the impact of the containment

```
Operand

I...1 rightOp

Expression

Operand

I...1 leftOp

Operand

Operand

I...1 leftOp

Operand

II...1 leftOp

Operand

III...1 leftOp

III...1 leftOp

IIII...1 leftOp

III...1 leftOp

II
```





- Impact of containment:
 - The life of contained object(s) is your responsibility

```
Unary::Unary(Expression& pe)
{
    operand = pe.clone();
}

Expression* UnaryPlus::clone()
{
    return new UnaryPlus(*this);
}
```





- Impact of containment:
 - The life of contained object(s) is your responsibility

```
Unary::Unary(Expression& pe)
{
    operand = pe.clone();
}

    Du type dynamique de "pe"

Expression* UnaryPlus::clone()
{
    return new UnaryPlus(*this);
}
```



- Impact of containment:
 - The life of contained object(s) is your responsibility

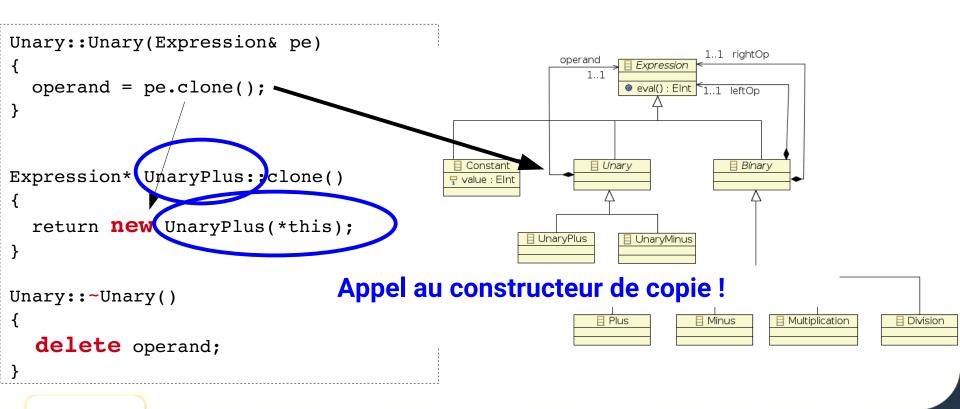
```
Unary::Unary(Expression& pe)
  operand = pe.clone();
                                                                               🗏 Binary
Expression* UnaryPlus::clone()
                                                🖵 value : EInt
  return new UnaryPlus(*this);
                                                                     UnaryMinus
Unary::~Unary()
                                                                            ■ Minus

☐ Multiplication

  delete operand;
```

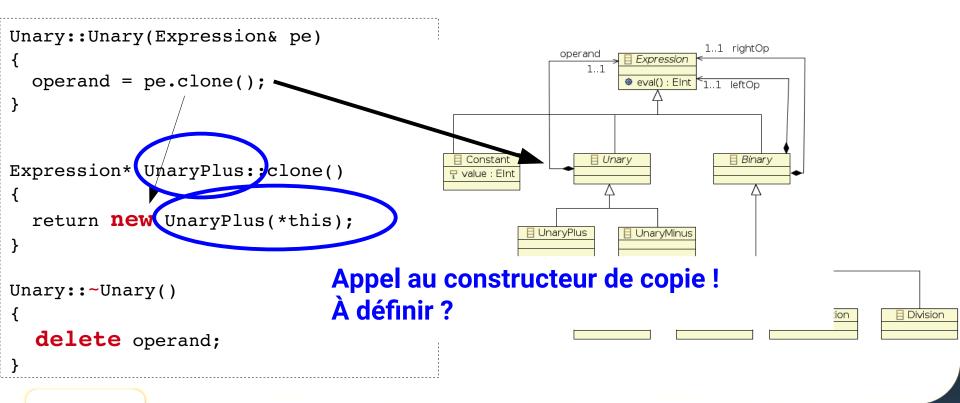


- Impact of containment:
 - The life of contained object(s) is your responsibility





- Impact of containment:
 - The life of contained object(s) is your responsibility





- Impact of containment:
 - The life of contained object(s) is your responsibility

```
Unary::Unary(const Unary& un)
                                 Suffisant ou à redéfinir dans UnaryPlus?
  operand = un.op->clone();
                                                                          1..1 rightOp
Unary::Unary(Expression& pe)
                                                                 eval() : EInt 
  operand = pe.clone(); -
                                                                            ☐ Binary
Expression* UnaryPlus::clone()
                                                        UnaryPlus
                                                                 📘 UnaryMinus
  return new UnaryPlus(*this);
                                                                         Minus

    ■ Multiplication

Unary::~Unary()
  delete operand;
```



- Impact of containment:
 - The life of contained object(s) is your responsibility

```
Unary::Unary(const Unary& un)
  operand = un.op->clone();
                                                                            1..1 rightOp
Unary::Unary(Expression& pe)
                                                                   eval() : EInt |
  operand = pe.clone(); =
                                                                               ☐ Binary
Expression* UnaryPlus::clone()
                                                          UnaryPlus
                                                                   📘 UnaryMinus
  return new UnaryPlus(*this);

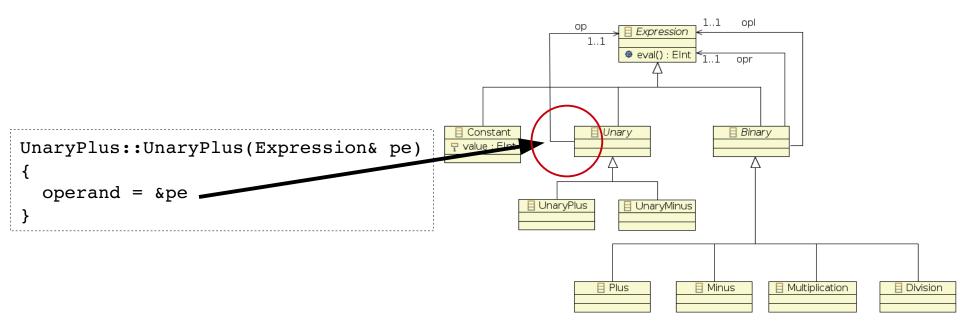
    ■ Multiplication

                                                                           🗏 Minus
Unary::~Unary()
  delete operand;
                                                   Et bien sûr ça impose la définition de
                                                   l'opérateur d'affectation...
```

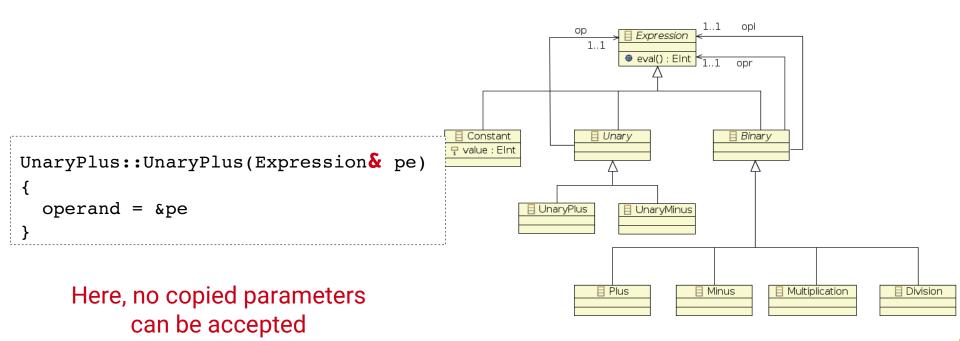




- Impact of containment:
 - The life of contained object(s) is your responsibility









Take care of object life duration

```
main(){
Constant a(3);
}
```

```
UnaryPlus::UnaryPlus(Expression& pe)
{
  operand = &pe
}
```

a:Constant

value = 3

Take care of object life duration

```
main(){
Constant a(3);
UnaryPlus plus_a(a);
}
```

```
UnaryPlus::UnaryPlus(Expression& pe)
{
  operand = &pe
}
```

a:Constant

value = 3

plus_a:UnaryPlus

operand*



Take care of object life duration

```
main(){
Constant a(3);
UnaryPlus plus_a(a);
}
```

```
UnaryPlus::UnaryPlus(Expression& pe)
{
  operand = &pe
}
```

a:Constant
pe:Constant
value = 3

```
plus_a:UnaryPlus
```

operand*



```
main(){
Constant a(3);
UnaryPlus plus_a(a);
}
```

```
UnaryPlus::UnaryPlus(Expression& pe)
{
  operand = &pe
}
```

```
a:Constant
pe:Constant
value = 3

plus_a:UnaryPlus
operand*
```



```
main(){
Constant a(3);
UnaryPlus plus_a(a);
}

UnaryPlus::UnaryPlus(Expression& pe)
{
  operand = &pe
```

```
a:Constant
pe:Constant
value = 3

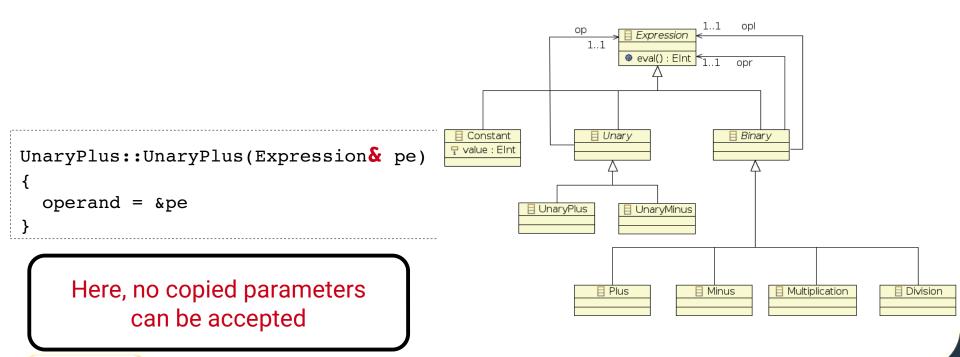
plus_a:UnaryPlus
operand*
```

```
main(){
Constant a(3);
UnaryPlus plus_a(a);
}

plus_a:UnaryPlus

unaryPlus::UnaryPlus(Expression pe)
{
   operand = &pe
}
```







Take care of object life duration

```
main(){
Constant a(3);
}
```

```
UnaryPlus::UnaryPlus(Expression pe)
{
  operand = &pe
}
```

a:Constant

value = 3



Take care of object life duration

```
main(){
Constant a(3);
UnaryPlus plus_a(a);
}
```

```
UnaryPlus::UnaryPlus(Expression pe)
{
  operand = &pe
}
```

a:Constant

value = 3

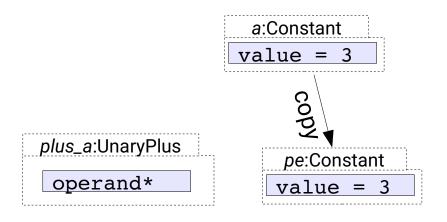
plus_a:UnaryPlus

operand*



```
main(){
Constant a(3);
UnaryPlus plus_a(a);
}
```

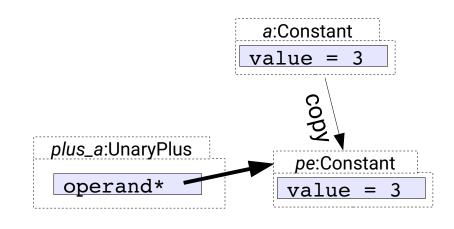
```
UnaryPlus::UnaryPlus(Expression pe)
{
  operand = &pe
}
```





```
main(){
Constant a(3);
UnaryPlus plus_a(a);
}

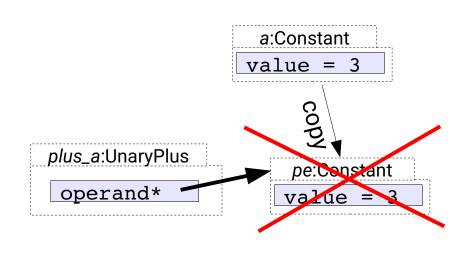
UnaryPlus::UnaryPlus(Expression pe)
{
   operand = &pe
```





```
main(){
Constant a(3);
UnaryPlus plus_a(a);
}

UnaryPlus::UnaryPlus(Expression pe)
{
  operand = &pe
}
```





```
UnaryPlus::UnaryPlus(Expression pe)
{
  operand = &pe
}
```

```
a:Constant
value = 3
```

```
plus_a:UnaryPlus

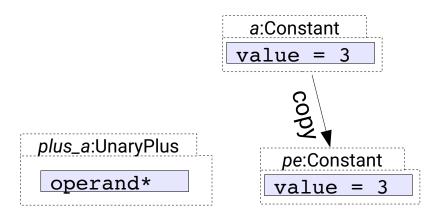
operand*
????
```



Take care to truncation...

```
main(){
Constant a(3);
UnaryPlus plus_a(a);
}
```

```
UnaryPlus::UnaryPlus(Expression pe)
{
  operand = &pe
}
```





Take care to truncation...

```
main(){
Constant a(3);
UnaryPlus plus_a(a);
}
```

Triple wrong because here:

- 1. Expression is abstract
- 2. truncation occurred
- 3.@ of temporary object is stored

UnaryPlus::UnaryPlus(Expression pe)
{
 operand = &pe
}

plus_a:UnaryPlus

operand*

a:Constant

value = 3

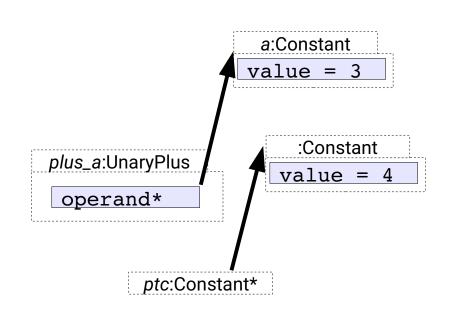
pe:Expression



```
Main(){
Constant* ptc = new Constant(4);

Constant a(3);
UnaryPlus plus_a(a);
}
```

```
UnaryPlus::UnaryPlus(Expression& pe)
{
  operand = &pe
}
```





```
Main(){
Constant* ptc = new Constant(4);

Constant a(3);
UnaryPlus plus_a(a),
}

UnaryPlus::UnaryPlus(Expression& pe)
{
    operand = &pe
}

ptc:Constant

ptc:Constant*
```





```
Main(){
                                                                     a:Constant
Constant* ptc = new Constant(4);
                                                                   value
Constant a(3);
UnaryPlus plus a(a),
                                                                        :Constant
                                                 plus_a:UnaryPlus
                                                                      value = 4
                                                  operand.
UnaryPlus::UnaryPlus(Expression& pe)
                                                                       Memory leak
  operand = &pe
                                                         ptc:Constan
```

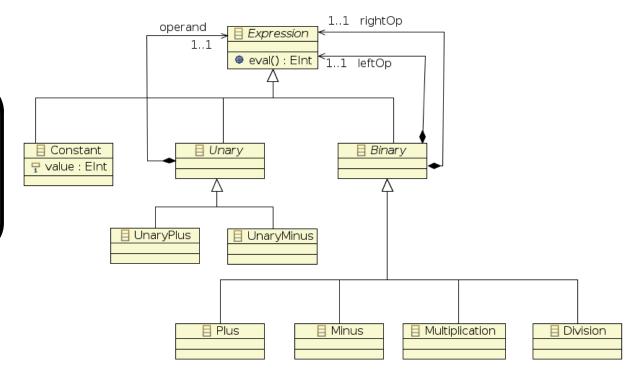




- A class diagram gives
 - structural aspects
 - relational aspects

Almost equivalent to a set of .h files: Expression.h Unary.h UnaryPlus.h

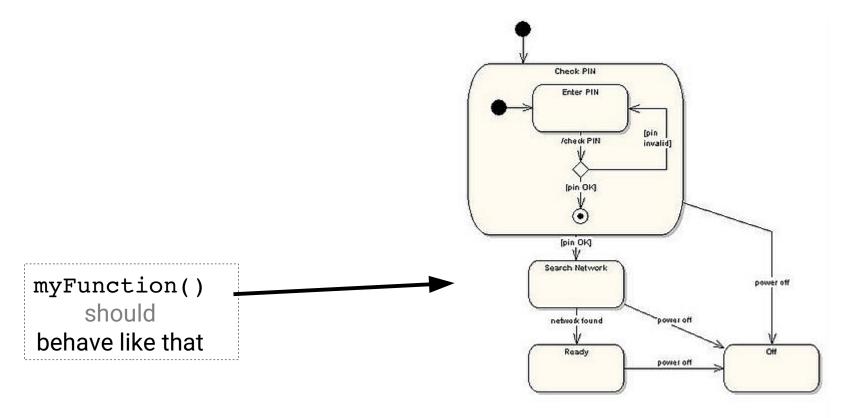
Be aware of previous comments!!







- UML can give much more (caricature)
 - Behavioral aspects
- Textual specification







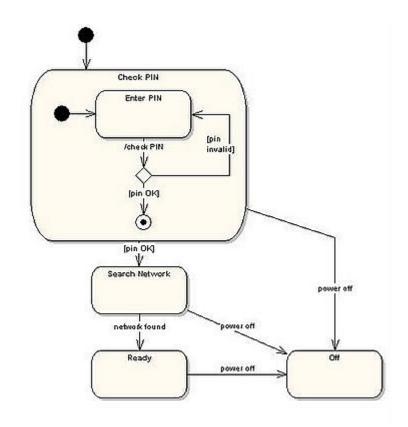


- UML can give much more (caricature)
 - Behavioral aspects
- Textual specification

Almost equivalent to a set of .cpp files:

Expression.cpp Unary.cpp UnaryPlus.cpp

should





- UML can give much more (caricature)
 - Behavioral aspects
 - •

Almost equivalent to how the main uses objects: system.h system.cpp main.cpp Leave
Comment

<< extends >>

Publish Weblog

Visitor

Author

Search Content

Google
or DB?

The system behave like that





• Implementing a system:

1. Classes realization

- Declaration / definition of the class(es) --> .h
 - ≃ reflects the class diagram
- Implementing the class(es) --> .cpp
 - = implements the (member) function(s) (activity diagram, stateCharts, sequence diagram, ...)

2. system realization

- Make use of class(es)
 - Predefined classes (for instance the STL ones)
 - The one previously defined (at step 1.)
 - Definition / implementation / uses of a « system » class







Implementing a system:

1. Classes realization

- Declaration / definition of the class(es) --> .h
 - ≃ reflects the class diagram
- Implementing the class(es) --> .cpp
 - = implements the (member) function(s) (activity diagram, stateCharts, sequence diagram, ...)

2. system realization

- Make use of class(es)
 - Predefined classes (for instance the STL ones)
 - The one previously defined (at step 1.)
 - Definition / implementation / uses of a « system » class







Implementing a system:

1. Classes realization

- Declaration / definition of the class(es) --> .h
 - ≃ reflects the class diagram
- Implementing the class(es) --> .cpp
 - = implements the (member) function(s) (activity diagram, stateCharts, sequence diagram, ...)

2. system realization

- Make use of class(es)
 - Predefined classes (for instance the STL ones)
 - The one previously defined (at step 1.)
 - Definition / implementation / uses of a « system » class







Implementing a system:

1. Classes realization

- Declaration / definition of the class(es) --> .h
 - □ ≃ reflects the class diagram
- Implementing the class(es) --> .cpp
 - = implements the (member) function(s) (activity diagram, stateCharts, sequence diagram, ...)

2. system realization

- Make use of class(es)
 - Predefined classes (for instance the STL ones)
 - The one previously defined (at step 1.)
 - Definition / implementation / uses of a « system » class

May be explicit in the model

