



Introduction to Eiffel

Martin Nordio

ETH Zurich

martin.nordio@inf.ethz.ch

**Distributed and Outsourced Software
Engineering - ETH course, Fall 2012**

Part 1: Language Constructs

- Basics: class definition, if then else, expressions, loops and across, creation procedures
- Inheritance: redefinition and multiple inheritance
- Exception Handling
- Once Routines
- Style rules
- Generics
- Agents and Tuples
- Information Hiding

Part 2: Contracts

- Preconditions, postconditions and class invariants
- Contracts in inheritance

Part 3: Libraries

- EiffelBase and Vision2

Part 4: EiffelStudio

- Editor, browsing, compiler and debugger

Version 7.1

The full version number is 7.1.8.8986

http://sourceforge.net/projects/eiffelstudio/files/EiffelStudio%207.1/Build_88986/

Make sure to have a look at all the resources listed in the wiki:

<https://github.com/DOSE-ETH/dose2012/wiki/Eiffel>

Part 1: Language constructs

1.1 BASICS



Class declaration: Eiffel vs Java:



```
class  
  ACCOUNT  
end
```

```
public class Account {  
  
}
```

Constructors



```
class
  ACCOUNT
create
  make,
  make_balance
feature
  make
    do ...
  end
  make_balance (i: INTEGER)
    do ...
  end

end
```

```
public class Account {
    public Account() {...}
    public Account (int b) {...}
}
```

Constructors

class

ACCOUNT

create

*make, make_balance,
make_name*

feature

make

do ...

end

make_balance (i: INTEGER)

do ...

end

make_name (s: STRING)

do ...

end

end

```
public class Account {  
    public Account() {...}  
    public Account (int b) {...}  
    public Account (string s) {...}  
}
```

Constructors can have any name; use the **create** clause to declare a routine as constructor



Overloading

class

PRINTER

feature

print_int (a_int: INTEGER)
do ... end

print_real (a_real: REAL)
do ... end

print_string (a_str: STRING)
do ... end

end

```
public class Printer {  
    public void print(int i) {...}  
    public void print(float f) {...}  
    public void print(String s) {...}  
}
```

Eiffel does not
support
overloading!



Creating Objects



class

BANK

feature

pay_bill

local

b1: ACCOUNT

do

create *b1.make*

end

end

```
public class Bank {  
    public void payBill() {  
        Account b1 = new Account();  
    }  
}
```

Creating Objects



```
class
  BANK

feature
  pay_bill
    local
      b1, b2: ACCOUNT
    do
      create b1.make
      create b2.make_balance (2)
    end
end
```

```
public class Bank {
  public void payBill() {
    Account b1 = new Account();
    Account b2 = new Account (2);
  }
}
```

Create objects using the **create** keyword; declare the local variables in the **local** clause



Creating Objects: default create

```
class
  MAIN

feature
  root
  local
    b1: BANK
  do
    create b1
    -- corresponds to
    -- create b1.default_create
    b1.pay_bill
  end

end
```

```
class
  BANK

feature
  pay_bill
  do
    ...
  end

end
```



All classes inherit from ANY (Object in Java). If no creation procedure is specified, *default_create* is used (inherited from ANY)

Creating Objects: default create



```
class
  BANK
inherit
  ANY
  redefine
    default_create
end
```

```
create
  default_create
```

```
feature
  ...
end
```

*The routine default_create
can be redefined*



Features

```
class
  ACCOUNT
feature -- Initialization
  make do ... end
  make_balance (i: INTEGER)
    do ... end
  make_name (s: STRING)
    do ... end

feature -- Basic operations
  deposit (i: INTEGER) do ... end
  withdraw (i: INTEGER) do ... end
  transfer (b: ACCOUNT) do ... end

feature -- Access
  balance: INTEGER do ... end
end
```

```
public class Account {
  public Account() {...}
  public Account (int b) {...}
  public Account (string s) {...}
  public void deposit (int i) {...}
  public void withdraw (int i) {...}
  public void transfer(Account b) .
  public int balance() {...}
}
```

The **feature** clause is used to group routines and for information hiding (see 1.8)



Expressions and Conditionals



```
feature
  foo
  do
    if b and (c or d) then
      x := 5
      ...
    end
  end
end
```

```
foo
do
  if b and then (c or else d) then
    ...
  end
end
end
```

```
public foo() {
  if (b & (c | d)) {
    x = 5;
    ...
  }
}
```

```
public foo() {
  if (b && (c || d)) {
    ...
  }
}
```

Return and breaks



```
class  
  B
```

```
feature
```

```
  foo: INTEGER
```

```
  do
```

```
    Result := 5
```

```
  end
```

```
end
```

```
public class B {  
  public int foo() {  
    return 5;  
  }  
}
```

Eiffel does not support neither breaks, continues nor return



Loops



```
print  
  local  
    i: INTEGER  
  do  
    from  
      i := 1  
    until  
      i >= 10  
    loop  
      ...  
      i := i + 1  
    end  
end
```

```
public class Printer {  
    public void print() {  
        for(int i=1;i<10;i++) {  
            ...  
        }  
    }  
}
```


Loops: Example 2



```
print  
local  
  i: INTEGER  
do  
  from  
    i := 1  
  until  
    i >= 10  
  loop  
    i := i + 1  
  end  
end
```

```
public class Printer {  
    public void print() {  
        int i=1;  
        while(i<10) {  
            i++;  
        }  
    }  
}
```

Loops: Traversing a list



```
print_1  
do  
  from list.start  
  until list.after  
  loop  
    list.item.print  
    list.forth  
  end  
end
```

```
print_2  
do  
  across list as e loop  
    e.item.print  
  end  
end
```

```
public class Printer {  
    public void print() {  
        for(Element e: list) {  
            e.print();  
        }  
    }  
}
```

Eiffel:

BOOLEAN

CHARACTER

INTEGER

INTEGER_64

REAL

DOUBLE

STRING

Java:

boolean

char, byte

short, int

long

float

double

String