Advanced Material

The following slides contain advanced material and are optional.
Outline

- Syntax comparison: Eiffel vs Java
- Naming in Eiffel
- Feature comments: Less is better (sometimes...)
Eiffel vs Java: Class declaration

class ACCOUNT
end

class Account {
}

Eiffel vs Java: inheritance

class ACCOUNT
inheriti
    ANY
end

public class Account
    extends Object {
    
}

Eiffel vs Java: feature redefinition

class
  ACCOUNT
inherit
  ANY
  redefine out end

feature
  out: STRING
  do
    Result := "abc"
  end
end

class Account extends Object {

  String toString() {
    return "abc";
  }
}

Eiffel vs Java: Precursor vs. super call

class ACCOUNT
inhibit
  ANY
    redefine out end

OTHER_PARENT
  redefine out end

feature
  out: STRING
    do
      Result := Precursor
    end
end

class Account extends Object {
  String toString() {
    return super();
  }
}

deferred class ACCOUNT

feature
  deposit (a_num: INT)
    deferred
  end
end

abstract class Account {
  abstract void deposit(int a);
}

Eiffel vs Java: deferred vs. abstract
Eiffel vs Java: genericity vs. generics

class OBJECT_QUERY [G]

feature
  result_cursor: RESULT_SET [G]

end

class ObjectQuery <E> {
  ResultSet<E> resultCursor;
}

Eiffel vs Java: frozen vs. final

```plaintext
frozen class ACCOUNT
inherit ANY
end

class ACCOUNT
feature
  frozen deposit (i: INTEGER)
    do end
end

final class Account
extends Object {
}

class Account {
  final void deposit(final int i) {}
}
```
Eiffel vs Java: expanded vs. primitive types

expanded class ACCOUNT
end

int, float, double, char
Eiffel vs Java: creation features vs. constructors

```java
public class Account {
    public Account() {}
}
```

```eiffel
class ACCOUNT
    create 
        make
    feature
        make
            do 
            end 
    end
end
```
Eiffel vs Java: constructor overloading

class ACCOUNT
create
    make, make_amount

feature
    make
        do end

    make_amount (a_amount: INT)
        do end

end

public class Account {
    public Account() {}
    public Account(int a) {}
}
Eiffel vs Java: method 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 print(int i) {}
  public print(float f) {}
  public print(String s) {}
}

class PRINTER

feature

print_int (a_int: INTEGER)
local
  l_retried: BOOLEAN

do
  if not l_retried then
    (create {DEVELOPER_EXCEPTION}).raise
  else
    -- Do something (e.g. continue)
  end
rescue
  l_retried := True
  -- Fix object state
retry
end
end
public class Printer {
    public print(int i) {
        try {
            throw new Exception()
        }
        catch(Exception e) {  //handle exception  }

        finally {  //clean-up  }
    }
}
Eiffel vs Java: Conditional

class PRINTER

feature print do
  if True then
    ...
  else
    ...
  end
end

public class Printer {
  public print() {
    if (true) {
      ...
    }
    else {
      ...
    }
  }
}
Eiffel vs Java: Assignment and equality

class PRINTERS

feature
  print (j: detachable JOB)
  do
    if j = Void then
      ...
    else
      count := j.num_pages
    end
  end
end

public class Printer {
  public print(Job j) {
    if (j == null) {
      ...
    } else {
      count = j.num_pages;
    }
  }
}
Eiffel vs Java: Loop 1

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

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

Eiffel vs Java: Loop 2

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

class Printer {
   public print() {
      int i=1;
      while(i<10) {
         i++;
      }
   }
}
Eiffel vs Java: Loop 3

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

print_2
  do
    across list as ic loop
      ic.item.print
    end
  end

public class Printer {
  public print() {
    for(Element e: list) {
      e.print();
    }
  }
}
Eiffel Naming: Classes

- Full words, no abbreviations (with some exceptions)
- Classes have global namespace
  - Name clashes may arise
- Usually, classes are prefixed with a library prefix
  - Traffic: TRAFFIC_
  - EiffelVision2: EV_
  - EiffelBase2: V_ (stands for verified)
- Base is not prefixed
Eiffel Naming: Features

- Full words, no abbreviations (with some exceptions)

- Features have namespace per class hierarchy
  - Introducing features in parent classes can cause clashes with features from descendants
  - Not possible to hide feature or introduce hidden feature. No `private` like in Java.
Eiffel Naming: Locals / Arguments

- Locals and arguments share namespace with features
  - Name clashes arise when a feature is introduced, which has the same name as a local (even in parent)

- To prevent name clashes:
  - Locals are prefixed with `l_`
  - Some exceptions like “i” exist (max 3 letters)
  - Arguments are prefixed with `a_`
tangent_ from (a_point: POINT): LINE
  -- Return the tangent line to the current circle
  -- going through the point `a_point', if the point
  -- is outside of the current circle.

require
  outside_circle: not has (a_point)
tangent_from (a_point : POINT): LINE
  -- The tangent line to the current circle
  -- going through the point `a_point`, if the point
  -- is outside of the current circle.

require
  outside_circle: not has (a_point)
tangent_ from (a_point : POINT): LINE
  -- Tangent line to current circle from point `a_point`
  -- if the point is outside of the current circle.

require
  outside_circle: not has (a_point)
**tangent_ from (a_point : POINT): LINE**

-- Tangent line to current circle from point `a_point`.

**require**

outside_circle: not has (a_point)
tangent_from (a_point : POINT): LINE
   -- Tangent from `a_point`.

require
   outside_circle: not has (a_point)
tangent_from (a_point : POINT): LINE
    -- Tangent from `a_point`.
    --
    -- `a_point`: The point from ...
    -- `Result`: The tangent line ...
    --
    -- The tangent is calculated using the
    -- following algorithm:
    -- ...

require
    outside_circle: not has (a_point)
tangent_from (a_point : POINT): LINE
   -- <Precursor>

require
   outside_circle: not has (a_point)