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
inherit
    ANY
end
```

```
public class Account
    extends Object {


}
```

Eiffel vs Java: feature redefinition

```eiffel
class
    ACCOUNT
inherit
    ANY
        redefine out end

feature

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

```java
public class Account
    extends Object {

    String toString() {
        return "abc";
    }

}
```

Eiffel vs Java: Precursor vs. super call

```
class
     ACCOUNT
inherit
     ANY
          redefine out end

feature

  out: STRING
     do
     Result :=
          Precursor {ANY}
     end
end
```

```
public class Account
    extends Object {

    String toString() {
         return super();
    }

}
```

Eiffel vs Java: deferred vs. abstract

```
deferred class
    ACCOUNT

feature
    deposit (a_num: INT)
        deferred
        end
end
```

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

Eiffel vs Java: frozen vs. final

frozen class
    ACCOUNT
inherit
    ANY

end

```
final class Account
    extends Object {

}
```

Eiffel vs Java: expanded vs. primitive types

expanded class
     ACCOUNT
end

int, float, double, char

Eiffel vs Java: creation features vs. constructors

```eiffel
class
    ACCOUNT
create
    make

feature
    make
        do
        end

end
```

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

Eiffel vs Java: constructor overloading

```eiffel
class
    ACCOUNT
create
    make, make_amount

feature
    make
        do end

    make_amount (a_amount: INT)
        do end

end
```

```java
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) {}
}
```

Eiffel: Exception Handling

```
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 alternate.
            end
        rescue
            l_retried := True
            retry
        end
end
```

Java: Exception Handling

```
public class Printer {
    public print(int i) {
        try {
            throw new Exception()
        }
    catch(Exception e) {   }
    }
}
```

Eiffel vs Java: Conditional

```
class
    PRINTER

feature
    print
        do
            if True then
                …
            else
                …
            end
        end
end
```

```
public class Printer {
    public print() {
        if (true) {
            …
        }
        else {
            …
        }
    }
}
```

```
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++) {
            ...
        }
    }
}
```

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

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

Eiffel vs Java: Loop 3

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


print_2
    do
        -- Enable "provisional syntax" in
    -- the project properies to
        -- use "across"
        across  list  as  e  loop
            e.item.print
        end
    end
```

```java
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_
  ➢ 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

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
  ➢ Arguments are prefixed with **a_**

Feature comments: Version 1

```
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)
```

Example from http://dev.eiffel.com/Style_Guidelines

```
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)
```

Feature comments: Version 3

```
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)
```

Feature comments: Version 4

```
tangent_ from (a_point : POINT): LINE
        -- Tangent line to current circle from point `a_point'.
    require
        outside_circle: not has (a_point)
```

Feature comments: Final version

```
tangent_ from (a_point : POINT): LINE
        -- Tangent from `a_point'.
    require
        outside_circle: not has (a_point)
```

Feature comments: More information

```
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)
```

Feature comments: Inherited comments

```
tangent_ from (a_point : POINT): LINE
        -- <Precursor>
    require
        outside_circle: not has (a_point)
```