



The following slides contain advanced material and are optional.



- Constants and global variables
- Constants in OO programming
- Once routines
 - Definition
 - Use
 - Sharing objects
 - Arguments and contracts

Constants and global variables



➤ Constants for basic types are easy

```
class CONSTANTS
```

```
  Pi: Real = 3.1415926524
```

```
  Ok: Boolean = True
```

```
  Message: STRING = "abc"
```

```
end
```

```
class APPLICATION
```

```
inherit CONSTANTS
```

```
feature
```

```
  foo do print (Pi) end
```

```
end
```



- What about user defined types?

```
class CONSTANTS
```

```
  i: COMPLEX = ???
```

```
  Hans: PERSON = ???
```

```
  Paris: MAP = ???
```

```
end
```

- In other languages
 - Static variables
 - Singleton pattern
- In Eiffel
 - Once routines

What are once routines?



- Executed the first time
- Result is stored
- In further calls, stored result is returned

```
foo: INTEGER
```

```
once
```

```
Result := factorial (10)
```

```
end
```

```
test_foo
```

```
do
```

```
io.put_integer (foo) -- 3628800, calculated
```

```
io.put_integer (foo) -- 3628800, directly returned
```

```
end
```

Once for whom?



- By default, computation is once per class hierarchy
 - Result is shared among all objects of a class and its subclasses
- Once routines can take a special flag
- This flag is used to indicate that execution is e.g. one of
 - Once per object
 - Once per thread

Use of once routines



- Constants, other than basic types

i: COMPLEX

once create Result.make (0, 1) end

- Lazy initialization

settings: SETTINGS

once create Result.load_from_filesystem end

- Initialization procedures

init_graphics_system

once ... end

- Sharing of objects (see next)

Sharing objects I



- You can share objects
- Can be used to achieve effect of global/static variables

- How?
 - Once routine returning a reference
 - Will always return the same reference
 - Create a **SHARED_X** class and inherit from it

Sharing objects II



```
class SHARED_X
  the_one_and_only_x: attached X
  once
    create Result.make
  end
end
```

```
class X
  create {SHARED_X}
  make
  feature {NONE}
  make
  do
  end
end
```

Pitfalls of once and constants



- No guarantee that only one instance will be created
 - Inheriting classes can also call creation routine
- Problems can arise when once references are shared with external C code due to the garbage collector
- Strings are not expanded!

```
message: STRING = "abc"  
foo  
do  
  message.append("def")  
  -- from now, "message" will be "abcdef"  
end
```

Arguments and contracts



```
foo (i: INTEGER): INTEGER
  require
    i > 0
  once
    Result := i * 2
  ensure
    Result = i * 2
  end
```

➤ What is the output of the following code block

```
do
  io.put_integer (foo (2)) -- 4
  io.put_integer (foo (3)) -- postcondition violation
  io.put_integer (foo (-2)) -- precondition violation
end
```

Don't write once functions taking arguments.
Don't write complex postconditions in once functions.



8.23.26 – Semantics: General Call Semantics

The effect of an Object_call of feature sf is, in the absence of any exception, the effect of the following sequence of steps:

1. Determine the target object O through the applicable definition.
2. Attach Current to O.
3. Determine the dynamic feature df of the call through the applicable definition.
4. For every actual argument a, if any, in the order listed: obtain the value v of a; then if the type of a converts to the type of the corresponding formal in sf, replace v by the result of the applicable conversion. Let arg_values be the resulting sequence of all such v.
5. Attach every formal argument of df to the corresponding element of arg_values by applying the Reattachment Semantics rule.
6. **If the call is qualified and class invariant monitoring is on, evaluate the class invariant of O's base type on O.**
7. **If precondition monitoring is on, evaluate the precondition of df .**
8. If df is not an attribute, not a once routine and not external, apply Non-Once Routine Execution Semantics to O and df .
9. **If df is a once routine, apply the Once Routine Execution Semantics to O and df.**
10. If df is an external routine, execute that routine on the actual arguments given, if any, according to the rules of the language in which it is written.
11. **If the call is qualified and class invariant monitoring is on, evaluate the class invariant of O's base type on O.**
12. **If postcondition monitoring is on, evaluate the postcondition of df.**