Object Definition Language

Short introduction

Arūnas Janeliūnas

Object Databases

Types

```
<type> ::= <atomic_type> |
           <class_name> |
           <collection_type> |
           <tuple_type> |
           <enumerative_type>
```

```
<atomic_type> ::= long | short | unsigned long |
                  unsigned short | float | double |
                  boolean | octet | char | string |
                  date | interval | time | timestamp |
                  void
```

```
<class_name> ::= <name>
```

Collection types

```
<collection_type> ::= <collection_constructor> < <type> > |
                      <array>
<collection_constructor> ::= set | bag | list
<array> ::= <type> [<size>][{, [<size>]}]
```

Examples

```
set <Person>

octet [3][3];

char [256]
```

Tuple types

```
<tuple_type> ::= struct <name>
                 {
                   <type> <name>
                   [{; <type> <name>}]
                 }
```

Examples

```
struct Date {
  octet day;
  octet month;
  unsigned short year;
};

struct Student {
  string name;
  string surname;
  short grades [10];
}
```

Enumerative types

```
<enumeratgive_type> ::= enum <name>
                        {
                          <name>
                          [{, <name>}]
                        }
```

Examples

```
enum Colours {
  Red, Green, Blue, Yellow, Black, White, Green, Purple, NonDescriptive
};

enum WorkingDays {
  Monday, Tuesday, Wednesday, Thursday, Friday
}
```

Type definition

```
<type_definition> ::= typedef <type> <name>
```

Examples

```
typedef char[256] Stack

typedef unsigned short SimpleNumber
```

Classes

```
<class> ::= interface <class_name>
                    [: <superclass> [{, <superclass>}]]
            {
              <property> {;<property>}
            };
<class_name> ::= <name>
<superclass> ::= <name>
<property> ::= <attribute> |
               <association> |
               <method>
```

Attributes

```
<attribute> ::= attribute <type> <name>
```

Examples

```
interface Employee : Person
{
  attribute string name;
  attribute string surname;
  attribute struct TypeAddress
                    { string city;
                      string street;
                      short house;
                      short flat;
                    } address;
  attribute Person[5] children;
}
```

Associations

```
<association> ::= relationship <type> <name>
                        [inverse <class_name> :: <association_name>]
<class_name> ::= <name>
<association_name> ::= <name>
```

Examples

```
interface Person {
   relationship Flat lives_in inverse Flat::resident;
};

interface Flat {
   relationship Person resident inverse Person::lives_in;
};

interface person {
   relationship Set<Person> parents
                            inverse Person::children;
   relationship List<Person> children
                            inverse Person::parents;
};
```

Methods

<method> ::= <type> <method_name> (<argument> {, <argument>})

<method_name> ::= <name>

<argument> ::= <argument_qualifier> <type> <name>

<argument_qualifier> ::= **in** | **out** | **inout**

Examples

```
interface Person {
   attribute String name;
   attribute String surname;
   attribute Person spouse;

   void mariage ( in Person whomToMarry );
}
```