

CHAPTER 16

The Object Constraint Language

The Object Constraint Language (OCL) is a language for expressing constraints on model elements. OCL expressions specify rules in the form of conditions and restrictions attached to model elements. This includes expressions specifying invariants or constraints attached to model elements, preconditions and postconditions attached to operations and methods, guard conditions, and navigation between model elements.

Expressions

Expressions are strings that evaluate to a value of a particular type. Expressions

- May be attached to model elements. These expressions specify conditions that must be maintained as true for all instances of the model elements.
- May be attached to operations. These expressions specify conditions that must hold true for the invocation of an operation (preconditions), or conditions that must hold true after an operation finishes (postconditions).
- May specify guard conditions attached to model elements. These expressions specify conditions that are attached to trigger messages, loops, and transitions.
- Are expressed in the context of an instance of a specific type. The "self" keyword
  - Refers to the instance to which the expression is attached. This instance is called the *contextual instance* since it provides the context in which an expression is evaluated. The type of the "self" keyword is the associated class to which the expression is attached, or the class enclosing or which owns the operation or guard condition.
  - Must start with a lowercase character.
  - May be omitted in an expression.
- Are evaluated from left to right. Each part (subexpression) of the overall expression results in a specific value or object of a specific type. The object

may be a single item or a collection of items. Based on the type or class of the object, a feature (attribute and operation) may be applied to the object to derive the next value or object. Associations of the object may also be used, or navigated, to reference associated objects and association objects. The evaluation of an overall expression starts with the "self" keyword to derive an object and continues with the application of features on intermediate objects from left to right until the complete expression is evaluated.

- May utilize basic values and types. Basic types are predefined by the OCL. Basic values are values of predefined basic types. These types include Boolean, Integer, Real, and String. Basic values evaluate to a single item.
- May utilize collection values and types. Collection types are predefined by the OCL. Collection values are values of predefined collection types. These types include Collection, Set, Bag, and Sequence. Collection values evaluate to a single item that is a collection containing items.
- Are evaluated in accordance with the following precedence order (from highest to lowest):
  - Dot (".") and arrow ("->")
  - Unary "not" and unary minus ("-")
  - Multiplication ("*") and division ("/")
  - Addition ("+") and subtraction ("-")
  - Logical "and," "or," and "xor"
  - Logical "implies"
  - Logical "if then else endif"
  - Logical comparison operators "<", ">", "<=", ">=", and "="

  Parentheses ("( )") may be used to change the order in which an expression is evaluated, where expressions are evaluated from innermost to outermost parentheses.
- May include comments. These are denoted using two dashes. Everything after the two dashes up to and including the end of line is a comment.
- May refer to model classes and their features, and associations and their features. All classes within a model are types within OCL expressions attached to the model elements. Features (attributes and operations) of objects are called *properties* within the OCL. Operations used within these expressions must be stereotyped using the "query" keyword; that is, operations must not have any side effects (or modify their containing class or association class). For postconditions, the value of a property represents the value it has upon completion of the operation. Within postconditions, properties may be postfixed with the at sign followed by the keyword "pre" ("@pre") to reference the value of the property at the start of an operation.
- May reference queries that are undefined. In such a case, the whole expression is undefined. However, subexpressions adhere to the following rules:
  - The value true ORed with any other value results in true.
  - The value false ANDed with any other value results in false.

Object Properties

Object properties are features of model elements defined in class diagrams.

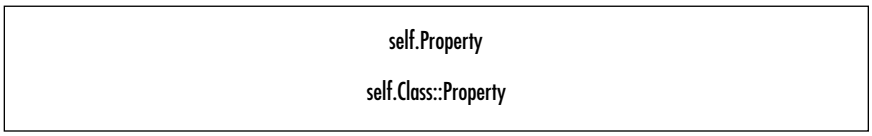


Figure 16-1: Object Properties

Object properties

- May be referenced using the "self" keyword (Figure 16-1). This is expressed using the "self" keyword followed by a dot and the property name or string. The expression results in the value of the property.
- May reference properties using pathnames. This is used to resolve ambiguity when two properties have the same name within the same object. For example, one class may inherit a property with the same name as an existing property. Selecting one of the properties is expressed by specifying the class or superclass name followed by two colons (":") and the property name.

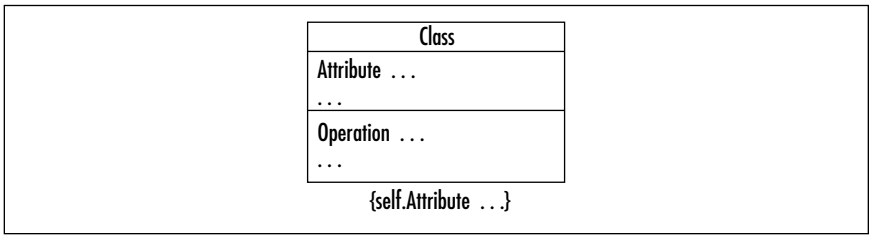


Figure 16-2: Attribute

- May reference attributes (Figure 16-2). This is expressed by specifying the property string as the attribute name. The result of the expression is the value of the attribute, and the type of the result is the type of the attribute.

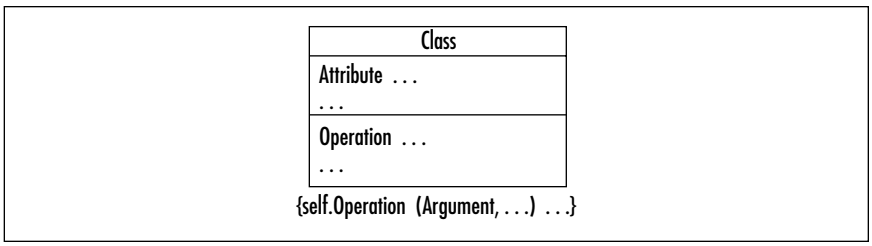


Figure 16-3: Operation

- May reference operations (Figure 16-3). This is expressed by specifying the property string as the name of the operation followed by parentheses containing an argument list consisting of a comma-separated list of arguments passed to the method. The result of the expression is the value returned from the operation, and the type of the result is the return type of the operation. The

"result" keyword may be used to reference the result of the operation. The arguments must match the order, number, and types specified in the formal parameter list of the operation.

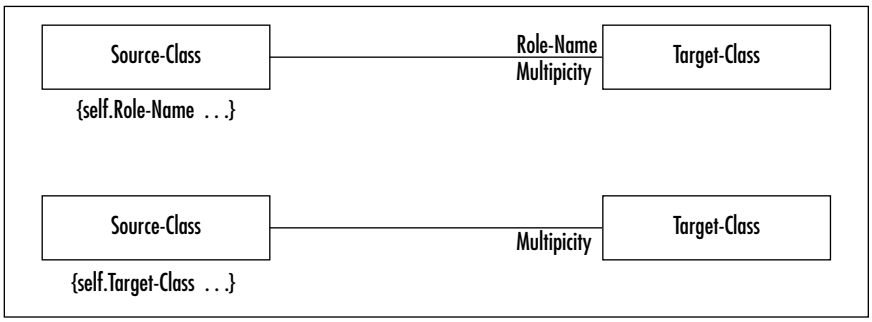


Figure 16-4: Associated Classes

- May be used to navigate associations to reference other target classes and their properties (Figure 16-4). This is expressed by specifying the property string as the role name or class of the target element.
  - If the multiplicity of the association end corresponding to the role name has a maximum multiplicity value of 1, the result of the expression is the associated object, and the type of the result is the target class.
  - If the multiplicity of the association end corresponding to the role name has a maximum multiplicity value of more than 1 and is not adorned with the "ordered" keyword constraint, the result of the expression is the set of associated target objects attached to the role name that are associated with the source objects, and the type of the result is a Set of the target class. This type is denoted as "Set (Target-Class)."
  - If the multiplicity of the association end corresponding to the role name has a maximum multiplicity value of more than 1 and is adorned with the "ordered" keyword constraint, the result of the expression is the set of associated target objects attached to the role name that are associated with the source objects, and the type of the result is a Sequence of the target class. This type is denoted as "Sequence (Target-Class)."
  - If the role name is omitted, the name of the class at the association end, starting with a lowercase character, is used as the role name. Fundamentally, the same target class name is utilized; however, the first character is converted to lowercase.

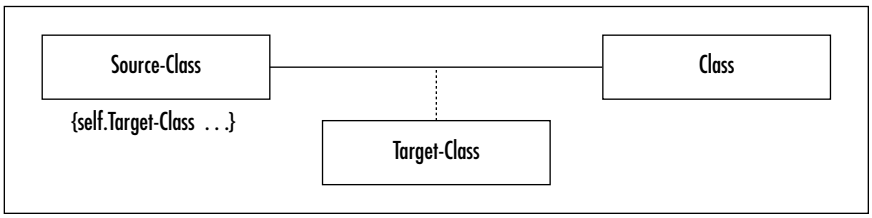


Figure 16-5: Association Classes

- May be used to navigate or reference association classes (Figure 16-5). This is expressed by specifying the property string as the name of the association class starting with a lowercase character. The result of the expression is the set of association objects that associate the source objects, and the type of the result is a Set of the target class.

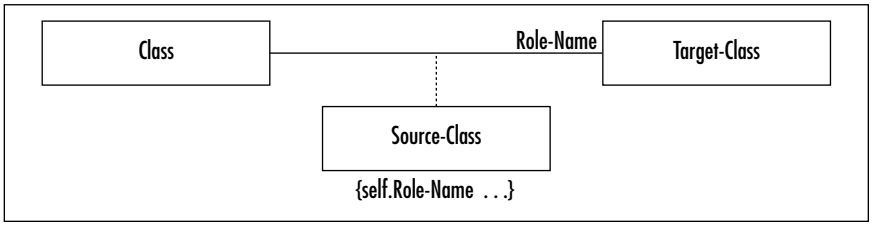


Figure 16-6: Associated Classes

- May be used to navigate from association classes to reference the classes that participate in the association (Figure 16-6). This is expressed by specifying the property string as the role name of the target element. If the role name is omitted, the name of the class at the association end, starting with a lowercase character, is used as the role name. The result of the expression is the associated object, and the type of the result is the target class.

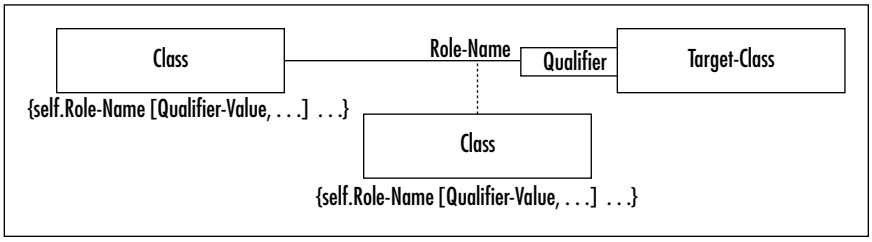


Figure 16-7: Qualifiers

- May be used to navigate through qualified associations (Figure 16-7). This is expressed by specifying the property string as the role name of the target element followed by a comma-separated list of qualifier values enclosed in square braces. The qualifier values and square braces may be omitted, in which case the results will be all objects on the association end.

Collections

Collections are groups of objects or elements that may result during the evaluation of an expression.



Figure 16-8: Collection Property

Collections

- Must have their properties referenced by using an arrow "->" followed by the name of the property (Figure 16-8). The arrow may also be applied to a single item, in which case the single item behaves as if it were a collection containing one item.
- May be Sets. Sets are groups of objects that do not contain duplicate elements and are unordered.
- May be Bags. Bags are groups of objects that may contain duplicate elements and are unordered.
- May be Sequences. Sequences are groups of objects that may contain duplicate elements and are ordered.

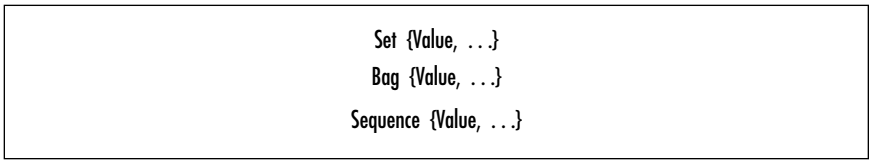


Figure 16-9: Set, Bag, and Sequence Literals

- May be specified by a literal (Figure 16-9). This is expressed by specifying the type of collection followed by curly braces containing a comma-separated list of values. When Integers are specified, a multiplicity expression may be used (Chapter 6).
- Must not be nested. That is, all collections of collections are automatically flattened so that the elements of both collections are combined into a single collection.



Figure 16-10: Iteration Expression

- May be projected from existing collections (Figure 16-10). That is, for each element in an existing collection, an iteration expression is evaluated. Based on the result of the evaluation, the element may be included in the result collection. This applies to the "select," "reject," "collect," "forAll," and "exists" features of collections. Iteration expressions
  - May have an iterator variable name or identifier string that is enumerated over all the elements in the collection.
  - May have a colon followed by a type expression that indicates the type of the iterator variable. If no iterator variable is specified, no type expression is used and the colon is omitted.
  - May have a comma-separated list of iterator variables and types. This enables the iteration expression to utilize multiple iterator variables.
  - Must have an iteration expression that indicates the expression to evaluate for each element in the collection. If an iterator variable name is

used, a vertical bar precedes the iteration expression, and the expression may utilize the iterator variable.

– For the "select" feature, the iteration expression specifies a Boolean expression that must evaluate to true for an element in order for the element to be selected into the result collection.

– For the "reject" feature, the iteration expression specifies a Boolean expression that must evaluate to false for an element in order for the element to be selected into the result collection.

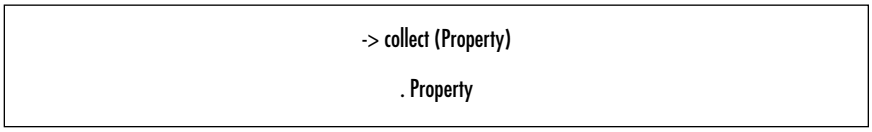


Figure 16-11: "Collect" Shorthand Notation

– For the "collect" feature, the result is the collection of the results of all the evaluations of the iteration expression over elements in the original collection. If the expression is simply a property, a shorthand notation may be used, in which the "collect" feature is suppressed and only the property name is specified (Figure 16-11). In general, when a property is applied to a collection, it will automatically be interpreted as a "collect" over the members of the collection with the specified property.

– For the "forAll" feature, the iteration expression specifies a Boolean expression that must evaluate to true for all the elements in the collection in order for the expression to result in a value of true. If the Boolean expression is false for one or more of the elements in the collection, the complete expression evaluates to false.

– For the "exists" feature, the iteration expression specifies a Boolean expression that must evaluate to true for at least one element in the collection in order for the expression to result in a value of true. If the Boolean expression is false for all the elements in the collection, the complete expression evaluates to false.

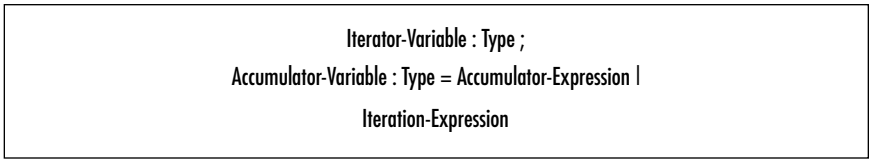


Figure 16-12: Iterate Expressions

- May be accumulated from existing collections (Figure 16-12). That is, for each element in an existing collection, an iteration expression is evaluated. The result of the evaluation is accumulated in another collection. This scheme of evaluating the expression is used by the "iterate" feature of collections. This is a very generic operation that may be used to define the "select," "reject," "collect," "forAll," and "exists" features of collections. Iterate expressions

  – Must have an iterator variable name or identifier string that is enumerated over all the elements in the collection.

– Must have a colon followed by a type expression that indicates the type of the iterator variable.
– Must have a semicolon followed by an accumulator variable name or identifier string that specifies a collection in which the result will be accumulated.
– Must have a colon followed by a type expression that indicates the type of the accumulator variable.
– Must have an equal sign followed by an expression that specifies the initial value for the accumulator variable.
– Must have a vertical bar followed by an iteration expression that indicates the expression to evaluate for each element in the collection. Within this expression, the results are accumulated or added to the accumulator variable.

Standard Types

The OCL defines a set of basic types (Boolean, Integer, Real, and String) and collection types (Collection, Set, Bag, and Sequence) that are related and supplemented with other fundamental OCL types (Figure 16-13). These types are organized in a type hierarchy adhering to the following rules:

- All types defined in a model or within the OCL have a type that is an instance of the OCL type OclType.
- The OclAny type is the supertype of all types in a model.
- The OclExpression type is the type of all OCL expressions.
- The Real, String, Boolean, Enumeration, and Collection types are subtypes of the OclAny type.
- The Integer type is a subtype of the Real type.
- The Set, Bag, and Sequence types are subtypes of the Collection type.
- Each instance of a given type may be substituted for an instance of its supertypes. This is a transitive rule. That is, if T1 is a supertype of T2, and T2 is a supertype of T3, an instance of T3 may be substituted for an instance of T1 or T2. Collection instances may be substituted in accordance with this rule if they are collections of element types that adhere to the rule.

The features of these types are depicted using the standard UML notation and are described in the following sections. Operations of these types are declared using parentheses for their parameters as the standard UML notation specifies, but OCL operations that are depicted as operators do not use parentheses to enclose their arguments when used in expressions. That is, the declaration "I + (I2 : Integer) : Integer" is actually used in the expression "I + I2," and not "I + (I2)," where "I2" is of type Integer and the result is of type Integer. Those operations that do use parentheses when used in an expression are indicated.

OclType Type

The OclType type is the type of all the OCL predefined types and the types defined in a model. The OclType allows access to the meta-level of the model.

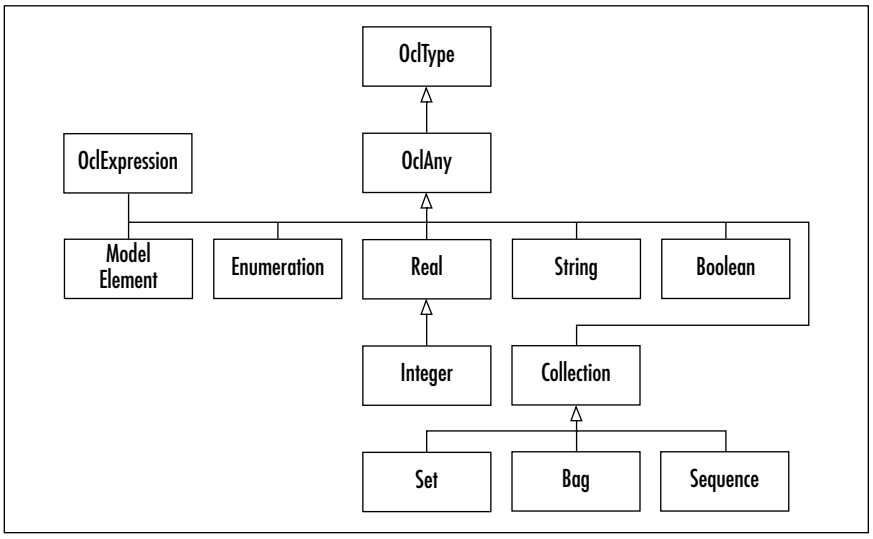


Figure 16-13: Standard OCL Types

The features of the OclType type are depicted in Figure 16-14 and described in Table 16-1, where *T* represents the instance on which the feature is applied.

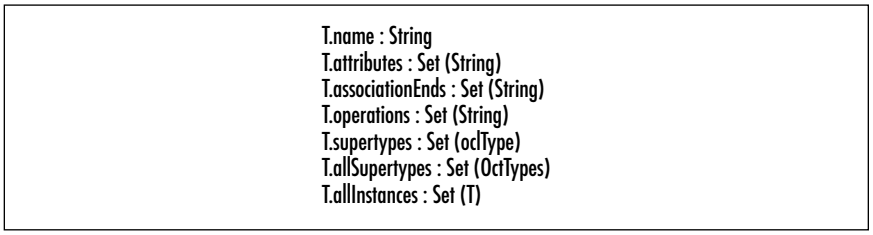


Figure 16-14: OclType Type Features

Table 16-1: OclType Type Features

Feature	*Description*
allInstances	Returns a set of all instances of T and all its subtypes.
allSupertypes	Returns a set of all supertypes of T.
associationEnds	Returns a set of names of the navigable association ends of T.
attributes	Returns a set of names of the attributes of T.
name	Returns the name of T.
operations	Returns a set of names of the operations of T.
supertypes	Returns a set of all direct supertypes of T.

OclAny Type

The OclAny type is the supertype of all types in a model. All classes in a model inherit the features defined by the OclAny type. The features of the OclAny type

are depicted in Figure 16-15 and described in Table 16-2, where *O* represents the instance on which the feature is applied.

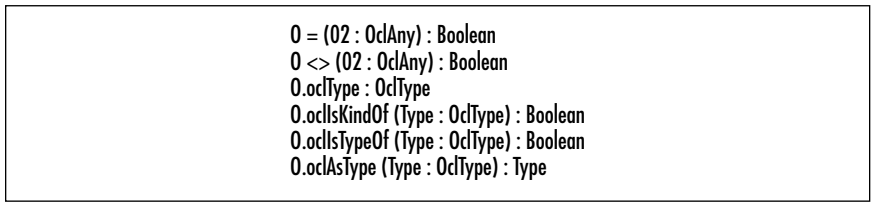


Figure 16-15: OclAny Type Features

Table 16-2: OclAny Type Features

Feature	*Description*
=	Returns true if O is the same object as O2.
<>	Returns true if O is not the same object as O2.
oclAsType	Returns O, but of the specified type. The result is undefined if the type of O is not the specified type or one of its subtypes.
oclIsKindOf	Returns true if the specified type is a supertype of the type of O.
oclIsTypeOf	Returns true if the specified type is the same as the type of O.
oclType	Returns the type of O.

OclExpression Type

The OclExpression type is the type of all OCL expressions. This type only has one feature: "E.evaluationType : OclType," where *E* represents the instance on which the feature is applied, and the feature returns the type of the object that results from evaluating E.

Real Type

The Real type represents the mathematical concept of real numbers. These are numbers with fractional parts expressed following a decimal point. The features of the Real type are depicted in Figure 16-16 and described in Table 16-3, where *R* represents the instance on which the feature is applied.

Integer Type

The Integer type represents the mathematical concept of integer numbers. The features of the Integer type are depicted in Figure 16-17 and described in Table 16-4, where *I* represents the instance on which the feature is applied.

String Type

The String type represents a sequence of ASCII characters. As a literal, a string is enclosed in single quotes. The features of the String type are depicted in Figure

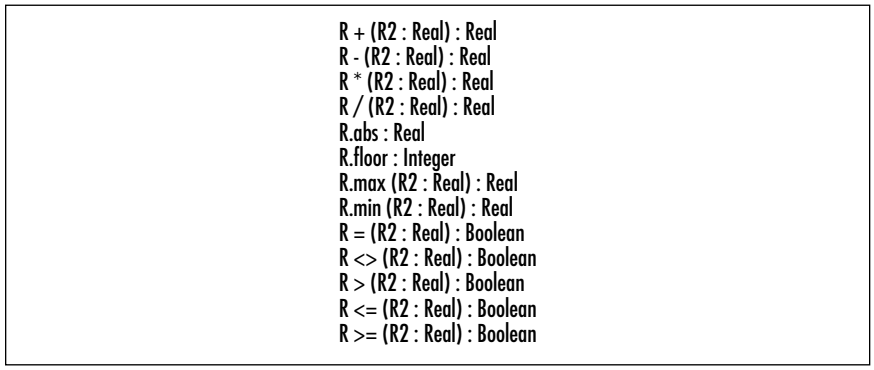


Figure 16-16: Real Type Features

Table 16-3: Real Type Features

Feature	*Description*
+	Returns the sum of R and R2.
-	Returns the value of subtracting R2 from R.
*	Returns the value of multiplying R and R2.
/	Returns the value of dividing R by R2.
abs	Returns the absolute value of R. If the value is negative, its positive counterpart is returned.
floor	Returns the largest integer that is less than or equal to R.
max	Returns the maximum of R and R2.
min	Returns the minimum of R and R2.
=	Returns true if R is equal to R2.
<>	Returns true if R is not equal to R2.
<	Returns true if R is less than R2.
>	Returns true if R is greater than R2.
<=	Returns true if R is less than or equal to R2.
>=	Returns true if R is greater than or equal to R2.

16-18 and described in Table 16-5, where *S* represents the instance on which the feature is applied.

Boolean Type

The Boolean type represents the values true and false. The features of the Boolean type are depicted in Figure 16-19 and described in Table 16-6, where *B* represents the instance on which the feature is applied.

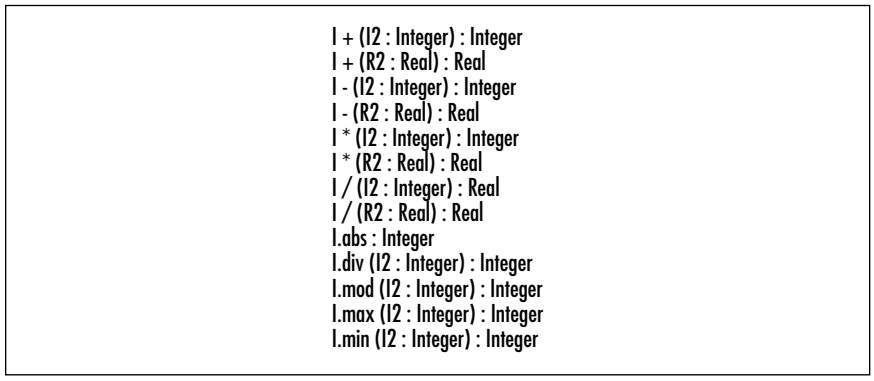


Figure 16-17: Integer Type Features

Table 16-4: Integer Type Features

Feature	*Description*
+	Returns the sum of I and I2. The result is of type Integer.
+	Returns the sum of I and R2. The result is of type Real.
-	Returns the value of subtracting I2 from I. The result is of type Integer.
-	Returns the value of subtracting R2 from I. The result is of type Real.
*	Returns the value of multiplying I and I2. The result is of type Integer.
*	Returns the value of multiplying I and R2. The result is of type Real.
/	Returns the value of dividing I by I2. The result is of type Real.
/	Returns the value of dividing I by R2. The result is of type Real.
abs	Returns the absolute value of I. If value is negative, its positive counterpart is returned.
div	Returns the number of times that I2 fits completely within I.
max	Returns the maximum of I and I2.
min	Returns the minimum of I and I2.
mod	Returns the remainder after the number of times that I2 fits completely within I, that is, the value of I modulo I2.

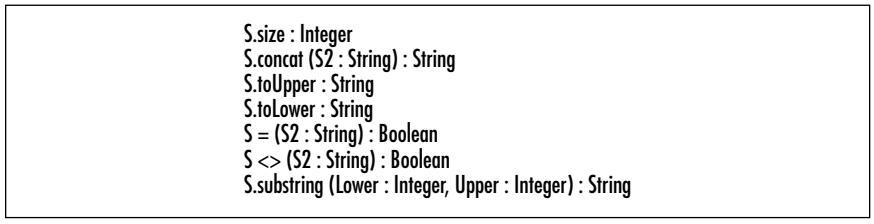


Figure 16-18: String Type Features

Table 16-5: String Type Features

Feature	*Description*
concat	Returns the concatenation of S2 to the end of S.
size	Returns the number of characters in S.
substring	Returns the substring of S starting at the character number specified by the "Lower" argument, up to and including the character number specified by the "Upper" argument. This feature requires the use of parentheses when used in an expression.
toLower	Returns S with all uppercase characters converted to lowercase characters.
toUpper	Returns S with all lowercase characters converted to uppercase characters.
=	Returns true if S contains the same characters in the same order as S2.
<>	Returns true if S does not contain the same characters in the same order as S2.

```
B or (B2 : Boolean) : Boolean
B xor (B2 : Boolean) : Boolean
B and (B2 : Boolean) : Boolean
B implies (B2 : Boolean) : Boolean
B = (B2 : Boolean) : Boolean
B <> (B2 : Boolean) : Boolean

not B : Boolean

if B then
    (Expr1 : OclExpression)
else
    (Expr2 : OclExpression)
endif : Expression1.evaluationType
```

Figure 16-19: Boolean Type Features

Table 16-6: Boolean Type Features

Feature	*Description*
and	Returns true if both B and B2 are true.
if then else endif	Returns the result of evaluating expression-1 if B is true; otherwise, returns the result of evaluating expression-2.
implies	Returns true if B is false, or if B is true and B2 is true.
not	Returns true if B is false, and false if B is true.
or	Returns true if either B or B2 is true.
xor	Returns true if either B or B2 is true, but not both.
=	Returns true if B is equal to B2.
<>	Returns true if B is not equal to B2.

Enumeration Types

Enumeration types (Figure 16-20) specify the values an object of the type may be assigned. An Enumeration type is expressed by specifying the keyword "enum" followed by curly braces containing a comma-separated list of values that an object of the type may be assigned. When a specific value is referenced, it is expressed with an additional "#" symbol. The features of the Enumeration type are depicted in Figure 16-21 and described in Table 16-7, where *E* represents the instance on which the feature is applied.

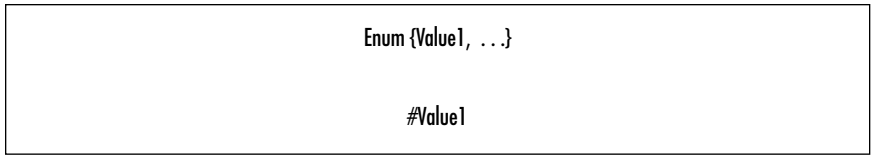


Figure 16-20: Enumeration Types

Table 16-7: Enumeration Type Features

Feature	*Description*
=	Returns true if E is equal to E2.
<>	Returns true if E is not equal to E2.

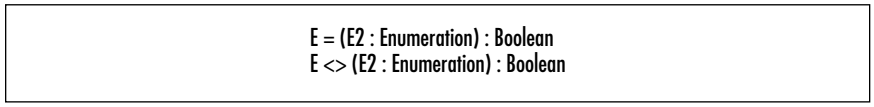


Figure 16-21: Enumeration Type Features

Collection Type

The Collection type is the abstract supertype of all collections in the OCL. An object within a collection is called an *element*. The Collection type is a template with a "T" parameter denoting the type of the collection. A concrete collection type substitutes a type for the "T" parameter of the Collection type and is denoted as "Set (T)," "Bag (T)," or "Sequence (T)." The features of the Collection type are depicted in Figure 16-22 and described in Table 16-8, where *C* represents the instance on which the feature is applied.

Set Type

The Set type is a subtype of the Collection type in which the group of objects does not contain duplicate elements and is unordered. The features of the Set type are depicted in Figure 16-23 and are described in Table 16-9, where *S* represents the instance on which the feature is applied.

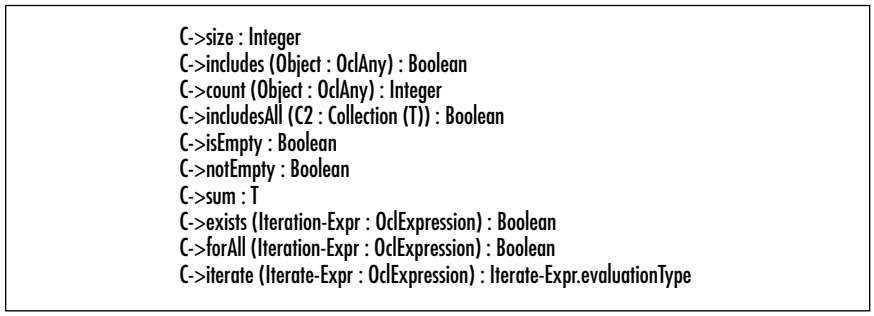


Figure 16-22: Collection Type Features

Table 16-8: Collection Type Features

Feature	*Description*
count	Returns the number of times the object occurs in C.
exists	Returns true if the expression evaluates to true for at least one element in C. This feature requires the use of parentheses when used in an expression.
forAll	Returns true if the expression evaluates to true for each element in C. This feature requires the use of parentheses when used in an expression.
includes	Returns true if the object is an element of C.
includesAll	Returns true if C contains all the elements of C2.
isEmpty	Returns true if C has no elements.
iterate	Iterates over C. This feature requires the use of parentheses when used in an expression.
notEmpty	Returns true if C has at least one element.
size	Returns the number of elements in C.
sum	Returns the sum of all elements in C. The elements must be of a type supporting addition (Integer or Real).

Bag Type

The Bag type is a subtype of the Collection type in which the group of objects may contain duplicate elements and is unordered. The features of the Bag type are depicted in Figure 16-24 and described in Table 16-10, where *B* represents the instance on which the feature is applied.

Sequence Type

The Sequence type is a subtype of the Collection type in which the group of objects may contain duplicate elements and is ordered. The features of the Sequence type are depicted in Figure 16-25 and described in Table 16-11, where *Q* represents the instance on which the feature is applied.

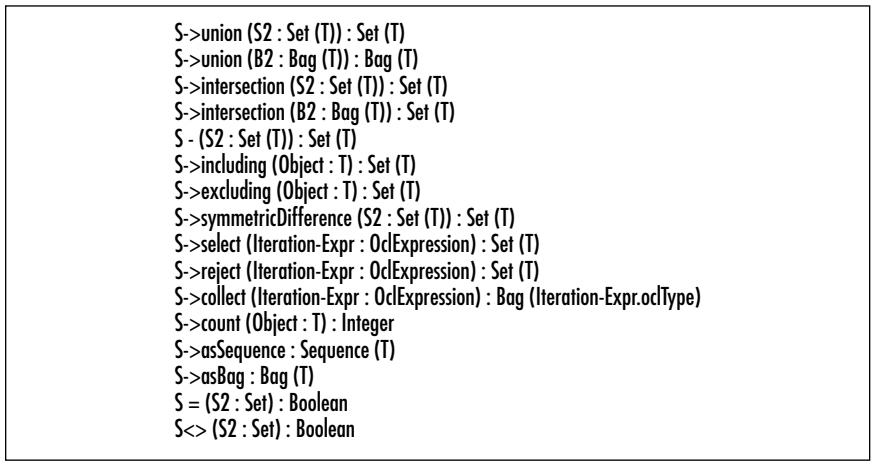


Figure 16-23: Set Type Features

Table 16-9: Set Type Features

Feature	*Description*
-	Returns the elements of S that are not in S2.
asBag	Returns the bag that contains all the elements in S.
asSequence	Returns a sequence that contains all the elements in S in random order.
collect	Returns the bag of elements that results from applying the expression to every member of S. This feature requires the use of parentheses when used in an expression.
count	Returns the number of occurrences of the object in S.
excluding	Returns the set containing all elements of S without the object.
including	Returns the set containing all elements of S plus the object.
intersection	Returns the intersection of S and S2. The result is of type Set. The result set contains all the elements that are in both sets; elements in only one set are not in the result set.
intersection	Returns the intersection of S and B2. The result is of type Set. The result bag contains all the elements that are in both collections; elements in only one collection are not in the result set.
reject	Returns the set of elements of S for which the expression is false. This feature requires the use of parentheses when used in an expression.
select	Returns the set of elements of S for which the expression is true. This feature requires the use of parentheses when used in an expression.

Table 16-9: Set Type Features (continued)

Feature	*Description*
symmetricDifference	Returns the set containing all the elements that are in S or in S2, but not in both.
union	Returns the union of S and S2. The result is of type Set. The result set contains all the elements in either set.
union	Returns the union of S and B2. The result is of type Bag. The result bag contains all the elements in either collection.
=	Returns true if S and S2 contain the same elements.
<>	Returns true if S and S2 do not contain the same elements.

```
B->union (B2 : Bag (T)) : Bag (T)
B->union (S2 : Set (T)) : Bag (T)
B->intersection (B2 : Bag (T)) : Bag (T)
B->intersection (S2 : Set (T)) : Set (T)
B->including (Object : T) : Bag (T)
B->excluding (Object : T) : Bag (T)
B->select (Iteration-Expr : OclExpression) : Bag (T)
B->reject (Iteration-Expr : OclExpression) : Set (T)
B->collect (Iteration-Expr : OclExpression) : Bag (Iteration-Expr.oclType)
B->count (Object : T) : Integer
B->asSequence : Sequence (T)
B->asSet : Set (T)
B = (S2 : Bag) : Boolean
B<> (S2 : Bag) : Boolean
```

Figure 16-24: Bag Type Features

Table 16-10: Bag Type Features

Feature	Description
asSequence	Returns a sequence that contains all the elements from B in random order.
asSet	Returns the set that contains all the elements from B with duplicates removed.
collect	Returns the bag of elements that results from applying the expression to every member of B. This feature requires the use of parentheses when used in an expression.
count	Returns the number of occurrences of the object in B.
excluding	Returns the bag containing all elements of B apart from all occurrences of the object.
including	Returns the bag containing all elements of B plus the object.
intersection	Returns the intersection of B and B2. The result is of type Bag. The result bag contains all the elements that are in both bags; elements in only one bag are not in the result bag.

Table 16-10: Bag Type Features (continued)

Feature	Description
intersection	Returns the intersection of B and S2. The result is of type Set. The result bag contains all the elements that are in both collections; elements in only one collection are not in the result set.
reject	Returns the bag of elements of B for which the expression is false. This feature requires the use of parentheses when used in an expression.
select	Returns the bag of elements of B for which the expression is true. This feature requires the use of parentheses when used in an expression.
union	Returns the union of B and B2. The result is of type Bag. The result bag contains all the elements in either bag.
union	Returns the union of B and S2. The result is of type Bag. The result bag contains all the elements in either collection.
=	Returns true if B and B2 contain the same elements the same number of times.
<>	Returns true if B and B2 do not contain the same elements the same number of times.

```
Q->union (Q2 : Sequence (T)) : Sequence (T)
Q->append (Object : T) : Sequence (T)
Q->prepend (Object : T) : Sequence (T)
Q->subSequence (Lower : Integer, Upper : Integer) : Sequence (T)
Q->at (I : Integer) : T
Q->first : T
Q->last : T
Q->including (Object : T) : Sequence (T)
Q->excluding (Object : T) : Sequence (T)
Q->select (Iteration-Expr : OclExpression) : Sequence (T)
Q->reject (Iteration-Expr : OclExpression) : Sequence (T)
Q->collect (Iteration-Expr : OclExpression) : Sequence (Iteration-Expr.oclType)
Q->iterate (Iterate-Expr : OclExpression) : Iterate-Expr.evaluationType
Q->count (Object : T) : Integer
Q->asBag : Bag (T)
Q->asSet : Set (T)
Q = (Q2 : Sequence) : Boolean
Q<> (Q2 : Sequence) : Boolean
```

Figure 16-25: Sequence Type Features

Table 16-11: Sequence Type Features

Feature	Description
append	Returns the sequence consisting of all elements in S followed by the object.
asBag	Returns a bag that contains all the elements from Q, including duplicates.

Table 16-11: Sequence Type Features (continued)

Feature	Description
asSet	Returns the set that contains all the elements from Q with duplicates removed.
at	Returns the *i*-th element in Q.
collect	Returns the sequence of elements that results from applying the expression to every member of Q. This feature requires the use of parentheses when used in an expression.
count	Returns the number of occurrences of the object in Q.
excluding	Returns the sequence containing all elements of Q apart from all occurrences of the object. The order of the elements is not changed.
first	Returns the first element in Q.
including	Returns the sequence containing all elements of Q plus the object added as the last element.
iterate	Iterates over Q. The iteration will be done from the first element to the last element in the order of the sequence. This feature requires the use of parentheses when used in an expression.
last	Returns the last element in Q.
prepend	Returns the sequence consisting of the object followed by Q.
reject	Returns the sequence of elements of Q for which the expression is false. This feature requires the use of parentheses when used in an expression.
select	Returns the sequence of elements of Q for which the expression is true. This feature requires the use of parentheses when used in an expression.
subSequence	Returns the subsequence of Q starting at the element number specified by the “Lower” argument, up to and including the element number specified by the “Upper” argument. This feature requires the use of parentheses when used in an expression.
union	Returns the sequence consisting of all elements in S followed by all elements in Q2.
=	Returns true if Q and Q2 contain the same elements the same number of times in the same order.
<>	Returns true if Q and Q2 do not contain the same elements the same number of times in the same order.

,ch16.19360 Page 262 Wednesday, May 9, 2001 4:06 PM