12 Summary of virtual classes and virtual methods

Description

In this chapter, we summarize virtual classes and virtual methods, including using virtuals as parameters. The purpose of this chapter is to have a complete description of virtual classes and virtual methods at one place in the book. To learn about these mechanism, the reader should consult the section on and the chapter on .

Virtual classes

A virtual class declaration has one of the following forms:

Self-constrained virtual class

```
class ClassName(Parameters) ReturnValue:< SuperClass
   Declarations
   Statements
```

The only difference from the declaration of a non-virtual class (see) is the use of `':<'` instead of just `':'`.

A declaration of this form specifies a *self-constrained virtual class* attribute () in the enclosing method or class.

Class-constrained virtual class

```
class ClassName(Parameters) ReturnValue:< SuperClass
```

A declaration of this form specifies a *class-constrained virtual class* attribute () in the enclosing method or class.

Further binding

A further binding of a virtual class has the form:

```
class ClassName::< SuperClass
   Declarations
   Statements
```

The symbol `'::<'` specifies that this is a *further binding*. The class is still virtual and may be further bound/extended in subclasses/submethods of the enclosing class/method.

The above form is the general form for a further binding of a virtual class. For a self-constrained virtual class, no `SuperClass` can be specified since the virtual binding in the enclosing superclass is the implicit superclass. Consider the example:

```
class GhostA:
   class C:<
      ...
class GhostB: GhostA
   class C::<
      ...
```

This example shows the general form of a further binding of a self-constrained virtual method. The virtual class `C` defined in `GhostA` is the implicit superclass of the further binding of `C` in `GhostB`.

A further binding of a class-defined virtual class may look like:

```
class GhostX:
   class C:< T
class GhostY: GhostA
```

```
   class C::< TT
```

The class-defined virtual `C` is constrained by the class `C`. In `GhostY`, a further binding of `C` is is made in the form of the class `TT`, which must be a subclass of `T`.

It is possible to add declaratios and/or statements to a further binding as indicated here:

```
class GhostX:
   class C:< T
class GhostY: GhostA
   class C::< TT
      Dcl1
      Dcl2
      ...
      Stm1
      Stm2
      ...
```

If declarations and/or statements are added in the further binding, the virtual class then becomes a self-constrained virtual class.

Final binding

```
class ClassName:: SuperClass
   Declarations
   Statements
```

The symbol '`::`' specifies that this is a *final binding.* The class is no longer virtual, which means that it is not possible to make further bindings in subclasses.

As for further binding, the above form is the general form of a final binding. With respect to *`SuperClass`*, the same rules apply here as for a further binding as described above.

Virtual methods

The declaration and bindings of a virtual method is analogous to the declaration and bindings of a virtual class.

A virtual method declaration has one of the following form:

Self-constrained virtual method

```
MethodName(Parameters) ReturnValue:< SuperMethod
   Declarations
   Statements
```

The only difference from the declaration of a non-virtual method (see) is the use of `':<'` instead of just `':'`.

A declaration of this form specifies a *self-constrained virtual method attribute* in the enclosing method or class. Note, we have not used the term *self-constrained virtual method* before.

Method-constrained virtual method

```
MethodName(Parameters) ReturnValue:< SuperMethod
```

A declaration of this form specifies a *method-constrained virtual method attribute* in the enclosing method or class. Note, we have no examples of method-constrained virtual methods in this version of the book. There are included here for completeness and to illustrate that virtual methods and virtual classes have analogous forms with respect to syntax and semantics.

Further binding

A further binding of a virtual method has the form:

```
MethodName::< SuperMethod
   Declarations
   Statements
```

The symbol '`::<`' specifies that this is a *further binding*. The method is still virtual and may be further bound/extended in subclasses/submethods off the enclosing class/method.

The above form is the general form for a further binding of a virtual method. For a self-constrained virtual method as used in this book, no `SuperMethod` can be specified since the virtual binding in the enclosing superclass is the implicit supermethod. Consider the example:

```
class GhostA:
   M:<
      ...
class GhostB: GhostA
   M::<
      ...
```

This example shows the general form of a further binding of a self-constrained virtual method. The virtual method `M` defined in `GhostA` is the implicit supermethod of the further binding of `M` in `GhostB`.

For completeness, we also sketch how the binding of a method-defined virtual method may look like:

```
class GhostX:
   M:< A
class GhostY: GhostA
   M::< AA
```

The method-defined virtual `M` is constrained by the method `A`. In `GhostY`, a further binding of `M` is is made in the form of the method `AA`, which must be a submethod of `A`.

It is possible to add declarations and/or statements to a further binding. The virtual method then becomes a self-constrained virtual method.

Final binding

```
MethodName:: SuperMethod
   Declarations
   Statements
```

The symbol '`::`' specifies that this is a *final binding.* The method is no longer virtual, which means that it is not possible to make further bindings in subclasses.

As for further binding, the above form is the general form of a final binding. With respect to *SuperMethod*, the same rules apply here as for a further binding as described above.

Virtuals as parameters

A parameter of a method or class may be declared as a class-constrained virtual class.

In chapter , the parameters `ElmType` of class `Set`, class `Array` and class `OrderedList` are examples of class-constrained virtual class parameters.

The invocation of the method or instantiation of the class may then specify a final binding of the virtual class parameter.

In a similar way, a parameter of a method or class may be declared as a method-constrained virtual method.

In section , the parameters `thenPart` and `elsePart` of `if:then:else` are examples of method-constrained virtual method parameters.

The invocation of the method or instantiation of the class may then specify a final binding of the virtual method parameter.