# AspectJ 1.1 Quick Reference

## Aspects *at top-level or* **static** *in types*

**aspect** *A* { … }
defines the aspect *A*
**privileged aspect** *A* { … }
*A* can access private fields
**aspect** *A* **extends** *B* **implements** *I, J* { … }
*B* is a class or abstract aspect, *I* and *J* are interfaces
**aspect** *A* **percflow**( *call(void Foo.m())* ) { … }
an instance of *A* is instantiated for every control flow through calls to *m()*

general form:
[ **privileged** ] [ *Modifiers* ] **aspect** *Id*
[ **extends** *Type* ] [ **implements** *TypeList* ] [ *PerClause* ]
{ *Body* }
where *PerClause* is one of
**pertarget** ( *Pointcut* )
**perthis** ( *Pointcut* )
**percflow** ( *Pointcut* )
**percflowbelow** ( *Pointcut* )
**issingleton**

## Pointcut definitions *in types*

**private pointcut** *pc*() : *call(void Foo.m())* ;
a pointcut visible only from the defining type
**pointcut** *pc*(*int i*) : *set(int Foo.x) && args(i)* ;
a package-visible pointcut that exposes an *int*.
**public abstract pointcut** *pc*() ;
an abstract pointcut that can be referred to from anywhere.
**abstract pointcut** *pc*(*Object o*) ;
an abstract pointcut visible from the defining package. Any pointcut that implements this must expose an *Object*.

general form:
**abstract** [*Modifiers*] **pointcut** *Id* ( *Formals* ) ;
[*Modifiers*] **pointcut** *Id* ( *Formals* ) : *Pointcut* ;

## Advice declarations *in aspects*

**before** () : *get(int Foo.y)* { ... }
runs before reading the field *int Foo.y*
**after** () **returning** : *call(int Foo.m(int))* { ... }
runs after calls to *int Foo.m(int)* that return normally
**after** () **returning** (*int x*) : *call(int Foo.m(int))* { ... }
same, but the return value is named *x* in the body
**after** () **throwing** : *call(int Foo.m(int))* { ... }
runs after calls to *m* that exit abruptly by throwing an exception
**after** () **throwing** (*NotFoundException e*) : *call(int Foo.m(int))* { ... }
runs after calls to *m* that exit abruptly by throwing a *NotFoundException*. The exception is named *e* in the body
**after** () : *call(int Foo.m(int))* { ... }
runs after calls to *m* regardless of how they exit
**before**(*int i*) : *set(int Foo.x) && args(i)* { ... }
runs before field assignment to *int Foo.x*. The value to be assigned is named *i* in the body
**before**(*Object o*) : *set(* Foo.*) && args(o)* { ... }
runs before field assignment to any field of *Foo*. The value to be assigned is converted to an object type (*int* to *Integer*, for example) and named *o* in the body
*int* **around** () : *call(int Foo.m(int))* { ... }
runs instead of calls to *int Foo.m(int)*, and returns an *int*. In the body, continue the call by using **proceed**(), which has the same signature as the around advice.
*int* **around** () **throws** *IOException* : *call(int Foo.m(int))* { ... }
same, but the body is allowed to throw *IOException*
*Object* **around** () : *call(int Foo.m(int))* { ... }
same, but the value of **proceed**() is converted to an *Integer*, and the body should also return an *Integer* which will be converted into an *int*

general form:
[ **strictfp** ] *AdviceSpec* [ **throws** *TypeList* ] : *Pointcut* { *Body* }
where *AdviceSpec* is one of
**before** ( *Formals* )
**after** ( *Formals* )
**after** ( *Formals* ) **returning** [ ( *Formal* ) ]
**after** ( *Formals* ) **throwing** [ ( *Formal* ) ]
*Type* **around** ( *Formals* )

## Special forms *in advice*

**thisJoinPoint**
reflective information about the join point.
**thisJoinPointStaticPart**
the equivalent of **thisJoinPoint.getStaticPart()**, but may use fewer resources.
**thisEnclosingJoinPointStaticPart**
the static part of the join point enclosing this one.

**proceed** ( *Arguments* )
only available in **around** advice. The *Arguments* must be the same number and type as the parameters of the advice.

## Inter-type Member Declarations *in aspects*

*int Foo* . m ( *int i* ) { ... }
a method *int m(int)* owned by *Foo*, visible anywhere in the defining package. In the body, **this** refers to the instance of *Foo*, not the aspect.
**private** *int Foo . m* ( *int i* ) **throws** *IOException* { ... }
a method *int m(int)* that is declared to throw *IOException*, only visible in the defining aspect. In the body, **this** refers to the instance of *Foo*, not the aspect.
**abstract** *int Foo . m* ( *int i* ) ;
an abstract method *int m(int)* owned by *Foo*
*Point* . **new** ( *int x, int y* ) { ... }
a constructor owned by *Point*. In the body, **this** refers to the new *Point*, not the aspect.
**private static** *int Point . x* ;
a static *int* field named *x* owned by *Point* and visible only in the declaring aspect
**private** *int Point . x = foo()* ;
a non-static field initialized to the result of calling *foo()*. In the initializer, **this** refers to the instance of *Foo*, not the aspect.

general form:
[ *Modifiers* ] *Type Type . Id* ( *Formals* )
[ **throws** *TypeList* ] { *Body* }
**abstract** [ *Modifiers* ] *Type Type . Id* ( *Formals* )
[ **throws** *TypeList* ] ;
[ *Modifiers* ] *Type* . new ( *Formals* )
[ **throws** *TypeList* ] { *Body* }
[ *Modifiers* ] *Type Type . Id* [ = *Expression* ] ;

## Other Inter-type Declarations *in aspects*

**declare parents** : *C* **extends** *D*;
declares that the superclass of *C* is *D*. This is only legal if *D* is declared to extend the original superclass of *C*.
**declare parents** : C **implements** *I, J* ;
*C* implements *I* and *J*
**declare warning** : *set(* Point.*) && !within(Point)* : *"bad set"* ;
the compiler warns *"bad set"* if it finds a set to any field of *Point* outside of the code for *Point*
**declare error** : *call(Singleton.new(..))* : *"bad construction"* ;
the compiler signals an error *"bad construction"* if it finds a call to any constructor of *Singleton*
**declare soft** : *IOException* : *execution(Foo.new(..))*;
any IOException thrown from executions of the constructors of *Foo* are wrapped in **org.aspectj.SoftException**
**declare precedence** : *Security, Logging, * ;
at each join point, advice from *Security* has precedence over advice from *Logging*, which has precedence over other advice.

general form
**declare parents** : *TypePat* **extends** *Type* ;
**declare parents** : *TypePat* **implements** *TypeList* ;
**declare warning** : *Pointcut* : *String* ;
**declare error** : *Pointcut* : *String* ;
**declare soft** : *Type* : *Pointcut* ;
**declare precedence** : *TypePatList* ;

## Primitive Pointcuts

**call** ( *void Foo.m(int)* )
a call to the method *void Foo.m(int)*
**call** ( *Foo.new(..)* )
a call to any constructor of *Foo*
**execution** ( * *Foo.*(..) throws IOException* )
the execution of any method of *Foo* that is declared to throw *IOException*
**execution** ( *!public Foo .new(..)* )
the execution of any non-public constructor of *Foo*
**initialization** ( *Foo.new(int)* )
the initialization of any *Foo* object that is started with the constructor *Foo(int)*
**preinitialization** ( *Foo.new(int)* )
the pre-initialization (before the **super** constructor is called) that is started with the constructor *Foo(int)*
**staticinitialization**( *Foo* )
when the type *Foo* is initialized, after loading
**get** ( *int Point.x* )
when *int Point.x* is read
**set** ( *!private * Point.** )
when any non-private field of *Point* is assigned
**handler** ( *IOException+* )
when an *IOException* or its subtype is handled with a catch block
**adviceexecution()**
the execution of all advice bodies
**within** ( *com.bigboxco.** )
any join point where the associated code is defined in the package *com.bigboxco*
**withincode** ( *void Figure.move()* )
any join point where the associated code is defined in the method *void Figure.move()*
**withincode** ( *com.bigboxco.*.new(..)* )
any join point where the associated code is defined in any constructor in the package *com.bigoxco.*
**cflow** ( *call(void Figure.move())* )
any join point in the control flow of each call to *void Figure.move()*. This includes the call itself.
**cflowbelow** ( *call(void Figure.move())* )
any join point below the control flow of each call to *void Figure.move()*. This does not include the call.
**if** ( *Tracing.isEnabled()* )
any join point where *Tracing.isEnabled()* is **true**. The boolean expression used can only access static members, variables bound in the same pointcut, and **thisJoinPoint** forms.

**this** ( *Point* )
any join point where the currently executing object is an instance of *Point*
**target** ( *java.io.InputPort* )
any join point where the target object is an instance of *java.io.InputPort*
**args** ( *java.io.InputPort, int* )
any join point where there are two arguments, the first an instance of *java.io.InputPort*, and the second an *int*
**args** ( **, int* )
any join point where there are two arguments, the second of which is an *int.*
**args** ( *short, .., short )*
any join point with at least two arguments, the first and last of which are *short*s
**Note:** any position in **this**, **target**, and **args** can be replaced with a variable bound in the advice or pointcut.

general form:
**call**(*MethodPat*)
**call**(*ConstructorPat*)
**execution**(*MethodPat*)
**execution**(*ConstructorPat*)
**initialization**(*ConstructorPat*)
**preinitialization**(*ConstructorPat*)
**staticinitialization**(*TypePat*)
**get**(*FieldPat*)
**set**(*FieldPat*)
**handler**(*TypePat*)
**adviceexecution()**
**within**(*TypePat*)
**withincode**(*MethodPat*)
**withincode**(*ConstructorPat*)
**cflow**(*Pointcut*)
**cflowbelow**(*Pointcut*)
**if**(*Expression*)
**this**(*Type* | *Var*)
**target**(*Type* | *Var*)
**args**(*Type* | *Var* , …)
where *MethodPat* is:
[*ModifiersPat*] *TypePat* [*TypePat* . ] *IdPat* (*TypePat* | .. , … )
[ **throws** *ThrowsPat* ]
*ConstructorPat* is:
[*ModifiersPat* ] [*TypePat* . ] **new** (*TypePat* | .., …)
[ **throws** *ThrowsPat* ]
*FieldPat* is:
[*ModifiersPat*] *TypePat* [*TypePat* . ] *IdPat*
*TypePat* is one of:
*IdPat* [ + ] [ **[]** … ]
**!** *TypePat*
*TypePat* **&&** *TypePat*
*TypePat* || *TypePat*
( *TypePat* )