The object of class Cat to the right shows that Cat extends class Animal. It is used later in the discussion.

equals()

Animal

a

Cat

c

Object

Animal

Cat@6dfe

Cat

Suppose we have the following cast:

(String) a

From the type of variable a, which is Animal (look to the right) and the fact that class String extends Object and cannot be extended, it is clear that no Animal object has a partition named String. If the program were compiled and the program run, every evaluation of this expression would throw a ClassCastException. Therefore, this is expression is deemed a compile-time error, a syntax error, and the expression will not be compiled.

We state the rule more generally

**Compile-time casting rule**. Consider a cast-expression

(*name*) *expression*

where *name* is the name of some class[[1]](#footnote-1) and the type of *expression* is some class-type C.

1. If it can be determined solely from the declaration of C (and its subclasses and superclasses) that no object that has a C partition also has a *name* partition, then this expression is syntactically incorrect, and it will not be compiled.
2. If at least one object that has a C partition also has a *name* partition, then the expression is OK and will be compiled.

**When casting at runtime may throw a ClassCastException**

Now consider this code, where variable a is as described in the diagram in the upper right.

a= **new** Cat();  
 …

… (Cat) a …

The type of variable a is Animal, and it is known some object with an Animal partition also has a Cat partition. Therefore the compile-time casting rule allows this to be compiled. It is syntactically OK.

Here is a case to be analyzed carefully, for jumping to what seems an obvious conclusion leads to an error. Consider the expression

(Cat)(**new** Animal())

What do *you* think — is it syntactically OK? Will it be compiled?

The type of the *expression* **new** Animal() is Animal. As said above, objects exist that have both Animal and Cat partitions. Therefore, the *expression* is syntactically OK and can be compiled!

Here is the important point: *The compile-time casting rule does not look at the particular value of the expression but only at its type*. You and I know that evaluation of this expression will throw a ClassCastException, but according to the compile-time casting rule, it is allowed. The compiler does not look at the object itself, but only its type.

1. For simplicity, we mention only classes. But in all generality, interfaces should be included. For example, *name* can be an interface and C can be an interface. [↑](#footnote-ref-1)