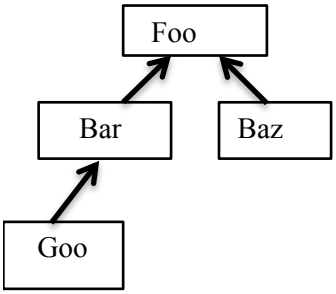
Inheritance Hell, a place of evil and suffering

"Inheritance Hell" refers to complex and seemingly contradictory behavior by the compiler and runtime system when resolving types and methods in an inheritance hierarchy. However, through a proper understanding of how the compiler does type checking and how the runtime system works, it is possible to be able to correctly predict the behavior of even the most complex examples. First, some rules.

A. Compile Time Rules:



1) We can assign up the inheritance hierarchy

```
Foo b = new Bar();
Bar g = new Goo();
Object o = new String("abc");
Container c = new JPanel();
```

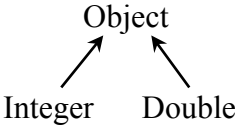
2) We cannot assign down the inheritance hierarchy

```
Bar g2 = new Foo();          // Type mismatch: cannot convert from Foo to Baz
Goo b2 = new Bar();          // Type mismatch: cannot convert from Bar to Goo
String str = new Object();   // Type mismatch: cannot convert from Object to String
JPanel p = new Container(); // Cannot convert from Container to JPanel
```

3) Casting up and casting down are both okay

```
Object anInt = new Integer(4);
int n = ((Integer) anInt).intValue();
System.out.println(((Object) anInt).getClass()); // class.java.lang.?_______?
```

4) Casting sideways is ***not*** okay




```
Double aDouble = 1.2;
Integer int2 = 12;
((Integer) aDouble).toString(); // Cannot cast from Double to Integer
((Double) int2).toString();     // Cannot cast from Integer to Double
```

B. Runtime Rules

1) Casting sideways at runtime is ***not*** okay either

```
Object anInt = new Integer(123);
Object aDouble = new Double(4.56);
Object obj = (Double)anInt; // Compiles, but throws a CastClassException
```

2) Casting below the object's runtime type is ***not*** okay.

```
Foo f = new Bar();
// Compiles, but this is a Runtime error: Bar cannot be cast to Goo
((Goo)f).one();  // Assume Bar and Goo both have method one()
```

Recommended Process for Answering Inheritance Hell Questions

Here is a recommended process that should help derive the correct answers.

1) First consider how the compiler will react

- What is the type at compile time?
- If there is a cast involved, based on the *compile-time type* of the variable, can we cast that type to that type?
- And, once we do that cast, does that class type have a *.method*() method, either directly, or inherited from one of its superclasses, found by traversing up the inheritance tree?

If yes to both, the code compiles. If not, write **CTE** for CompileTime Error

2) If the message compiles consider the following that could happen at runtime

Based on the *run-time type*, which is the type of the actual object rather than the type into which it is stored, can that object be cast to the variable's class type? If yes, some message will be sent (code will run). Otherwise write **RE** for Runtime Error**.** The answer would be "no" at runtime if

- there is a sideways cast
- the cast is to a type below the runtime type of the object being casted

3) If there is no CTE or RE, determine the output

The Java runtime starts with the *run-time type*. It finds which version of the method it should use by traversing back up the inheritance tree. It will always find the method, eventually.

There is also a chance a method may send a message to the object that exists in that class and in the object's runtime type. In this case, the actual method called is the method in the run-time type.