The class invariant

Each instance of this class describes a time of day in some part of the world. The time itself is in seconds. Field `zone` contains a three-letter designation of the time zone for the current time, like `"GMT"` for Greenwich Mean Time, or `"EST"` for Eastern Standard Time. And field `ampm` indicates whether this time should be displayed in AM-PM time or in 24-hour time.

The class has a bunch of methods: three constructors, one of which allows the time to be given not in seconds but in hours, minutes, and seconds; getter methods; one setter method; and function `toString`.

It also has a function that yields an object with the same time but in zone GMT. For example, if we call this function in an object that has a time of 8 hours in EST, it produces a new object that has the same time in GMT —the time is 13 hours, because EST, which is New York time, is 5 hours behind GMT time, which is London time. But if object `a0` has time 23 hours, New York time, then the new object will have time 28 hours in GMT. This kind of conversion might also end up with field `time` containing a negative value.

Now, consider writing the constructor that is given the time, in seconds, and the zone. In order to ensure that objects have no mistakes in them, we require that zone z be a legal zone. But what is a legal zone? How many different ones are there? Which ones will this program handle? We have not been given this information.

There *should* have been a comment on the declaration that defines the zones that can be used.

Next, the constructor should make sure that the time, s seconds, is appropriate as well. But what is appropriate? Are there any constraints on it? We can't tell. There *should* have been a comment on the declarations that explains what field time is and gives constraints on it. The main property is that in *some* time zone, the time is in the range 0 to 24 hours. And this helps us make the specification of the constructor more precise.

Finally, there should be a comment explaining what field `ampm` is for.

The class invariant

These comments together constitute thee *class invariant*. This is a description of the fields of an instance. It must include all constraints on them. For example, in this case, only certain values are allowed for the zone.

The class invariant is used as follows. Every method in an object can assume that the class invariant is true when the method is called, and the class invariant must be true when execution of the method body ends. The programmer can rely on all the constraints when writing the method body. In return, the programmer is responsible for ensuring that the class invariant is true when the body terminates.

It is your duty as programmer to write the class invariant when you declare the fields. And the more careful and clear and precise and thorough you are in writing the class invariant, the easier time *you* will have in writing the methods. In fact, you, the programmer, are the one who will benefit the most from writing the class invariant early in the game. Let me show you why.

Consider writing the constructor. From the specification, the body has to check whether the zone is legal. But what is a legal zone? Well, just look at the class invariant.

In fact, the class invariant should be looked at continually when writing almost every method. First, knowing the constraints on the fields generally makes writing the body easier. Second, continual referral to the class invariant will help you ensure that the constraints are true when the method body ends.

Position of the class invariant

In this blecture, we placed the class invariant as a single comment before the declarations. As an alternative, parts of the class invariant may be interspersed with the declarations and may appear as comments to the right of a declaration. The important point is not the particular format used but the actual writing of the class invariant.