```
public abstract class Event    {
    protected int    numLanes;        //used in each subclass
    protected Vector swimmers;

   public abstract Seeding getSeeding();
   public abstract boolean isPrelim();
   public abstract boolean isFinal();
   public abstract boolean isTimedFinal();
}
```

This defines the methods without our having to fill in the methods. Then we can derive concrete classes from the Event class, called PrelimEvent and TimedFinalEvent. The only difference between these classes is that one returns one kind of seeding and the other returns a different kind of seeding.

We also define an abstract Seeding class having the following methods:

```
public abstract class Seeding {
   public abstract Enumeration getSwimmers();
   public abstract int getCount();
   public abstract int getHeats();
}
```

Next, we create two concrete seeding subclasses: StraightSeeding and CircleSeeding. The PrelimEvent class will return an instance of CircleSeeding, and the TimedFinalEvent class will return an instance of StraightSeeding. Thus we see that we have two hierarchies: one of Events and one of Seedings. We see these two hierarchies illustrated in Figure 4.1.

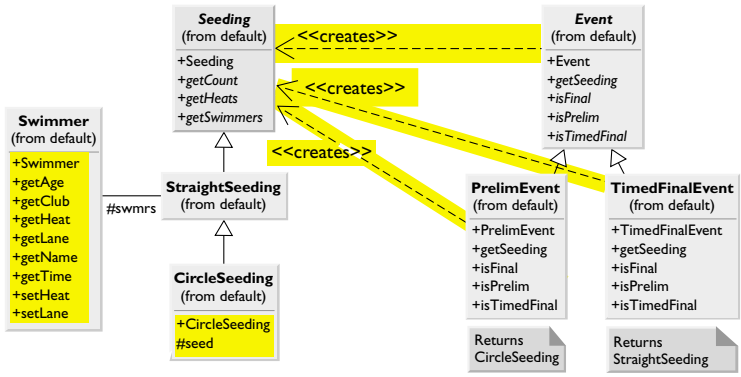

Figure 4.1 The class relationships between Event and Seeding classes.

In the Event hierarchy, you will see that both derived Event classes contain a *getSeeding* method. One of them returns an instance of StraightSeeding, and the other returns an instance of CircleSeeding. So you see, there is no real factory deci-