Project 2 Notes
Eric Lemar
Section AA

Error propagation:

When calling a function that may return an error, ALWAYS make sure to deal with the error in an appropriate manner. If you call acceptGameResults(String,int,String,int) from acceptGameResults(String), make sure to return the return value of the called function. This is very good code reuse, but make SURE to return the error value from this sub-call. How might have using exceptions avoided this error?

Code reuse in inheritance:

It should almost **NEVER** be necessary to copy any entire function verbatim from a super class into subclasses. Just directly use the inherited function.

There are many other cases where you'll find yourself copying large pieces of code with only minor changes from the superclass. Sometimes this is unavoidable, but there are several common strategies to avoid this, none of which are perfect. Dealing with acceptGameResults(String name1, int score1, String name2, int score2) is a good example. Options:

1. **Wrap the superclass function**. Sometimes the needed change can be accomplished just by doing something before or after the original. If so, override the function, but call supper.acceptGameResults() at the appropriate place in the body in the body. This strategy MAY work for acceptGameResults(), but is troublesome since you really want to do score checking in the middle of the old method(after the checks for null, but before adding to the list).
2. **Split the function into reusable protected helper methods**. For instance, write a checkForNull() and addContestant() function. Override the old acceptGameResults() and don't call super.acceptGameResults(), but do use the parent class functions to do most of the work.
3. **Design the original function to call overridable helper methods for functionality that may change**. For addContests(), you might have acceptGameResults() call functions isValidScore() and findWinner() respectively. Instead of overriding acceptGameResults(), golf and baseball can simply override the much smaller isValidScore() and findWinner() methods.

Hiding fields

Don't redefine instance fields in subclasses. If you declared a field in the superclass as protected, the subclass can still see it. If you do redefine it, the new definition actually "shadows" the field from the superclass, essentially making a completely new variable. Accessing the field through a static type of the superclass accesses the parent field, accessing through the static type of the subclass accesses the child field. This almost certainly isn't what you intended.

Initialization with inheritance:
A minor point, but if you declare an instance variable in a class, it's USUALLY considered best to initialize it in the constructor for that class rather than in child classes. This maintains the abstraction of having the superclass present a full interface to the subclasses.

Testing
There were a fair number of projects where 99.9% of the code to behave correctly was there, but there was some tiny detail that kept it from working. For instance, they used a null value before checking for null or they forgot to return the error code from a call to an overloaded function. A small amount of work to test a range of test cases would have avoided most of this. Any time you see yourself putting in code to deal with some special case, also think about adding a test to check that special case. Feel like you're taking as much time testing and debugging as writing code? Good, you're probably starting to do ALMOST enough testing and debugging :)

this/super
A number of solutions tended to call functions using the "this.func()" syntax. Although technically correct, it's stylistically better to just call "func()". The effect is identical. "this" is only used when you need a reference to the current object to pass to a function you're calling. For instance, if your contestant object wanted to know what gamekeeper it was a member of, acceptGameResults() might use "this" to get a pointer to itself which can then be passed on to the Contestant constructor. The only other major reason to use "this" is to access an instance variable if there is a local with the same name. This is done frequently in constructors.

There were also some cases where people called "super.func()" to call a function that was defined in the super class. ONLY do this if you specifically need the version of func() that was defined in the super class, not the version in the current class. Otherwise just call func(). Calling super.func() in other cases makes it unclear to the reader WHY you are specifically calling the non-overriden version. It is also fragile since you later decide to override the function.