Solution 1: Dig through dots

```
Athlete betterCyclist1(Athlete other) {
    if (this.cycResult.pointsEarned() < other.cycResult.pointsEarned()) {
        return this;
    } else return other;
}
```

This is potentially the most popular solution new Java programmers will use. It involves using the dot operator to directly access methods or variables within an object. This is generally bad practice because it exposes the inner workings of a class to another class. The idea behind classes is to separate the concerns. This does not achieve that separation.

Solution 2: Use a method that handles everything in CyclistResult

```
Athlete betterCyclist2(Athlete other) {
    if (this.cycResult.betterScoreThan(other)) {
       return this;
    } else return other;
}
```

This is a better approach to the first one and it is getting closer to what we want. We know that the CyclistResult should have a method that handles at least most of the computation here. However, it would be nice if we didn't have to pass in an Athlete to the betterScoreThan method at all.

Solution 3: Use getters

```
Athlete betterCyclist3(Athlete other) {
    if(this.cycResult.pointsEarned() < other.getCycResult().pointsEarned())
        return this;
    else
        return other;
    }
```

This is getting awfully close to a great solution. We aren't exposing the inner workings of the CyclistResult class and we aren't passing an Athlete into a method within CyclistResult. However, we are using a getter. Our ideal solution involves removing the need for getter functions. We will continue to talk about a concept called encapsulation throughout the class.

Solution 4: Encapsulation and Separation of Concerns

```
Athlete betterCyclist4(Athlete other) {
    if(this.cyclistPointsEarned() < other.cyclistPointsEarned())
        return this;
    else
        return other;
}

double cyclistPointsEarned() {
    return this.cycResult.pointsEarned();
}
```

Here you see what an encapsulated solution looks like. Your methods are doing what they need to and nothing more. As the class goes on, we will talk more about why this solution works so well. As it turns out, CyclistResult does not need to know about the math comparing the two Athletes.