

Sketches to answers to Sample Midterm Questions.

2. The emphasis in this question is on getting good coverage. In my sample answer, I list all the cases that should be covered but only show exact test input for 1 of the test cases. A complete answer would have exact test input for ALL of the cases.

The following test cases should report that a full house WAS found:

Full house in community cards

Community cards: Ace of Clubs, Ace of Hearts, Ace of Diamonds, King of Clubs, King of Diamonds

Hole cards: 2 of Hearts, 5 of Clubs

Triple in community cards; pair in hole

Triple split across community & hole; pair in community

Triple in community cards; pair split across community & hole

Triple split across community and hole cards; pair split across community and hole cards

The following cases should indicate that a full house WAS NOT found:

Three of a kind, but no pair

Two pairs

The specification is ambiguous about how the following test case should turn out:

Four of a kind and a pair in any combination. The problem here is that the hand includes a full house, but there is a better hand. If the purpose of the method being tested is to determine if a full house is present, then the method should find it. If its purpose is to find the highest-ranking hand, it should instead identify this as four-of-a-kind.

3. As with question 2, a complete answer would identify the values in the numInRank array that is used by the isFullHouse method. I do not include that level of detail here.

To get complete coverage, there should be following test cases:

Not a full house - needed to execute the "return false" at the end of the method. Also ensures that we take the "else" branch on the if-statement directly inside the for

loop.

Triple at a higher rank than the pair - gets the false branch of the “if (pairFound)” if and the true branch of the “if (tripleFound)” if.

Pair at a higher rank than the triple - gets the true branch of the “if (pairFound)” if and the false branch of the “if (tripleFound)” if.

4.

- a. NullPointerException if c is null.
NullPointerException if o is null.
Object added and true is returned.
Object already in collection, false is returned.
UnsupportedOperationException if the actual object in c does not support add.
ClassCastException if the collection only allows objects of a particular class and o is not of that class
IllegalArgumentException if the object is not in the collection already and the collection does not add the object for any reason than those given above. For example, maybe the collection is only intended to hold positive Integers and a negative value was passed in.
- b. Collection should have 2 subclasses: ImmutableCollection and MutableCollection. MutableCollection adds mutator methods like add, remove and clear. ImmutableCollection does not add any operators that allow mutation. For every kind of collection currently in the collections hierarchy, like Set, LinkedList, etc., there would now be two versions. One version is immutable; the other is mutable. Since add is not defined on the immutable classes, any attempt to call add would result in a compile-time error rather than the run-time error of the current organization.
- c. One big problem is that immutability is not enforceable, short of making a class final. We can't make ImmutableCollection final because then we couldn't have things like ImmutableSet, ImmutableList, etc. Because of this, there is no way to prevent someone from writing a subclass that is mutable but claiming it extends ImmutableCollection. A programmer declaring an object to be an ImmutableCollection but ending up with a mutable object in the variable would be surprised (or worse!).

Since MutableSet is a lot like ImmutableSet except that it has additional methods. So, it would be very tempting to make MutableSet as subclass of ImmutableSet so that it could inherit the common code. This would lead to the problem that a mutable set could masquerade as an immutable set.

The main problem with these solutions is that if a mutable set masqueraded as an immutable set, the problem would not be detected by the compiler or runtime

system. With Sun's solutions if we try to mutate an immutable collection, we will at least get a runtime error. This is better than no error at all, which our proposed alternative is susceptible to.