Questions

Bookmark this page

AF & RI

2/2 points (graded)

Which of the following should be known (visible and documented) to the client of an abstract data type? Check all that apply.

- [x] abstract value space
- [ ] abstraction function
- [x] creators
- [x] observers
- [ ] rep
- [ ] rep invariant

Explanation
In order to preserve representation independence, the client should know about things related to the abstraction, but not about the rep. From this list, the abstraction includes the abstract value space (e.g., CharSet's abstract value space is "sets of characters"), creators (e.g., CharSet's constructor), observers (e.g., CharSet.size()). But the rep, rep invariant, and abstraction function involve knowledge of the rep, so they should not generally be known to the client.

Which of the following should be known (visible and documented) to the maintainer of an abstract data type? Check all that apply.

- [x] abstract value space
- [x] abstraction function
- [x] creators
- [x] observers
- [x] rep
- [x] rep invariant

Explanation
The maintainer of an abstract data type has to know about both the abstraction and the rep, so all elements should be known.

Submit

Show Answer

Answers are displayed within the problem

AF & RI, part 2

1/1 point (graded)

Suppose C is an abstract data type whose representation has two String fields:

```
class C {
    private String s;
    private String t;
    ...
}
```

Assuming you don't know anything about C's abstraction, which of the following might be part of a rep invariant for C?

- [x] s contains only letters
- [x] s.length() == t.length()
- [ ] s represents a set of characters
- [ ] C's observers
- [x] s is the reverse of t
- [ ] s+t

✔

Explanation

Recall that the rep invariant is a function from rep values (pairs of Strings s,t) to boolean, so the only good answers to this question are boolean predicates (true or false statements) that constrain legal values of s and t.
"s represents a set of characters" belongs in an abstraction function.
C's observer operations are part of the abstraction, not the rep.
s+t is not a boolean predicate.

Submit

Show Answer

Answers are displayed within the problem

AF & RI, part 3

1/1 point (graded)

Suppose we are implementing CharSet with the following rep:

```
public class CharSet {
    private String s;
    ...
}
```

But we neglect to write down the abstraction function (AF) and rep invariant (RI). Here are four possible AF/RI pairs, which were also mentioned in the reading.

SortedRep:

```
// AF: represents the set of characters found in s
// RI: s[0] < s[1] < ... < s[s.length()-1]
```

SortedRangeRep:

```
// AF: represents the union of the ranges {s[i]...s[i+1]} for each adjacent pair of characters in s
// RI: s.length is even, and s[0] < s[1] < ... < s[s.length()-1]
```

NoRepeatsRep:

```
// AF: represents the set of characters found in s
// RI: s contains no character more than once
```

AnyRep:

```
// AF: represents the set of characters found in s
// RI: true
```

Three programmers are working on CharSet, each on a different method: add(), remove(), and contains().

Which possible AF/RI pairs are consistent with this programmer's implementation of `add()` ?

```
public void add(char c) {
    s = s + c;
}
```

- [ ] SortedRep
- [ ] SortedRangeRep

- [ ] NoRepeatsRep
- [x] AnyRep

Explanation
The programmer who wrote add() did it the easiest way possible.
It isn't consistent with SortedRep or SortedRangeRep, because those reps require the character to be put in a particular place, depending on the value of c, not just at the end of the string.
It isn't consistent with NoRepeatsRep because c may already occur in the string, and add() isn't checking for that.
But it is consistent with AnyRep, whose RI allows any string of characters and whose AF interprets the string in such a way that c is considered part of the resulting set.

Submit

Show Answer

Answers are displayed within the problem

AF & RI, part 4

0/1 point (graded)

Which possible AF/RI pairs are consistent with this programmer's implementation of `remove()` ?

```
public void remove(char c) {
    int position = s.indexOf(c);
    if (position >= 0) {
        s = s.substring(0, position) + s.substring(position+1, s.length());
    }
}
```

- [x] SortedRep
- [ ] SortedRangeRep
- [x] NoRepeatsRep
- [ ] AnyRep

Explanation
This implementation of remove() finds the first occurrence of c in the string and removes it.
It is consistent with SortedRep, because it still keeps the string ordered. Note also that if you read SortedRep's RI carefully, you'll see that it also forbids duplicates, so we're guaranteed that the string won't have any other occurrences of c that we need to remove.
It isn't consistent with SortedRangeRep, because it will make an even-length string into an odd-length string by throwing away one end of a range pair.
It is consistent with NoRepeatsRep, because the string will have at most one occurrence of c to remove.
It isn't consistent with AnyRep, because of the possibility that c is duplicated in the string. If the string is "caac" and we remove just the first 'c', the string will become "aac". We'll have failed to remove 'c' from the set.

Submit

Show Answer

AF & RI, part 3

1/1 point (graded)

Finally, which possible AF/RI pairs are consistent with this programmer's implementation of `contains()` ?

```
public boolean contains(char c) {
    for (int i = 0; i < s.length(); i += 2) {
        char low = s.charAt(i);
        char high = s.charAt(i+1);
        if (low <= c && c <= high) {
            return true;
        }
    }
    return false;
}
```

- [ ] SortedRep
- [x] SortedRangeRep
- [ ] NoRepeatsRep

☐ AnyRep

✔

Explanation

This version of contains() strongly assumes the SortedRange rep.

It isn't consistent with SortedRep. It may even throw an exception sometimes, when it reaches the end of an odd-length string and s.charAt(i+1) tries to access beyond the end of the string. But in many cases, it will just quietly give the wrong answer. For example, the SortedRep `"az"` is supposed to represent just `{a,z}` but this version of contains() will return true for all the letters from `a` to `z`.

It is consistent with SortedRangeRep -- this is how `contains()` should be written for that rep.

It isn't consistent with NoRepeatsRep for the same reason as SortedRep -- it may throw exceptions and give wrong answers.

It isn't consistent with AnyRep, again for the same reason.

Submit

Show Answer

Answers are displayed within the problem

Previous Next