Questions

Bookmark this page

immutable shapes

3/3 points (graded)

Let's define an interface for rectangles:

```
/** An immutable rectangle. */
public interface ImmutableRectangle {
    /** @return the width of this rectangle */
    public int getWidth();
    /** @return the height of this rectangle */
    public int getHeight();
}
```

It follows that every square is a rectangle:

```
/** An immutable square. */
public class ImmutableSquare {
    private final int side;
    /** Make a new side x side square. */
    public ImmutableSquare(int side) { this.side = side; }
    /** @return the width of this square */
    public int getWidth() { return side; }
    /** @return the height of this square */
    public int getHeight() { return side; }
}
```

Does `ImmutableSquare.getWidth()` satisfy the spec of `ImmutableRectangle.getWidth()` ?

Yes ✔ **Answer:** Yes

Does `ImmutableSquare.getHeight()` satisfy the spec of `ImmutableRectangle.getHeight()` ?

Yes ✔ **Answer:** Yes

Does the whole `ImmutableSquare` spec satisfy the `ImmutableRectangle` spec?

Yes ✔ **Answer:** Yes

Explanation

`ImmutableSquare` is a subtype of `ImmutableRectangle` and it implements the interface appropriately.

Submit

Show Answer

Answers are displayed within the problem

mutable shapes

4/4 points (graded)

Now let's consider a mutable rectangle:

```
/** An mutable rectangle. */
public interface MutableRectangle {
    // ... same methods as above ...
    /** Set this rectangle's dimensions to width x height. */
    public void setSize(int width, int height);
```

Surely every square is still a rectangle?

```
/** A mutable square. */
public class MutableSquare {
    private final int side;
    // ... same constructor and methods as above ...
    // TODO implement setSize(..)
```

For each possible `MutableSquare.setSize(..)` implementation below, is it a valid implementation?

1.

```
/** Set this square's dimensions to width x height.
 *  Requires width = height. */
public void setSize(int width, int height) { ... }
```

No -- stronger precondition ✔ **Answer:** No -- stronger precondition

2.

```
/** Set this square's dimensions to width x height.
 *  @throw BadSizeException if width != height */
public void setSize(int width, int height) throws BadSizeException { ... }
```

Specifications are incomparable ✔ **Answer:** Specifications are incomparable

3.

```
/** If width = height, set this square's dimensions to width x height.
 *  Otherwise, new dimensions are unspecified. */
public void setSize(int width, int height) { ... }
```

No -- weaker postcondition ✔ **Answer:** No -- weaker postcondition

4.

```
/** Set this square's dimensions to side x side. */
public void setSize(int side) { ... }
```

Specifications are incomparable ✔ **Answer:** Specifications are incomparable

Explanation

In #1, the stronger requirement `width = height` violates the contract defined by `MutableRectangle.setSize(..)`.
In #2, the postcondition requires different behavior, incompatible with the original spec, when `width != height`. If `BadSizeException` is a checked exception, this error is caught statically by the compiler.
In #3, the spec doesn't impose a precondition, but instead provides a weaker postcondition.
In #4, instead of implementing the `setSize` method from `MutableRectangle`, this *overloads* that name with another, different method. If `MutableSquare` doesn't also implement the required 2-int-argument `setSize`, that's a static error.
Indeed, there is no correct way for `MutableSquare` to implement `MutableRectangle.setSize(..)` and **mutable square is not a subtype of mutable rectangle**.

Submit

Show Answer

Answers are displayed within the problem

Previous Next