Ask me anything

0 questions
0 upvotes
We are designing a hierarchy of classes for geometric shapes. Should Square inherit from Rectangle or vice versa?

- A Square is a type of Rectangle
- Square is a specific rectangle
- Square from Rectangle because every Square is a rectangle, however, not every rectangle is a square
- Square should inherit from rectangle
- an abstract class cannot be instantiated
- Yes because a Square is a Rectangle
<table>
<thead>
<tr>
<th>abstract class can have instance variables</th>
<th>an abstract class cannot be instantiated</th>
<th>unlimited interfaces can be implemented but only one class can be inherited from</th>
</tr>
</thead>
<tbody>
<tr>
<td>A class can implement multiple interfaces. By contrast you can only extend one superclass</td>
<td>you can only inherit from one abstract class but implement several interfaces</td>
<td>abstract class cannot be implemented</td>
</tr>
<tr>
<td>Abstract class can inherit another class using extends keyword and implement an interface. Interface can inherit only one interface. Abstract class can be inherited using extends keyword. Interface can only be implemented using implements key-word.</td>
<td>An interface would be more flexible, because you can implement multiple interfaces</td>
<td></td>
</tr>
</tbody>
</table>
Why is it better to declare method arguments using interfaces than (abstract or concrete) classes?

- not depending on a specific implementation
- more flexibility
- easier
- documentation

When a class implements an Interface, it must implement all the methods. When a class inherits from an abstract class, it doesn’t necessarily have to override them all.

If you only need one class with that behavior.

We might not need an instance of the object being passed in.
When is it unnecessary to define an interface for a class?

- if only one class would use this interface
- if only one class implements it
- Assertions become clearer
Why is a Null Object better than a null value?

- NullPointerException
- no null check necessary
- it's better to handle null values as they can cause Null...Exception
Why do we refactor the TicTacToe game before adding new features, instead of adding the new features at the same time?

- clean up class hierarchy -> more flexibility
- Changing too much at the same time just sounds like a huge mess. If an error occurs, we wouldn't know where it stems from.
- responsibility management
When would you use the automated “Extract Method” refactoring?

- Renaming
- Change input variables
- When the original method is too long -> extract helper methods
- Responsibility management
- Redistribute responsibilities
- If I need a similar method
- To make a messy method a little bit cleaner or help distribute tasks
Why is a “simple” refactoring like renaming a method not really so simple?

- self explaining names
- must make certain that the new name is not already in use
- if two classes have the same method
- if it is overridden or overloaded it can get confusing
Last chance for questions