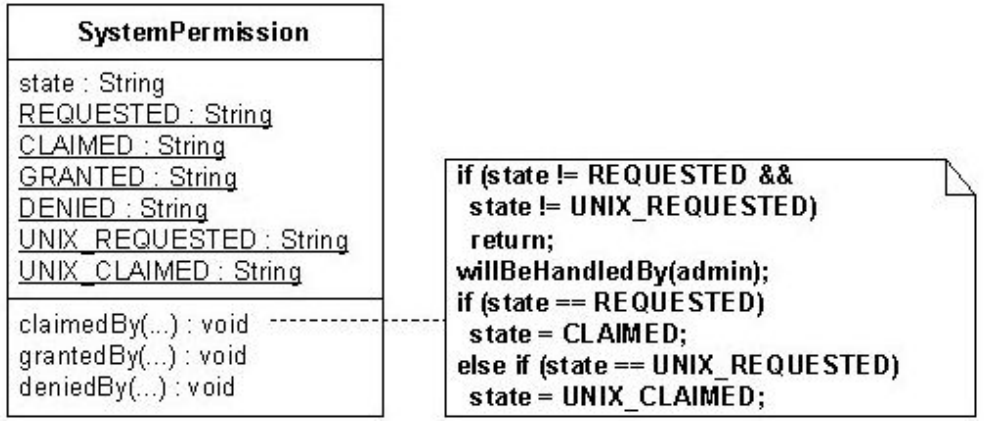
Replacing State-Altering Conditionals with State

Sam Poquette
Jon Schulberger

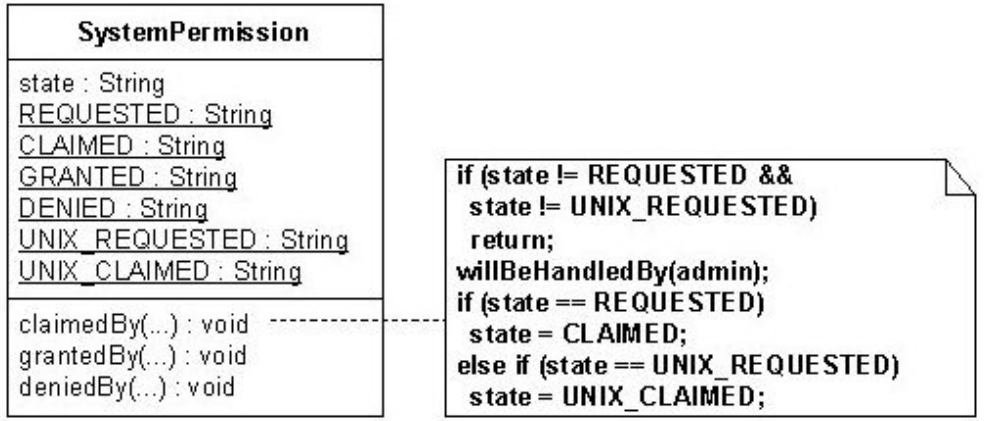
What's the issue?

When an object's state and the transitions between them are controlled by complex conditional statements.



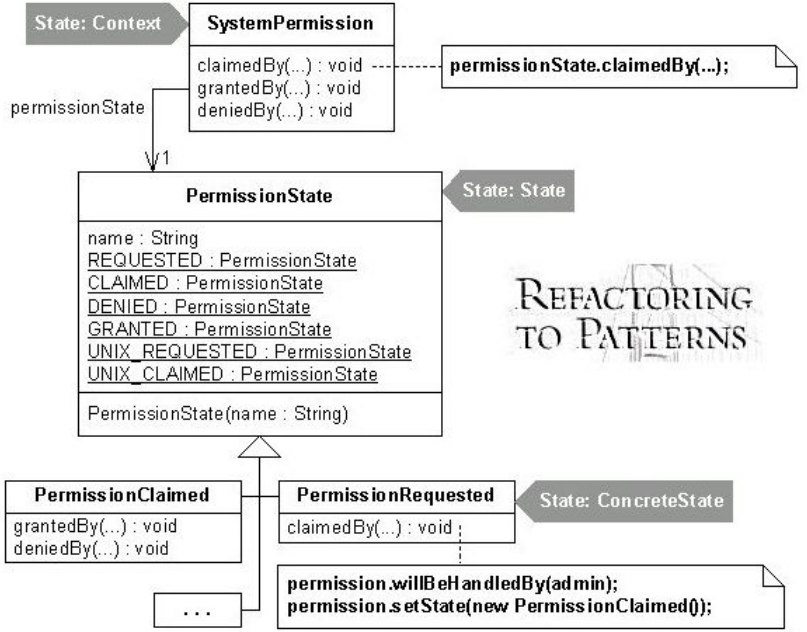
What to do

Replace the conditionals with State classes that handle specific states and transitions between them.



What to do

Replace the conditionals with State classes that handle specific states and transitions between them.



When should we not use state?

If the conditionals are simple and easy to understand, then using the state pattern will only make it more complex.

If the state only applies to a single algorithm, it would make more sense to use strategy rather than state.

Conditional code

Solution (summarized)

- Pull all state-related behavior into an abstract class
- Extract state-related methods and create default methods
- Create subclasses for each state
- Override only the required inherited methods
- Modify test class to account for state

Refactored code

Questions

- What are some other potential uses for this refactoring technique?
- Are there any cases where the default behavior should not be to error out?
- When should the state pattern be used instead of a basic class to replace type code?

References

- https://industriallogic.com/xp/refactoring/alteringConditionalsWithState.html
- http://www.informit.com/articles/article.aspx?p=1398607&seqNum=4
- https://www.codeproject.com/Articles/27515/Replace-State-Altering-Conditionals-with-State
- https://refactoring.guru/replace-type-code-with-state-strategy
- http://blog.xebia.in/index.php/2015/09/07/scaling-state-altering-conditional-logic-with-state-pattern