**Parking System Use Cases Reflection**

Software Design & Programming

Object-Oriented Mthd &Pgm I

Joseph DiBiasi

University of Denver College of Professional Studies

9/19/2025

Faculty: Nirav Shah

**Introduction**

Designing use cases for a university parking system involves a few base scenarios with several variations on each. The core concept of the parking system is registering a car with the parking office to receive a permit, scanning this permit to record parking in a university parking lot, and the parking office sending a monthly bill to the customer with charges.

**Use Case Implementation**

Each of these use cases had alternative flows from their base usage. Customers were able to register multiple cars and receive multiple permits. I did not specify this in the use case diagram as I wanted to keep it simple and it seemed that registering multiple permits would be considered the same workflow. Different parking lots had different styles of charging the permits. The base flow for parking in these lots involved scanning on entry to the parking lot, as that was always required; the alternative flow was also scanning that permit on exit from the parking lot. This alternative flow was extended from the base flow as a parking lot that scanned a permit on exit would also be scanning that same permit on entry. Therefore the initial permit functionality would be extended to include extra steps. The charging of the permit on a temporal basis also varied based on these flows. I decided to display this difference in charging rate as an includes because each flow had a variation on the charge rate. Finally, the monthly bill sent to the customer also had its alternative flow modeled as an extend. This was something I was less sure of modeling in the diagram. If this were an application, this alternative flow would likely be an interface implemented on the monthly bill as opposed to the inheritance structure of the parking lot. That is because in this instance we want to modify the bill that is sent to the customer by collating all of their permit(s) charges and discounting the charges by 20% whenever it involved a compact car. It is possible this is better represented in a different way as a result.

**Dependencies and Assumptions**

Several dependencies are required for this parking system to function properly, chief among them is the hardware and software needed for this process to happen. If a customer is unable to register with the parking office, how are they supposed to park their car? What happens if the machine that scans permits for entry is down, will we ban customers from using the parking lot in question until it is fixed or do they get free parking? These dependencies that we rely on for this parking system to function all brought with them their own questions, so writing these use cases had to plan for these systems functioning when the use case was executed. Then there were the assumptions that we had to operate under to write our basic use cases. Customers cannot cheat the system was a big one; the 20% discount offered to compact car users was significant. Would we see customers in real life trying to cheat the system so that they can park SUVs but take advantage of the smaller rate? This would lead to a whole series of use cases on parking system enforcement that would have their own dependencies and assumptions; it was easier to assume our system was solid than to dive into so many unknowns. A final simpler consideration involved the daily rates. If a system charged a permit only on entry and on remaining overnight, it seemed safe to assume that it only charged daily rates and customers could leave that lot multiple times in a day and return. Parking lots that charge daily rates have mixed enforcement on this issue.

**Summary**

Developing use cases for a university parking system is something that starts simple but can easily scale to be massively complex. Even just 3 simple use cases can have alternative flows, dependencies, and assumptions that add much to their complexity when included. This complexity needs to be balanced as a use case that is too bloated is better served being broken down into multiple smaller use cases.

**References**

Agile Modeling. 2025. *UML Use Case Diagrams: An Agile Introduction* Ambysoft Inc. Accessed September 19, 2025. <https://agilemodeling.com/artifacts/useCaseDiagram.htm>.

Agile Modeling. 2025. *UML Use Case Diagrams: Diagramming Guidelines* Ambysoft Inc. Accessed September 19, 2025. https://agilemodeling.com/style/usecasediagram.htm.