36.9. Example: Monopoly

First, let's briefly review the new domain rules and requirements in iteration-3: If a player lands on a property square (a lot, railroad, or utility) then they buy it if they have enough cash and it's not owned. If it is owned by another player, they pay rent according to square-specific rules.

Let's also review the essential design, as shown in Figure 36.23 and Figure 36.24. Polymorphism is applied; for each kind of square that has a different landed-on behavior, there is a polymorphic *landedOn* method. When a *Player* software object lands on a *Square*, it sends it a *landedOn* message.

Figure 36.23. DCD for the polymorphic *landedOn* design strategy.

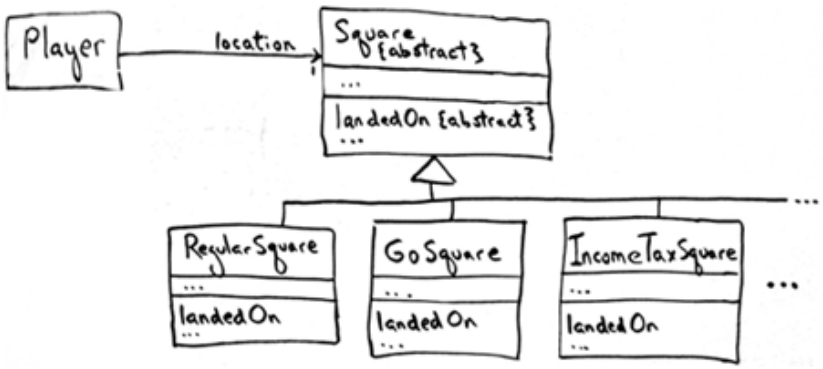
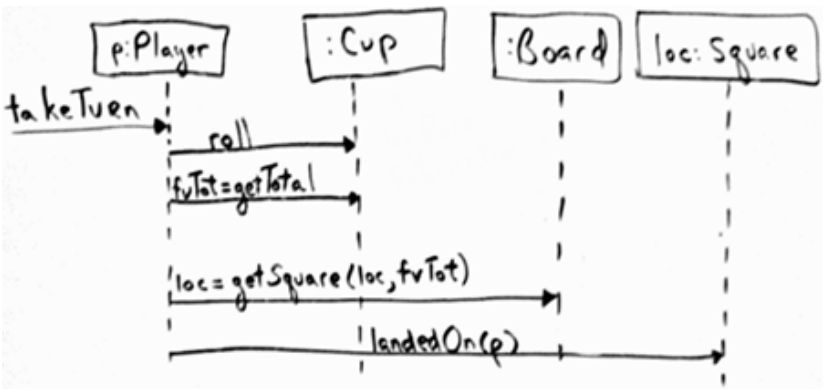


Figure 36.24. Dynamic collaborations for the *landedOn* design strategy.



The existing design shows off the beauty of polymorphism to handle new, similar cases. For this iteration, we will simply add new square types (*LotSquare, Rail-RoadSquare, UtilitySquare*) and add more polymorphic *landedOn* methods.

Figure 36.26. Attempting to purchase a property.

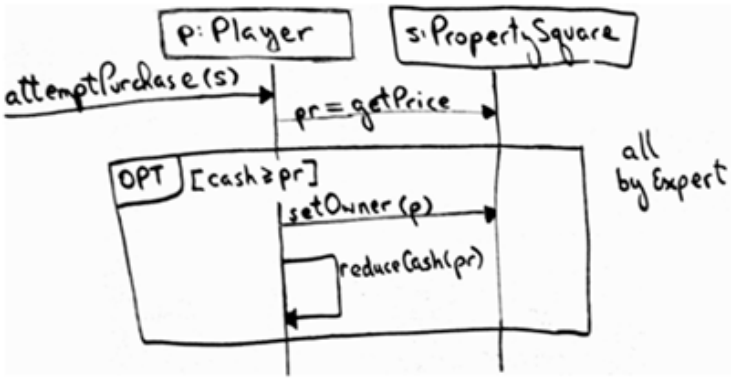


Figure 36.27. Paying rent.

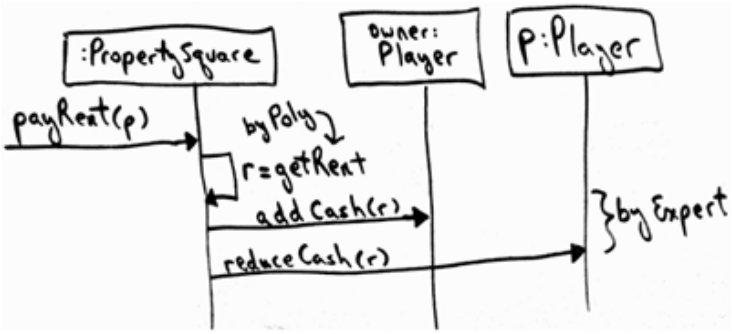
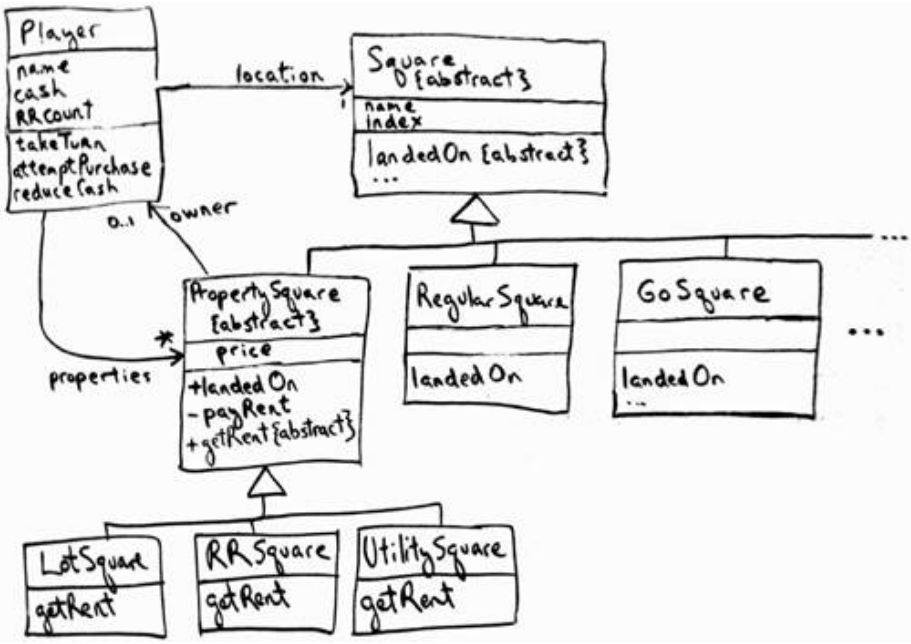


Figure 36.29. Partial DCD for iteration-3 of Monopoly.



Notice in Figure 36.25 that all the *PropertySquares* have identical *landedOn* behavior, so this method can be implemented once in the superclass and inherited by the subclasses of *PropertySquare*. The only behavior that is unique to each subclass is the calculation of the rent; thus by the Polymorphism principle, there is a *getRent* polymorphic operation in each subclass (see Figure 36.28).

Figure 36.25. Landing on a PropertySquare.

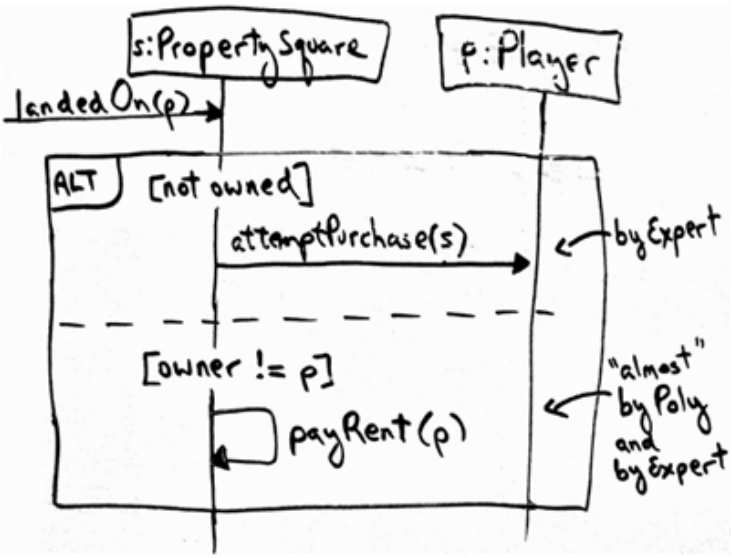


Figure 36.28. Polymorphic *getRent* methods.