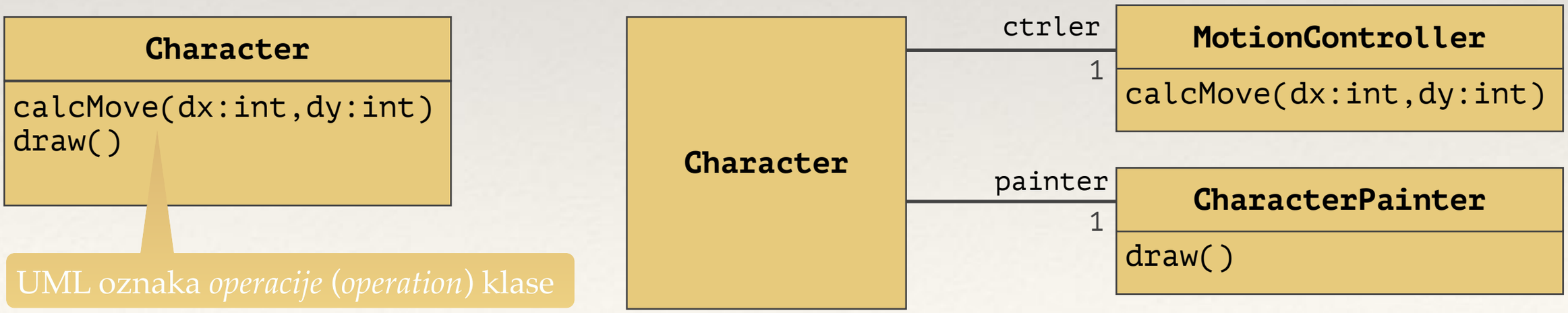
Raspodela odgovornosti

- Na primer, logično je da u odgovornosti klase za lika u arkadnoj igrici bude izračunavanje njegovih sledećih koordinata tokom kretanja po polju, kao i iscrtavanje tog karaktera na polju; međutim, ako je neka od ove dve odgovornosti suviše složena (npr. zavisi od složenih stanja, struktura ili algoritama), ili se mogu nezavisno kombinovati, treba razmisliti o njihovom razdvajanju u različite klase: lik, kao apstrakcija, može imati, kao deo svoje implementacije, objekte klasa zaduženih za izračunavanje pomeraja i za iscrtavanje
- Objekat klase *Character delegira* (*delegates*) odgovornost za ove obaveze tako što poziva odgovarajuće operacije tih pridruženih objekata kada mu je potrebno izračunavanje pomeraja ili iscrtavanje
- Na taj način, način izračunavanja pomeraja i/ili iscrtavanja može se nezavisno menjati specijalizacijama ovih klasa, jer navedeni pozivi mogu biti polimorfni
- Naravno, sve ovo ima smisla samo ako su ove promene predvidive: nema potrebe raditi dodatan posao i činiti softver složenijim ako takve promene nisu očekivane. Čak i ako nisu predviđene, ukoliko je ispoštovana enkapsulacija apstrakcije lika, softver se može *refaktorisati* (*refactoring*) sa ciljem ovakve dekompozicije



Raspodela odgovornosti

- Jedan od osnovnih principa softverskog inženjerstva, srodan sa navedenim, jeste princip *lokalizacije projektnih odluka* (*localization of design decisions*): manifestacija neke projektne odluke u programu treba da bude *lokalizovana* na jednom mestu, a ne rasuta na više mesta
- Ovo stoga što bi promena te odluke bila teška za sprovođenje ukoliko su njene manifestacije rasute po programu: izmena je teška i podložna greškama, a rizik domino efekta veliki; lokalizacija projektne odluke olakšava njenu promenu i povećava šansu da ta promena prođe bez problema
- Projektna odluka može da bude različite vrste, složenosti i nivoa apstrakcije: od najprostije odluke da je neka promenljiva nekog tipa ili neka konstanta ima neku vrednost, pa sve do odluke o načinu sprovođenja nekog scenarija u programu ili o arhitekturi datog softvera
- Najjednostavniji primer jeste korišćenje konstanti u programu:
  - korišćenje neposrednog literala (npr. broja 100) za neku konstantu (npr. dimenziju nekog niza) na više mesta u kodu (npr. prilikom deklarisanja niza i prilikom iteriranja kroz niz do njegove granice) predstavlja prestup ovog principa: promenu odluke da niz bude baš te veličine na neku drugu: a) zahteva prolazak kroz kod i sva mesta korišćenja tog literala i pažljivu zamenu drugom vrednošću, b) podložno je greškama, jer se na nekom mestu ista ta vrednost (npr. 100) može koristiti sa potpuno drugim značenjem, i ovakvom zamenom pogrešno promeniti
  - umesto toga, uvođenje simboličke konstante koja ima željenu vrednost predstavlja doslednu primenu ovog principa: odluka da je data dimenzija baš određena je lokalizovana na jednom mestu - u deklaraciji te konstante
- Naravno, i ovo ima ograničene domete i važi samo za odluke koje imaju perspektivu da se promene: celokupnu arhitekturu datog softvera nije lako promeniti, dok se neke činjenice verovatno neće promeniti za života softvera (npr. činjenica da sat ima 60 minuta ili sedmica 7 dana itd.)
- Klasa može da bude način i mesto za lokalizaciju projektnih odluka određenog nivoa granularnosti i apstrakcije: raspodela neke odgovornosti u neku klasu jeste jedna pojava lokalizacije projektne odluke