Vysvětlení úprav v implementaci

Původní implementace již využívala kompozici objektů (třída Faktura obsahovala objekty tříd Prijemce a CastkyDokladu), ale nyní jsem přidal následující funkce k třídě Faktura:

- Zprostředkování dílčích hodnot - Do třídy `Faktura` jsem přidal metody, které delegují volání na metody kompozitních objektů. Například metoda `getJmenoPrijemce()` volá metodu `getJmeno()` na objektu prijemce. Tímto způsobem třída `Faktura` zprostředkovává přístup k dílčím hodnotám svých komponent.
- Aktualizace testů - V testovací třídě jsem přidal nový test `testDilciHodnoty()`, který demonstruje přímý přístup k dílčím hodnotám přes delegované metody třídy `Faktura`.
- UML diagram - Vytvořil jsem UML diagram, který znázorňuje vztahy mezi třídami v našem řešení. Kompozice je znázorněna plnou čárou s diamantem na straně celku.

Změny v implementaci testů

- S přidáním delegovaných metod do třídy Faktura se mění i způsob, jakým můžeme testy implementovat. Nyní můžeme testovat dílčí hodnoty přímo přes rozhraní třídy Faktura, aniž bychom museli pracovat s jejími vnitřními objekty.

Hlavní rozdíly

Původní přístup (bez delegace):

```
javaFaktura faktura = new Faktura(...);
String jmeno = faktura.getPrijemce().getJmeno();
double cenaBezDph = faktura.getCastkyDokladu().getCenaBezDph();
```

Nový přístup (s delegací):

```
javaFaktura faktura = new Faktura(...);
String jmeno = faktura.getJmenoPrijemce();
double cenaBezDph = faktura.getCenaBezDph();
```

Tento nový přístup má několik výhod:

- Jednodušší API - Klient může přistupovat k dílčím hodnotám přímo přes rozhraní třídy `Faktura`, což je jednodušší a přímější.
- Abstrakce implementace - Klient nemusí znát vnitřní strukturu objektu `Faktura`.
- Flexibilita změn - Pokud se v budoucnu změní implementace vnitřních objektů, klientský kód nemusí být měněn.