CodeArt Web Code is Art

Apstraktne klase i interfejsi

April 4th, 2010 by Senad Crljenković Leave a reply »

Obećao sam u jednom od prethodnih postova, da ću pojasniti dilemu korištenja **interfejsa** u odnosu na **apstraktne klase**. U osnovi, interface i apstraktna klasa su slični, ali ipak se koriste na različite načine i za različite namjene, te se mogu izvući određene smjernice kada koristiti jedno a kada drugo.

Ukoliko naše klase nasljeđuju apstraktnu klasu, u mogućnosti smo da na jednostavan način imamo ponovnu upotrebu koda (*code-reuse*). Određene funkcionalnosti možemo imati dijeljene unutar apstraktne klase i u mogućnosti smo pružiti **podrazumijevanu (default) implementaciju**.

S druge strane, interfejsi popunjavaju nedostatak korištenja višestrukog nasljeđivanja u nekim jezicima – **klasa može implementirati koliko je potrebno interfejsa, a naslijediti samo jednu klasu.** Ovdje neću uzimati u obzir mogućnost korištenja višestrukog nasljeđivanja, pošto je prisutna tendencija onemogućavanja istog u novijim jezicima, a poznati su problemi korištenja višestrukog nasljeđivanja u jezicima koji ga podržavaju – stoga se ono ionako ne preporučuje.

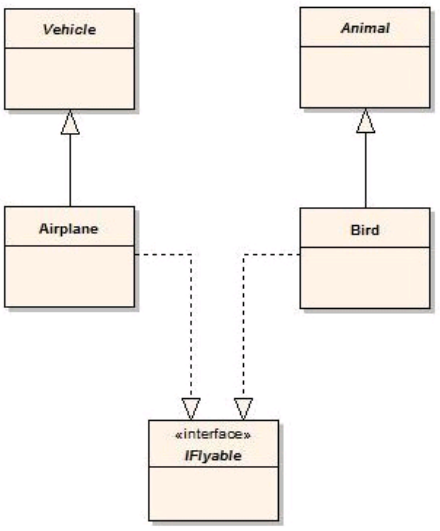
Interfejs ne pruža mogućnost korištenja zajedničke baze koda – on je jednostavno **ugovor** koji propisuje šta klase koje ga implementiraju moraju podržati, ali na svakoj klasi ostaje da za sebe implementira datu funkcionalnost. Zbog toga, može izgledati praktičnije kada god imamo potrebu za dijeljenjem koda, koristiti apstraktnu klasu, ali to nije tako. Nasljeđivanje, ako se koristi neispravno, može rezultirati vrlo lošim dizajnom. Code-reuse **nije** dovoljan razlog za preferiranje apstraktne klase nad interfejsom – jer isto to možemo ostvariti na mnogo fleksibilniji način korištenjem kompozicije [http://codeart.tech.ba/2010/03/11/naslijediti-ili-sastaviti/] .

S druge strane, korištenje interfejsa uvodi dodatni problem. Kako isti ne podržavaju mogućnost podrazumijevane implementacije, svaka promjena interfejsa, zahtijeva promjenu postojeće implementacije. Ako dodamo novi metod na interfejs, sve klase koje su izvedene iz tog interfejsa moraju implementirati taj metod. To može biti poseban problem kod frameworka. Recimo da imate klijente koji koriste vaš framework, i implementiraju određeni interfejs(e). Oni moraju izmijeniti/kompajlirati/redeployati sve klase koje su izvedene iz tog interfejsa kad god dodate novi metod na interfejs. Kod apstraktnih klasa to možete izbjeći praveći virtuelni metod i pružajući podrazumijevanu implementaciju.

Iz svega ovoga može se zaključiti da apstraktne klase i interfejsi, ponaosob, imaju svoje prednosti i nedostake u određenim okolnostima. Interfejsi imaju svoju namjenu, ali ne mogu izbaciti potrebu za nasljeđivanjem, kao osnovnim konceptom OOP-a.

Kada onda koristiti apstraktnu klasu, a kada interfejs?

- Ako između dvije apstrakcije imamo jasnu i čistu klasifikacionu "**je**" (***is a***) vezu, odnosno, ako možemo reći da "B **je** A", onda B treba da nasljeđuje iz A, i A je kandidat za apstraktnu baznu klasu. Primjer: pas **je** sisar.



[http://codeart.tech.ba/wp-content/uploads/2010/04/abs_interfaces.jpg]

- Ako pak, želimo označiti da neki objekat ima određenu **mogućnost** (*ability*), koristićemo interfejs, kao atributsku "je" vezu. Pri tome, više konceptualno nepovezanih klasa može koristiti isti interfejs. Primjer, ptica može letjeti i avion može letjeti. Ni u kom slučaju oni ne trebaju biti izvedeni iz neke zajedničke klase. U ovom slučaju, kandidat za interfejs je mogućnost letenja. Označava zajedničku osobinu među pomenutim klasama, iako su klase međusobno nepovezane (ptica je životinja, avion je vozilo). Ove klase mogu imati i proizvoljan broj drugih interfejsa, a vrlo vjerovatno će imati kao bazu – apstraktnu klasu životinja i vozila. Bitno je da će klijent moći posmatrati ove obje klase kao "***Flyable***" i polimorfno pozivati na njima metod **Fly()**, a pri tome ga ne treba zanimati da li se radi o ptici ili avionu. Svaka od njih ponaosob će implementirati svoj način letenja. U mnogo slučajeva za implementaciju je uputno koristiti kompoziciju, posebno ako postoje zajednički implementacijski detalji.

- Previous Entry: Kupovina preko Amazona
- Next Entry: POCOYO

Posted in OO koncepti
Tags: apstraktna klasa interface nasljedjivanje
You can follow any responses to this entry through the RSS 2.0 Feed. Both comments and pings are currently closed.

2 komentar(a)

Dodaj komentar

Muhamed A. *05/10/2010 at 9:29 pm*

Odličan članak!

Senad Crljenković *21/01/2011 at 9:41 pm*

Hvala.

Natrag na vrh