Sveučilište J.J. Strossmayera u Osijeku
Odjel za matematiku
Sveučilišni diplomski studij matematike i računarstva

Kristina Nikolić

Povijest objektno-orijentiranih programskih jezika

Seminarski rad

Nositelj kolegija : doc. dr. sc. Alfonzo Baumgartner

Osijek, 2016.

Sadržaj

1 UVOD

Objektno orijentirano programiranje postoji od 1960-ih godina. Iako je nastalo prije mnogo godina, još uvijek postoje uređaji koji su programirani i koji se održavaju u funkcionolnim jezicima, a razlog tome je taj što oni rade sasvim dobro te nema razloga prouzrokovati potencijalnu katastrofu mijenjajući ih.

U posljednjih 10 godina, uslijed razvoja Weba, objektno orijentirani programski jezici dobili su puno bolju poziciju među ostalim programskim jezicima budući da u ovom području nije postojalno ništa što se trebalo održavati onakvim kakvim je bilo do sada.

Objektno-orijentirano programiranje postepeno je, u određenim segmentima, zamijenilo dotadašnje proceduralno programiranje zbog prednosti koje nudi, ali i zbog činjenice da je objektno-orijentirano programiranje jednostavnije za shvatiti iz razloga što je ljudima prirodno razmišljati u vidu objekata.

2 Povijest objektno-orijentiranog programiranja

2.1 Prvi objektno-orijentirani programski jezici

Ideja o objektno-orijentiranim jezicima pojavila se 1960ih godina kada je norveški znanstvenik iz područja računarstva, Kirsten Nygaard(1926-2002), trebao bolji način za opisivanje heterogenosti operacijskih sustava prilikom pisanja simulacija računalnih programa. U svrhu toga, počeo je razvijati prvi objektni jezik, nazvan Simula. Ubrzo mu je u razvoju navedenog jezika pridružio i kolega, Ole-Johan Dahl(1931-2002). Nygaard i Dahl, na osnovu ALGOL-a 60, u Norveškom računarskom centru u Oslu razvili navedeni jezik.

Simula je ime dva simulacijska programska jezika Simula I i Simula 67. Simula 67 predstavila je objekte, klase, nasljeđivanje i podklase, virtulane procedure, subrutine itd.

Ovaj programski jezik smatra se pretečom i bazom za razvoj objektno-orijentiranih programskih jezika, kao što su C++, Object Pascal, Java, C#.

Nekoliko godina kasnije, američkom znanstveniku, Alanu Keyu (1940 -), koji je u to vrijeme bio na Sveučilištu u Utah-u, svidjelo se što je vidio u Simula-i. On je imao viziju osobnog računala koje bi imalo grafički orijentirane aplikacije i smatrao je da bi jezik kao što je Simula mogao ostvariti to da ljudi koji nisu stručnjaci stvore takve aplikacije.

Budući da je Alan Key imao diplomu iz matematike i biologije, pokušao je svoja znanja objediniti sa svojom vizijom. Naime, Key je o objektima razmišljao kao o stanicama i/ili zasebnim računalima u mreži koji međusobno mogu komunicirati jedino preko poruka. Također, želio se riješiti podataka na način da "<-" pridruživanje bude samo još jedan token poruke. Sam je rekao da mu je trebalo puno vremena da smisli simbole kao što su imena za funkcije i procedure.

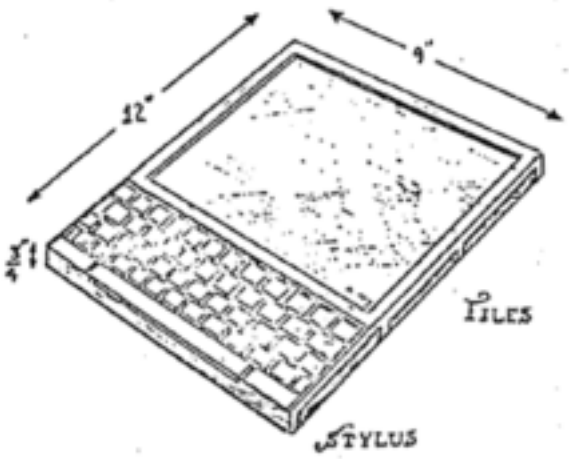
Zbog svoje matematičke pozadine dosjetio se da bi svaki objekt mogao imati nekoliko cjelina koje bi bile povezane s njim i da bi oni mogli tvoriti familije te da bi to bilo jako korisno.

Svoju viziju prodao je Xerox Parc-u i u ranim 1970ima, tim predvođen Alanom Keyom u Xerox Parcu stvorio je Smalltalk. Smalltalk je bio objektno-orijetnirani jezik razvijen za programiranje Dynabooka (osobnog računala koje je prema Keyovoj ideji nalikovalo na današnje tablete).

Na pitanje što za njega znači objektno-orijentirano programiranje, Alan Key je u jednom intervjuu rekao sljedeće:

"Objektno-orijentirano programiranje za mene znači prenošenje poruka, lokano zadržavanje i zaštita te skrivanje stanja procesa i ekstremno kasno povezivanje svih stvari. Ono može biti napravljeno u Smalltalku ili u LISPu. Vjerojatno postoje i drugi sustavi u kojima to može biti ostvarno, ali meni oni nisu poznati."

Smalltalk je objektno orijenitrani, dinamički pisan programski jezik. To je jedan od prvih objektno orijentiranih jezik dizajniran i stvoren djelomično za edukacijske svrhe, ali iz konstruktivno učenje unutar Learning Research Group-a (LRG) u Xerox PARC. Razvili su



Slika 2.1: Originalna ilustracija Dynabooka, rad Alana Keya iz 1972.

ga Alan Key, Dan Ingalls (1944 -), Adele Goldberg(1945 -), Ted Kaehler, Scott Wallace i drugi 1970ih godina.

Postoji puno varijanti Smalltalka, no najčešće se govori o Smalltalk-80 jeziku čija je prva verzija postala javno dostupna 1980.

Smalltalk je proizvod istraživanja predvođenog Alanom Keyom u Xerox Palo Alto Research Centru. Alan Key je dizajnirao većinu rane verzije Smalltalk-a, dok je Dan Ingalls implementirao. Prvu verziju, poznatu kao Smalltalk-71, kreirao je sam Key u nekoliko dana zbog oklade da jezik inspiriran Simulom s idejom o prenošenju poruka može biti implenetrian sa stranicom koda.

Nako toga razvijeni su i Smalltalk-72, Smalltalk-76 te Smalltalk-80. Smalltalk-80 uveo je koncept „sve je objekt“ te je bio prva verzija jezika koji je postao dostupan izvan PARC-a i dan je na korištenje malom broju tvrtki (Hewlett-Packard, Apple Computer, Tekronix i DEC) i sveučilištu (UC Berkeley).

1998. ANSI Smalltalk je postavljen kao standard kada se govori o Smalltalku.

Kasnije se iz Smalltalka u Applu razvio Squeak, te se razvio i VisualWorks u Xerox PARCu. VisualWorks je podloga na kojoj su rađeni mnogi operacijski sustavi poput Windowsa, Mac OS X, Linuxa i nekih verzija Unixa.

Pod utjecajem Smalltalka, a posredno i Simula-e, kasnije su se razvili i drugi OO jezici poput Objective-C-a, Jave, Pythona, Ruby, itd.

Razlika između Smalltalka i Jave i C++-a je tu tome što u Smalltalku nema razlike između primitivnih tipova podataka i objekata. Ondje je i svaki primitivni tip, poput integera, charactera također objekt.

2.2 Neki od popularnijih objektno-orijentiranih programskih jezika

Objektno-orijentirani programski jezici bazirani su na konceptu objekata. Objekti kao takvi imaju atribute i (operacije) metode koje mogu izvršavati. Objekti pomoću svojih procedura mogu mijenjati vrijednosti atributa objekata koji su povezani s njima - objekti koriste anotacije "this" i "self". U objektno-orijentiranom programiranju, programi su dizajnirani na način da objekti mogu biti u interakciji jednis drugima. Postoji jako puno objektno orijentiranih programskih jezika, ali najpopularniji su oni koji su bazirani na klasama.

Neki od popularnijih programskih jezika koji su zasnovani na objektno-orijenitranim programskim jezicima su Python, C++, Objective-C, Java, Swift, C#, Perl, Ruby i PHP.

U osnovi, svi navedeni programski jezici koirste strukturu objekta. Objekti imaju atribute(varijable) i metode. Varijable pohranjuju informacije, dok metode primaju ulazne parametre, vraćaju izlazne i manupiliraju varijablama.

3 Osnovna svojstva objektno-orijentiranih programskih jezika

3.1 Što je to objekt?

Na pitanje "Što je objekt?" istovremeno može biti i jednostavno i kompliciranо dati odgovor.

Naime, jednostavno je iz razloga što većina ljudi i razmišlja na način objekata. Jednako tako, kompliciranо je jer taj isti pojam sada treba pretočiti na razinu učenja.

Primjerice, kada pogledamo automobil, vidimo automobil kao objekt. Objekt je definiran s 2 termina - atributima i ponašanjem. Jednako tako, i automobil ima atribute, poput boje, motora, marke, tipa itd. Osoba, osim toga, ima i ponašanja, poput primjerice vožnje, trubljenja, upaljenih svjetala i tome slično. U svojoj osnovonoj definiciji, objekt je enititet koji sadrži oboje - i atribute i ponašanje. Upravo u ovome je razlika između proceduralnog i objektno orijentiranog programiranja.

U proceduralnom programiranju, kod je smješten u potpuno neovisne funkcije ili procedure. Idealno, ove procedure su "crne kutije" u koje unosimo ulazne i dobivamo izlazne parametre.

U objektno orijentiranom programiranju nema pojmova poput globalne varijable. Objekti su puno primitivniji tipovi podatka poput integera i stringova koji predstvaljaju atribute i također mogu sadržavati metode koje predstavljaju ponašanje. U objektu se mogu koristiti metode za upravljanje podacima.

Budući da objektno orijentirani jezici rade po principu „sve je objekt“, za razliku od funkcionalnih jezika kao što je C koji imaju samo jedan thread kontrole, objektni jezici omogućuju kontrolu unutar objekata. Dakle, u svakom trenutku, više objekata može izvršavati operaciju (barem je to tako zamišljeno, računalo to odrađuje na način koji mi hardware dopušta). Objekti međusobno komuniciraju prenošenjem poruka. Poruka je jednostavno poziv metode drugog objekta.

3.2 Osnovna svojstva OOP

Odlična analogija na ono kako su zamišljeni objektno-orijetirani programski jezici je radio. Svrha radija je da pretvara odaslane signale u zvukove koje ljudi mogu čuti. Radio ima različite tipke koje nam dopuštaju da postavimo frekvenciju stanice koja trenuto svira, da promijenimo glasnoću, bass, upalimo/ugasimo radio itd. Ove tipke predstavljaju operacije kojima možemo upravljati radijem. Međutim, implementacija radija skrivena je od nas. Poanta je da mi ni ne moramo znati tu implementaciju. Činjenica da je implementacija skrivena od nas omogućuje proizvođaču da nadogradi tehnologiju bez da ponovo mora učiti korisnike kako da koriste radio.

Ideja s objektima je jednaka. Objekt je predstavljen kao skup funkcija koje nešto izvršavaju, ali ne otkrivaju svoju implementaciju. To možemo smatrati crnom kutijom. Primjerice,

promotrimo primjer stoga. Stog nudi operacije poput push, pop, isempty i top. Ako je stog implementiran kao objekt, implementacije će biti skrivena od programa. Stog može biti implemntiran kao polje, red ili neka druga struktura podataka. Jedino što zanima program je to da stog nudi specificirane operacije koje osiguravaju željeno ponašanje.

Skup operacija koje nudi objekt zove se interface. Interface definira imena operacija i kako će se operacije ponašati. Suština interfacea je da djeluje između objekata i programa koji koriste objekte. Objekt jamči da će pružiti sve operacije koje treba te da će se one ponašati na zadani način. Zbog strukture objekta, sama njegova implementacija može se izmjeniti bez da to utječe na ponašanje programa. Primjerice, možemo zamijentiti implementaciju stoga pomoću polja sa stogom pomoću reda bez da utječemo na ponašanje programa.

Nadalje, postoje i klase. Na klase možemo gledati kao na tvornice koje u sebi mogu imati instancirano jako puno objekata. Programer stvara klasu na način da u nju postavi željene objekte koji se mogu sastojati od:

1. deklaracije skupa varijabli koje će objekt sadržavati
2. deklaracije skupa operacija koje će objekt nuditi
3. skupa funkcija koje implementiraju svaku od tih operacija

Skup varijabli nazivamo instance varijabli, dok skup operacija koje nudi objekt zovemo se metode.

Kada program želi napraviti novu instancu objekta, traži od određene klase da stvori objekt za nju. Klasa zauzima memoriju u kojoj čuva instancu varijabli i vraća objekt programu. Svaki objekt zna koja klasa ga je stvorila tako da kada je određena operacija zatražena od objekta, on može pogledati funkciju koja implemetira tu operaciju nad njegovim varijablama.

3.3 Nasljeđivanje

Objektno-orijentirano programiranje nudi i koncept nasljeđivanja. Vratimo se ponovo na analogiju s radijem, no ovaj put promatrajmo radio budilicu. Radio budilica ima sve iste funkcionialnosti kao radio uz dodatnu funkcionalnost da ima i funkciju budilice. Nasljedimo li funkcionalnost koju nudi radio unutar radio-budilice, osoba koja zna upravljati radijem, znati će upravljati i tim dijelom radio budilice. Dakle, umjesto da dizajniramo radio-budilicu od samog početka, mi proširujemo interface definiran za radio s funkcionalnostima koje nudi radio-budilica.

U objektno orijentiranom programiranju, nasljeđivanje znači da nasljeđujemo dio interfacea drugog objekta s mogućnošću da osim nasljeđivanja imena funkcija, nasljedimo i dio implementacije.

Nasljeđivanje se radi na način da kažemo da je nova klasa podklasa postojeće klase. Klasa iz koje nasljeđujemo zove se nadklasa(superklasa). Podklasa uvijek nasljeđuje cijeli interface nadklase. Ona može proširiti interface, ali ne može obrisati niti jednu operaciju koja postoji u interfaceu nadklase. Osim toga, podklasa nasljeđuje i sve implementacije nadklase, drugim

riječima, funkcije koje implemetiraju operacije nadklase.

Međutim, podklasa može definirati nove implementacije funkcija. To se zove „overriding“ implementacije nadklase. Podklasa može selektivno odabrati koje će implementacije nadklase overrideati, a koje nasljediti.

4 ZAKLJUČAK

Budući da živimo u vrijeme kada su objektno-orijetnirani programski jezici standard, mislim da bismo trebali iskoristi prednosti koje oni nude. Objektno-orijentirani programski jezici se danas koriste u velikom dijelu razvoja aplikacija te smatram da bi svatko tko ima želju baviti se programiranjem u današnje vrijeme trebao znati osnovne koncepte OOP.

Osim toga, kako su svi objektno-orijentirani programski jezici zasnovani na istoj ideji, oni su, generalno gledajući, vrlo slični. Razlike, međutim, i dalje postoje te se stoga ponekad nije jednostavno samo prebaciti iz jednog programskog jezika u drugi.

Smatram da je poznavanje koncepata na kojima je razvijeno objektno-orijentirano programiranje te znanje programiranja u barem jednom objektno-orijentiranom programskom jeziku, jako dobra podloga za relativno brzo savladavanje ostalih objektno-orijentiranih programskih jezika.

Literatura

[1] http://web.eecs.utk.edu/ huangj/CS302S04/notes/oo-intro.html, 30.05.2016.

[2] http://userpage.fu-berlin.de/řam/pub/pub_jf47ht81Ht/doc_kay_oop_en, 03.06.2016.

[3] https://en.wikipedia.org/wiki/Object-oriented_programming, 30.05.2016.

[4] https://en.wikipedia.org/wiki/Simula, 31.05.2016.

[5] https://en.wikipedia.org/wiki/Smalltalk, 31.05.2016.

[6] http://www.developer.com/lang/article.php/10924_3304881_2/The-Object-Oriented-Thought-Process.htm, 30.05.2016.