Primjena OOP principa u PHP programskom jeziku

Begović, Nikolina

Sveučilište J.J. Strossmayera u Osijeku

Filozofski fakultet

Preddiplomski studij Informatologije

Nikolina Begović

Primjena OOP principa u PHP programskom jeziku

Završni rad

Mentor doc.dr.sc., Boris Badurina

Osijek, godina 2015.

Sadržaj

SAŽETAK

Objektno orijentirano programiranje (ili OOP) jedan je od mogućih pristupa programiranju koji je računalnim stručnjacima pružio novu sintaksu, ali i novi način razmišljanja o programskim problemima. Dva temeljna pojma OOP-a su klase i objekti. Klasa je razred koji definira varijable i metode zajedničke skupini objekata, a objekti su instance klase. Objekt može biti bilo koji predmet ili pojam iz stvarnog ili apstraktnog svijeta. Načela koja ga karakteriziraju su učahurivanje, nasljeđivanje i polimorfizam. Učahurivanje (ili enkapsulacija) odnosi se na štićenje podataka koji nisu namijenjeni za korištenje i mijenjanje iz drugih klasa. Tri tipa vidljivosti pomoću kojih se učahurivanje koristi su javna, zaštićena i privatna. Nasljeđivanje omogućuje da jedna klasa naslijedi sve metode, karakteristike i ponašanje neke druge klase, a polimorfizam programeru daje mogućnost definiranja nekoliko metoda istim imenom,a svaka kao parametre prima objekte različitih tipova. OOP principi nalaze svoju primjenu u brojnim programskim jezicima, a jedan od nih je i PHP programski jezik. Njegov naziv temelji se na rekurzivnoj definiciji PHP: Hypertext preprocessor, no ta definicija se gotovo više niti ne koristi. Svi principi objektno-orijentiranog programiranja su primjenjivi u PHP-u na te olakšavaju programerima proces razvoja aplikacija. Klase se kreiraju pomoću ključne riječi *class*, objekti pomoću riječi *new*. Vidljivost se definira riječima *public*, *protected* i *private*, ovisno o potrebi. Klase nasljeđuju pomoću riječi *extends,* a polimorfizam se očituje u definiranju funkcija istog imena, ali različitih svojstava. Trenutna inačica, PHP5, ima zaista dobru podršku OOP-a što mu omogućuje lakšu ponovnu upotrebu koda, bolju razvijenost timskog rada, neki uzorci vode do puno učinkovitijeg koda i više odgovara marketinškim potrebama.

Ključne riječi: objektno-orijentirano programiranje, PHP programski jezik, programska paradigma, objekt

1. UVOD

U ovo vrijeme intenzivnog razvoja računalnih tehnologija kada su svi okruženi računalima, pametnim mobitelima i tabletima, potreba za kadrom koji se bavi programiranjem je u stalnom porastu, a mogućnosti zapošljavanja su više nego dobre. Kako bi programer bio uspješan mora poznavati više programskih jezika, a koji će to biti ovisi o njegovim afinitetima. Također, ono što je bitno da svaki programer poznaje su programske paradigme koje mu daju više načina gledanja na problem i njegova rješavanja. U radu će se govoriti o objektno-orijentiranoj programskoj paradigmi jer se upravo ta paradigma nametnula kao najpoželjnijom među programerima u posljednje vrijeme. Smatra se da je to zbog njezine mogućnosti za ponovnu upotrebu i lakšeg razumijevanja za ljude zbog svojih koncepata klasa i objekata. Objektno-orijentirano programiranje u ovome će se radu prikazati kroz PHP programski jezik. Upravo taj jezik odabran je zbog svoje rasprostranjenosti među programerskim krugovima te podržava objektno-orijentiranu paradigmu u mnogim aspektima. Zbog toga je dobar programski jezik kroz koji se mogu detaljno prikazati i objasniti objektno-orijentirani koncepti. Koncepti koji će biti detaljno razrađeni su: klase, objekti, učahurivanje, nasljeđivanje i polimorfizam. Ti koncepti čine najvažniji dio objektno-orijentiranog programiranja koje programer mora razumjeti ako želi koristiti ovu paradigmu. Posao programera je posao u kojem se mora cijeli život učiti i nadograđivati svoje znanje zbog stalnog napretka u računalnim tehnologijama.

2. OBJEKTNO ORIJENTIRANO PROGRAMIRANJE

Kako bi programeri što manje vremena potrošili na pisanje koda, ispravljanje grešaka i ponavljanje istih poslova potreban im je pravi način razmišljanja, tj. prava programska logika. Postoji više programskih paradigmi koje im u tome pomažu, npr. strukturalna, proceduralna, imperativna, objektno-orijentirana paradigma itd.[1] U ovom radu govorit će se o objektno-orijentiranom programiranju jer je ta paradigma način razmišljanja prema kojem danas rade brojni programeri diljem svijeta.

Prvi programski jezik sa svojstvima objektno-orijentirane paradigme je SIMULA, čija je svrha izgradnja sustava za simulaciju, a prvi puta pojavio se 1967. godine. Sa SIMULA-om uveden je pojam klase (razreda) što je jedan od ključnih pojmova objektno-orijentiranog programiranja.[2] Idući bitan korak je razvoj Smaltalk-a 1972. godine. On je prvi „pravi" (čisti) objektno-orijentirani programski jezik, razvijen u laboratoriju Xerox PARC i najkorištenija verzija 80-ih. 1983. godine Bjorn Stroustrup u Bell Labs-u integrirao je objektno-orijentirano programiranje u programski jezik C. Rezultat toga bio je jezik C++ koji je postao prvi naširoko komercijalno korišteni objektno-orijentirani programski jezik.[3] Standardizacija je dovršena 1998., a 2003. izdana je standardna verzija s ispravljenim greškama.[4]

Uz objektno-orijentirano programiranje često se spominje i proceduralno programiranje, a ta dva pojma čine suprotne koncepte u programiranju. S oba pristupa, i proceduralnim i objektno-orijentiranim, mogu se riješiti problemi i izraditi učinkoviti programi, ali glavna razlika je u fokusu programera tijekom najranije faze planiranja faza projekta. Problemi kod proceduralnog programiranja su odvojenost ponašanja od podataka; strukture se koriste za sadržavanje podataka, a procedure za rad nad tim podacima. Zbog te odvojenosti iznimno je teško održavati i graditi velike sustave.[5] Koristeći objektno-orijentirani pristup problemu znači definiranje objekata kako bi se ostvario zadatak i razvijanje objekata kako bi svaki imao svoje vlastite podatke te izvodio zadatke kad drugi objekt to zatraži. Za objektno-orijentirano programiranje može se reći kako je „prirodnije" jer je ljudima prirodnije razmišljati o svijetu

[1] Usp. Programming Paradigms. URL: http://cs.lmu.edu/~ray/notes/paradigms/ (2015-08-24)
[2] Usp. Vanjak, Zvonimir. Uvod u objektno-orijentirano programiranje u programskom jeziku C++. URL: http://www.ieee.hr/_download/repository/ASP-Objektno.pdf (2015-08-15)
[3] Usp. A Brief History of Object-Oriented Programming. URL: http://web.eecs.utk.edu/~huangj/CS302S04/notes/oo-intro.html (2015-27-08)
[4] Usp. Vanjak, Zvonimir. Nav. dj.
[5] Usp. Cihlar, Davor. Osnove izrade PHP aplikacija : Uvod u OO PHP, 2012. URL: https://www.fer.unizg.hr/_download/repository/3._PHP.pdf (2015-08-03)

objekata i načinu njihove interakcije nego o svijetu sustava, stavki podataka i logici koja je potrebna za njihovu manipulaciju. [6]

Objektno-orijentirano programiranje predstavlja pokušaj da se programe učini bližima načinu na koji ljudi razmišljaju o svijetu. U starijim načinima programiranja, programer koji je suočen s nekim problemom morao bi identificirati računalni zadatak koji će se izvesti za rješenje problema. Programiranje se tada sastoji od pronalaženja slijeda instrukcija koje će postići taj zadatak. Ali, u objektno-orijentiranom programiranju umjesto zadataka nalaze se objekti – entiteti koji imaju svoje ponašanje, koji posjeduju informaciju te mogu imati interakciju jedni s drugima, no detaljnije o objektima će biti govora u daljnjem tekstu. Ovakvo programiranje sastoji se od dizajniranja setova objekata koji modeliraju dani problem. Objekti u programu mogu predstavljati stvarne ili apstraktne entitete u domeni problema. To bi trebalo učiniti dizajn programa prirodnijim i stoga lakšim za napisati ispravno i razumjeti ga. [7]

Osnovni koncept OOP-a je objedinjavanje podataka (varijabli) i operacija (funkcija) koje manipuliraju podacima u jednu cjelinu koja se naziva objekt. Program se sastoji od skupa objekata koji međusobnom komunikacijom rješavaju problem. Ovaj način razvoja pogodan je za razvoj složenih programskih sustava. Razvoj je brži jer se omogućuje jednostavno korištenje postojećeg programskog koda. Posebno je pogodan za razvoj aplikacija s grafičkim korisničkim sučeljem.[8] Originalno, objektno-orijentirano programiranje najviše se koristilo za dva tipa aplikacija: računalne simulacije, koje pokušavaju oponašati aktivnosti stvarnoga svijeta; grafičko korisničko sučelje (GUI), koje omogućuje korisniku da komunicira s programom u grafičkom okruženju.[9] Pet najvažnijih značajki objektno-orijentiranih jezika su: klase, objekti, polimorfizam, nasljeđivanje i učahurivanje.[10]

2.1. KLASE I OBJEKTI

Aplikacija je program koji se izvodi da bi ostvario neki zadatak, kao što je pripremanje plaće, stvaranje korisničke narudžbe, igranje igrica i sl. U stvarnom svijetu objektima se smatraju ljudi, životinje i stvari. U objektno-orijentiranom programiranju objektima se smatraju i karakteristike (npr. boja, okus, miris), događaji (povezivanje, ometanje), stanja (smirenost, kretanje, ljutnja) i

[6] Usp. Farrell, Joyce. An Object-Oriented Approach to Programming Logic and Design. Canada : Course Technology, Cengage Learning, 2011., str. 27.
[7] Usp. Pillay, Anban. Object Oriented Programming using Java : Notes for the Computer Science Module *Object Oriented Programming* COMP200. Durban : School of Computer Science, University of KwaZulu-Natal, 2007.
[8] Usp. Žubrinić, Krunoslav. Uvod u objektno orijentirano programiranje. Dubrovnik: Informatički klub FUTURA, 2014.
[9] Usp. Farrell, Joyce. Nav. dj., str. 27.
[10] Usp. Farrell, Joyce. Nav. dj., str. 285.

sve ostalo što se može imenovati imenicom. Ukoliko je program velik, može se dogoditi da unutar njega postoje tisuće objekata s kojima je onda teško raditi. Kako bi se rad olakšao, uvodi se grupiranje objekata sličnih karakteristika, a te grupe nazivaju se klasama. Objekti koji su dio pojedinih klasa nazivaju se instancama klasa. Razlike između objekata i instanci zapravo nema. Izraz objekt koristi se u situacijama kad se govori o općenitim objektima, a izraz instanca koristi se kad se želi istaknuti da je određeni objekt dio neke klase.[11]

Klasa ili razred je obrazac (nacrt) koji definira varijable i metode zajedničke skupini objekata te opisuje atribute i metode povezane sa srodnim objektima. Klasa nije konkretan objekt već predložak koji se koristi za stvaranje objekata. Razred je jedan, a iz njega se može instancirati proizvoljan broj objekata koji će svi biti 'isti' u smislu da svi imaju isti skup članskih varijabli i članskih funkcija.[12] Kada se klasa dizajnira, mora se razmišljati o objektima koji će se kreirati iz te klase - kako izgledaju ti objekti (varijable), kako se ponašaju ti objekti (metode). Na primjer, klasa Bicikl sadržavat će varijable : Visina, Boja, Brzina, Kočnica. Sadržaj tih varijabli za određeni objekt mijenjat će se pomoću metoda: promijeniVisinu(), obojajBicikl(), promijeniBrzinu(), stisniKočnicu().[13]

Objekt je instanca klase. Na primjer, crveniBiciklSPetBrzina je instanca klase koja se sastoji od svih bicikala. Klasa (razred) je poput nacrta iz kojeg se može konstruirati puno kuća, ili kao recept s kojim se može napraviti puno jela. Jedna kuća i jedno jelo su instance svojih klasa.[14] Objekt može biti bilo koji predmet ili pojam iz stvarnog ili apstraktnog svijeta. Identitet predstavljaju osobine objekta po kojima se on razlikuje od drugih objekata.[15]

Kada se razmišlja objektno-orijentirano, sve je objekt i svaki objekt je član klase. Stol je objekt, računalo je objekt, kuća je objekt. Također, cvijet je objekt, mačka je objekt, članovi obitelji su objekti. Događaji su isto tako objekti – obavljena kupovina namirnica, obljetnica godišnjice. Svaki objekt je član klase. Stol je član klase koja uključuje sve stolove, a mačka je član klase koja uključuje sve mačke. Programerskim rječnikom može se reći kako je stol u nečijem uredu instanca klase Stol, a mačka na ulici je instanca klase Mačka.[16] Razmišljanje o predmetima kao instancama klasa omogućuje da se općenito znanje o klasama primijeni na njezinim pojedinim članovima. Posebne instance objekta preuzimaju atribute iz općenite kategorije. Na primjer, ako nečiji prijatelj kupi Automobil, zna se da on ima naziv modela.

[11] Usp. Rudolf Pecinovsky. OOP - Learn Object Oriented Thinking and Programming, 2013. URL: http://files.bruckner.cz/be2a5b2104bf393da7092a4200903cc0/PecinovskyOOP.pdf (2015-07-25)
[12] Usp. Vanjak, Zvonimir. Nav. dj.
[13] Usp. Žubrinić, Krunoslav. Nav. dj.., str. 40.
[14] Usp. Farrell, Joyce. Nav. dj., str. 285.
[15] Usp. Žubrinić, Krunoslav. Nav. dj., str. 35.
[16] Usp. Farrell, Joyce. Nav. dj., str. 286

Možda se ne zna koji je to točno model ili neke druge karakteristike, ali zna se da su to postojeći atributi za svaki Automobil.[17]

2.2. UČAHURIVANJE

Učahurivanje (ili enkapsulacija) bitan je princip objektno-orijentiranog programiranja. Kad računalni stručnjak kreira klase, želi osigurati da neophodni podaci budu dostupni metodama koje će raditi na njima. Zbog toga se podaci enkapsuliraju. No ipak, želi da su samo bitne informacije vidljive korisnicima klasa iz kojih će se kreirati instance, mijenjati vrijednosti dostupnih atributa i zvati dostupne metode. Stoga želi sakriti ili zaštiti neke podatke koji nisu potrebni za internu upotrebu da ne bi došlo do slučajnih promjena osjetljivih podataka.[18]

Učahurivanje je proces kombiniranja svih atributa i metoda koje objekt posjeduje u jedan „paket“. Postoje tri tipa vidljivosti za varijable i metode:

- Javna (public) – podrazumijeva se da je varijabla ili metoda javna ukoliko se ne definira drukčije. To znači da joj može pristupiti bilo tko te iz bilo kojeg dijela koda.
- Privatna (private) – ukoliko je varijabla ili metoda definirana kao privatna pristup je moguć samo unutar razreda u kojem se ona nalazi.
- Zaštićena (protected) – ukoliko je varijabla ili metoda zaštićena pristup je moguć unutar razreda u kojem se ona nalazi, ali i unutar njegovih naslijeđenih razreda.[19]

Skrivanje informacija je koncept koji omogućuje da druge klase ne mogu mijenjati atribute objekta – samo metode unutar objektnih klasa imaju tu privilegiju. Izvan klasa može biti dopušteno jedino podnošenje zahtjeva za promjenom atributa, a metode unutar klasa zadužene su za procjenu je li zahtjev primjeren. Na primjer, kad osoba koristi vrata, uglavnom ne razmišlja o konstrukcijskim dijelovima tih vrata niti ima pristup unutrašnjim dijelovima kvake niti zna koja boja je korištena za bojanje unutrašnjosti vrata. Bitna joj je funkcionalnost i sučelje tih vrata. Slično je i s objektima; objekti koji se kreiraju unutar objektno-orijentiranih programa mogu biti skriveni od ostatka koda i vanjskog dijela programa. U slučajevima kada su detalji skriveni, programeri se mogu fokusirati na funkcionalnost i sučelje, kao što ljudi rade s objektima u stvarnom svijetu.[20]

[17] Usp. Farrell, Joyce. Nav. dj., str. 286.
[18] Usp. Hillar C., Gaston. Learning Object-Oriented Programming. Birmingham : Packt Publishing, 2015., str. 65.
[19] Usp. Cihlar, Davor. Nav. dj.
[20] Usp. Farrell, Joyce. Nav. dj., str. 290.

Ako se u programu koristi privatno ili zaštićeno svojstvo, dobro je koristiti „gettere“ i „settere“. Metoda *set* koristi se za postavljanje podataka u polja koja nisu dostupna, a metoda *get* koristi se za čitanje podataka iz nedostupnih polja.[21] Te metode omogućuju da se pristupi podacima kojima se ne može pristupiti izravno izvan klase jer su zaštićeni ili privatni.

Enkapsulacija spaja povezane varijable i metode u uredan softverski paket te je jednostavan, ali moćan način koji programerima daje dvije prednosti:

- Modularnost: izvorni kod objekta može biti napisan i uređivan neovisno o izvornom kodu drugih objekata. Također, objekt se može lako prenositi u sustavu i radit će bez problema.
- Skrivanje informacija: objekt je prema zadanim postavkama javan kako bi drugi objekti mogli komunicirati s njim. Objektu se mogu postaviti postavke tako da informacije i metode budu privatne te se mogu mijenjati bilo kada, a da ne utječu na druge objekte.[22]

2.3. NASLJEĐIVANJE

Drugi važan koncept u objektno-orijentiranom programiranju je nasljeđivanje, koje je proces stjecanja obilježja prethodnika. U stvarnom svijetu primjer nasljeđivanja može se usporediti s novim vratima sa staklenim prozorom koja nasljeđuju većinu svojih obilježja od standardnih vrata. Imaju istu svrhu, otvaraju se i zatvaraju na isti način te imaju istu kvaku. Vrata sa staklenim prozorom samo imaju jedno dodatno obilježje – stakleni prozor. Ako se netko i nije nikada susreo s vratima koja imaju stakleni prozor, svejedno će znati što su i kako ih koristiti jer razumije karakteristike svih vrata. Kod objektno-orijentiranog programiranja, jednom kada se kreira objekt, mogu se razvijati novi objekti koji posjeduju sva obilježja originalnog objekta plus bilo koja nova obilježja koja programer poželi.[23]

Prilikom nasljeđivanja kreira se „opća“, bazna ili nadređena klasa (*superclass*). Podređenim ili izvedenim klasama (*subclass*) je bazna klasa nadređena i one nasljeđuju njezina svojstva. Često se koriste i nazivi roditeljska klasa (*parent class*) i klasa dijete (*child class*). Roditeljska klasa je klasa koja se koristi kao baza za nasljeđivanje, a od nje nasljeđuje klasa dijete. Jedan od glavnih pokazatelja odnosa između baza je veličina. Iako nije pravilo, u većini slučajeva izvedena klasa je veća od bazne klase u smislu da ima dodatna polja i metode. Ona

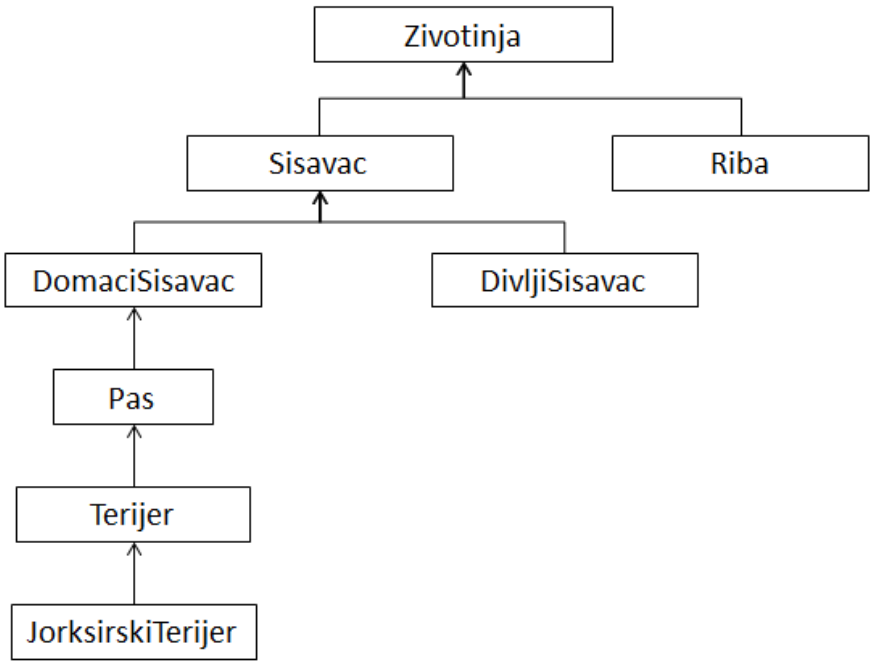
[21] PHP Documentation : Overloading. URL: http://php.net/manual/en/language.oop5.overloading.php#object.get (2015-08-30)
[22] Usp. Pillay, Anban. Nav. dj., str. 16.
[23] Usp. Farrell, Joyce. Nav. dj., str. 288.

možda izgleda manjom zbog manje veličine ispisanog koda, ali je veća jer je naslijedila sva polja i metode roditeljske klase te još tome pridodaje vlastita nova polja i metode. Izvedena klasa može biti i dalje proširena, to jest može imati vlastite izvedene klase. Te klase mogu dalje imati svoje izvedene klase i tako dalje dokle god je potrebno za funkcioniranje aplikacije. Cijela lista roditelja i djece klase čini popis predaka bazne klase.[24]

OOP razlikuje tri vrste nasljeđivanja. Prva vrsta nasljeđivanja odgovara s nasljeđivanjem sučelja u programima. Ovo nasljeđivanje je ključno u programiranju. Ako objekt nasljeđuje sučelje svoga roditeljskog objekta, onda taj objekt nudi isto što i roditeljski objekt, te je stoga sposoban oponašati roditeljsku instancu. Ovo nasljeđivanje ne primjenjuje se samo na sučeljima, nego i u implementaciji posebnih tipova datoteka od strane klasa. Drugo je nasljeđivanje implementacije koje se nameće u nasljeđivanju klasa. Objekt preuzima ne samo sučelje nego i cijelu implementaciju. Zbog toga taj objekt ne mora definirati svoju vlastitu metodu, nego koristi metode roditeljskog objekta. I zadnje, treće, je prirodno nasljeđivanje koje govori o tome kako se gleda na specijalizaciju objekata bez obzira na programiranost. Dobro programiran program sadrži sve tri vrste nasljeđivanja koje su u skladu.[25]

Primjer koji bi mogao bolje opisati ovu značajku OOP-a je primjer nasljeđivanja klase Životinja. Klasa Životinja je klasa koja generalizira sve članove životinjskog carstva te čini bazu za sve različite klase životinja koje se mogu pojaviti. Klasa Sisavac je klasa koja generalizira sisavce. Ova klasa nije bazna jer postoje životinje koje nisu sisavci poput reptila i ptica. Klasa DomaciSisavac se kreira jer neke životinje spadaju u istu skupinu, no nikako nisu na istoj razini. Tako tigar spada u porodicu mačaka, no vrlo je drugačiji od domaće mačke. Uz DomaciSisavac kreira se i DivljiSisavac, klasa koja je suprotnost prethodnoj klasi. Dalje ide klasa Pas koja nasljeđuje klasu DomaciSisavac. TerijerPas nasljeđuje klasu Pas kao skupina pasa terijer. Zadnja klasa je JorksirskiTerijer koja je podklasa klase TerijerPas, jer jorkširski terijer spada u skupinu terijera.[26] Ovako izgleda stablo nasljeđivanja klasa iz primjera:

[24] Usp. Farrell, Joyce. Nav. dj., str. 345.
[25] Usp. Pecinovsky, Rudolf. Nav. dj., str. 110.
[26] Usp. Hillar C. Nav. dj., str. 98.



Slika 1. Stablo nasljeđivanja

Razumijevanje nasljeđivanja pomaže programeru pri preciznijem organiziranju klasa. Svi objektno-organizirani programski jezici koriste nasljeđivanje iz istog razloga - kako bi organizirali korištenje objekata programa i kako bi se novi objekti lakše razumjeli na temelju znanja o njihovim naslijeđenim obilježjima. Prednosti nasljeđivanja su brojne:

- Ušteda vremena jer se ne mora ponovno pisati polje i metode
- Reducira se šansa za greške jer su prethodne metode već korištene i testirane
- Svakome tko je koristio prethodnu klasu je lakše razumjeti novu klasu koja nasljeđuje karakteristike prethodne
- Reducira se šansa za greške i nedosljednost u dijeljenim poljima
- Mogućnost korištenja nasljeđivanja čini programe lakšima za pisati, lakšima za razumjeti i manje sklonima greškama.[27]

2.4. POLIMORFIZAM

Stvarni svijet prepun je objekata. Za primjer se ponovno mogu uzeti vrata. Vrata se moraju otvarati i zatvarati. Otvaraju se pomoću sučelja jednostavnog za korištenje – kvake. Objektno-

[27] Usp. Farrell, Joyce. Nav. dj., str. 341.

orijentirani programeri bi rekli da se prenosi poruka vratima kad im se „kaže“ da se otvore hvatanjem kvake. Ista ta poruka (hvatanje kvake) ima različiti rezultat kad se primijeni na radio, a ne na vrata. Procedura koja se koristi kako bi se nešto otvorilo – procedura otvaranja – radi različito na vratima sobe i na ladici stola, bankovnom računu, računalnoj datoteci ili ljudskim očima. No, iako ove procedure rade drugačije koristeći drugačije objekte, svaka od ovih procedura može se nazvati procedurom otvaranja. S objektno-orijentiranim programiranjem fokus je na objektima koji će biti manipulirani programom. Definiraju se karakteristike tih objekata i metode koje će svaki objekt koristiti te se definiraju informacije koje moraju biti prenesene tim metodama. Mogu se kreirati višestruke metode s istim imenom, koje će se ponašati različito i u skladu s tipom objekata s kojima se koriste. Ovaj koncept naziva se polimorfizmom.[28]

Riječ polimorfizam znači „više oblika“, a potječe iz grčke riječi „poly“ koja znači mnogi i „morphos“ koja znači oblik.[29] To je objektno-orijentirani koncept koji se odnosi na sposobnosti varijable, funkcije ili objekta da preuzme više oblika. U programskim jezicima koji podržavaju polimorfizam, objekti klasa koji pripadaju istom hijerarhijskom stablu mogu posjedovati funkcije koje nose isti naziv, ali svaka ima različito ponašanje. Na primjer, nadklasa Životinja sadrži izvedene podklase Konj, Riba i Ptica. Klasa Životinja ima funkciju nazvanu Kretanje koju nasljeđuju sve njezine podklase. Polimorfizam omogućuje da svaka podklasa ima svoj način implementacije te funkcije. Na primjer, kad se funkcija Kretanje pozove u objektu iz klase Konj, funkcija može uzvratiti prikazivanjem galopiranja na ekranu. U drugu ruku, kad se ista funkcija pozove u objektu iz klase Riba, na ekranu će se prikazati plivanje. U slučaju klase Ptica, prikazat će se naravno letenje.[30]

Polimorfizam zapravo skraćuje posao programeru jer kreira općenitu klasu sa svim atributima i ponašanjima koja zamisli. Kad je potrebno da programer kreira specifične podklase s određenim jedinstvenim atributima i ponašanjem, on jednostavno prilagodi kod u specifičnim dijelovima gdje će se ponašanje razlikovati. Svi drugi dijelovi koda mogu ostati kakvi jesu.[31]

2.5. ŽIVOTNI VIJEK OBJEKTA

[28] Usp. Farrell, Joyce. Nav. dj., str. 288.
[29] Usp. OOP – Polymorphism, 2015. URL: http://ccm.net/contents/424-oop-polymorphism (2015-09-01)
[30] Usp. Polymorphism. URL: https://www.techopedia.com/definition/28106/polymorphism-general-programming (2015-09-04)
[31] Usp. Polymoprhism. Nav. dj.

U objektno-orijentiranom programiranju životnim vijekom objekta smatra se vrijeme od kreiranja objekta do njegova uništenja. Osnovni tijek života objekta je stvaranje, korištenje i uništenje, što se može usporediti s ljudskim životom koji se u osnovi sastoji od rađanja, života i umiranja. Ono što se događa u tom vremenu korištenja objekta ovisi o programskom jeziku koji se koristi, kao i što način življenja ovisi o osobi.

Najveća razlika između programskih jezika kad je u pitanju životni vijek objekta je razlika u njihovom uništavanju. U nekim programskim jezicima uništenje objekata je nužno za izvođenje aplikacija dok u drugima uništenje nije moguće, a to ovisi o memorijskoj lokaciji objekta. Objektima sa statičkom memorijskom lokacijom nemoguće je odrediti redoslijed kreiranja i uništavanja tj. koji statički objekt je kreiran prvi, koji drugi itd., pa stoga se objekti ne mogu uništavati. Objektima s automatskom ili dinamičnom memorijskom lokacijom se kreiranje objekata može odrediti pa stoga uključuju i uništavanje objekata.

Važno je spomenuti i da neki programski jezici koriste sakupljača smeća (GC – „Garbage Collector“) koji prati „žive“ objekte, a ostale označava kao smeće. To znači da kad se objekt više ne koristi, sakupljač smeća uzima memoriju koju je taj objekt koristio te ju ponovno koristi za buduće objekte. Tako se objekti ne brišu i ne vraća se memorija operacijskom sustavu već se samo čuva za objekte koji će biti kreirani. Kako bi znao koji se objekti više ne koriste, sakupljač smeća naizmjenično pokreće algoritam koji se sastoji od dva koraka:

1. Algoritam prelazi sve objektne reference i označava svaki „živi“ objekt koji pronađe.
2. Sva memorija programa koja nije zauzeta označenim objektima se vraća i označava slobodnom.[32]

3. PHP PROGRAMSKI JEZIK

PHP je svoju povijest započeo kao kolekcija CGI (Common Gateway Interface) skripti koje je napisao Rasmus Lerdorf kako bi pratio posjete na svom online životopisu. Taj oblik PHP-a je bio potpuno drukčiji od onog kojeg mi danas poznajemo. Lerdorfov PHP nije bio skriptni jezik, nego alati koji su pružali rudimentalne varijable i automatsko formiranje interpretacije varijabli koristeći HTML ugrađenu sintaksu. Nakon brojnih revizija kojima je PHP podvrgnut od 1994. do 1998. godine, programeri Tel Aviva Andi Gutmans i Zeev Suraski udružili su se s Rasmusom Lerdorfom kako bi transformirali PHP iz male kolekcije CGI alata u punopravni programski

[32] Usp. Java Memory Management : Memory Management. URL: http://www.dynatrace.com/en/javabook/how-garbage-collection-works.html (2015-09-06)

jezik s više sintakse i potporom za objektno orijentirano programiranje. Svoj konačni proizvod nazvali su PHP 3 i objavili ga krajem 1998. godine. Ta verzija je prva koja se može usporediti s PHP-om koji danas poznajemo. Aktualna inačica PHP-a je PHP5.[33]

PHP svoj naziv temelji na rekurzivnoj definiciji. To je kratica za PHP: Hypertext predprocessor. Iako je svoj razvoj započeo kao „hypertext predprocessor" danas je on uvelike proširio svoje mogućnosti i ta definicija se gotovo više niti ne koristi.[34] PHP je skriptni jezik koji je vrlo popularan, a njegova prihvaćenost ima brojne razloge. Neki od njih su upotreba otvorenog koda jer PHP programski jezik ima u potpunosti otvoren kod, vrlo visoka razina integracije s HTML-om i potrebama Weba danas, mogućnost upotrebe i u najsloženijim sustavima, može se bez ikakvih preinaka koristiti i pod Linux, Windows i ostalim platformama, programi pisani u programskom jeziku PHP mogu se vrlo brzo izvršavati te ima dobru podršku zajednice koja radi na razvoju i upotrebi PHP-a.[35]

PHP je interpreter što znači da programe pisane u programskom jeziku PHP nije potrebno prevoditi u izvršni oblik (eng. Compile) već se oni izvode prilikom pokretanja. Programi se pišu kao dio HTML stranice tako da se unutar HTML oznake dodaju PHP oznake koje počinju s „<?php", a završavaju s „?>". Sve unutar te oznake smatra se PHP programom, a tip datoteke mora biti .php kako bi se PHP dio programa uspješno izvršio.[36]

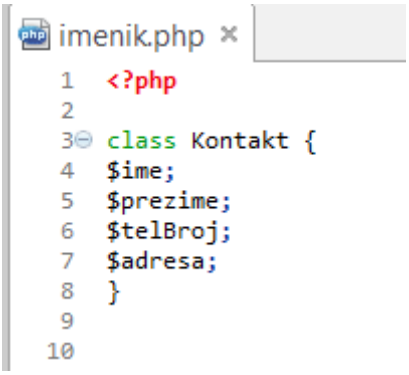
3.1. PRIMJENA OBJEKTNO-ORIJENTIRANIH PRINCIPA

Primjena objektno orijentiranih mogućnosti je omogućena u novijim inačicama, tj. u inačici PHP5. Kao što je već spomenuto, pet glavnih značajki objektno-orijentiranih jezika su: klase, objekti, učahurivanje, nasljeđivanje i polimorfizam. S obzirom da je objektno-orijentirano programiranje programerska paradigma, način provođenja takvog programiranja ovisi o programskom jeziku u kojem se koristi.

Počevši od osnovnih koncepata OOP-a, klasa i objekata, u PHP-u osnovno definiranje klase počinje s ključnom riječi *class* nakon koje slijedi naziv klase, a zatim par vitičastih zagrada koje zatvaraju svojstva te klase. Naziv klase može biti koji, osim rezerviranih PHP riječi. Te riječi su predefinirani identifikatori koje programer ne može koristiti u svojim skriptama. Ispravan naziv klase počinje sa slovom ili donjom crtom, nakon čega mogu ići slova, brojevi i

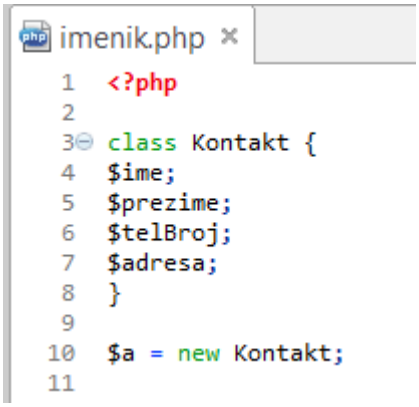
[33] Usp. Lockhart, Josh. Modern PHP. O'Reilly Media, 2015., str.16.
[34] Usp. Paunović, Vlatka; Tomić, Siniša. PHP: Priručnik uz seminar, listopad 2006. // Hrvatska udruga za otvorene sustave i Internet. URL: http://www.open.hr/wp-content/uploads/2012/04/PHP_prirucnik.pdf, str.11
[35] Usp. Paunović, Vlatka; Tomić, Nav. dj., str. 9.
[36] Usp. Paunović, Vlatka; Tomić, Siniša. Nav. dj., str. 14.

donje crte u količini koju programer želi. Klasa može sadržavati vlastite konstante, varijable i funkcije (metode).[37]



Slika 2. Kreiranje klase Kontakt

Kako bi se kreirala instanca klase, koristi se ključna riječ *new* uz naziv klase unutar koje se objekt želi kreirati. Klase bi trebale biti definirane prije kreiranja objekta. Ukoliko niz sadrži naziv klase uz ključnu riječ *new*, kreirat će se nova instanca te klase. Kad se već kreirana instanca klase dodjeljuje novoj varijabli, nova varijabla će imati pristup istim instancama kao i dodijeljeni objekt.[38]



Slika 3. Kreiranje objekta

Princip učahurivanja u potpunosti se koristi i u PHP-u sa svojim obilježjima koja su već spomenuta. Dakle, moguće je varijablu deklarirati kao javnu, zaštićenu ili privatnu pomoću ključnih riječi *public, protected* i *private*. To se još naziva i definiranjem vidljivosti. Svakoj varijabli koja je deklarirana u PHP-u je već određena razina vidljivosti. Naime, ukoliko se ne

[37] Usp. PHP Documentation : The Basics. URL: http://php.net/manual/en/language.oop5.basic.php (2015-09-02)
[38] Usp. PHP Documentation : The Basics. Nav. dj.

navede neka od tri razine vidljivosti pri kreiranju varijable, varijabla se automatski deklarira javnom. To se također odnosi i na metode kreirane unutar klase.

imenik.php

```php
<?php

class Kontakt {
public $ime;
private $prezime;
protected $telBroj;
protected $adresa;
}

$a = new Kontakt;

```

Slika 5. Učahurivanje

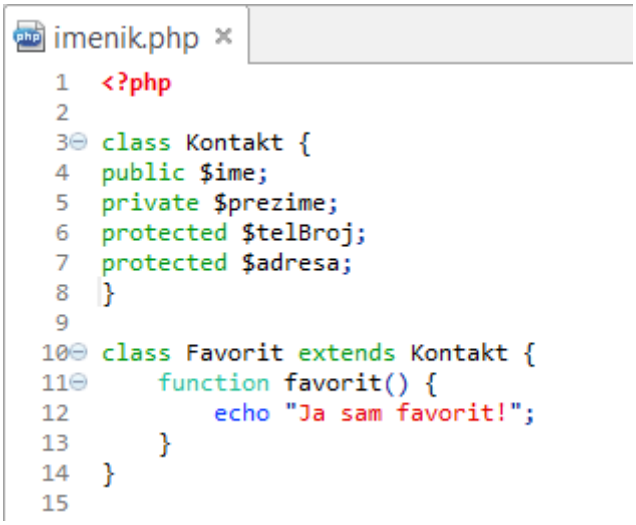
Kako bi se moglo pristupiti zaštićenim i privatnim varijablama i izvan klase u kojima su definirane koriste se dvije metode PHP programskog jezika, *get* i *set*. *Get* metoda koristi se za čitanje podataka iz polja kojima nije moguć pristup. *Set* metoda koristi se za kreiranje podataka unutar nedostupnih polja.

imenik.php

```php
<?php

class Kontakt {
public $ime;
private $prezime;
protected $telBroj;
protected $adresa;

    public function setPrezime($prezime) {
        $this->prezime = $prezime;
    }
    public function getPrezime($prezime) {
        return $this->prezime;
    }
        public function setTelBroj($telBroj) {
        $this->telBroj = $telBroj;
    }
    public function getTelBroj($telBroj) {
        return $this->telBroj;
    }
        public function setAdresa($adresa) {
        $this->adresa = $adresa;
    }
    public function getAdresa($adresa) {
        return $this->adresa;
    }

```

Slika 6. *Get* i *set* metode

Objektno-orijentirani princip nasljeđivanja koristi se i u PHP programskom jeziku. U PHP se koristi ključna riječ *extends* pomoću koje se jedino mogu kreirati podklase. Proširivanjem klase, podklasa nasljeđuje sve javne i zaštićene metode bazne klase. Te metode zadržavaju svoju izvornu funkciju, osim ako ih podklasa ne nadglasa. Ukoliko programer želi koristiti nasljeđivanje, klasa od koje se nasljeđuje mora biti definirana prije pisanja podklase.[39]



Slika 6. Nasljeđivanje

Već spomenuto nadglašavanje dio je zadnjeg koncepta objektno-orijentiranog programiranja koji se analizira u ovome radu, polimorfizma. Polimorfizmom se naziva postupak u kojem se pozivanjem iste metode dobivaju različiti rezultati. Ponekad programeri, zbog potreba svojih aplikacija, žele da se naslijeđene metode u podklasama ponašaju drukčije od roditeljskih klasa. U PHP-u se to postiže kreiranjem metode unutar podlase koja ima naziv jednak metodi unutar roditeljske klase. Stoga, kad god se poziva metoda za objekt iz te podklase, PHP će pokrenuti metoda podklase, a ne metode roditeljske klase.[40] Metoda koja se nalazi u roditeljskoj klasi može se zbog toga smatrati zadanom metodom.

[39] Usp. PHP Documentation : Object Inheritance. URL: http://php.net/manual/en/language.oop5.inheritance.php (2015-09-05)
[40] Usp. Doyle, Matt. Object-Oriented PHP: Working with Inheritance, 2011. URL: http://www.elated.com/articles/object-oriented-php-working-with-inheritance/ (2015-09-05)

imenik.php

```php
<?php

class Kontakt {
public $ime;
private $prezime;
protected $telBroj;
protected $adresa;

    public function setPrezime($prezime) {
        $this->prezime = $prezime;
    }
    public function getPrezime($prezime) {
        return $this->prezime;
    }

/*funkcija koja ispisuje prezime kontakta*/
    public function prezime() {
        echo "Moje prezime je" + getPrezime();
    }
}

class Favorit extends Kontakt {
    public function favorit() {
        echo "Ja sam favorit!";
    }
/*funkcija koja nadglašava funkciju prezime() iz roditeljske klase*/
    public function prezime() {
        echo "Prezime favorita je" + gerPrezime();
    }
}

```

Slika 8. Nadglašavanje

4. ZAKLJUČAK

Objektno-orijentirano programiranje je programska paradigma nastala tijekom 70-ih godina 20. stoljeća. Prvi programski jezik sa svojstvima objektno-orijentirane paradigme je SIMULA, a nakon toga bitan događaj je dodavanje objektno-orijentiranih svojstava u programski jezik C, čime je nastao programski jezik C++. S godinama razvoja i usavršavanja ovakvo programiranje postalo je jedno od najviše prihvaćenih i korištenih paradigmi.

Ključni koncepti OOP-a su klase, objekti, učahurivanje, nasljeđivanje i polimorfizam. Klasa je obrazac koji definira varijable i metode zajedničke skupini objekata te opisuje atribute i metode povezane sa srodnim objektima, a objekt je instanca klase. Na primjer, klasa je poput nacrta iz kojeg se može konstruirati puno kuća, a objekt je jedna kuća nastala iz tog nacrta tj. iz te klase. Princip učahurivanja je bitan princip u objektno-orijentiram programima. Taj princip omogućuje programeru da odredi koji će podaci biti dostupni drugim klasam i metodama te da može zaštiti podatke koje je potrebno zaštiti. U učahurivanju postoje tri tipa vidljivosti: javna (*public*), privatna (*private*) i zaštićena (*protected*). Idući princip je nasljeđivanje koji predstavlja proces stjecanja obilježja prethodnika. Prilikom nasljeđivanja kreira se bazna klasa, a zatim se kreiraju nove klase koje nasljeđuju svojstva bazne klase. Klase koje nasljeđuju svojstva od bazne klase nazivaju se podklasama ili izvedenim klasama. Treći princip je polimorfizam. To je objektno-orijentirani koncept koji se odnosi na sposobnosti varijable, funkcije ili objekta da preuzme više oblika.

PHP programski jezik svoju povijest započinje kao skriptni jezik, a trenutna najnovija verzija je PHP5. Primjena objektno-orijentiranih principa omogućena je u inačici PHP5. Za kreiranje klasa ključna riječ je *class* nakon koje slijedi naziv klase te je time kreirana nova klasa. Za instanciranje objekata koristi se ključna riječ *new* uz naziv klase iz koje se objekt instancira. Za učahurivanje koriste se ključne riječi *public, private* i *protected.* Kako bi neka klasa naslijedila svojstva druge klase koristi se ključna riječ *extends*. Jedino se polimorfizam razlikuje, tako da se kreiraju funkcije istog naziva kao i funkcija koju se želi nadglasati.

Objektno-orijentirano programiranje u PHP programskom jeziku je svakako veliki napredak za programere jer im omogućuje manje pisanje koda, višestruko korištenje dijelova koda, pravi se manje grešaka te stoga štedi vrijeme programeru što je najvažnije. Zbog toga bitno je da svatko tko se bavi razvojem računalnih programa razumije objektno-orijentirano programiranje te ga koristi u praksi.

LITERATURA:

1. A Brief History of Object-Oriented Programming. URL: http://web.eecs.utk.edu/~huangj/CS302S04/notes/oo-intro.html (2015-27-08)
2. Cihlar, Davor. Osnove izrade PHP aplikacija: Uvod u OO PHP. URL: https://www.fer.unizg.hr/_download/repository/3._PHP.pdf (2015-06-26)
3. Doyle, Matt. Object-Oriented PHP: Working with Inheritance, 2011. URL: http://www.elated.com/articles/object-oriented-php-working-with-inheritance/ (2015-09-05)
4. Farrell, Joyce. An Object-Oriented Approach to Programming Logic and Design. Canada : Course Technology, Cengage Learning, 2011.
5. Hillar C., Gaston. Learning Object-Oriented Programming. Birmingham : Packt Publishing, 2015.
6. Java Memory Management : Memory Management. URL: http://www.dynatrace.com/en/javabook/how-garbage-collection-works.html (2015-09-06)
7. OOP – Polymorphism, 2015. URL: http://ccm.net/contents/424-oop-polymorphism (2015-09-01)
8. Paunović, Vlatka; Tomić, Siniša. PHP: Priručnik uz seminar, listopad 2006. // Hrvatska udruga za otvorene sustave i Internet. URL: http://www.open.hr/wp-content/uploads/2012/04/PHP_prirucnik.pdf (2015-06-26)
9. PHP Documentation : Object Inheritance. URL: http://php.net/manual/en/language.oop5.inheritance.php (2015-09-05)
10. PHP Documentation : Overloading. URL: http://php.net/manual/en/language.oop5.overloading.php#object.get (2015-08-30)
11. PHP Documentation : The Basics. URL: http://php.net/manual/en/language.oop5.basic.php (2015-09-02)
12. Pillay, Anban. Object Oriented Programming using Java : Notes for the Computer Science Module *Object Oriented Programming* COMP200. Durban : School of Computer Science, University of KwaZulu-Natal, 2007.
13. Polymorphism. URL: https://www.techopedia.com/definition/28106/polymorphism-general-programming (2015-09-04)
14. Programming Paradigms. URL: http://cs.lmu.edu/~ray/notes/paradigms/ (2015-08-24)
15. Rudolf Pecinovsky. OOP - Learn Object Oriented Thinking and Programming, 2013. URL: http://files.bruckner.cz/be2a5b2104bf393da7092a4200903cc0/PecinovskyOOP.pdf (2015-07-25)

16. Vanjak, Zvonimir. Uvod u objektno-orijentirano programiranjeu programskom jeziku C++, 2007. URL: http://www.ieee.hr/_download/repository/ASP-Objektno.pdf (2015-06-26)
17. Žubrinić, Krunoslav. Uvod u objektno orijentirano programiranje, ožujak 2014. // Informatički klub Futura. URL: http://www.futura.com.hr/materijali/startup-2014/OOP_od_ideje_do_aplikacije.pdf (2015-06-25)