Мр Бранислава Бајковић Лазаревић, Мијодраг Ђуришић

НОВИ ТРЕНДОВИ У НАСТАВИ ПРОГРАМИРАЊА

Увод

Почеци озбиљније наставе програмирања код нас датирају с краја седамдесетих и везују се за чувену Шуварову реформу средњег усмереног образовања. Први програмски језик који су тада учили ученици трећег разреда математичко-техничке струке је био симболички језик хипотетичког рачунара NAR1. Убрзо је уследио FORTRAN, као језик за научно-техничку примену, а потом и COBOL као представник језика за пословну примену.

Наравно, врло брзо се увидело да је само малом броју програмера неопходно познавање симболичких језика па је асемблер уступио место, у програмима наших средњих школа, савременим процедуралним језицима попут Pascal-a и C-a.

И све би било у реду да нису сви одједном почели да користе рачунаре, па је понестало програма и програмера. Мудри програмери схватише да би много једноставније било да уместо да сваку апликацију зидају из почетка, направе шаблоне за готове објекте, па да их по потреби праве и уклапају једне уз друге. Тако настаде објектно оријентисано програмирање а ми, у настави програмирања у нашој средњој школи, опет се нађосмо у процепу. Баш смо усавршили Pascal, помало натуцамо и C, а деца нас питају за Delphi, VB, C#, ... Тако, у нужди, дођосмо до аргумента: џаба вам објекти ако не знате алгоритме. А самоука деца седоше, укључише рачунаре, покренуше своје „генераторе“ апликација, набацаше неке сличице, дугмиће, боксове на форму, притиснуше “play” и гле, пред нама текстуални едитор, са менијем, копирањем, форматирањем слова ... Несумњиво, далеко испред оног којим смо се поносили после много дана рада и безброј линија кода.

Шта нам је чинити? Ако не можеш да их победиш, придружи им се. А да бисмо се лакше уклопили, подсетимо се још једном како су се развијали програмски језици и покушајмо да направимо везу са објектно оријентисаним језицима.

Основни задатак рачунара, па самим тим и програма, јесте моделирање реалних система и ситуација. Управо по приступу моделирању реалних система и ситуација можемо уочити три различите групе програмских језика:

- Машински зависни језици моделирају системе из перспективе рачунара. Алати који су нам на располагању су на најнижем нивоу апстракције.
- Виши програмски језици као нова етапа у развоју, имају на располагању алате блиске човеку али и даље моделирају системе из перспектива рачунара. Алати који су на располагању су на вишем нивоу апстракције.

- Објектно оријентисани језици исте системе моделирају на још вишем нивоу апстракције. Када користимо ове алате не размишљамо о реализацији решења већ размишљамо о систему и моделирамо га онако како он у реалној ситуацији и постоји.

Сви системи у реалном свету су такви да над њима можемо извршавати неке акције а и они сами могу произвести акције. Код процедуралних језика имамо програм који моделира систем преко скупа објеката и низа акција и процедура. Акције се изводе строго утврђеним редоследом. Овакав концепт не одговара природи објеката и система које моделирамо.

У ООП програмирању имамо програме као скупове објеката. Објекти реагују на догађаје, комуницирају међусобно и размењују поруке. Редослед реакција зависи од догађаја и поруке.

Овај концепт је природан концепт. Ако посматрамо човека као систем, он прима разне поруке, али све поруке нису исте важности и он неће одговорити истим приоритетом на њих. У истом моменту човек ће запазити да му је укључена рингла на шпорету као и да се опекао јер му је прст на шпорету, али реакција „склони прст са рингле“ је реакција од највишег приоритета.

Први кораци у моделирању су били блиски машини, данас су блиски човеку. Програмирање је створено и развијано не због програмера већ због крајњег корисника. Крајњи корисници су људи који су главни корисници информација које можемо извући нашим програмима. Због њих цео систем и постоји. Данас се развијају алати све ближи крајњем кориснику и то на начин да он може самостално већину својих проблема да разреши. Тиме се концепт програмирања своди на развој алата у помоћи крајњем кориснику а то ће бити привилегија малог броја корисника. Са друге стране ниво образовања крајњег корисника стално расте, алати који су му на располагању су све моћнији и богатији. На тај начин, програмирање, онакво какво га знамо у прошлости, већ је историја. Алати који су доступни већ данас су такви да крајњи корисник програмирањем прилагођава алате својим потребама.

Основни принципи објектно оријентисаног програмирања

Основна намена рачунара је да помогне човеку у решавању разних проблема из своје или наше околине, што се постиже креирањем модела те исте околине. У моделирању користимо интелектуалне операције, а у њима је основни алат *апстракција*. Апстракција је контролисано укључивање и искључивање неких детаља о објектима. Моделирање је успостављање аналогије између оригинала и модела.

Можемо рећи да је машински језик апстракција ниског нивоа за решавање задатака. Виши програмски језици попут FORTRAN-а, PASCAL-а, C-а, ... јесу апстракција машинског језика, тако да представљају напредак у односу на машински језик. Недостатак ових језика је у томе што њихова примарна апстракција ипак захтева да онај ко решава проблем у тим језицима размишља из угла структуре рачунара уместо из угла проблема који решава. У процедуралним језицима сваки објекат из реалног света третирамо двострано: са једне стране су његове

статичке особине или структура, а са друге његове динамичке особине или акције које може да изведе. Објектно оријентисани приступ иде корак даље. У моделирању се полази од реалног света. Апстракције које су на услузи блиске су човеку а не машини. Објекти реалног света осим статичких особина поседују и динамичке. Аутомобил, осим што има својства попут боје, броја седишта, кубикаже, има и особину да може да се помери, може да стане, окрене итд. Акције које објекат може да уради су саставни део самог објекта. У објектно оријентисаном програмирању се самим тим не ограничавамо само на одређену врсту проблема као што је то случај са језицима попут LISP-а и PROLOG-а. Сваки објекат у ООП језицима има своја унутрашња стања као и скуп метода или акција које може да изврши. Програмирање је склапање више објеката у целину. Аналогија објеката у објектно оријентисаном програмирању са објектима у природи је очигледна. Решавање проблема на раунару тиме би било растерећено од размишљања о машини и програмском језику а више усмерено на размишљање о самом проблему који се решава. Друга велика добит која се остварује овим приступом је што се лакше могу организовати цели тимови програмера на развијању истог пројекта. Сваки члан неког тима лако може користити све објекте које су други чланови тима реализовали.

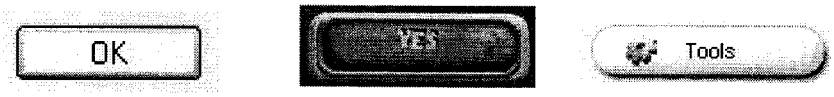
Smalltalk је један од првих успешних објектно оријентисаних језика. Alan Kay је навео пет основних карактеристика Smalltalk-а:

1. *Све је објекат.* Посматрајте објекат као побољшану променљиву; он чува податке, али могу му се поставити и захтеви које ће испуњавати вршећи операције над подацима.
2. *Програм је скуп објеката, који једни другима порукама саопштавају шта да раде.*
3. *Сваки објекат се смешта у меморијски простор у коме се могу налазити и други објекти.*
4. *Сваки објекат је инстанца неке класе.* Најважнија одлика класе је које поруке јој можемо послати. Класу можемо схватити као неко уопштење типа података.
5. *Сви објекти из исте класе могу да примају исте поруке.*

Сваки објекат има интерфејс. Преко интерфејса упућују се захтеви објекту. Класа је апстракција (уопштавање) конкретних објеката. Сваки објекат у систему служи као модел апстрактног „учесника“ који може да извршава задатак, извештава о промени стања, и „комуницира“ са другим објектима у систему, а да при томе не морамо откривати како су они имплементирани. Процеси, функције или методи такође могу бити на исти начин апстраховани.

Када кажемо човек, онда имамо општу представу о томе шта све подразумевамо под тим појмом, за разлику од помињања неког нашег конкретног познаника, нпр. Стеван, ... У том случају објекат има конкретна својства примерена само њему. Кажемо да је Стеван инстанца класе човек.

Посматрајмо класу ДУГМЕ (button) на рачунару. На слици која следи видимо три објекта који су инстанце класе ДУГМЕ. Сви они могу да примају исте

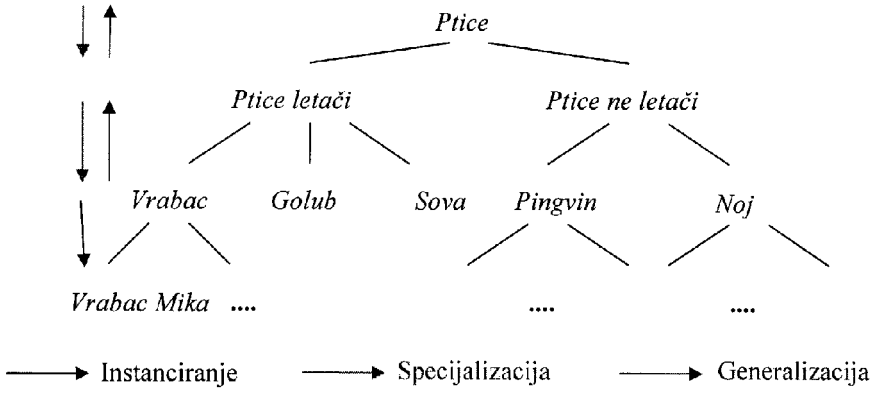


поруке. На њих се примењују исте методе (нпр. дугме је притиснуто, ...) и имају исту функционалност која се добија из опште класе ДУГМЕ.

Објекти су основа модуларности и структурираности објектно оријентисаног програмирања. Они су самосталне целине које можемо склапати са другим објектима у складу са потребама и тако решавати сложеније проблеме. Сваки објекат може примати, али и слати, поруке другим објектима. Објекат ДУГМЕ може примити поруку да је миш у његовом фокусу или да је притиснуто дугме миша. У овом другом случају објекат шаље поруку другом објекту којим се спроводи нека акција.

Једна од основних карактеристика објектно оријентисаних језика је и *Енкапсулација*. Енкапсулација подразумева да корисници неког објекта не могу мењати унутрашња стања и методе објеката. Само интерним методама објеката је омогућено да приступе тим стањима. Сваки објекат омогућава спољним корисницима да приступе и користе његове јавне методе помоћу интерфејса.

Наслеђивање је такође важна особина ООП-а. Она је последица генерализације као методе за моделирање објеката. На пример, можемо поћи од облика попут: троуглови, правоугаоници, кругови, ... и генерализацијом направити нову класу Облик. Обрнута операција се назива специјализација. Кажемо да се класа Облик специјализује у нпр. класу троугао. Све класе које су добијене специјализацијом наслеђују све особине полазне или генеричке класе. На пример, методу ротирај, троугао ће наследити из надкласе облик.



Посматрајмо класу Птице. Она је настала генерализацијом две класе Птице летачи и Птице нелетачи. Обе класе наслеђују особину Има крила. Класа Птице

летачи је такође настала генерализацијом класа: врабац, голуб, сова, ... Инстанца класе врабац (врабац кога тренутно гледам у свом дворишту) наслеђује све особине и све методе својих надкласа. За класу врабац кажемо да је потомак класе Птице летачи. Генерализацијом добијамо хијерархијску структуру. Ној као и врабац могу да примене методу брзо се помери. Метода ће деловати тако да ће Ној да потрчи а Врабац ће да полети. Када су у питању троугао и правоугаоник, метод промени величину извршава се на различите начине. Та особина, да иста метода делује различито у зависности на који се објекат примењује, назива се *полиморфизам*. Класа Птице летачи је апстрактна класа. Она не резултује конкретном птицом која може да полети. Класа врабац може да резултује конкретном инстанцом. Ова особина је такође важна у развоју објектно оријентисаног моделирања система и назива се *апстракција*. Можемо да правимо апстрактне класе које не резултују конкретним објектима са којима манипулишемо али помоћу њих и наслеђивања можемо да пренесемо неке особине објекту који је њихов потомак.

Како је ОО приступ решавању проблема најближи реалним ситуацијама, природно је да је и само склапање апликација на овај начин заправо најједноставније. Вођени том чињеницом, лако можемо закључити да неки од објектно оријентисаних програмских језика заиста може бити први корак у програмирању будућих програмера. Наравно да на почетку треба користити богате библиотеке класа, које поседује сваки оперативни систем, и уклапањем њихових инстанци креирати апликације. Самостално креирање класа, самим тим и програмирање сложенијих метода које захтевају и одређени процедурални приступ, требало би да представља виши ниво у развоју будућих програмера, ниво до кога ће доћи само они којима ће програмирање бити професија.