**DOKUMENTACIJA PROJEKTA**

**PAINT**

Objektno orjentirano programiranje

Anđela Čolak

9. veljače 2015.

Objektno orjentirano programiranje je paradigma programiranja zasnovana na konceptu objekta. Osnovni princip je definiranje klase kao samostalne programske cjeline.

Podrazumijeva se metoda programiranja kojom se definiranje neke klase vrši korištenjem svojstava postojećih klasa.Objekti koji se definiraju pomoću takvih klasa iskazuju dva svojstva: *Nasljeđivanje* i *Polimorfizam*.

Osnovni pojmovi objektnog oblikovanja su *objekt* i *klasa* , a osnovni elementi su : *apstrakcija* , *enkapsulacija* , *modularnost* , *nasljeđivanje* i *polimorfizam*.

***Klasa:***

Klasa je osnovica modularnosti i strukture objektno-orjentiranog programa. Sadrži informaciju (npr. ime, godište, spol) , ponašanje i funkcionalnost (metode).

Moj projekt ima klasu *Olovka*. Ta klasa ima polje veličina koja se može dohvatiti **Get** metodom.

**Set** metodom se ta veličina može postaviti samo na vrijednosti između 5 i 30 u ovom konkretnom primjeru..

public int Velicina

{

get { return velicina; }

set

{

if (value < 30 && value > 5)

velicina = value;

else

{

if (value >= 30)

velicina = 30;

else if(value<=5)

velicina = 5;

}

}

}

Klasa ima i bool olovka koji označava radi li se o punom ili praznom tragu koji olovka ostavlja na platnu. Njegova vrijednost može se dohvatiti i postavljati na novu vrijednost u bilo kojem trenutku.

Takav mehanizam pristupa varijablama zove se **Enkapsulacija.**

***Nasljeđivanje:***

Nasljeđivanje je tehnika kojom se definiranje neke klase vrši korištenjem definicije postojeće (*osnovne*) klase. Tako dobivena klasa zove se *izvedena klasa.*

Izvedena klasa nasljeđuje sva svojstva i metode osnovne klase uz definiranje nekih svojih svojstava ili metoda. Izvedena klasa može i pregaziti metode osnovne klase.

Sve definirane klase nasljeđuju već ugrađenu superklasu “*object”*.

Nasljeđivanje je moguće jedino ako su prava pristupa osnovne klase postavljena na *public* ili *protected*. U slučaju da je osnovna klasa postavljena na *private*, ona je nevidljiva svim ostalim klasama.

**public** - član je javan i dostupan je svim klasama.

**protected** - član je dostupan unutar klase i svim naslijeđenim klasama

**private** -član je dostupan samo unutar klase.

***Apstrakcija:***

[Apstrakcija](http://bs.wikipedia.org/wiki/Ra%C4%8Dunarska_apstrakcija) je pojednostavljivanje karakteristika objekta, gdje se zanemaruju detalji i uzimaju se samo zajedničke karakteristike klase. Čest je slučaj da se neki objekti minimalno razlikuju, i zasebno definiranje svakog od njih je redundantno. Osim toga za efikasno projektiranje praktično je pojednostavljivanje konkretnog problema.

Apstraktne klase ne služe stvaranju objekata, nego samo kao osnova za stvaranje podklasa.

Služe samo da bi se izrazila zajednička svojstva svih podklasa.

Npr.

I pravokutnik i trokut su Geometrijska tijela koja imaju svoje duljine stranica i površinu, ali njihove površine računaju se drugačije. Tada obje klase nasljeđuju klase Lik koja ima metodu RačunajPovršinu() koja će biti definirana u Izvedenim klasama.

protected abstract class Lik

{

protected int a;

protected int b;

protected int Povrsina;

protected abstract void RacunajPovrsinu()

{

}

}

private class Pravokutnik :Lik

{

protected override void RacunajPovrsinu()

{

Povrsina = a \* b;

}

}

private class Trokut : Lik

{

protected override void RacunajPovrsinu()

{

Povrsina = a \* b / 2;

}

}

***Modularnost:***

Modularnost je osobina sistema da se cijepa u skup kohezivnih i slabo uparenih modula kako bi se smanjila složenost problema. U velikim projektima se logički povezane (kohezivne) klase i objekti deklariraju u istom paketu, a izlažu se samo elementi koje ostali moduli moraju vidjeti. U malim se programima sve klase deklariraju u istom paketu. Modularizacija mora omogućiti da se implementacija promijeni, a da to ne utječe na implementaciju u ostalim modulima (minimalna međuovisnost).

***Polimorfizam:***

Polimorfizam označava mogućnost da se metode koje se isto zovu ponašaju različito u ovisnosti o parametrima koje primaju ili o tipu objekta nad kojim se metoda poziva. Polimorfizam se postiže premošćivanjem metoda bazne klase.

Vratimo se na primjer s trokutom i pravokutnikom. Iako se matoda RacunajPovrsinu() isto zove, ona će se drugačije ponašati s obzirom nad kojim objektom pozivamo metodu.

Pravokutnik p = new Pravokutnik();

p.RacunajPovrsinu(2,3);

Površina objekta p biti će postavljena na vrijednost 6.

Trokut t = new Trokut();

t.RacunajPovrsinu(2,3);

Površina objekta t biti će postavlljena na vrijednost 3.

Isto tako metode se mogu razlikovati i po broju parametara koje primaju.

Kada bi imali metodu Dodaj() ona bi se drugačije ponašala u slučaju da primi string i da primi vrijednost int.

private void Dodaj(string s)

{

Console.WriteLine("Dodali ste string "+s);

}

private void Dodaj(int i)

{

Console.WriteLine("Dodali ste broj "+i.ToString());

}