**Šta je objektno orijentisano programiranje?**

Objektno orijentisano programiranje jedna je od programskih paradigmi koje određuju stil programiranja. [Ovaj oblik programiranja](https://python.swaroopch.com/oop.html) bavi se radom sa objektima. Reč je o elementima programa koji imaju svoja stanja i ponašanje, tj. svaki od objekata ima neke osobine koje ga definišu. To praktično znači da se elementi programa predstavljaju kroz objekte, a da se program realizuje kroz menjanje stanja objekata.

Pri programiranju u nekom od jezika koji su objektno orijentisani fokus se stavlja na klase. Reč je o definisanju atributa objekta i njegovog ponašanja u odnosu na stanja u kojima se nalazi. Svaki put kada poželite da kreirate objekat, program će prvo pogledati klasu. 

Nakon toga će napraviti objekat prema pravilima koja su definisana u toj klasi. To znači da je objekat praktično pojedinačna instanca neke klase. Na ovaj način možete kreirati više drugačijih objekata istog tipa istih ili različitih vrednosti atributa koje poseduju.

Ovim načinom programiranja projekat se razlaže na manje logičke celine. Ove celine, tj. objekti ili klase imaju svoje specifične karakteristike i funkcije. To što se ove celine razdvajaju ne znači da ne komuniciraju međusobno. Kod ovog pristupa programiranju akcenat se stavlja na problem koji treba rešiti. U tom slučaju se fokusiramo na delove sistema koji imaju određene zadatke. 

Jedna od glavnih prednosti ovog pristupa programiranju jeste to što se programerima na ovaj način omogućava da kreiraju module koji ne moraju da se menjaju kada se doda nova vrsta objekta. Programer u tom slučaju jednostavno kreira novi objekat koji nasleđuje osobine od postojećeg objekta. Zbog toga se objektno orijentisani programi lako menjaju. 

Iako svi programski jezici nisu objektno orijentisani, usled popularnosti i ovog pristupa mnogi su usvojili ove principe.

**Osnove objektno orijentisanog programiranja**

Da bismo bolje razumeli kompletan proces, potrebno je da znamo osnovne principe, tj. koncepte na kojima se zasniva ovaj proces. Objektno orijentisano programiranje (OOP) bazira se na klasama i objektima. To znači da je moguće gledati na neki program, tj. softversko rešenje kao na skup objekata koji su definisani klasama.

**Klasa**

Klasa u objektno orijentisanom programiranju sadrži polja sa opisom informacija i funkcionalnosti (metode). Polja definišu osobine objekata. Ove osobine se tokom izvršavanja programa menjaju, čime se menja i kompletno unutrašnje stanje objekta.

Funkcionalnosti predstavljaju skupove naredbi koje utiču na polja kako bi se promenilo unutrašnje stanje objekata. Iz pomenutog zaključujemo da klasa poseduje atribute i operacije koje se nad njima izvršavaju. Koncept klase lakše je shvatiti kroz konkretan primer. 

Ako je, na primer, klasa IT kurs, opis će u sebi imati informacije o tome. To su podaci koji se odnose na konkretnu klasu, u ovom slučaju IT kurs (npr. trajanje, broj časova, predavač…). Ukoliko promenimo kurs, menjaće se podaci u opisu (duže trajanje, veći broj časova, drugi predavač…). Na ovaj način klasa definiše model na osnovu kog se neki objekat kreira.

U nekim objektno orijentisanim programskim jezicima postoji razlika, pa tako JavaScript, za razliku od jezika Java i C#, ne koristi klase u tradicionalnom obliku.

**Objekat**

Objekat predstavlja pojedinačni primerak klase koji poseduje sve osobine i ponašanja koje ta klasa definiše. Svaki objekat ima svoje karakteristike koje su definisane od strane klase. Na osnovu podataka koje definiše klasa moguće je napraviti praktično beskonačan broj objekata. 

Treba napomenuti da ovde ne govorimo o istovetnim kopijama. Naime, svaki objekat ima svoj identitet, u skladu sa postavljenim načelima klase. 

Ovako stvoreni objekti nezavisni su jedan od drugog. Ako je u našem slučaju *IT kurs*klasa, objekti mogu biti Advanced JavaScript Programming kurs, HTML & CSS Fundamentals kurs itd. Dakle, u ovom primeru reč je o IT kursevima koji imaju imaju iste principe, ali drugačije podatke.

**Apstrakcija**

Apstrakcija podrazumeva pojednostavljivanje karakteristika neke klase kako bi se naglasile važne osobine, a ignorisale nebitne (definisane od slučaja do slučaja).

Kod apstrakcije se zanemaruju određeni detalji i uzimaju, pre svega, zajedničke karakteristike klase, u zavisnosti od potrebe datog programa. Na ovaj se način biraju osobine kojima klasa definiše neki objekat. 

Ako je objekat televizor, u nekom slučaju može biti dovoljno da se navede dijagonala, na primer 102 cm. Sa druge strane, ako je potrebno detaljno prikazati konkretan TV uređaj, potrebno je navesti i, na primer, rezoluciju ekrana, materijal od kog je napravljen, broj HDMI ulaza…

**Enkapsulacija**

Enkapsulacija ili zatvorenost predstavlja jedan od osnovnih koncepata objektne orijentisanosti. Reč je o procesu gde se informacija o klasi štiti od direktnog pristupa. U tom slučaju je jedini način pristupa putem definisanih metoda. Na ovaj se način skrivaju detalji, pa korisnik samo zna šta može da radi sa određenim objektom. 

To znači da možemo, na primer, da odštampamo nešto klikom na dugme, ali da pritom ne znamo šta se tačno događa kada se to desi. Ovaj način programiranja zove se „crna kutija”, znamo kako da nešto aktiviramo, ali ne znamo detaljno kako se odvija proces. Na ovaj način se smanjuje šansa da dođe do greške, jer jednostavno nema detaljnijeg pristupa. Ovaj proces se sprovodi tako što članove klase delimo na javne i privatne.

Javni članovi se koriste u bilo kom delu programa, dok je privatnim moguće pristupiti samo u okviru definisane klase. Na ovaj način se privatni članovi klase menjaju na jasno definisan način. Na ovaj način se kreira određena zaštita koja štiti ove članove. Ipak, treba znati da su, bez obzira na zaštitu, svi članovi u okviru tela klase međusobno dostupni.

**Nasleđivanje**

Nasleđivanje podrazumeva identifikaciju zajedničkih osobina različitih klasa i uveliko olakšava programiranje, jer gubimo potrebu da iste atribute pišemo nekoliko puta. Ovaj proces omogućava jednoj klasi da nasledi karakteristike druge klase.

To je koncept po kojem se može definisati klasna hijerarhija, što znači da objekte definišu samo jedintvene karakteristike. Na ovaj način se jasno definiše na koji se način može izvesti novi objekat. Izvedena klasa je potklasa (potomak), a opštija je nadklasa (predak). Na ovaj način se kreira hijerarhijska nasledna linija.

**Polimorfizam**

Polimorfizam omogućava da osnovna klasa definiše funkcije koje će biti zajedničke za sve izvedene klase, s tim da te izvedene klase na svoj način mogu da implementiraju te funkcije. To znači da objekti nekog specifičnog tipa mogu da se posmatraju kao objekti osnovnog tipa. Međutim, polimorfno ponašanje poziva omogućava da izvedeni objekat izrazi svoju različitost.

Važno je znati i da osnovna klasa i one izvedene formiraju hijerarhiju koja se postavlja od uže generalizacije ka široj. U tom slučaju, osnovna klasa poseduje sve što izvedene klase mogu da koriste. Izvedene klase na ovaj način imaju mogućnost da samostalno implementiraju funkcije.