# Objektno-orijentisano programiranje

## Uvod u objektno-orijentisano programiranje

Nevena Ćirić
www.matf.bg.ac.rs/~nevena_ciric

# Osnovni pojmovi

- Objektno orijentisano programiranje je zasnovano na konceptu objekta.
- Objekti su strukture podataka (atributa) sa pridruženim skupom procedura i funkcija (metodi) koji služe za operisanje podacima koji čine objekat.
- U objektno-orijentisanim programskim jezicima metodi su jedini način operisanja podacima koji su sadržani u objektima.
- Svaki objekat predstavlja primerak (instancu) neke klase. Definisanjem klase definiše se kako sadržaj objekata te klase (skup atributa), tako i skup metoda za manipulisanje objektima te klase.

# Definisanje klase

- Za definisanje klase koristimo ključnu reč class koju prati ime klase i u okviru vitičastih zagrada sama definicija strukture klase (članova klase – atributi i metodi).

```
class ImeKlase {
        // članovi klase ( atributi i metodi )
}
```

- Svaka klasa u Javi je sadržana u nekom paketu, a ako se ne navede eksplicitno, naše klase se smeštaju u podrazumevani (default) paket koji nema ime.
- Stavljanje klase u neki paket se vrši navođenjem odgovarajuće deklaracije paketa pre same definicije klase (kao prva naredba fajla koji sadrži definiciju klase).
- Deklaracija paketa je oblika: package imePaketa;

# Definisanje klase

- Svaka klasa ima direktan pristup svim ostalim klasama istog paketa, ali atributi i metodi koji su članovi tih klasa ne moraju nužno biti dostupne.
- Dostupnost klasa (u smislu mogućnosti kreiranja objekata te klase), atributa i metoda se kontroliše modifikatorima kontrole vidljivosti, tj. pristupa.
- Za klase postoje 2 nivoa vidljivosti:
  - podrazumevana (paketna) vidljivost - ukoliko ispred definicije klase nije naveden modifikator kontrole vidljivosti, definicija te klase je dostupna samo metodima klasa koje se nalaze u istom paketu
  - javna vidljivost - ukoliko želimo da klasa bude dostupna i van paketa u kome je definisana, potrebno je deklarisati klasu koristeći public modifikator u prvoj liniji definicije klase.

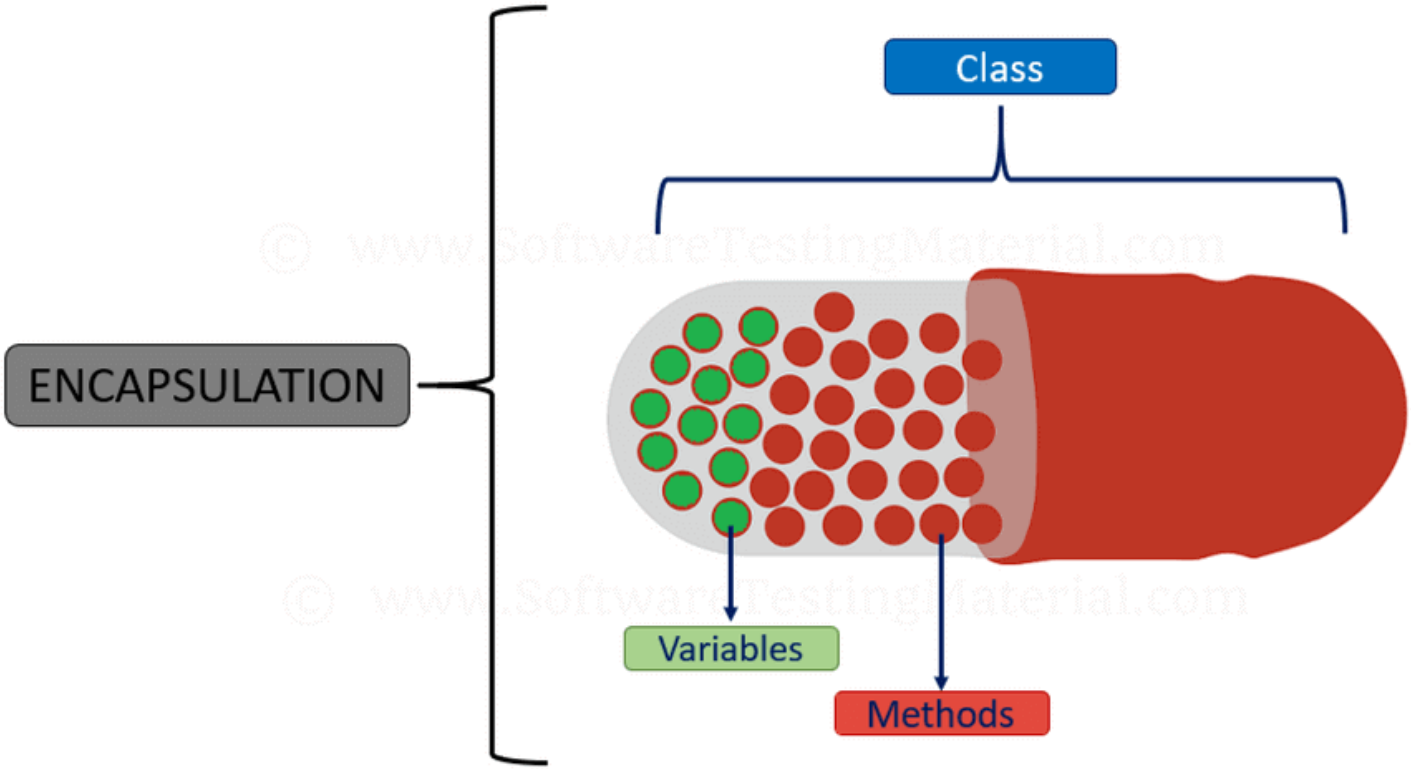
# Modifikatori kontrole vidljivosti (pristupa)

- Za atribute i metode postoje 4 nivoa vidljivosti:
  - public – dopušten pristup iz metoda proizvoljne klase (ne nužno iz istog paketa), sve dok je klasa čiji je to član deklarisana kao public
  - <package> – dopušten pristup iz metoda proizvoljne klase iz istog paketa; karakteriše ga nepotrebnost navođenja modifikatora, to je podrazumevani nivo vidljivosti
  - protected – dopušten pristup iz metoda proizvoljne klase istog paketa i iz proizvoljne potklase (ne nužno iz istog paketa)
  - private – dopušten pristup samo iz metoda unutar klase, nikakav pristup izvan klase nije moguć

# Modifikatori kontrole vidljivosti

| Modifier | Class | Package | Subclass | Global |
|---|---|---|---|---|
| Public | Yes | Yes | Yes | Yes |
| Protected | Yes | Yes | Yes | No |
| Default | Yes | Yes | No | No |
| Private | Yes | No | No | No |

# Enkapsulacija



# Enkapsulacija

- Enkapsulacija je još jedan od osnovnih principa objektno-orijentisanog programiranja.
- Enkapsulacija podrazumeva skrivanje podataka* unutar objekta od spoljnih pristupa.
- Enkapsulacijom se obezbeđuje da objekti imaju strogo kontorlisan pristup i izmene nad podacima, čime se smanjuje mogućnost grešaka i logičkih nedoslednosti u programu.
- Takođe, enkapsulacija omogućava da se sakriju (neki) detalji implementacije klase i da se način na koji je ta implementacija izvršena može menjati bez potrebe za dodatnim izmenama programa koji koriste tu klasu, sve dok se metodi koji se mogu pozivati izvan klase pozivaju na isti način kao i ranije.

*Prvenstveno atributa, ali i nekih metoda

# Enkapsulacija

- Skrivanje podataka se postiže upotrebom private modifikatora vidljivosti. Takvim atributima (i metodima) direktno mogu pristupati i menjati njihove vrednosti samo metodi koji pripadaju istoj klasi kao i posmatrani atributi (i metodi).
- Svako pristupanje podacima van klase vrši se isključivo preko javnih metoda kreiranih za te potrebe – tzv. getter i setter metoda.
- Getter metodi služe za dobijanje vrednosti atributa, a setter metodi za postavljanje nove vrednosti atributa datog objekta.
- Definisanjem samo get ili set metoda možemo učiniti atribut objekta read-only ili write-only. Na taj način kontrolišemo koju vrstu pristupa želimo da omogućimo za podatke koji se čuvaju u okviru objekta.

# Enkapsulacija

- Postvalja se pitanje zašto koristiti public metode da bismo omogućili pristup vrednosti private atributa kada možemo jednostavno da ga definišemo kao public?
  - Glavna prednost korišćenja getter i setter metoda jeste u tome što pružaju mogućnost da dopustimo pristup atributu samo za dobijanje vrednosti, a da menjanje vrednosti ostane onemogućeno (dok bi definisanje atributa javno vidljivim automatski omogućilo i dobijanje i menjanje vrednosti).
  - Čak i kada je potrebno da bude dopušteno i dobijanje i menjanje vrednosti atributa, bolje je to omogućiti putem getter i setter metoda zato što se u okviru set metoda može izvršiti provera nove vrednosti (pre nego što izmena bude napraviljena) kako bi se sprečilo pridruživanje neodgovarajućih vrednosti atributima.

# Konstruktori klase

- Atributima se u definiciji klase mogu dodeliti inicijalne vrednosti – tada će atributima svih objekata te klase prilikom kreiranja biti dodeljene navedene inicijalne vrednostimi.
- Ukoliko se ne navede inicijalna vrednost, prilikom kreiranja objekta svim atributima biće pridružena podrazumevane vrednost, i to:
  - numeričke vrednosti – 0
  - boolean – false
  - char – '\u0000'
  - klasni tip – null
- Neposredno nakon kreiranja objekta poziva se konstruktor klase – metod koji vrši inicijalizaciju konkretnog (novokreiranog) objekta odgovarajućim vrednostima atributa.

# Konstruktori klase

- Konstruktor je metod čiji potpis ima specifičan oblik:
  - nema povratnu vrednost (ni void!)
  - ima isto ime kao i klasa kojoj pripada
- Jedna klase može imati definisano više konstruktora, pri čemu se njihovi potpisi razlikuju po broju i tipu argumenata.
- Ukoliko ne definišemo nijedan konstruktor za našu klasu, kompajler će obezbediti podrazumevani konstruktor klase koji nema parametre i ne radi ništa.
- Ukoliko sami napišemo bar jedan konstruktor, kompajler ne pravi podrazumevani konstruktor. Ako nam je i on potreban, moramo ga eksplicitno definisati .

# This referenca

- Svaki nestatički metod klase ima promenljivu this koja predstavlja referencu na tekući objekat za koji je metod pozvan.
- Kompajler implicitno koristi this referencu svaki put kada neki metod referiše na nestatiči atribut klase objekta nad kojim je pozvan.
- Postoje i situacije kada je neophodno eksplicitno koristiti this referencu da bi se pristupilo članovima klase (atributima i metodima).
- Imena promenljivih koje se deklarišu unutar metoda, lokalna su za taj metod i ne smeju biti deklarisane dve promenljive sa istim imenom. Dozvoljeno je koristiti imena lokalnih promenljivih, kao i imena argumenata metoda koja su ista kao imena atributa klase. U tom slučaju, unutar tog metoda neophodno je koristiti this referencu za referisanje na atribute klase. Sâmo ime promenljive će se uvek odnositi na lokalnu promenljivu metoda, ne na atribut klase.
- Da bi se u okviru metoda pozvao neki drugi metod iste klase nad istim objektom potrebno je pristupiti mu preko this reference.

# Klasa Object i predefinisanje nasleđenih metoda

- U Javi imamo klasu Object koja je natklasa svih klasa, kako klasa iz Java standardne biblioteke klasa, tako i klasa koje mi definišemo.
- To znači da svaka klasa nasleđuje sve metode definisane u klasi Object, a jedan od njih je i metod toString().
- Implementacija metoda toString() u klasi Object:

```
public String toString() {
        return getClass().getName() + "@" +
                Integer.toHexString(hashCode());
}
```

- Moguće je definisati metod klase koji ima isti potpis kao i nasleđeni metod iz natklase. Time se vrši predefinisanje nasleđenog metoda, odnosno promena ponašanja ili dodavanje novog ponašanja tog metoda u potklasi.
- Predefinisani metodi se označavaju @Override anotacijom.