Uvod u objektno-orjentisano programiranje

Objektno-orjentisano programiranje 1

Objektno-orjentisano programiranje

- OO program – **skup objekata** koji su u **međusobnim interakcijama**
- Objekat je struktura koja se sastoji od
  - **atributa/polja** – promenljive članice objekta
  - **metoda** – funkcije članice objekta
- Ideja da se podaci i funkcije koje te podatke obrađuju upakuju u jednu celinu i predstave jednom promenljivom
- Vrednost atributa određuje **stanje objekta**, metodama je definisano **ponašanje objekta**
- Objekti su u interakciji, međusobno razmenjuju poruke
  - **poruka – objekat A pozove metod nad objektom B**

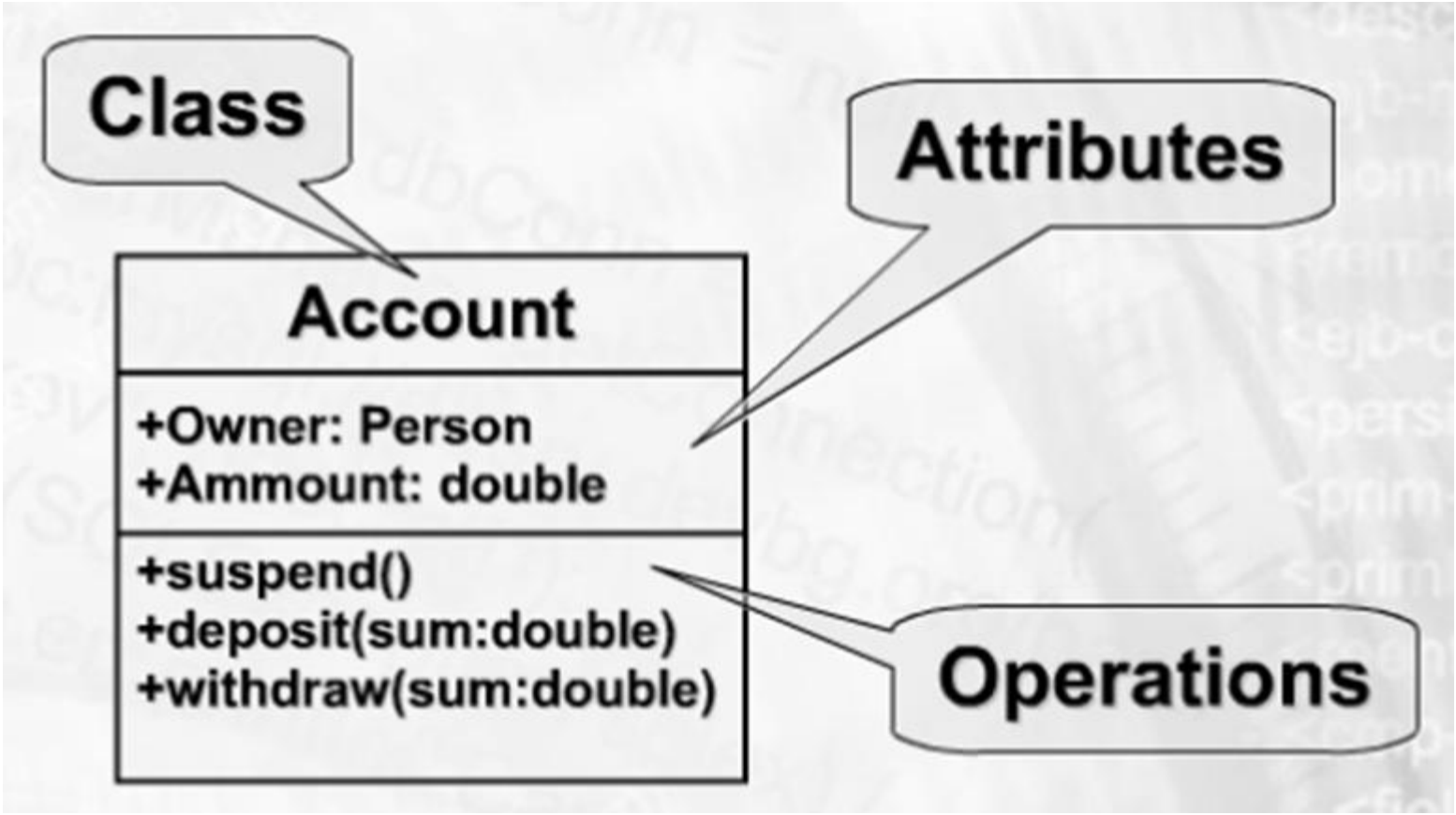
Objektno-orjentisano programiranje

- **Klasa – definicija skupa istorodnih objekata**
  - objekti koji imaju istu specifikaciju stanja i ponašanja (iste atribute i metode)
- **Sve promenljive u nekom programu imaju ime i tip**
  - Tip određuje skup mogućih vrednosti promenljive i skup operacija dozvoljenih nad promenljivom
- **Klase su tipovi objekata**
  - u PJ Java klase su **referencijalni tipovi**
  - **objekat A kao atribut sadrži referencu na neki drugi objekat B → objekat A može poslati poruku objektu B**
- Kada pišemo OOP programe **definišemo klase** i **instanciramo (kreiramo) objekte** definisanih klasa

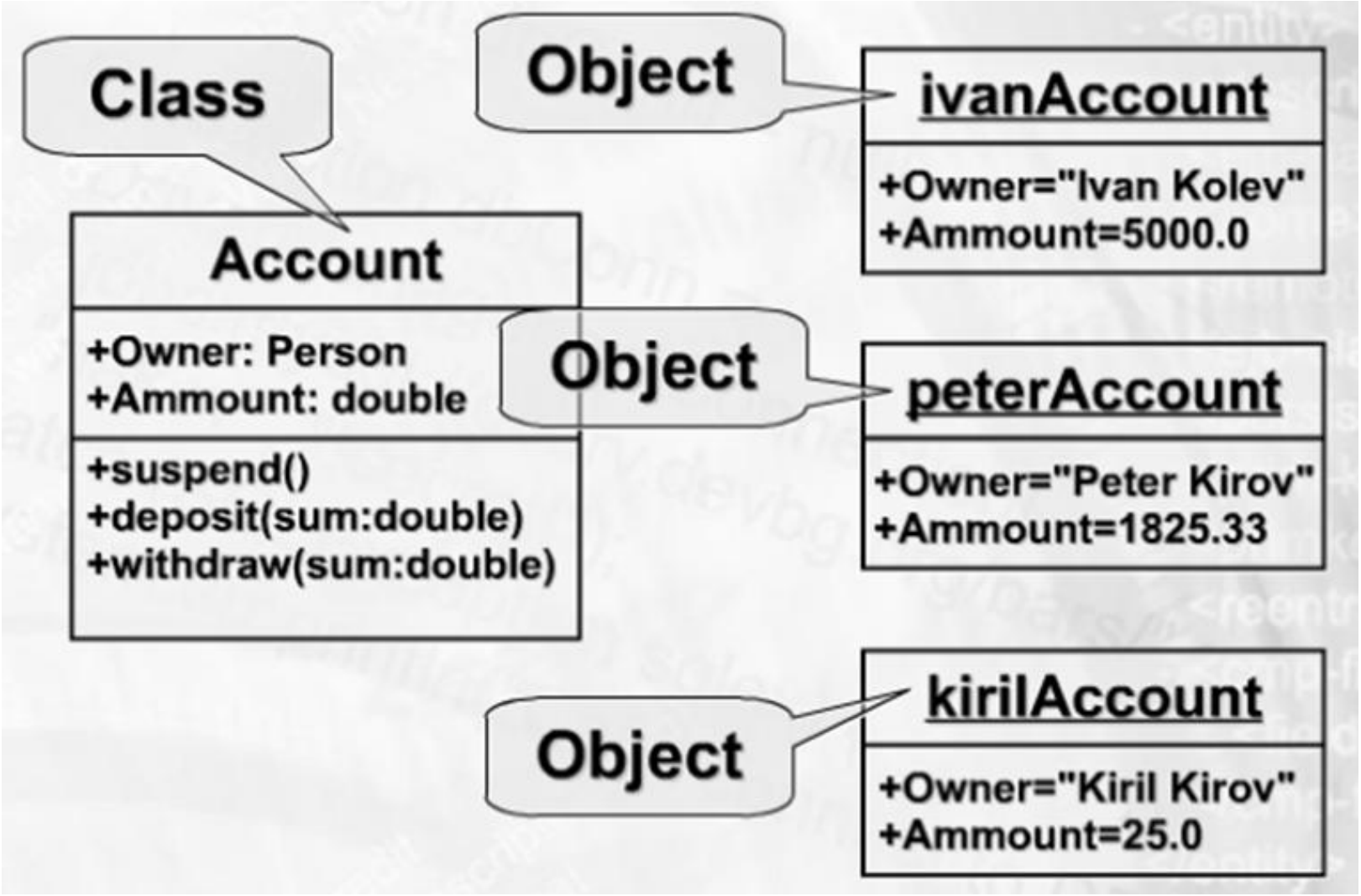
Imperativni programski jezici

- Program – formalan opis **procesa izračunavanja** ili **specifikacije izračunavanja** u nekom **programskom jeziku**
  - imperativni VS deklarativni programski jezici
- Program – skup **naredbi** koje tokom izvršavanja menjaju vrednosti skupa **promenljivih** (stanje programa)
- Naredba dodele vrednosti promenljivoj kao osnovna naredba
- Naredbe kontrole toka (if-then-else, switch, while, do-while, for)
- **Kategorije imperativnih programskih jezika**
  - **Proceduralni** – dekompozicija programa u funkcije i procedure, lokalne i globalne promenljive
  - **Modularni** – dekompozicija programa u module, modul sadrži logički srodne definicije promenljivih, funkcija i tipova podataka, privatni i javni deo modula
  - **Objektno-orjentisani** – dekompozicija programa u klase, klasa sadrži logički srodne definicije promenljivih i funkcija, klase kao tipovi podataka, privatni i javni deo klase, **nasleđivanje klasa**

Primer klase



Klase i objekti



Konstruktori i destruktori

- Konstruktor – specijalni metod koji se poziva kada se instancira klasa koji služi da inicijalizuje stanje objekta
- Destruktor – specijalni metod koji se poziva kada se objekat uništava
- U programskom jeziku Java
  - klase se instanciraju operatorom new
  - rezultat operatora new je referenca na instancirani objekat
  - može postojati više referenci na jedan objekat
  - nema destruktora, automatsko uništavanje objekata (*garbage collector*)
    - objekta pokupi GC kada ne postoji nijedna referenca na objekat

Apstrakcija, enkapsulacija i sakrivanje informacija

- **Sakrivanje informacija** – neki delovi objekta mogu biti javni, neki privatni
  - **modifikatori vidljivosti (pristupa) prilikom definisanja atributa i metoda klase**
  - privatni atributi su vidljivi jedino unutar klase
  - privatne metode mogu pozivati jedino metode iz iste klase
  - javnim atributima i metodama objekta se može pristupati i iz drugih klasa
- **Enkapsulacija** – dizajniranje objekta na način da su detalji nebitni za korišćenje objekta skriveni od korisnika objekta
  - **Ako su svi atributi objekta privatni tada objekat ima potpunu kontrolu nad sopstvenim stanjem**

Apstrakcija, enkapsulacija i sakrivanje informacija

- Suština programskih jezika je da omoguće mehanizme dekompozicije i apstrahovanja
  - **Dekomponovati – razložiti entitet (problem) u pod-entitete (pod-probleme) koji su manje kompleksnosti**
  - **Apstrahovati – sakriti (zanemariti, odložiti, zaboraviti) nebitne (tehničke, implementacione) detalje (smanjiti kompleksnost)**
  - Proceduralni programski jezici
    - Dekompozicija programa u procedure
    - Korisnik procedure mora znati samo zaglavlje, a ne telo procedure da bi koristio proceduru
  - Objektno-orjentisani programski jezici
    - Dekompozicija programa u objekte
    - Korisnik objekta mora znati samo zaglavlja javnih metoda objekta da bi koristio objekat

Agregacija i kompozicija

- **Osnova za interakciju između objekata: objekat *a* klase *A* ima referencu na objekat *b* klase *B***
  - U definiciji klase *A* imamo atribut *p* tipa *B*
- Ako klasa *A* **instancira** *p* tada su *A* i *B* u relaciji **kompozicije**
  - *b* je deo *a* koji ne može da postoji bez *a*
  - uništavanjem objekta *a* uništava se i objekat *b*
  - *A* = Čovek, *B* = Srce
- Ako klase *A* **ne instancira** *p* tada su *A* i *B* u relaciji **agregacije**
  - *b* je deo *a* koji može da postoji bez *a*
  - uništavanjem objekta *a* ne uništava se objekat *b*
  - *A* = FudbalskiKlub, *B* = Fudbaler

Nasleđivanje

- **Klase se mogu međusobno nasleđivati**
- Ako klasa *A* nasleđuje klasu *B* tada
  - *A* nasleđuje sve atribute i metode klase *B*
  - objekat klase *A* sadrži atribute i metode definisane u klasi *B*
  - **u klasi A se mogu redefinisati nasleđeni atributi i metode (*overriding*)**
  - A – izvedena klasa, podklasa, klasa dete
  - B – bazna klasa, nadklasa, super klasa, klasa roditelj
- **Ako klasa A nasleđuje klasu B tada su te dve klase u relaciji JE ("is a")**
  - Mačka je Životinja
  - Pas je Životinja
  - Pravugaonik je GeometrijskaFigura
  - Kvadrat je Pravugaonik

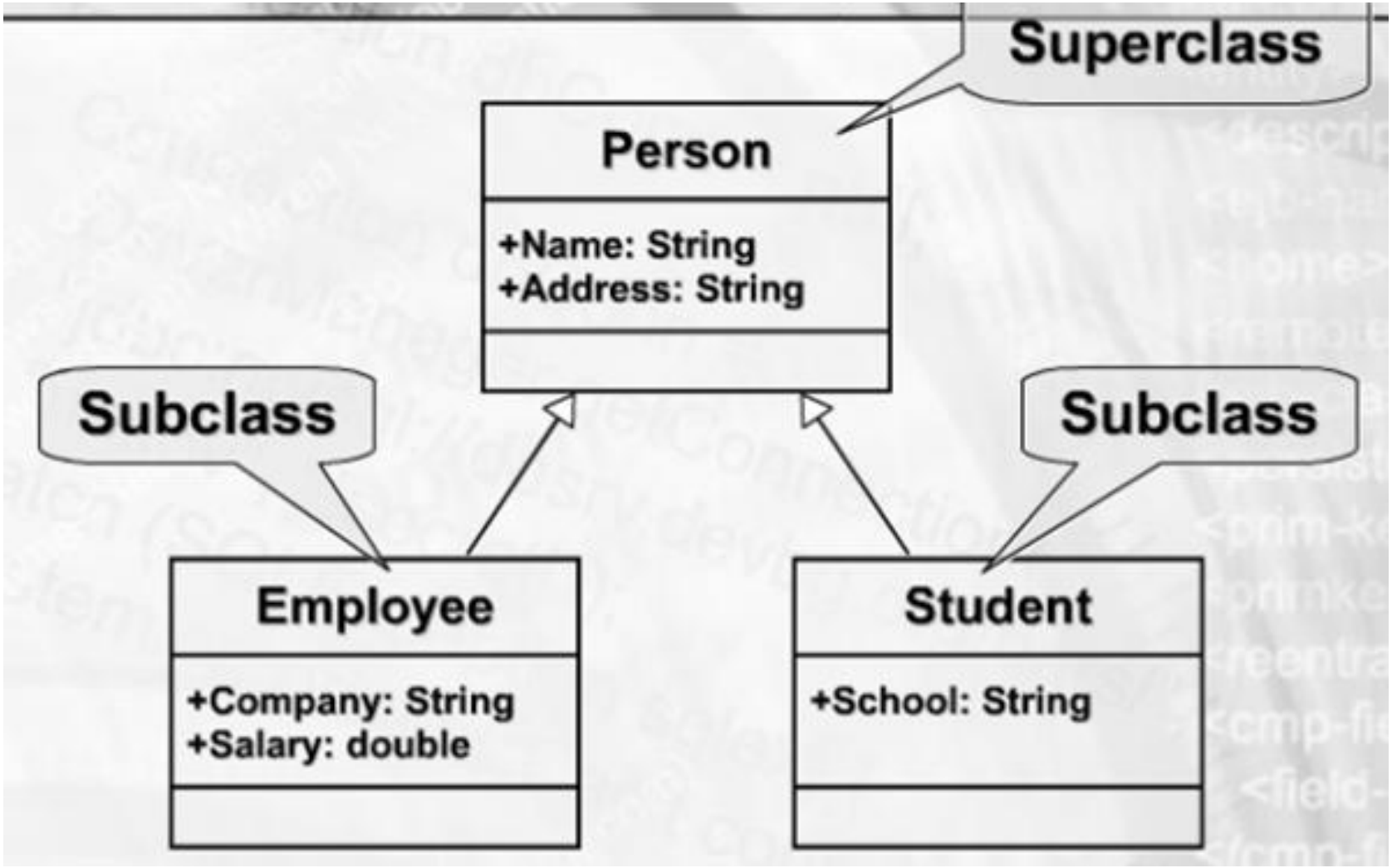
Nasleđivanje

- **Nasleđivanje klasa je mehanizam ponovnog iskorišćenja koda (eng. *code reuse*) koji omogućava**
  - **Specijalizaciju**
    - Izvedena klasa dodaje nove atribute i nove metode
  - **Proširenje postojeće funkcionalnosti**
    - Izvedena klasa dodaje nove metode
  - **Modifikaciju postojeće funkcionalnosti**
    - Izvedena klasa modifikuje nasleđene metode
  - **Kod iz bazne klase se ne kopira**
    - Nema klonova u programskom kodu
    - Negativne posledice klonova su glomazniji programi koji se teže održavaju
    - ***Copy-paste* programiranjem dupliciramo postojeće semantičke greške (*bug*-ove)**

Nasleđivanje – primer

- **class Pravugaonik**
  - atributi a i b tipa int (dužina i širina stranice)
  - konstruktor za inicijalizaciju a i b na neke početne vrednosti
  - metod za računanje obima: 2 * (a + b)
- **Svaki kvadrat je pravugaonik, ali nije svaki pravugaonik kvadrat**
  - **Kvadrat je pravugaonik sa specijalnim osobinama**
- **class Kvadrat extends Pravugaonik**
  - konstruktor koji inicijalizaciju a i b na identičnu vrednost
  - možemo redefinisati metod za računanje obima: 4 * a
    - jedna operacija manje

Nasleđivanje – primer 2



Apstraktne klase

- **Apstraktne klase sadrže apstraktne metode**
  - Apstraktan metod – metod koji nije implementiran, dato samo zaglavlje
  - Apstraktne klase se ne mogu instancirati
- Smisao apstraktnih klasa je da
  - Obezbede neke opšte funkcionalnosti
  - Da se specifične funkcionalnosti implementiraju i izvedenim klasama (**apstraktne klase se uvek nasleđuju**)

```
abstract class Figura {
	abstract double obim();

	public String toString() {
		return "Figura obima " + obim();
	}
}
```

- **Hijerarhije klasa u OO programima → od apstraktnih ka specifičnim klasama**

Liskov princip supstitucije

- **Ako klasa *S* nasleđuje klasu *T* tada objekat klase *S* može da se koristi gde god se očekuje objekat klase *T***

```
class Pravugaonik { ... }

class Kvadrat extends Pravugaonik { ... }

class Ekran {
    void nacrtaj(Pravugaonik p) { ... }
}

Ekran e = new Ekran();

Pravugaonik p = new Pravugaonik();
e.nacrtaj(p);

Kvadrat k = new Kvadrat();
e.nacrtaj(k);
```

Liskov princip supstitucije

- **Ako klasa *S* nasleđuje klasu *T* tada objekat klase *S* može da se koristi gde god se očekuje objekat klase *T***
- `T a = new T()`
- `T b = new S()`
  - *a* i *b* – reference tipa *T*
  - **Referenca tipa *T* može da pokazuje na objekte klase *T*, ali i objekte klasa izvedenih iz *T***
  - Tip u vremenu kompajliranja i tip u vremenu izvršavanja

```
class Pravugaonik { ... }
class Kvadrat extends Pravugaonik { ... }
Pravugaonik p = new Kvadrat();
```

Polimorfizam i dinamičko vezivanje

- **Polimorfizam – različito ponašanje nekog operatora ili metode u zavisnosti od tipova argumenata**
  - 2 + 2 → 4
  - “ana” + “marija” → “anamarija”
  - **Ponašanje izraza a + b zavisi od tipa a i b**
  - **Preopterećeni (*overloaded*) operatori i metodi**
- **Na osnovu tipa argumenta se određuje koji operator se primenjuje / koji se metod poziva**
  - Statičko vezivanje – određivanje se vrši u vremenu kompajliranja
  - **Dinamičko vezivanje – određivanje se vrši u vremenu izvršavanja**
- **Polimorfizam u PJ Java – koja od preopterećenih metoda se poziva se određuje u vremenu izvršavanja na osnovu tipova referenci u vremenu izvršavanja (a ne u vremenu kompajliranja)**

Polimorfizam – primer

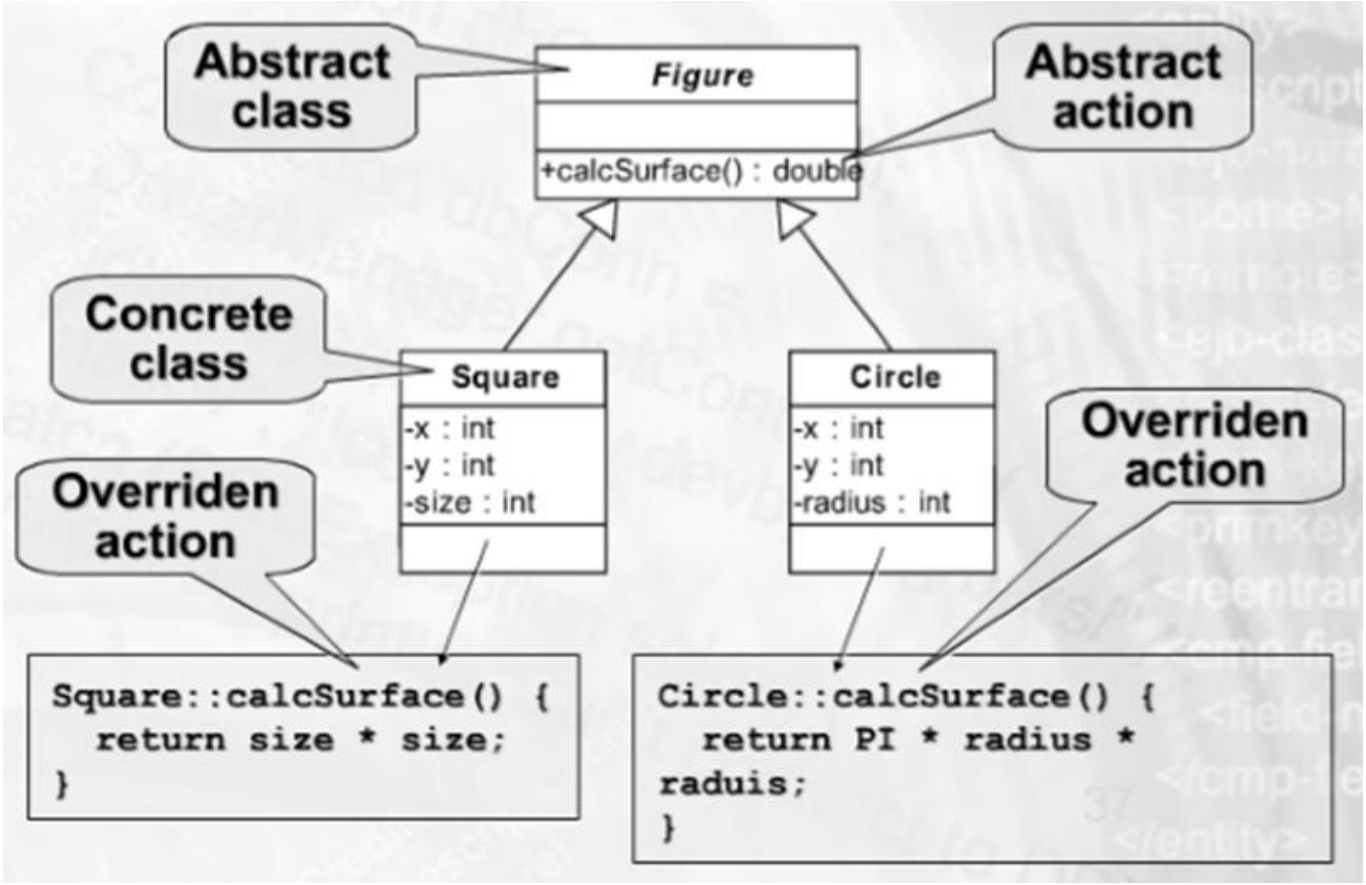


Figure f1 = new Square();

Figure f2 = new Circle();

double d1 = f1.calcSurface(); → **poziva se calcSurface() iz Square**

double d2 = f2.calcSurface(); → **poziva se calcSurface() iz Circle**

Jednostruko i višestruko nasleđivanje

- **Jednostruko – klasa može naslediti tačno jednu klasu**
- **Višestruko – klasa može naslediti više od jedne klase**
- **Diamond problem**
  - `abstract class Figura` – apstraktni metod obim
  - `class Kvadrat extends Figura` – implementira metod obim
  - `class Krug extends Figura` – implementira metod obim
  - `class KvKrug extends Kvadrat, Krug`
    - Klasa KvKrug **ne implementira** metod obim
  - ```
    KvKrug kk = new KvKrug();
    kk.obim();
    ```
    koji obim() se poziva – iz klase Kvadrat ili klase Krug?
    ```

OOP u PJ Java

- Java program je kolekcija **klasa** i **interfejsa** koji se mogu grupisati po **paketima**
- Klase: apstraktne, konkretne (ne-finalne i finalne), ugnježdene (statičke i nestatičke), anonimne, lokalne
- **Interfejsi – specifikacija funkcionalnosti klase**
  - Interfejs – skup zaglavlja javnih metoda
  - Klase implementiraju interfejse
  - **Interfejsi su takođe referencijalni tipovi objekata**
  - Interfejs je ugovor – ako ne-apstraktna klasa implementira interfejs tada ona implementira sve metode propisane interfejsom
- **Paketi – prostori imena, mehanizam hijerarhijske dekompozicije**
  - Klase u istom paketu imaju različita imena, klase iz različitih paketa mogu da imaju isto ime
  - Paketi mogu da sadrže pakete, srodne klase i interfejse stavljamo u isti paket