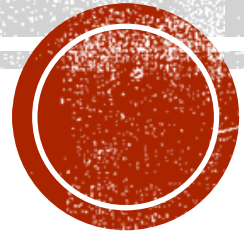
ОБЈЕКТНО ОРИЈЕНТИСАНО ПРОГРАМИРАЊЕ

ПРОГРАМСКИ ЈЕЗИК ЈАВА – 1

Објектно оријентисано програмирање (ООП)



УОПШТЕНО

- ООП је парадигма заснована на скупу објеката који интерагују.
- Интеракције:
  - манипулисање објеката (стање),
  - размена информација између објеката.
- Решавање проблема преко објеката који дејствују међусобно.
- Репрезентација објеката ни превише општа ни превише конкретна.
  - Универзалност.
  - Ефикасност.

КАРАКТЕРИСТИКЕ

- Објекти су ентитети који садрже податке.
  - Захтеви се могу проследити другим објектима.
  - Или сами реализовати у виду операција.
- Програм је скупина објеката који (преко порука) шаљу захтеве једни другима.
  - Порука је захтев да се позове метод који припада датом објекту.
- Сваки објекат поседује сопствену меморију.
  - У тој меморији се могу наћи и други објекти.
- Сваки објекат има свој тип.
  - Сваки од објеката је примерак неког типа.
- Сви објекти датог типа могу прихватати и процесирати исте поруке.

ИСТОРИЈАТ И РАЗВОЈ

- Први објектно оријентисани језик био је **Simula 67**, настао 1967. године.
  - Није стекао велику популарност.
- Главни промотери ООП парадигме су творци језика **SmallTalk.**
  - Алан Кеј и остали.
  - Агресивно рекламирање утицало на развој парадигме.
- Године 1983. појављује се језик **C++.**
  - Објектно оријентисана надградња тада популарног процедуралног језика C.
  - Доприноси популаризацији ООП стила програмирања.
- Нови, моћни програмски језици (**Java, C#, Python, …**), који подржавају ООП.
- ООП приступ омогућава лако коришћење већ написаног софтвера (тзв. рејузабилност софтвера)
  - Али и читав низ нових концепата.

ОСНОВНИ ПОЈМОВИ

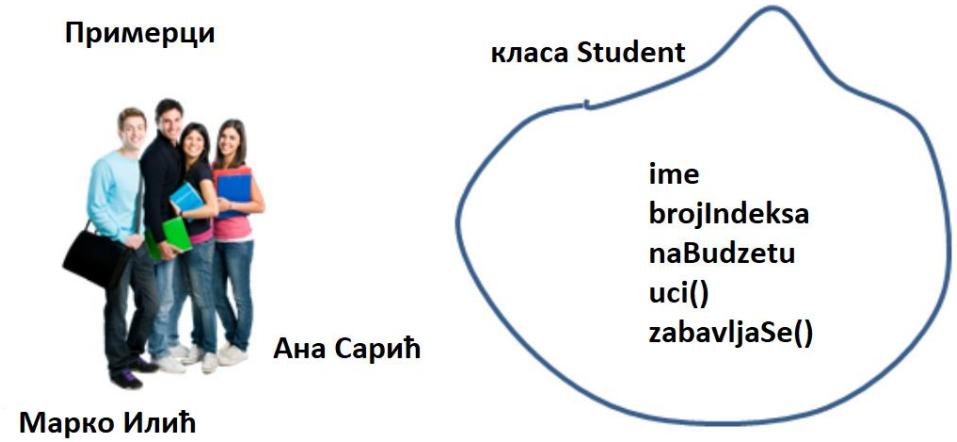
- **Објекат** је интегрална целина података и функција (процедура) за рад са њима.
- Функције (процедуре) се, под утицајем SmallTalk-а, називају **методи**.
- У објекту су **учаурени (енкапсулирани):**
  - подаци које тај објекат садржи, тзв. **атрибути**
  - и методи за рад са њима.
- **Стање** објекта је описано вредностима атрибута.
  - Промена стања објекта би требала да се мења само путем метода тог објекта. (Није пожељно да објекат није свестан промене свог унутрашњег стања.)
- **Метод** је функција (процедура) која је саставни део објекта.
- **Порука** је скуп информација које се шаље објекту.
  - Ефекат слања поруке је обично извршавање метода над објектом.
  - Информативне, упитне и командне поруке.

ОБЈЕКТИ

- Поред метода и атрибута, објекат нужно има и идентитет.
  - Обично информација изведена из адресе меморијске локације објекта.
- Програмер може:
  - сам дизајнирати нове објекте,
  - користити постојеће из програмских библиотека.
- Квалитетан дизајн:
  - објекат ради једну ствар добро,
  - не покушава да уради превише.

КЛАСЕ И ПРИМЕРЦИ

- **Класа** је шаблон за креирање конкретних објеката.
  - Описује структуру објекта (атрибуте).
  - Описује скуп функционалности (методе) објекта.
  - Сви методи, па и главни **main** метод смештени унутар неке класе.
- **Инстанца (примерак)** класе је конкретан објекат дате класе.
  - У даљем тексту су инстанца, примерак и објекат синоними.



ПРИМЕР 1

- По угледу на претходну слику, реализовати Јава програм.

ПРИМЕР 1 (2)

```
class Student {
    String ime;
    int brojIndeksa;
    boolean naBudzetu;

    void stampajPodatke() {
        System.out.println("Име студента је: "
              + ime + ", број индекса је: "
              + brojIndeksa + "/ На буџету: "
              + naBudzetu + ".");
    }
}
```

```
class StudentPokretanje {
    // улазна тачка програма
    public static void main(String[] args) {
        Student prvi = new Student();
        prvi.ime = "Марко Илић";
        prvi.brojIndeksa = 243;
        prvi.naBudzetu = false;

        Student drugi;
        drugi = new Student();
        drugi.ime = "Ана Сарић";
        drugi.brojIndeksa = 25;
        drugi.naBudzetu = true;

        prvi.stampajPodatke();
        drugi.stampajPodatke();
    }
}
```

```
Име студента је: Марко Илић, број индекса је: 243/ На буџету: false.
Име студента је: Ана Сарић, број индекса је: 25/ На буџету: true.
```

НАСЛЕЂИВАЊЕ

- **Потклаcа** класе А је класа В:
  - уколико све инстанце класе В имају иста својства као инстанце класе А.
- Потклаcе обично настају додавањем нових атрибута и метода постојећој класи.
- **Наткласом** класе В називамо класу А уколико је В потклаcа класе А.
- **Наслеђивање** је механизам за крерање нових класа из постојећих.
- Формирање **релација** између једне класе и више других:
  - **конкретизација**, на пример «Последипломац је конкретизација класе Студент»;
  - **уопштење**, на пример «Студент је уопштење класе Последипломац».
- Наслеђивање омогућава употребу метода и атрибута унутар потклаcа.
  - Једном написан метод не мора да се напише поново.

ПРИМЕР 2

- Дефинишимо класу **Poslediplomac** која је потклаcа класе **Student** и која додатно садржи: логичко поље **zaposlen**, целобројно **brojIspita** и још два метода, један за постављање броја испита, а други за враћање његове вредности.
- Потклаcа се генерише помоћу кључне речи **extends**.

ПРИМЕР 2 (2)

```
class Poslediplomac extends Student {
    boolean zaposlen;
    private int brojIspita;

    public void postavibrojIspita(int b) {
        brojIspita = b;
    }

    public int uzmibrojIspita() {
        return brojIspita;
    }
}
```

```
Poslediplomac novi = new Poslediplomac();
novi.postavibrojIspita(n);
novi.ime = "Петар Перић";
novi.brojIndeksa = 4;
novi.naBudzetu = true;
novi.stampajPodatke();
System.out.println(novi.ime + " је положио "
 + novi.uzmibrojIspita()
 + "        испита.");
```

```
Име студента је: Петар Перић, број индекса је: 4/ На буџету: true.
Петар Перић је положио 7 испита.
```

КОМПОЗИЦИЈА

- Релације **садржавања**:
  - име је објекат и притом **део** студента (или мотор је део аутомобила),
  - студент **садржи** објекат име (аутомобил садржи мотор).
- Сложени објекти могу бити композиције једноставнијих (**композиција**).
- Композицији и/или наслеђивање.

ДИНАМИЧКО (КАСНО) ВЕЗИВАЊЕ

- **Динамичко (касно) везивање** података.
  - Током извршавања се (на основу поруке) одређује која ће се операција извршити.
  - Меморијска локација се одређује током извршавања.
  - Попут динамичке алокације меморије у C-у.
- Насупрот томе је **статичко (рано) везивање**.
  - Начин оперисања познат већ у фази превођења.
  - (Релативна) меморијска локација позната већ у фази превођења.
  - Попут статичке алокације меморије у C-у.

УЧАУРИВАЊЕ (ЕНКАПСУЛАЦИЈА)

- Учауривањем се скривају подаци унутар објекта и онемогућава неконтролисан приступ овим подацима.
- На тај начин спречава се тзв. домино-ефекат (енг. side effect).
- Учауривањем се може постићи и боља расподела функционалности међу класама.
  - Свака класа располаже само подацима који су јој природно додељени.
  - Ово онемогућава мешање објекта једне класе у функционалности објеката друге класе.

ПРИМЕР 3

- Нека су дате две класе **Krug** и **Crtez**, при чему су на цртежу нацртана два круга.
- Природно је да **Krug** садржи учаурене податке који га описују, на пример центар круга и полупречник.
- **Crtez** садржи, као атрибуте, два круга.
- Објекат класе **Crtez** не треба да има директан приступ подацима који описују круг, већ да им приступа или манипулише њима искључиво позивањем метода над објектима класе круг, нпр. метод **transliraj**.
- **Crtez** има могућност транслирања свих кругова за дати вектор помераја.

ПРИМЕР 3 (2)

```
class Krug {
    int cx, cy;
    int r;

    Krug(int cx, int cy, int r) {
      this.cx = cx;
      this.cy = cy;
      this.r = r;
    }

    void transliraj(int dx, int dy) {
      cx += dx;
      cy += dy;
    }

    void prikazi() {
      System.out.println("Круг C = (" + cx + ", "
        + cy + ") и r = " + r);
    }
}
```

```
class Crtez {
    Krug krug1;
    Krug krug2;

    public Crtez() {
      krug1 = new Krug(10, 20, 11);
      krug2 = new Krug(17, 20, 4);
    }

    void translirajSve(int dx, int dy) {
      krug1.transliraj(dx, dy);
      krug2.transliraj(dx, dy);
    }

    void prikazi() {
      krug1.prikazi();
      krug2.prikazi();
    }
}
```

ПРИМЕР 4

- Написати Јава програм који одређује НЗД и НЗС за три броја.
- Рачунање НЗД и НЗС треба да буде учаурено унутар објекта.
- Класа у којој су учаурене атрибут вредности броја, методи за рачунање НЗД и НЗС, као и метод за приказ броја, налазе се у класи **CeoBroj**.

ПРИМЕР 4 (2)

- Решење у тексту поглавља.

ПРЕДНОСТИ ООП

- Погодан за: анализу, пројектовање и програмирање.
- Олакшано одржавање софтвера.
- Омогућава лако и једноставно уклапање модула.
- Поновна искористивост софтвера.
- Најпогоднији за симулирање догађаја.
- Омогућава паралелни рад више програмера на истом пројекту.

МАНЕ ООП

- Неефикасност — тежи се већем коришћењу централне процесорске јединице
- Понекад се генерише код који није неопходан.
- Може да доведе до дуплирања кода.
- Утиче на спорије извршавање програма (због динамичког везивања).
- Није погодно за све типове проблема (нпр. проблеми у којима су непходна интезивна нумерричка израчунавања).
- Компликованије је програмирање засновано на интеракцији објеката.

ПИТАЊА И ЗАДАЦИ

1. Која је основна идеја објектно оријентисане парадигме?
2. Кратко описати историјат објектно оријентисаног програмирања.
3. Шта је класа, а шта је инстанца/конкретан објекат класе?
4. Објаснити механизам наслеђивања и навести конкретне примере.
5. Упоредити динамичко и статичко везивање.
6. Шта се подразумева под учаурењем објекта?
7. Које су предности, а које недостаци објектно оријентисаног програмирања?