



Distance Learning System

Core Java Programming

Objektno Orjentisano Programiranje
Klase i Objekti

Objektno Orjentisano programiranje

- Objektno orjentisano programiranje je programiranje koje omogućava programerima da razvijaju programe, kreirajući objekte i relacije između tih objekata.

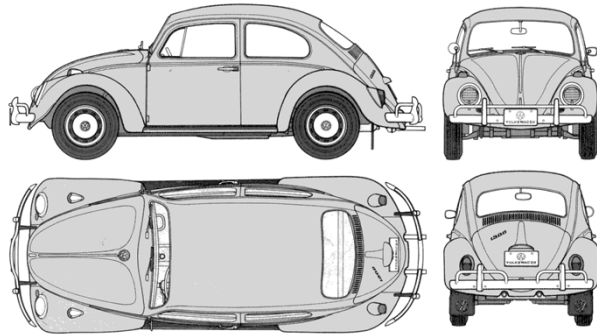
Java u svetu OOP-a

- Iako nije prvi ni jedini objektno orjentisani programski jezik, Java predstavlja simbol objektnog programiranja današnjice
- Java je potpuno objektno orjentisana, što znači da zapravo ne može postojati bez objekata
- Jezici koji mogu da se koriste u objektnom kontekstu ili bez njega, nazivaju se **hibridnim jezicima**
 - Poznatiji hibridni jezici su: C++, php, ...
 - Poznatiji potpuno objektno orjentisani jezici su: C#, Visual Basic
- Karakteristike objektnih jezika su: klase, objekti, metode, svojstva

Klase

- Klasa predstavlja šablon po kome se proizvode objekti koji su još poznati i kao instanca klase.
- Klase generalizuju entitete realnog sveta, dok objekti predstavljaju specifične manifestacije ovih entiteta.
- Na klasu se može misliti kao na modlicu (kalup) po kojoj se prave kolačići.

Klasa BUBA



Objekti klase BUBA



Klase

- Objekat ili instanca jeste jedan primerak nečega, napravljen prema određenom uzorku (klasi). Svaka ptica jeste jedan objekat (instancja) pomenute klase ptica, dok je svaki čovek, objekat klase čovek.
- Da li je ptica instanca klase avion? Verovatno ne. Ali i ptica i avion mogu biti instanca klase: leteći objekat

Kreiranje klasa i objekata

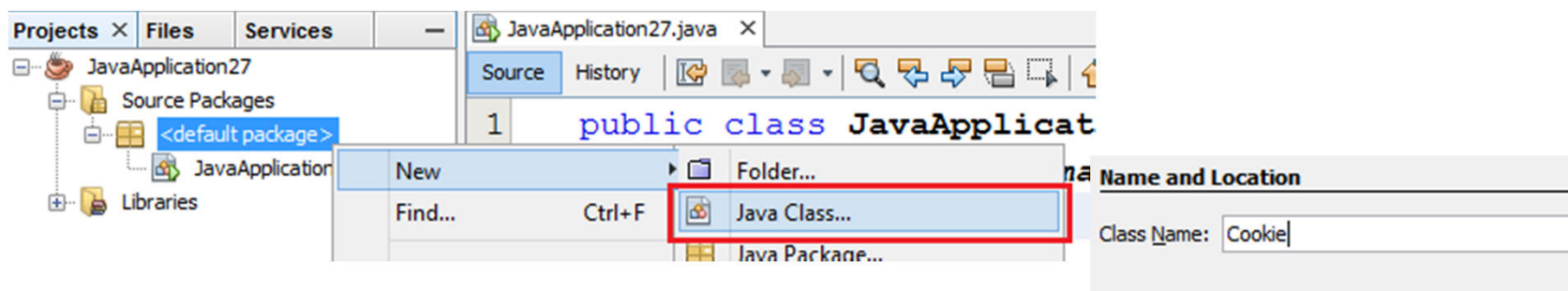
- Prvi korak u procesu kreiranja klasa i objekata jeste kreiranje klase.
- Iako je u nekim jezicima moguće kreirati objekat bez prethodnog kreiranja klase, u Javi to nije moguće
- Da bi u Javi kreirali klasu, kreiramo novi fajl (koji se zove kao i klasa koju hoćemo da kreiramo), u njemu pišemo:

```
class Cookie {  
    // various member declarations  
}
```

- Ovo nam je već poznato, zar ne?

Kreiranje klasa i objekata

- U NetBeansu već postoji šablon za kreiranje ovakvog fajla, pa treba samo aktivirati kontekstni meni iznad odgovarajućeg paketa, a zatim odabrati opciju **new->Java Class**



- Kod imenovanja klasa poštujemo ista pravila kao i kod imenovanja promenljivih, pri čemu postoji još jedno „nepisano“ pravilo: Nazive klasa pišemo početnim velikim slovom ukoliko je to moguće, a ako naziv ima više reči, onda svaki početak reči takođe.
- Na primer: class Person, class MyClass, class SomeOtherClass, class Car...

Kreiranje klasa i objekata

- Kada kreiramo klasu, možemo je koristiti za kreiranje objekata

```
Cookie cookie = new Cookie();
```

- *Napisana linija znači: napravi jedan objekat po šablonu **Cookie** i smesti ga u promenljivu koja se zove **cookie**.*
- Takođe smo mogli napisati i:

```
Cookie cookie;  
cookie = new Cookie();
```


Kreiranje klasa i objekata

- Klasa koja je kreirana u prethodnom primeru nema naročit smisao jer, iako se zove Cookie, ne sadrži nikakve informacije o bilo kakvim kolačima. Štaviše, ona ne sadrži nikakve informacije.

```
class Person {  
    String name;  
    String surname;  
    int age;  
    int height;  
    int weight;  
}
```

- Klasa Person sadrži neke informacije. Tačnije, ona sadrži polja u koja možemo smestiti informacije. Ime, prezime, godište, visina i težina su karakteristike kojima se može opisati jedan čovek

Kreiranje klasa i objekata

- Sledećim kodom bi mogli instancirati jednu osobu po šablonu Person:

```
Person person = new Person();  
person.name = "Peter";  
person.surname = "Jackson";  
person.age = 50;  
person.height = 176;  
person.weight = 80;
```

- Ako ovako nešto uradimo u kodu, ništa se neće dogoditi na izlazu, ali će program znati da postoji promenljiva (objekat) sa nazivom person, koja sadrži podatke koje karakterišu jednog čoveka.

Kreiranje klasa i objekata

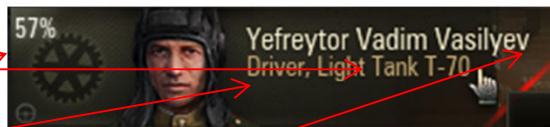
- Iako Klase / objekti oslikavaju objekte iz realnog sveta, oni ih ne moraju mapirati u potpunosti, već treba da odgovaraju potrebama sistema
- Na primer, za osobu sa slike, mogli bi kreirati klasu Person, ali za nju nam ne bi bila važna težina i visina, dok bi nam sa druge strane, bilo važno iskustvo



Kreiranje klasa i objekata

- Pogledajmo kako bi izgledala implementacija ove klase

```
public class TankCrewPerson {  
    int vehicle_id;  
    int experience;  
    byte category;  
    String name_and_surname;  
}
```



- Kreiranje objekta bi moglo izgledati ovako:

```
TankCrewPerson tc_person = new TankCrewPerson();  
tc_person.experience = 57;  
tc_person.category = 1;  
tc_person.name_and_surname = "Yefreytor Vadim Vasilyev";  
tc_person.vehicle_id = 5;
```

Kreiranje klasa i objekata

- Pored toga što imaju karakteristike, ljudi mogu nešto i da rade
- Mogućnost objekata da nešto rade, ostvaruje se pomoću metoda
- Metode su takođe sastavni delovi objekta, ali se malo drugačije ponašaju i označavaju od običnih podataka

```
class Person {  
    String name;  
    String surname;  
    int age;  
    int height;  
    int weight;  
    void sleep() {}  
    void run() {}  
}
```

Polja klase

Metode klase

Tri najbitnija koncepta objektnog programiranja

- Enkapsulacija
- Nasleđivanje
- Polimorfizam

Enkapsulacija

- Pojam enkapsulacije podrazumeva sakrivanje kompleksne logike neke funkcionalnosti.

Za ovo je odličan primer automobil. Kada okrenemo ključ u automobilu, dolazi do startovanja motora. Ono što mi, kao vozači, znamo, je da treba da okrenemo ključ da bismo aktivirali motor. Ali, ono što ceo proces aktivacije motora podrazumeva, mnogo je kompleksnije od okretanja ključa. Kreator automobila sakrio je kompleksnu logiku aktivacije motora od nas i dao nam pristup njegovim najbitnijim funkcijama.

Nije ga briga ko je
za volanom

Vozač

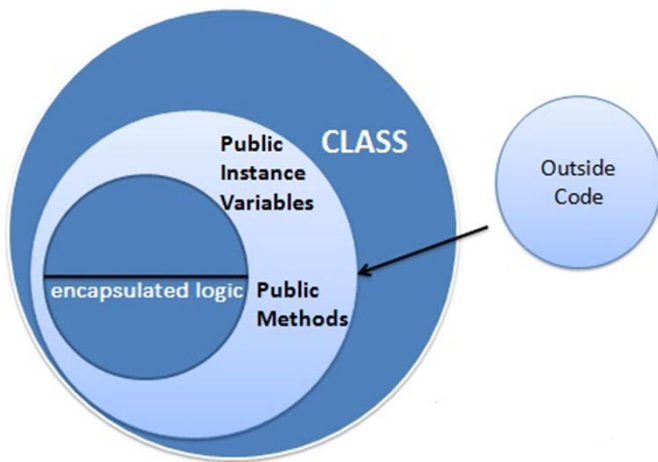


Automobil

Nije ga briga šta se
događa u pozadini

Enkapsulacija

- Ako postoji klasa Person, i poseduje metodu run, tada najverovatnije nećemo hteti da znamo šta se u toj metodi događa, već hoćemo samo da vidimo krajnji rezultat, a to je trčanje.



```
Person somePerson = new Person();  
somePerson.run();
```

Ne zanima nas kako ovaj metod funkcioniše, sve dok nam daje željene rezultate

Enkapsulacija

- Metode i svojstva ugrađenih Java klasa upravo oslikavaju primer enkapsulacije. Mi u našim programima koristimo ove metode i svojstva bez ikakvog znanja o načinu njihovog funkcionisanja.

```
System.out.println("What is happening inside me?");
```

```
public void println(String x) {  
    synchronized (this) {  
        print(x);  
        newLine();  
    }  
}
```

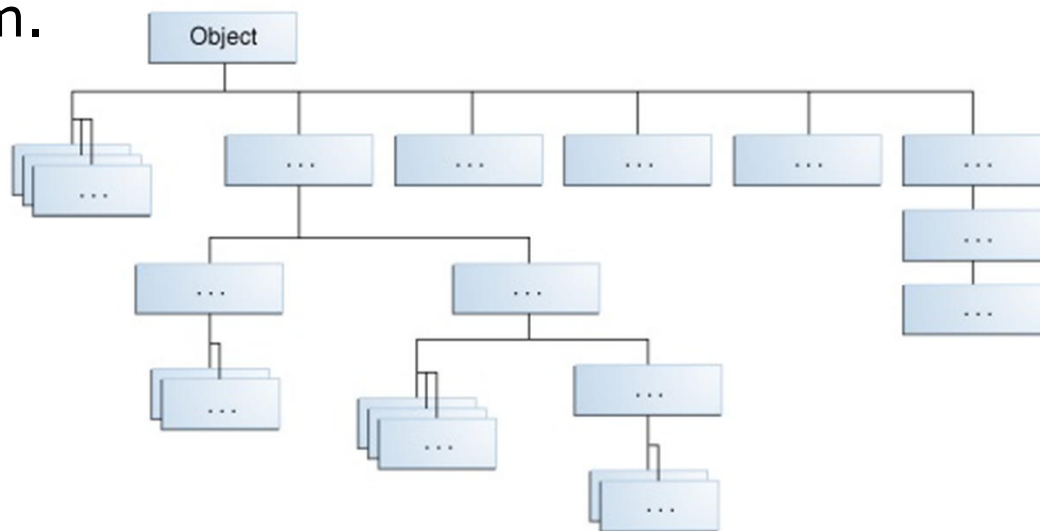
```
public void print(String s) {  
    if (s == null) {  
        s = "null";  
    }  
    write(s);  
}
```

```
private void write(String s) {  
    try {  
        synchronized (this) {  
            ensureOpen();  
            textOut.write(s);  
            textOut.flushBuffer();  
            charOut.flushBuffer();  
            if (autoFlush && (s.indexOf('\n') >= 0))  
                out.flush();  
        }  
    }  
}
```

...

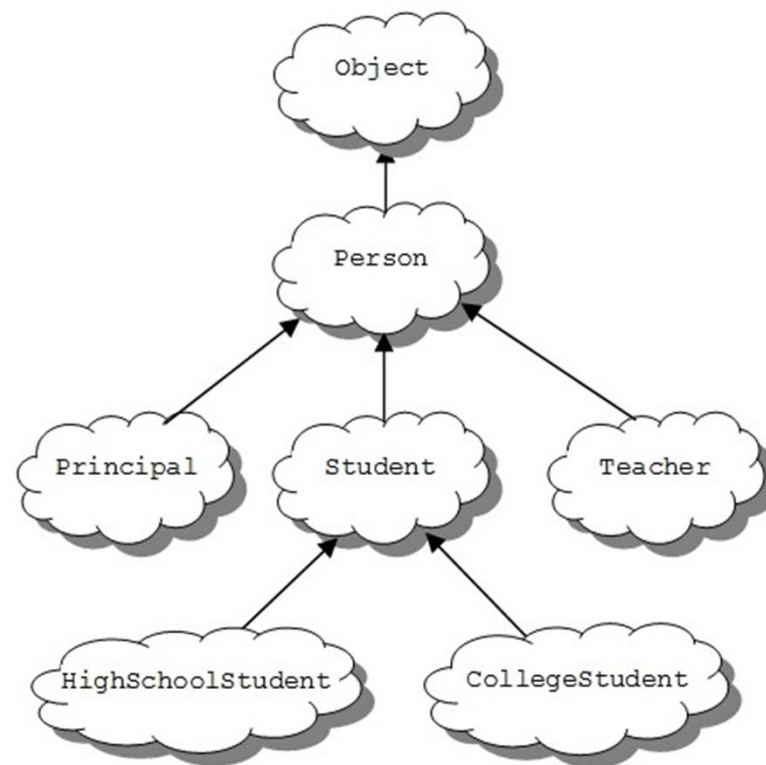
Nasleđivanje (inheritance)

- Nasleđivanje definiše odnos između klasa, a samim tim i objekata u jednom objektno-orientisanom sistemu.
- Sve klase u Javi imaju neku drugu klasu iznad, koja se može nazvati njihovim roditeljom.
- Specijalna klasa, jedina koja nema svog roditelja je klasa koja se zove Object.
- Svi drugi objekti u Javi predstavljaju naslednike ovog tipa.



Nasleđivanje (inheritance)

- Možemo reći da smo svi mi instance jedne klase Osoba (Person). Ipak svako od nas može obavljati različite poslove, pa se na toj osnovi mogu kreirati novi skupovi osoba u zavisnosti od posla koji obavljaju (Principal, Student, Teacher).
 - Student može biti na visokoj školi ili koledžu. Ako bismo to preveli na klasni model objektno-orijentisanog jezika, može se reći da klasa HighSchoolStudent nasleđuje klasu Student, a da klasa Student nasleđuje klasu Person.
- Svaka klasa koja nema kreiranog roditelja, kao svoju nadklasu ima klasu Object. Ova struktura je prikazana na ilustraciji.



Nasleđivanje - implementacija

- Da bismo naglasili da klasa u Javi nasleđuje neku drugu klasu koristimo ključnu reč `extends`. Pogledajmo to na primeru klasa `Person` i `Student`. Jasno je da klasa `Student` nasleđuje klasu `Person`.

```
class Person {  
  
}  
class Student extends Person {  
  
}
```

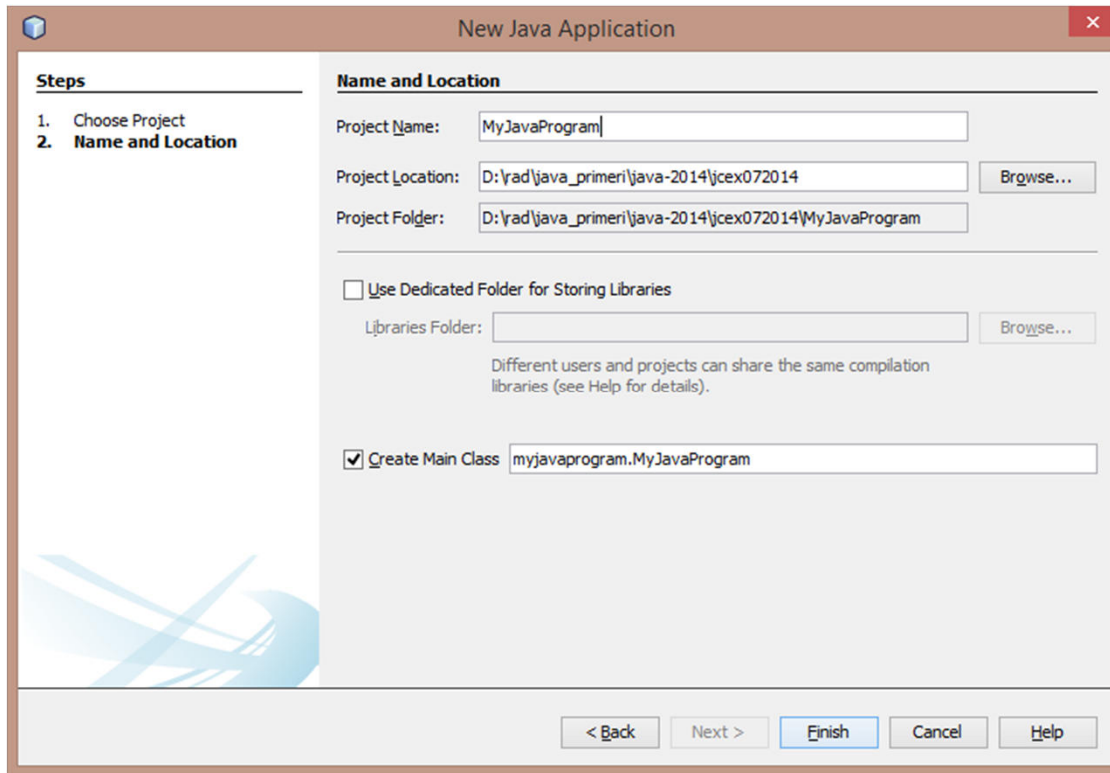


Distance Learning System

Core Java Programming

Članovi klase

„Prva“ Java klasa (jcex072014 MyJavaProgram)



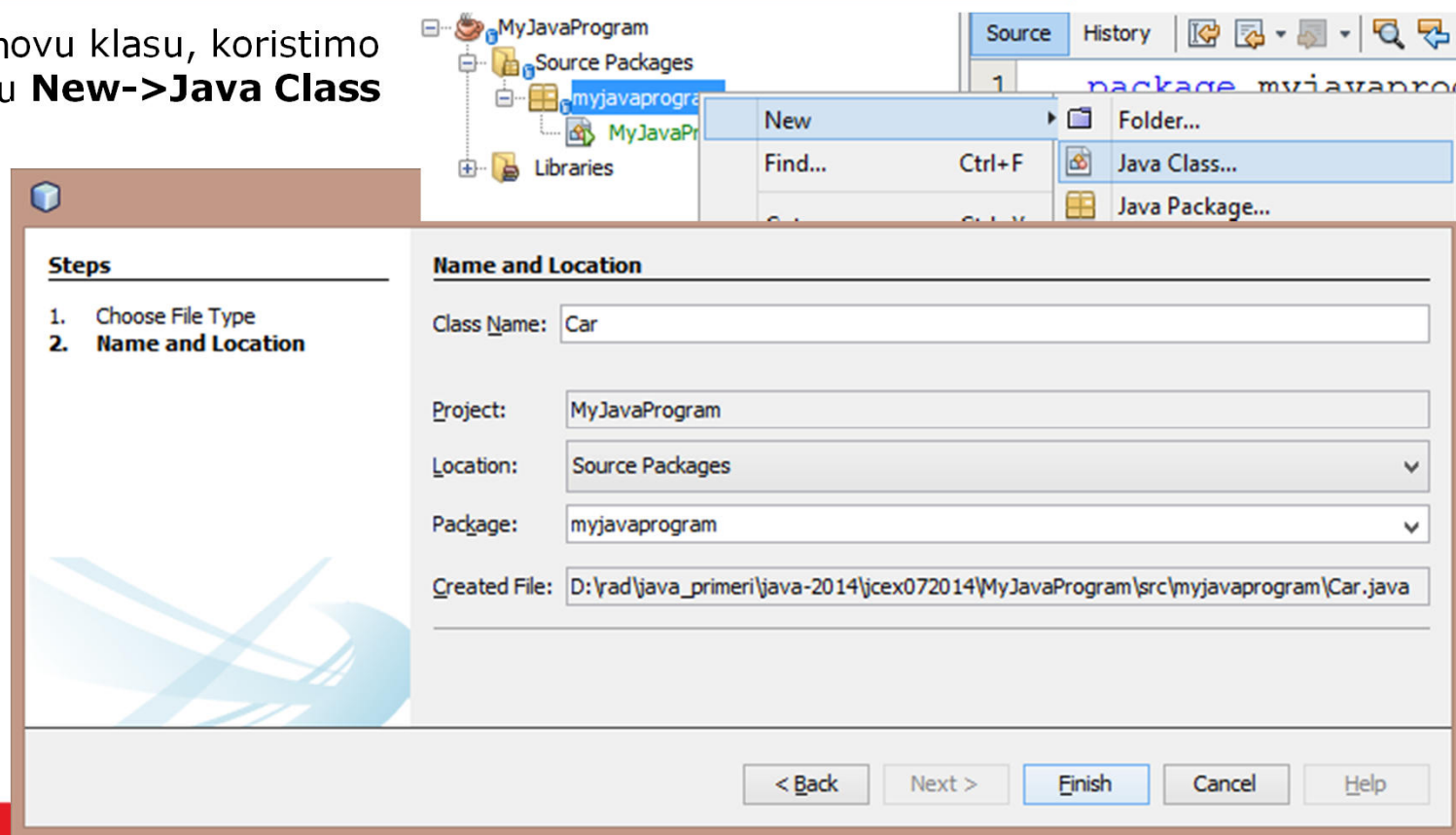
- Da bi ispratili primere u nastavku, treba kreirati novi Java Application projekat pod nazivom MyJavaProgram
- Projekat kreiran po navedenim parametrima daje sledeću strukturu



Prva Java klasa

- Da bi kreirali novu klasu, koristimo opciju **New->Java Class**

- U prozoru koji se otvara upisujemo naziv klase: Car (pod uslovom da pravimo klasu Car)



Struktura klase (jcex072014 MyJavaProgram)

- Prvom linijom definisan je paket u kome se nalazi klasa definisana u kodu ispod.
- Sledi definicija klase.

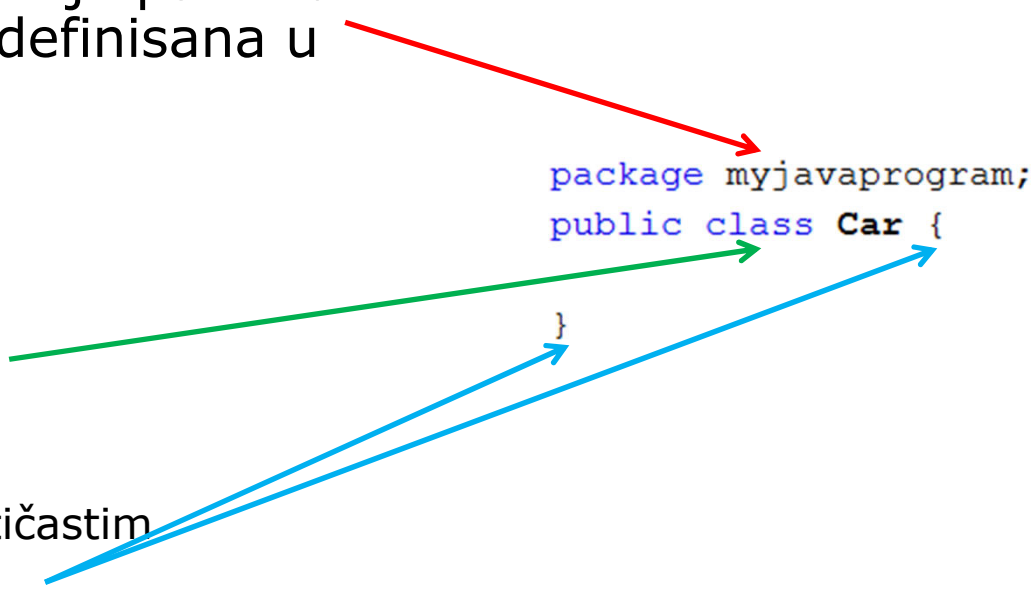
- Zaglavlje

- Public modifikator
- Ključna reč **class**
- Naziv klase

- Telo

- Predstavljeno je vitičastim zagradama
- Trenutno je prazno

```
package myjavaprogram;  
public class Car {  
    }  
}
```



Članovi klase

- Dele se na **instancne** i **statičke**
- Članovi su:
 - **Polja**
 - **Metode**
 - **Druge klase**

Modifikatori pristupa

- Modifikatori pristupa predstavljaju opseg dostupnosti određenih članova klase
- Modifikatori pristupa su:
 - **Private** - članovi su vidljivi samo u okviru klase u kojoj se nalaze
 - **Package-private** (često se naziva samo package) svi drugi članovi istog paketa imaju pristup komponenti
 - **Protected** elementima se može pristupiti iz same klase gde su definisani i iz izvedenih klasa
 - **Public** bilo koji objekat ima pristup nad elementom

Polja klase

- Polja su zapravo promenljive koje se nalaze u okviru klase
- Dele se na **instancna** i **statička**

```
package myjavaprogram;  
public class Car {  
    String make;  
    String model;  
    int numDoors;  
    static int wheels = 4;  
}
```

Instancno

```
public static void main(String[] args) {  
    Car c = new Car();  
    c.make = "BMW";  
}
```

Statičko

```
public static void main(String[] args) {  
    System.out.println(Car.wheels);  
}
```

Private modifikator pristupa

- Na prethodnom primeru ne postoje modifikatori pristupa, i zato su polja javna.
- Ako bi postavili privatni modifikator na neko od polja, ono ne bi bilo dostupno izvan klase:

```
package myjavaprogram;  
public class Car {  
    private String make;  
    String model;  
    int numDoors;  
    static int wheels = 4;  
}
```

**Pokušaj pristupanja
polju izvan klase
izaziva grešku**

```
public static void main(String[] args) {  
    Car car = new Car();  
    car.make = "BMW";  
}
```

make has private access in Car

(Alt-Enter shows hints)

Metode

- Pored polja, među članovima klase nalaze se i **metode**
- Metode nisu ništa drugo do funkcije koje se nalaze unutar klase
- Obzirom da je Java potpuno objektni jezik, funkcija kao pojam i ne postoji, već samo metoda
- Metod koji se aktivira automatski prilikom kreiranja klase, naziva se **konstruktor**
- Unutar konstruktora, možemo postaviti neke inicijalne vrednosti instanciranom objektu

Konstruktor (jcex072014 Constructor)

- Konstruktor se automatski aktivira prilikom instanciranja klase

```
1 package myjavaprogram;
  public class Car {
      String make;
      String model;
      int numDoors;
      static int wheels = 4;
      public Car() {
          System.out.println("The car is being created");
      }
  }

2 public static void main(String[] args) {
    Car car = new Car();
}

3 run:
The car is being created
BUILD SUCCESSFUL (total time: 0 seconds)
```

Metod main

- Pored konstruktora, još jedan veoma važan metod je: **main**
- Ovaj metod već jako dobro poznajemo, jer bez njega nije moguće startovati Java program (to ne znači da program mora imati ovaj metod)
- Podsetimo se metode main. Sada bi trebalo da nam neke stvari budu mnogo jasnije:

The diagram shows the following code snippet with annotations:

```
package myjavaprogram;  
public class MyJavaProgram {  
    public static void main(String[] args) {  
        Car car = new Car();  
    }  
}
```

Annotations and arrows:

- Javno dostupan** (red text) with a red arrow pointing to `public`.
- Potpis metode** (black text) with a green arrow pointing to `main`.
- Statički metod** (green text) with a green arrow pointing to `static`.
- Ne vraća vrednost** (orange text) with an orange arrow pointing to `void`.
- Prihvata parametre** (blue text) with a blue arrow pointing to `String[] args`.
- Telo metode** (black text) with a blue arrow pointing to the opening curly brace of the method body.

Statičné metode

- Statičné metode nemajú veze sa objektom klase, veé sa samom klasom. Zbog toga ne mogu pristupati instancnim élanovima, veé samo statiékim

```
package methods1;  
public class Car {  
    String make;  
    String model;  
    int numDoors;  
    static int wheels = 4;  
    public static void HowManyWheels() {  
        System.out.println(wheels);  
    }  
}
```

Poziv

`Car.HowManyWheels();`

Instancne metode (jcex072014 Methods1)

- Karakteristika instancnih metoda je da postoje u kontekstu objekta. To znači da su im dostupni svi članovi objekta klase u kojoj se nalaze
- Ove metode nije moguće startovati bez prethodno kreiranog objekta

```
package methods1;  
public class Car {  
    String make;  
    String model;  
    int numDoors;  
    static int wheels = 4;  
    public static void HowManyWheels() {  
        System.out.println(wheels);  
    }  
    public Car() {  
        model = "I5";  
    }  
}
```

```
Car car = new Car();  
System.out.println(car.model);
```

Isti podatak

```
run:  
I5  
BUILD SUCCESSFUL (total time: 0 seconds)
```

Instancne metode (jcex072014 Methods1)

```
public static void HowManyWheels() {...3 lines }  
void printDetails() {  
    System.out.println("Make " + make);  
    System.out.println("Model " + model);  
    System.out.println("Number of doors " + numDoors);  
}  
public Car() {...3 lines }
```

- Metoda sa leve strane ima naziv printDetails i služi za prikaz vrednosti polja kreiranog automobila:

- Da bi je aktivirali, kucamo:

```
public static void main(String[] args) {  
    Car car = new Car();  
    car.printDetails();  
}
```

- Izlaz programa je:

```
run:  
Make null  
Model IS  
Number of doors 0  
BUILD SUCCESSFUL (total time: 0 seconds)
```

- Pokušajte da modifikujete program tako da prikaže smislene rezultate

Instancne metode (jcex072014 Methods1)

- Pokušajte da ispravite program iz prethodnog primera, tako da prikaže smislene rezultate

Parametrizacija metoda

- Jedna od ključnih osobina metoda je mogućnost prihvatanja i vraćanja parametara
- Parametri koji ulaze u metod, nazivaju se **ulazni parametri**
- Parametri koje metod vraća, nazivaju se **izlazni parametri** (u pitanju je zapravo samo jedan izlazni parametar)
- Prilikom kreiranja metode, moramo u njegovom potpisu naznačiti koje će parametre vratiti i koje će prihvatiti
- Metod ne mora vratiti ništa, niti prihvatiti ništa, ali onog trenutka kada ga kreiramo, u obavezi smo da poštujemo njegovu definiciju (njegov potpis)
- Za vraćanje vrednosti iz metode, koristimo ključnu reč **return** kojoj sledi vrednost

Parametrizacija metoda (jcex072014 MethodParams)

- Jedan parametrizovan statički metod, može izgledati ovako:

Tip izlaznog parametra

Slanje na izlaz

```
public static int add(int a, int b){  
    int result = a + b;  
    return result;  
}
```

Tipovi ulaznih parametara

- Metod kasnije možemo pozvati na sledeći način:

```
public static void main(String[] args) {  
    int res = add(2,3);  
    System.out.println(res);  
}
```

Parametrizacija metoda - void

- Neretko, metod ne vraća nikakav rezultat
- Kod ovakvih metoda moramo dati do znanja sistemu da metod neće vratiti ništa, i to radimo ključnom rečju **void**

```
public static void emptyResult(){  
    System.out.println("Hello, nothing for you!");  
}
```

Vrednosti i reference

- Parametre možemo proslediti metodama kao reference ili kao vrednosti
- Ovo zavisi od tipa parametara
 - Svi složeni tipovi (objekti i nizovi) prosleđuju se uvek po referenci
 - Svi prosti tipovi prosleđuju se po vrednosti

Prosleđivanje parametara po referenci

- Ukoliko postoji sledeći metod

```
public static void passByReference(Car car) {  
    car.model = "Beetle";  
}
```

- I obratimo mu se na sledeći način

```
Car car = new Car();  
car.model = "Renault 4";  
passByReference(car);  
System.out.println(car.model);
```

- Šta možemo da očekujemo kao rezultat?
 - Pokušajte da samostalno uradite primer i vidite rezultat

Prosleđivanje parametara po vrednosti

- Ukoliko postoji sledeći metod

```
public static void passByValue(int x) {  
    x = 25;  
}
```

- I obratimo mu se na sledeći način

```
int x = 10;  
passByValue(x);  
System.out.println(x);
```

- Šta možemo da očekujemo kao rezultat?
 - Pokušajte da samostalno uradite primer i vidite rezultat

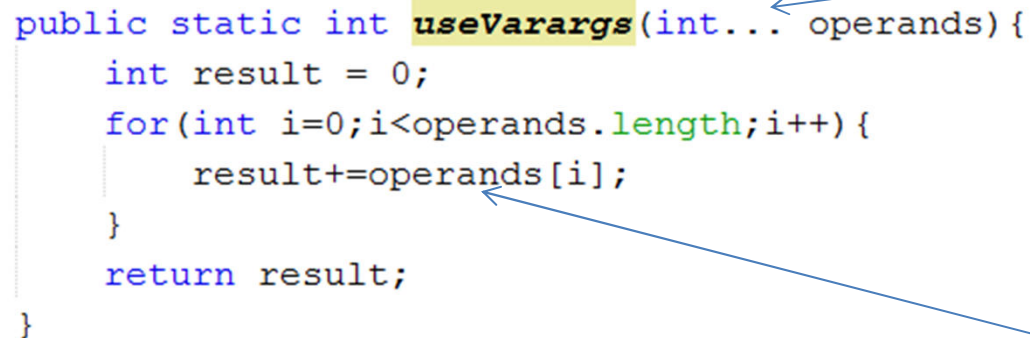
Proizvoljan broj argumenata

- U nekim slučajevima, hoćemo da metod sadrži proizvoljan broj parametara
- Ovo je moguće uraditi na više načina
 - Korišćenjem **varargs** parametrizacije
 - **Preopterećenjem** metode

Prosleđivanje proizvoljnog broja parametara

- Da bi prosledili proizvoljan broj parametara metodi, parametre metode moramo označiti ulazni parametar na specifičan način

```
public static int useVarargs(int... operands) {  
    int result = 0;  
    for(int i=0;i<operands.length;i++){  
        result+=operands[i];  
    }  
    return result;  
}
```

A blue arrow points from the 'int...' part of the method signature to the 'operands.length' property access in the for loop. Another blue arrow points from the 'operands[i]' array access to the 'operands' parameter name in the signature.

- Prilikom poziva, umesto jednog parametra, prosleđujemo ni jedan ili više.

```
System.out.println(useVarargs(1,5,12));
```

- U samoj metodi, parametre ćemo dobiti u formi **niza**

Vežba – Kalkulator (jcex072014 Calculator)

- Potrebno je kreirati klasu kalkulator koja ima dva svojstva: operand1 i operand2.
- Klasa poseduje metode:
 - **add**, koja kao rezultat vraća zbir dva operanda
 - **sub**, koja kao rezultat vraća razliku dva operanda
 - **mul**, koja kao rezultat vraća proizvod dva operanda
 - **div**, koja kao rezultat vraća količnik dva operanda
- Sve vrednosti su tipa double.