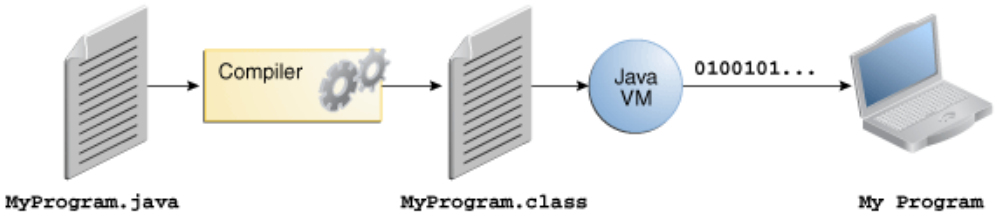
OOP u programskom jeziku Java

-Java programski jezik je jezik visokog nivoa koji se može opisati sledecim epitetma:

- Jednostavan
- Objektno orijentisan
- Distribuiran
- Multithreaded
- Dinamičan
- Nezavisan od arhitekture računara
- Portabilan
- Programski jezik visokih performansi
- Robustan
- Siguran

-Pisanje i kompajliranje JAVA koda:



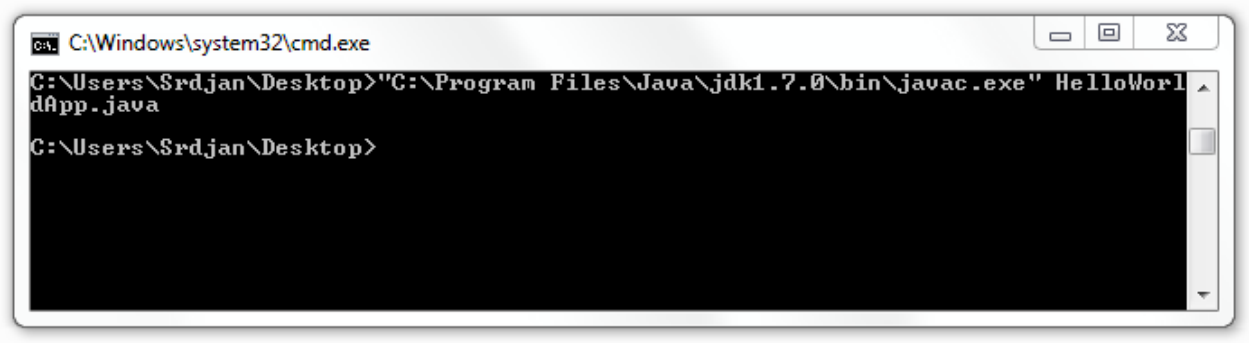
Primer 1.1

- Kreiranje izvornog fajla

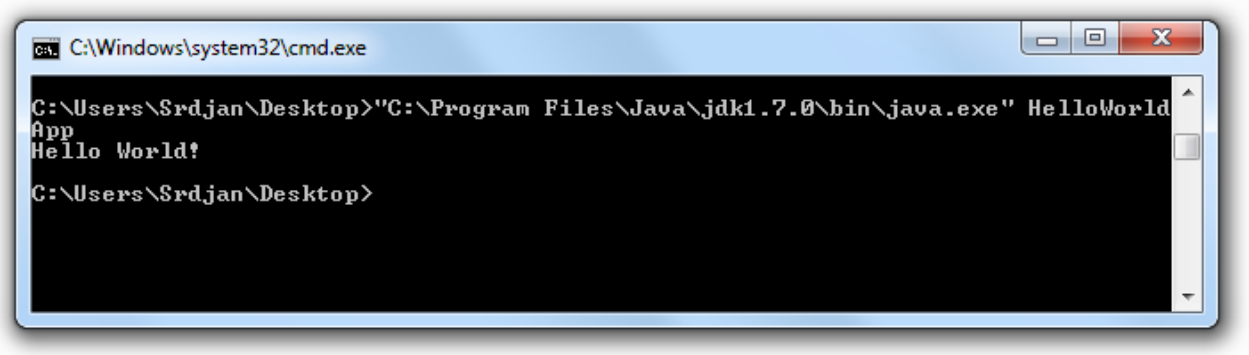
```
/**
* Stampanje stringa "Hello World!" na standardni izlaz.
*/
public class HelloWorldApp
{
	public static void main(String[] args)
	{
		System.out.println("Hello World!"); // Prikazi string.
	}
}
```

Prethodni kod treba sačuvati u fajl *HelloWorldApp.java*

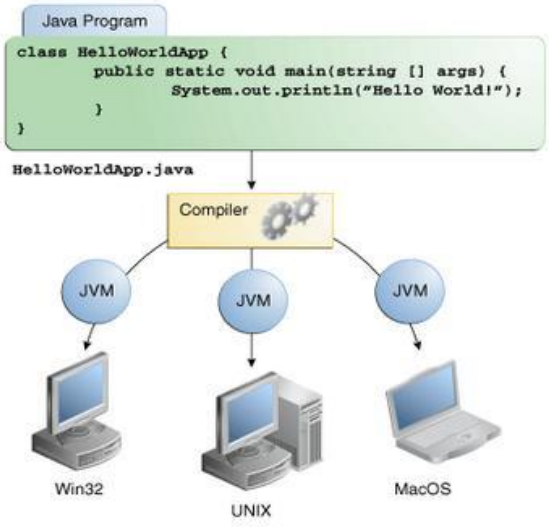
- Kompajliranje izvornog fajla u .class fajl



- Pokretanje izvršnog .class fajla



Java VM (virtuelna mašina) je dostupna na mnogim operativnim sistemima. Isti **.class** fajl moguće je pokrenuti na Windows-u, Linux-u, Solaris OS-u ili Mac OS-u.



Kompajliranje i pokretanje na Linux-u:

- svaka Java aplikacija mora sadržati barem jednu klasu s metodom main(String[] args)
- počinje svoje izvršavanje pozivom metode main
- ovako napisan program se prevodi izvršavajući

javac HelloWorld.java

- Ako nema grešaka prevodilac javac kreira datoteku HelloWorld.class koja sadrži bytecode instrukcije za JVM.
- JVM se pokreće sa

java HelloWorld

Razlika između proceduralnog i objektno-orijentisanog programiranja

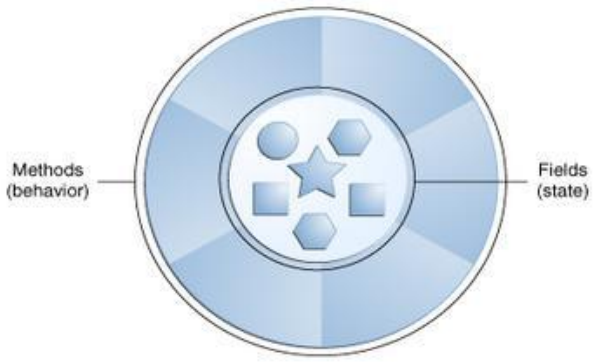
- Standardno, **proceduralno** programiranje(npr. C): Program započinje izvršavanjem funkcije **main** koja izvršava postavljeni zadatak pozivanjem drugih funkcija. Program završava kad se izvrše sve instrukcije funkcije main. Osnovni građevni blok programa je, dakle, **funkcija**. Postavljeni zadatak se rešava tako da što se razbije na niz manjih zadataka od kojih se svaka može implementirati u jednoj funkciji, tako da je program niz funkcijskih poziva.

- U **objektno-orijentisanom** programiranju osnovnu ulogu imaju **objekti koji sadrže i podatke i funkcije (metode)**. Program se konstruiše kao skup objekata koji međusobno komuniciraju. Podaci koje objekat sadrži predstavljaju njegovo **stanje**, dok pomoću metoda on to stanje može da menja i komunicira sa drugim objektima.

Objektno-orijentisani koncept programiranja

Šta je objekat?

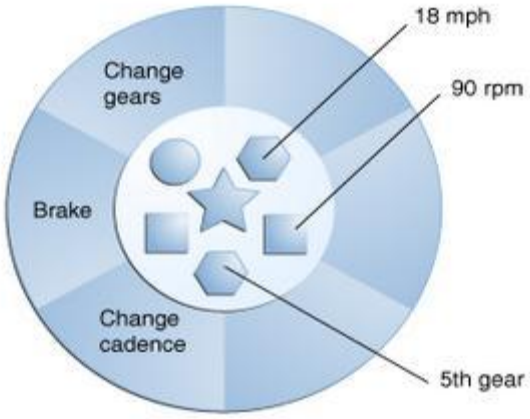
- Objekati su ključna stvar za razumevanje objektno-orijentisane tehnologije. U prirodi oko nas možemo uočiti mnoge primere objekata: Pas, računar, televizor, automobil...
- Objekti u prirodi dele dve karakteristike:
  - **Stanje.** Automobil ima nekoliko stanja: marka, klasa, boja, maksimalna brzina, tip menjača...
  - **Ponašanje.** Automobil može da se upali, ugasi, može da mu se promeni brzina...
- **Programski objekti** su konceptualno isti kao objekti u prirodi. Oni takođe imaju stanje i ponašanje. Objekti svoja stanja čuvaju u poljima (promenljivama), dok svoje ponašanje ispoljavaju uz pomoć metoda (funkcija?).



Prilikom uočavanja objekata oko nas, potrebno je za početak odgovoriti na dva pitanja:

- U kojim stanjima može biti objekat?
- Koja su moguća ponašanja objekta?

Pogledajmo na primeru automobila, stanja i neka njegova moguća ponašanja:



- Pisanje koda u individualne programske objekte nosi sa sobom nekoliko pogodnosti:
    1. Modularnost,
    2. Skrivanje informacija - promena stanja objekta spolja može se vršiti samo preko metoda,
    3. Ponovna upotreba koda,
    4. Jednostavnije otkrivanje i otklanjanje grešaka.

Šta je klasa?

- U prirodi se mogu pronaći mnogi objekti koji su istog tipa. Postoje hiljade automobila koji su od istog proizvođača i istog tipa. Svaki od tih automobila izrađen je prema istom spisku nacrta i ima iste komponente. U objektno-orijentisanoj terminologiji kažemo da je automobil instanca klase objekata automobila. Tačnije klasa je šematski plan za svaki objekat koji se kreira. To ćemo najbolje ilustrovati na primeru 1.2 u kome su kreirane dve klase Automobil i TestKlasa koja u sebi sadrži metodu main():

Primer 1.2

```
class Automobil
{

	int cadence = 0;
	int speed = 0;
	int gear = 1;

	void changeCadence(int newValue)
	{
		cadence = newValue;
	}

	void changeGear(int newValue)
	{
		gear = newValue;
```

```
        }

        void speedUp(int increment)
        {
                speed = speed + increment;
        }

        void applyBrakes(int decrement)
        {
                speed = speed - decrement;
        }

        void printStates()
        {
                System.out.println("cadence:" +
                        cadence + " speed:" +
                        speed + " gear:" + gear);
        }
}

class TestKlasa
{
        public static void main(String[] args)
        {

                // Kreiraj dva različita automobila
                Automobil automobil1 = new Automobil();
                Automobil automobil2 = new Automobil();

                // Pozovi metode kreiranih objekata
                automobil1.changeCadence(50);
                automobil1.speedUp(10);
                automobil1.changeGear(2);
                automobil1.printStates();

                automobil2.changeCadence(50);
                automobil2.speedUp(10);
                automobil2.changeGear(2);
                automobil2.changeCadence(40);
                automobil2.speedUp(10);
                automobil2.changeGear(3);
                automobil2.printStates();
        }
}
```

OVERLOADING

A software engineering process whereby multiple functions of different types are defined with the same name.

Primer 1.3

```
// Demonstrate method overloading.
class OverloadDemo
{
	void test()
	{
		System.out.println("No parameters");
	}

	// Overload test for one integer parameter.
	void test(int a)
	{
		System.out.println("a: " + a);
	}

	// Overload test for two integer parameters.
	void test(int a, int b)
	{
		System.out.println("a and b: " + a + " " + b);
	}

	// overload test for a double parameter
	double test(double a)
	{
		System.out.println("double a: " + a);
		return a*a;
	}
}

class Overload
{
	public static void main(String args[])
	{
		OverloadDemo ob = new OverloadDemo();
		double result;
		// call all versions of test()
		ob.test();
		ob.test(10);
		ob.test(10, 20);
		result = ob.test(123.2);
		System.out.println("Result of ob.test(123.2): " + result);
	}
}
```

Konstruktor

- Konstruktor je metod ima isto ime kao i klasa, poziva se isključivo pri instanciranju objekata (dakle, operatorom **new**),
- Nema povratne vrednosti,
- Klasa može imati više konstruktora (overloading na delu),
- Konstruktor bez parametara je default konstruktor i on postoji kada klasa nema posebno implementiran (naveden) ni jedan konstuktor.

Primer 1.4

```
public class Cube
{

	int length;
	int breadth;
	int height;

	public int getVolume()
	{
		return (length * breadth * height);
	}

	Cube()
	{
		length = 10;
		breadth = 15;
		height = 5;
		System.out.println("Constructor with no parameters");
	}

	Cube(int l, int b)
	{
		length = l;
		breadth = b;
		height = 10;
		System.out.println("Constructor with two parameters");
	}

	Cube(int l, int b, int h)
	{
		length = l;
		breadth = b;
		height = h;
		System.out.println("Constructor with three parameters");
	}

	public static void main(String[] args)
	{
		Cube cubeObj1, cubeObj2;
		cubeObj1 = new Cube();
		cubeObj2 = new Cube(10, 20, 30);
		System.out.println("Volume of Cube1 is : " + cubeObj1.getVolume());
```

```
            System.out.println("Volume of Cube2 is : " + cubeObj2.getVolume());
        }
}
```

Zadatak 1

Napisati program koji ima sledeće klase (sve u istom fajlu):

- Klasu **Prijava** koja sadrži
  - Atributi: naziv predmeta, broj indeksa studenta, ispitni rok
  - Konstruktor koji prihvata i setuje naziv predmeta, broj indeksa studenta i ispitni rok
- Klasu **StudentskaSluzba** koja ima
  - Metodu **PrihvatiPrijavu** koja kao argument prihvata prijavu ispita i ispisuje podatke o njoj
- Klasu **Student** koja sadrži
  - Atribute: broj indeksa, ime i prezime
  - Konstruktor koji prihvata i setuje broj indeksa, ime i prezime
  - Medodu **PrijaviIspit** koja kao argumente prihvata studentsku službu u kojoj treba prijaviti ispit, rok i naziv ispita. Zatim kreira prijavu sa odgovarajućim podacima i prijavljuje ispit studentskoj službi.
- *Public* klasu **Program** koja u **main** metodi treba u navedenom redosledu:
  - Kreira objekat tipa studentska služba
  - Kreira studenta Pera Perić sa indeksom 77/07
  - Kreira studenta Jova Jović sa indeksom 55/09
  - Jova prijavljuje ispit “OOP” u junskom roku
  - Pera prijavljuje ispit “SPA2” u junskom roku

```
class Prijava {
	String brInd;
	String rok;
	String nazivIspita;

	Prijava(String brInd1, String rok1, String nazivIspita1) {
		brInd = brInd1;
		rok = rok1;
		nazivIspita = nazivIspita1;
	}
```

```
}



class StudentskaSluzba {
	void PrihvatiPrijavu(Prijava p) {
		System.out.println(p.brInd + " " + "je prijavio " + p.nazivIspita + " u
roku " + p.rok);
	}
}

class Student {
	String ime;
	String prezime;
	String brInd;

	Student(String ime1, String prezime1, String brInd1) {
		this.ime = ime1;
		this.prezime = prezime1;
		this.brInd = brInd1;
	}

	void PrijaviIspit(StudentskaSluzba ss, String rok, String ispit) {
		Prijava p = new Prijava(brInd, rok, ispit);
		ss.PrihvatiPrijavu(p);
	}
}

public class Program {
	public static void main(String[] args) {

		StudentskaSluzba ss = new StudentskaSluzba();

		Student pera = new Student("Pera", "Peric", "77/07");
		Student jova = new Student("Jova", "Jovic", "55/09");

		jova.PrijaviIspit(ss, "jun", "OOP");
		pera.PrijaviIspit(ss, "jun", "SPA2");
	}
}
```