

# Izuzeci

Već je navedeno da se Java program ne može kompajlirati sve dok u njemu postoje sintaksne greške. U tom slučaju, program se ne može ni pokrenuti. Međutim, šta se dešava kada se desi greška u toku izvršavanja programa? Mehanizam “izuzetaka” (“exception”) u Javi postoji upravo da bi se ovakve situacije obrađivale i rešavale na uniforman način.

**Izuzeci** (u Javi) su sve one situacije koje nastupe u toku izvršenja programa a, pri tome, mogu da poremete normalan rad programa. Greške koje izazivaju izuzetke su uglavnom u vezi sa: unosom podataka (“User input errors”), radom perifernih uređaja računara (“Device errors”), radom memorijskih uređaja računara (“Physical limitations”) ili samim izvršavanjem metoda (“Code errors”). **Izuzeci nisu sintaksne greške**, pa program koji izaziva ovakve situacije može normalno da se kompajlira i pokrene.

Mehanizam za obrađivanje izuzetaka u Javi funkcioniše na sledeći način. Metoda u kojoj se desila greška **“baca” (“throws”) odgovarajući izuzetak**. Pored podataka o **vrsti greške**, izuzetak sadrži i podatke o tome **u kojoj metodi i na kojoj liniji koda je nastala greška (tzv. “stack-trace”)**. Kada neka metoda baci izuzetak, **izvršavanje te metode ali i celog programa se prekida**. To se može videti na jednom jednostavnom primeru.

## Primer 1

Napraviti Javnu klasu ***IzuzeciDemo*** koja ima:

- Atribut niz koji predstavlja niz celih brojeva.
- Metodu koja inicijalizuje niz na 10 elemenata i na ekranu ispisuje 11. element.

Napisati klasu ***TestIzuzeciDemo*** koja kreira jedan objekat klase *IzuzeciDemo* i poziva njegovu metodu za ispisivanje. Posle toga, potrebno je ispisati poruku da je 11. element niza ispisan. Pokrenuti program.

```
public class IzuzeciDemo {  
  
    int[] niz;  
  
    void ispisiJedanaestog() {  
        niz = new int[10];  
        System.out.println(niz[10]);  
    }  
  
}  
  
public class TestIzuzeciDemo {  
  
    public static void main(String[] args) {  
  
        IzuzeciDemo id = new IzuzeciDemo();  
  
        id.ispisiJedanaestog();  
  
        System.out.println("Jedanaesti element je ispisan.");  
    }  
}
```

```
}  
  
}
```

Kada se navedeni kod pokrene, izazvaće pojavljivanje izuzetka. Konkretno, metoda “ispisiJedanaestog” će da inicijalizuje niz na 10 elemenata i da pokuša da ispiše jedanaesti (što nije moguće). Program će da se prekine, a na ekranu će da se pojavi sledeća poruka:

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 10  
    at IzuzeciDemo.ispisiJedanaestog(IzuzeciDemo.java:7)  
    at TestIzuzeciDemo.main(TestIzuzeciDemo.java:7)
```

Iz same poruke se može zaključiti nekoliko stvari. Prva linija poruke sadrži podatke o tome koja je vrsta greške u pitanju - vrednost indeksa kojim se pristupa elementima niza je van granica (“java.lang.ArrayIndexOutOfBoundsException”). U produžetku iste linije je navedena i poruka koja dodatno objašnjava uzrok nastajanja greške - “10”. To je upravo ona nedozvoljena vrednost indeksa koja je izazvala pojavljivanje greške (niz je inicijalizovan na 10 elemenata, pa je indeks poslednjeg elementa 9). Ostale linije poruke sadrže informaciju o tome na kojoj liniji koda je nastala greška. Može se videti da je greška nastala na sedmoj liniji koda klase IzuzeciDemo (fajl je “IzuzeciDemo.java”) i to u okviru metode “ispisiJedanaestog” ove klase. Takođe, može se videti da se u okviru “main” metode klase TestIzuzeciDemo, na sedmoj liniji koda, poziva metoda koja je napravila grešku.

Na kraju, potrebno je primetiti da se program prekida čim se pojavi greška tj. baci izuzetak. Primer za to je komanda za ispisivanje poruke koja se nalazi u “main” metodi klase TestIzuzeciDemo. Ova komanda nije uopšte izvršena (poruka nije ispisana na ekranu) jer je bačen izuzetak pre nego što je mogla da se izvrši.

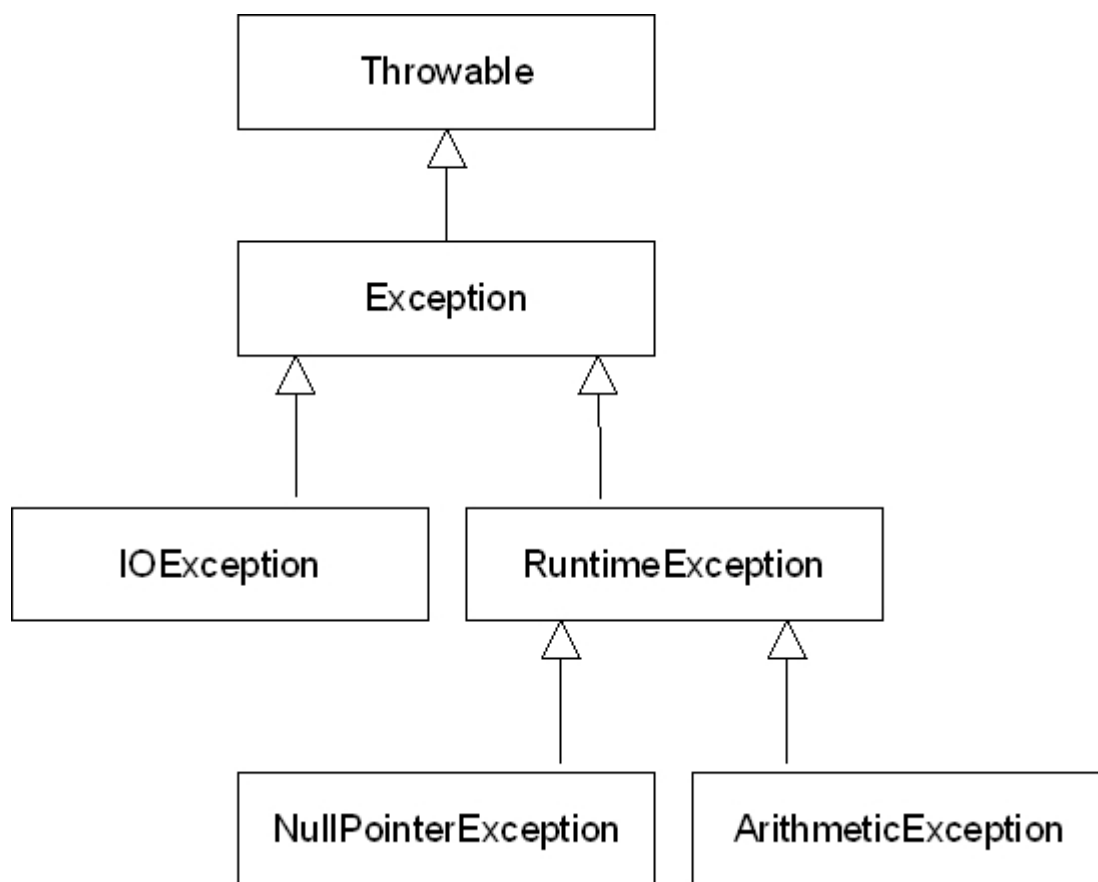
Izuzeci u Javi su **implementirani u vidu klasa**. Praktično, to znači da se, svaki put kada se desi greška, napravi po jedan objekat klase koja predstavlja izuzetak, napuni odgovarajućim podacima i prosledi Javinom mehanizmu obrađivanje izuzetaka. U zavisnosti od toga koji je tip greške u pitanju, baca se objekat odgovarajuće klase izuzetka.

Java sadrži neke **predefinisane klase izuzetaka** koje baca automatski. Sve ove klase pripadaju paketu “java.lang”. Kratka lista ovih klasa sa odgovarajućim opisima je data u sledećoj tabeli.

Tip izuzetka (klasa)	Situacija u kojoj se pojavljuje
ArithmeticException	Bilo kakva greška pri aritmetičkim operacijama (npr. deljenje sa nulom).
ArrayIndexOutOfBoundsException	Vrednost indeksa elementa niza je manja od nule ili veća od maksimalne dozvoljene vrednosti (npr. pristupa se 11. elementu niza čiji je kapacitet 10 elemenata).
StringIndexOutOfBoundsException	Vrednost indeksa kojim se pristupa znaku iz String-a je manja od nule ili veća od maksimalne dozvoljene vrednosti (npr. pristupa se petom znaku String-a koji ima dva slova).
NegativeArraySizeException	Inicijalizacija niza sa negativnim kapacitetom.
ClassCastException	Neispravna konverzija objekta u objekat druge klase.
NullPointerException	Pristupanje nizu ili objektu neke klase pre nego što je izvršena inicijalizacija (niz tj. objekat ima vrednost “null”).

NumberFormatException	Neispravna konverzija String vrednosti u neki realan ili ceo broj.
-----------------------	--

Klase izuzetaka formiraju određenu **hijerarhiju nasleđivanja** (Slika 1). Osnovna klasa iz koje su izvedeni svi izuzeci je klasa Throwable. Ova klasa sadrži samo neke osnovne funkcionalnosti i omogućava povezivanje sa mehanizmom za obrađivanje izuzetaka. Njene direktna podklasa je klasa Exception. Klasa Exception sadrži sve potrebne funkcionalnosti koje bi jedan izuzetak trebalo da ima, pa se može uslovno reći da je zapravo ova klasa nadklasa svih izuzetaka. Njene direktne podklase su, na primer, IOException (koja predstavlja greške u radu sa uređajima) i RuntimeException (koja predstavlja greške u toku izvršavanja koda koje nisu u vezi sa uređajima). Iz klase RuntimeException su izvedene predefinisane klase izuzetaka NullPointerException i ArithmeticException, ali i mnoge druge.



Slika 1: Hijerarhija klasa izuzetaka

Javini predefinisani izuzeci se bacaju automatski svaki put kada se pojavi odgovarajući tip greške. Međutim, nekada je potrebno baciti izuzetak u onim situacijama koje se smatraju greškama (za određeni program). Na primer, ako se kao vrednost ulaznog parametra neke metode unese neka nedozvoljena vrednost, metoda bi trebalo da baci izuzetak. Izuzetak se može **baciti “po potrebi” uz pomoć naredbe “throws”**. Ova naredba se koristi tako što se iza ključne reči “throws” navede (ili kreira) objekt klase izuzetka koji se želi baciti. Konstruktoru klase izuzetka se može (ali ne mora) proslediti poruka greške u vidu String vrednosti.

```
throw new NazivKlaseIzuzetka (poruka_greške);
```

## Primer 2

Napraviti Javnu klasu **Osoba** koja ima:

- Privatni atribut ime.
- Privatni atribut prezime.
- Odgovarajuće javne get i set metode za ova dva atributa. Nedoovoljene vrednosti za oba atributa su null String-ovi. U slučaju unosa nedovoljenih vrednosti, potrebno je baciti izuzetak sa odgovarajućom porukom.

```
public class Osoba {

    private String ime;
    private String prezime;

    public String getIme() {
        return ime;
    }
    public void setIme(String ime) {
        if (ime == null)
            throw new RuntimeException("Ime ne moze biti null");

        this.ime = ime;
    }
    public String getPrezime() {
        return prezime;
    }
    public void setPrezime(String prezime) {
        if (prezime == null)
            throw new RuntimeException("Prezime ne moze biti null");

        this.prezime = prezime;
    }
}
```

*U primeru se može videti da se u slučaju unosa nedovoljenih vrednosti baca izuzetak klase RuntimeException. Konstruktoru izuzetka je, u oba slučaja, prosleđena poruka koju je potrebno proslediti u slučaju greške.*

*Potrebno je primetiti da u obe set metode IF naredba nema ELSE deo u kojem se nalazi naredba za dodeljivanje vrednosti atributu. Razlog je taj što će se, ako se desi izuzetak, izvršavanje metode ali i celog programa prekinuti pa se naredba za dodelu svejedno neće izvršiti.*

Ako u toku izvršavanja programa bude bačen izuzetak, izvesno je da će se program istog trenutka prekinuti. Ali to ne mora da bude tako. Program će se **prekinuti samo ako bačeni izuzetak ne bude uhvaćen (“catch”)**. Komanda kojom se mogu hvatati i kontrolisati bačeni izuzeci je tzv. **“try-catch” blok**. Ovaj blok se sastoji od ključnih reči “try” i “catch” između kojih je potrebno postaviti sve one komande i pozive metodama za koje se sumnja da mogu baciti izuzetak. Funkcija “try-catch” bloka je da uhvati izuzetak, omogući njegovu obradu i spreči prekidanje programa. Takođe, ovaj blok se aktivira samo ako izuzetak bude bačen, a u suprotnom ne utiče na tok izvršavanja programa. Deklaracija ovog bloka se vrši na sledeći način.

```
try{

    ...komande koje mogu baciti izuzetke...
```

```

} catch (NazivKlaseIzuzetka izuzetak) {

    ...komande koje se izvršavaju samo ako je izuzetak bačen...

}

```

Blok počinje ključnom reči “try” iza koje slede vitičaste zagrade. U okviru ovih zagrada je potrebno navesti komande za koje se zna da mogu baciti izuzetke. Sledeća ključna reč je “catch” iza koje, u običnim zgradama stoji naziv klase izuzetka koja se hvata (npr. IOException ili ArithmeticException) i naziv promenljive koja će da primi referencu na bačeni izuzetak. Konačno, na kraju je blok naredbi koje se izvršavaju samo ako izuzetak bude bačen, a u suprotnom ne. U okviru ovog, poslednjeg bloka naredbi se može pozivati objekat koji predstavlja bačeni izuzetak.

Potrebno je napomenuti da će **“catch” naredba da uhvati samo izuzetke one klase (i bilo koje od njenih podklasa) čiji naziv je naveden u zagradi**. Znači, ako se hvata izuzetak klase IOException, a bude bačen izuzetak klase ArithmeticException, “try-catch” blok neće da ga uhvati i program će se prekinuti. Sa druge strane, ako se kao očekivani tip izuzetka postavi klasa Exception, uhvatiće se svi izuzeci jer je klasa Exception nadklasa svih izuzetaka.

### Primer 3

*Sledi primer upotrebe “try-catch” bloka. U ovom slučaju, baca se izuzetak klase RuntimeException, a “catch” naredba hvata upravo izuzetak ovog tipa. Kada se kod izvrši, program neće da se prekine, a na ekranu će se samo ispisati izuzetak - njegov tip i poruka.*

```

try{
    int a = 1;
    if (a==1)
        throw new RuntimeException("A je jedan");
} catch (RuntimeException e) {
    System.out.println(e);
}

```

*Prethodni primer može da bude napisan i tako da “catch” naredba hvata izuzetak klase Exception. U tom slučaju, efekat izvršavanja je isti jer je Exception nadklasa klase RuntimeException pa će svaki izuzetak klase Exception ili bilo koje njene podklase biti uhvaćen.*

```

try{
    int a = 1;
    if (a==1)
        throw new RuntimeException("A je jedan");
} catch (Exception e) {
    System.out.println(e);
}

```

*Sledeći “try-catch” blok, međutim, neće moći da uhvati izuzetak klase ArithmeticException ako bude bačen. Razlog je taj što se očekuje izuzetak klase NullPointerException.*

```

try{
    int a = 1;
    if (a==1)
        throw new ArithmeticException("A je jedan");
} catch (NullPointerException e) {

```

```
        System.out.println(e);
    }
```

Ako neki skup komandi može da izazove bacanje više različitih tipova izuzetaka, moguće je napraviti **“try-catch” blok sa nekoliko “catch” naredbi** i to tako da svaka od naredbi hvata po jedan tip izuzetka. Ako izuzetak bude bačen, proveravaće se očekivani tipovi izuzetaka svake “catch” naredbe i to redom u kojem su napisane sve dok se ne nađe tip koji odgovara bačenom izuzetku. Tek tada će se aktivirati odgovarajuća “catch” naredba (i to samo jedna). Ukoliko je potrebno da se obavezno izvrše neke naredbe nakon što se desio izuzetak, može se upotrebiti i **“finally” blok**. Komande iz ovog bloka će se izvršiti posle komandi odgovarajuće “catch” naredbe.

```
try{

    ...komande koje mogu baciti izuzetke...

}catch(NazivKlaseIzuzetka1 izuzetak){

    ...komande koje se izvršavaju samo ako je izuzetak 1 bačen...

}catch(NazivKlaseIzuzetka2 izuzetak){

    ...komande koje se izvršavaju samo ako je izuzetak 2 bačen...

}finally{

    ...komande koje se izvršavaju ako je bilo koji izuzetak bačen...

}
```

Potrebno je **obratiti pažnju i na redosled pisanja “catch” naredbi** u okviru “try-catch” bloka koji ima nekoliko “catch” naredbi. Podklase izuzetaka moraju biti navedene kao očekivani tipovi izuzetaka pre svojih nadklasa, inače će se aktivirati “catch” naredba nadklase izuzetka umesto “catch” naredbe podklase izuzetka. U Javi se ova situacija tretira kao sintaksna greška, pa program neće moći ni da se kompajlira.

#### **Primer 4**

*Sledi primer upotrebe “try-catch” bloka sa nekoliko “catch” naredbi. U ovom slučaju, baca se izuzetak klase RuntimeException ili ArithmeticException, a “catch” naredbe hvataju upravo izuzetke ovog tipa.*

```
try{
    int a = 1;
    if (a==1)
        throw new ArithmeticException("A je jedan");
    else
        throw new RuntimeException ("A nije jedan");
}catch(ArithmeticException e){
    System.out.println("A je jedan "+e);
}catch(RuntimeException e){
    System.out.println("A nije jedan "+e);
}
```

Ako je potrebno izvršiti iste naredbe u slučaju da se baci bilo koji od izuzetaka, prethodni primer je mogao da se napiše i na sledeći način. U ovom slučaju “catch” naredba može da uhvati bilo koji tip izuzetka.

```
try{
    int a = 1;
    if (a==1)
        throw new ArithmeticException("A je jedan");
    else
        throw new RuntimeException ("A nije jedan");
} catch (Exception e){
    System.out.println(e);
}
```

Konačno, sledi primer “try-catch” bloka sa “finally” blokom. “Finally” blok će dase aktivira svaki put kada neki izuzetak bude bačen, bez obzira na to koji je tip izuzetka u pitanju.

```
try{
    int a = 1;
    if (a==1)
        throw new ArithmeticException("A je jedan");
    else
        throw new RuntimeException ("A nije jedan");
} catch (ArithmeticException e){
    System.out.println("A je jedan "+e);
} catch (RuntimeException e){
    System.out.println("A nije jedan "+e);
} finally{
    System.out.println("Greska postoji!");
}
```

Potrebno je razmotriti i šta se tačno može uraditi u slučaju da izuzetak bude uhvaćen. U nekim situacijama, dovoljno je da se spreči prekidanje programa, pa blok naredbi posle ključne reči “catch” može da ostane prazan. Ali, nekada je potrebno ispisati podatke koje sadrži izuzetak, pa se moraju pozivati odgovarajuće metode klase izuzetka.

Kada su klase izuzetaka u pitanju, metoda “toString” vraća String sa nazivom klase izuzetka i porukom izuzetka. Metoda “getMessage” vraća String koji samo sadrži poruku izuzetka, dok metoda “printStackTrace” ispisuje “stack-trace” na ekranu.

### Primer 5

Sledi primer poziva metoda “toString”, “getMessage” i “printStackTrace” bačenog izuzetka.

```
public class PrimerPozivaMetodaIzuzetka {

    public static void main(String[] args) {

        try{
            throw new RuntimeException("Greska");
        } catch (RuntimeException e){
            //Ispisace se naziv klase izuzetka i poruka
            //java.lang.RuntimeException: Greska
        }
    }
}
```

```

        System.out.println(e);

        //Ispisace se samo poruka izuzetka
        //Greska
        System.out.println(e.getMessage());

        //Ispisace se ceo "stack trace"
        //java.lang.RuntimeException: Greska
        //at PrimerPozivaMetodaIzuzetka.main
        //(PrimerPozivaMetodaIzuzetka.java:6)
        e.printStackTrace();
    }
}

```

Izuzeci u Javi se dele u dve grupe: **proveravani izuzeci ("checked")** i **neproveravani izuzeci ("unchecked")**. Klasa RuntimeException i sve njene podklase spadaju u grupu neproveravanih izuzetaka, dok su sve ostale klase iz grupe proveravanih izuzetaka (npr. Exception, IOException...). Klase koje su navedene u tabeli Javinih predefinisanih izuzetaka nasleđuju klasu RuntimeException pa pripadaju grupi neproveravanih izuzetaka. Ali, u čemu je razlika?

Ako metoda baca neki neproveravani izuzetak, poziv te metode **može, ali ne mora biti uokviren "try-catch" blokom koji hvata taj izuzetak**. Ako metoda baca neki proveravani izuzetak, poziv te metode **mora da bude uokviren "try-catch" blokom koji hvata taj izuzetak, a metoda mora da bude označena ključnom reči "throws" i nazivom klase izuzetka koji baca**.

```

tip_vr nazivMetode(...ul. param. ...) throws NazivKlaseIzuzetka{

    //Telo metode

}

```

## Primer 6

Napisati klasu ProveraParnosti koja ima:

- Javnu statičku metodu proveraParnosti koja vraća TRUE ako je broj koji je unet kao ulazni parametar paran, a FALSE ako nije. Metoda baca PROVERAVANI izuzetak ako je uneti broj jednak 0.

Napisati klasu TestProveraParnosti koja poziva metodu klase ProveraParnosti.

```

public class ProveraParnosti {

    public static boolean proveraParnosti(int a) throws Exception{
        if (a==0)
            throw new Exception("Uneli ste nulu");

        if (a%2==0) return true;
        else return false;
    }
}

```



```

public class TestProveraParnosti {

    public static void main(String[] args) {
        //Svaki poziv ove metode MORA da bude uokviren
        //u "try-catch" blok jer metoda baca proveravani
        //izuzetak.
        try {
            ProveraParnosti.proveraParnosti(10);
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }
}

```

Na kraju, potrebno je dodati i to da se **u Javi mogu praviti i nove klase izuzetaka** i to nasleđivanjem odgovarajućih klasa (na primer Exception za proveravane i RuntimeException za neproveravane izuzetke). Na ovaj način se mogu pružati još detaljnije poruke o tome gde je i zašto nastala neka greška.

## Zadaci

### Zadatak 1

Napraviti klasu **Osoba** koja ima:

- Privatni atribut ime
- Privatni atribut prezime
- Privatni atribut visina (ceo broj) koji predstavlja visinu osobe u cm.
- Privatni atribut tezina (realan broj) koji predstavlja težinu osobe u kg.
- Privatni atribut godine (ceo broj) koji predstavlja starost osobe.
- Odgovarajuće javne get i set metode za ove attribute. Nedozevljene vrednosti za attribute ime i prezime su null String-ovi. Visina mora biti u granicama 50-250cm, težina od 2-250kg a godine 0-125. U slučaju unosa nedozvoljenih vrednosti za bilo koji atribut, potrebno je baciti izuzetak sa odgovarajućom porukom.
- Redefinisanu metodu toString klase Object. Ova metoda bi trebalo da vraća String koji sadrži vrednosti imena i prezimena osobe, njenih godina, visine i težine kao i odgovarajući tekst uz to.
- Redefinisanu metodu equals klase Object koja kao ulazni argument prima objekat klase Object. Prvo, potrebno je proveriti da li je unet objekat klase Osoba, pa ako nije baciti izuzetak sa odgovarajućom porukom. Ako jeste unet objekat klase Osoba, ova metoda vraća true ako su vrednosti atributa ime i prezime jednaka imenu i prezimenu unete osobe, a u suprotnom false.

Napraviti klasu **TestOsoba** koja kreira jedan objekat klase Osoba, unosi u njega odgovarajuće vrednosti za ime, prezime, visinu, težinu i godine i ispisuje na ekranu sve podatke o osobi. Startovati program. Probati onda sa unosom nedozvoljenih vrednosti za visinu i ponovo startovati program. Probati sa unosom nedozvoljenih vrednosti za ostale attribute i videti šta se dešava kada se startuje program.

### Rešenje:

```

public class Osoba {

    private String ime;
    private String prezime;
    private int visina;
    private double tezina;
    private int godine;

    public int getGodine() {
        return godine;
    }

    public void setGodine(int godine) {
        if (godine<0 || godine>125)

```

```

        throw new RuntimeException("Unete godine su van granica");
    else
        this.godine = godine;
}
public String getIme() {
    return ime;
}
public void setIme(String ime) {
    if (ime == null)
        throw new RuntimeException("Uneto ime ne sme biti null");
    else
        this.ime = ime;
}
public String getPrezime() {
    return prezime;
}
public void setPrezime(String prezime) {
    if (prezime == null)
        throw new RuntimeException("Uneto prezime ne sme biti null");
    else
        this.prezime = prezime;
}
public double getTezina() {
    return tezina;
}
public void setTezina(double tezina) {
    if (tezina<2 || tezina>250)
        throw new RuntimeException("Uneta tezina je van granica");
    else
        this.tezina = tezina;
}
public int getVisina() {
    return visina;
}
public void setVisina(int visina) {
    if (visina<50 || visina>250)
        throw new RuntimeException("Uneta visina je van granica");
    else
        this.visina = visina;
}

public String toString(){
    return "Ime: "+ime+" Prezime: "+prezime+" Visina: "+visina+
        " Tezina: "+tezina+" Godine: "+godine;
}

public boolean equals(Object o){
    if (! (o instanceof Osoba))
        throw new RuntimeException("Morate uneti objekat klase Osoba");
    else{
        Osoba os = (Osoba) (o);
        if (ime.equals(os.getIme()) &&
            prezime.equals(os.getPrezime()))
            return true;
        else
            return false;
    }
}

}

public class TestOsoba {

    public static void main(String[] args) {

```

```

        Osoba o = new Osoba();

        o.setIme("Pera");
        o.setPrezime("Peric");

        //Ako se umesto visine 175 unese visina koja nije dozvoljena
        //bice bacen izuzetak i program ce da se prekine uz prikazivanje
        //greske crvenim slovima.
        o.setVisina(175);

        //Ako se unese naredna linija koda umesto prethodne program se prekida.
        //Isto vazi i za unos drugih nedozvoljenih vrednosti.
        //o.setVisina(-5);

        //Ako je uneta pogresna visina, program ce da se prekine na tom mestu,
        //pa se naredne tri naredbe nece ni izvršiti.
        o.setTezina(75.3);
        o.setGodine(55);
        System.out.println(o);
    }
}

```

## Zadatak 2

Napraviti klasu **Automobil** koja ima:

- Privatni atribut marka.
- Privatni atribut model.
- Privatni atribut kubikaza (ceo broj).
- Odgovarajuće javne get i set metode za ove attribute. Nedozvoljene vrednosti za attribute marka i model su null String-ovi, a kubikaža mora biti veća od 500 i manja od 8000 kubika. U slučaju unosa ovih vrednosti, porebno je baciti izuzetak sa odgovarajućom porukom.
- Redefinisani metodu toString klase Object. Ova metoda bi trebalo da vraća String koji sadrži vrednosti marke, modela i kubikaže automobila kao i odgovarajući tekst uz to.

Napraviti klasu **TestAutomobil** koja kreira jedan objekat klase Automobil, u njega unosi odgovarajuće vrednosti i na ekranu ispisuje sve podatke o automobilu. Pozivati metode za unos (set metode) tako da se, u slučaju unosa nedozvoljenih vrednosti, program ne prekida zbog izuzetka koje bacaju ove metode već se samo ispisuje poruka o grešci koju sadrži bačeni izuzetak. Probati sa unosom nedozvoljenih vrednosti i videti šta se dešava kada se startuje program.

**Rešenje:**

```

public class Automobil {

    private String marka;
    private String model;
    private int kubikaza;

    public int getKubikaza() {
        return kubikaza;
    }

    public void setKubikaza(int kubikaza) {
        //Metoda moze da se napise i ovako - bez else dela.
        //Kada se throw naredba izvrši, izvršavanje metode se prekida
        //pa se nece izvršiti komanda "this.kubikaza = kubikaza;"
        if (kubikaza < 500 || kubikaza > 8000)
            throw new RuntimeException("Uneta kubikaza je van granica");

        this.kubikaza = kubikaza;
    }
}

```

```

    public String getMarka() {
        return marka;
    }
    public void setMarka(String marka) {
        //Metoda moze da se napise i ovako - bez else dela.
        //Kada se throw naredba izvrši, izvršavanje metode se prekida
        //pa se nece izvršiti komanda "this.marka = marka;"
        if (marka == null)
            throw new RuntimeException("Uneta marka ne sme da bude null");

        this.marka = marka;
    }
    public String getModel() {
        return model;
    }
    public void setModel(String model) {
        //Metoda moze da se napise i ovako - bez else dela.
        //Kada se throw naredba izvrši, izvršavanje metode se prekida
        //pa se nece izvršiti komanda "this.model = model;"
        if (model == null)
            throw new RuntimeException("Uneti model ne sme da bude null");

        this.model = model;
    }

    public String toString(){
        return "Marka: "+marka+" Model: "+model+" Kubikaza: "+kubikaza;
    }
}

public class TestAutomobil {

    public static void main(String[] args) {

        Automobil a = new Automobil();

        //Metoda setMarka baca izuzetak ako se unese null String.
        //Jedini nacin da se obezbedi da program nastavi sa
        //izvršavanjem cak i kada se unese null String je da se
        //metoda uokviri u try-catch blok. To vazi za sve metode
        //koje bacaju neki izuzetak.
        try {
            a.setMarka("Dodge");
            //Ako se umesto prethodne naredbe stavi
            //a.setMarka(null);
            //program nece da prekine sa radom vec ce samo da se
            //ispise poruka o gresci.
        } catch (Exception e) {
            System.out.println("Greska: "+e.getMessage());
        }

        try {
            a.setModel("Viper");
            //Ako se umesto prethodne naredbe stavi
            //a.setModel(null);
            //program nece da prekine sa radom vec ce samo da se
            //ispise poruka o gresci.
        } catch (Exception e) {
            System.out.println("Greska: "+e.getMessage());
        }

        try {
            a.setKubikaza(7900);

```

```

        //Ako se umesto prethodne naredbe stavi
        //a.setKubikaza(-2000);
        //program nece da prekine sa radom vec ce samo da se
        //ispise poruka o gresci.
    } catch (Exception e) {
        System.out.println("Greska: "+e.getMessage());
    }

    System.out.println(a);

}

}

```

### Zadatak 3

Napisati klasu **IspisivacKategorije** koja ima:

- Javnu statičku metodu koja kao ulazni argument dobija broj koji predstavlja visinu čoveka u cm. Metoda prvo proverava da li je visina u granicama (120 cm – 250 cm). Ako je visina van granica, baca se **PROVERAVANI** izuzetak sa odgovarajućom porukom. U suprotnom, metoda ispisuje na ekranu kojoj kategoriji čovek pripada (<160 – niski ljudi, 160<=185 srednje visine, 185<= visoki).
- Javnu statičku metodu koja kao ulazni argument dobija broj koji predstavlja godine čoveka. Metoda prvo proverava da li su godine u granicama (0 – 125). Ako je uneti broj van granica, baca se **PROVERAVANI** izuzetak sa odgovarajućom porukom. U suprotnom, metoda ispisuje na ekranu kojem dobu pripada čovek (0-29 mladost, 30 – 55 zrelost, 56 – starost).

Napisati klasu **TestIspisivacKategorije** koja poziva metode klase IspisivacKategorije.

**Rešenje:**

```

public class IspisivacKategorije {

    public static void ispisiKategorijuVisine(int visina) throws Exception{

        //Ako je u pitanju PROVERAVANI izuzetak, baca se izuzetak klase
        //Exception a ne RuntimeException. RuntimeException pripada NEPROVERAVANIM
        //izuzecima.
        if (visina<120 || visina>250)
            throw new Exception ("Visina je van granica");

        if (visina < 160)
            System.out.println("Covek pripada kategoriji niskih ljudi");
        if (visina >= 160 && visina < 185)
            System.out.println("Covek pripada kategoriji srednje visokih ljudi");
        if (visina >= 185)
            System.out.println("Covek pripada kategoriji visokih ljudi");

    }

    public static void ispisiKategorijuGodina(int godine) throws Exception{

        if (godine<0 || godine>125)
            throw new Exception ("Godine su van granica");

        if (godine <= 29)
            System.out.println("Covek pripada kategoriji mladih ljudi");
        if (godine > 29 && godine <= 55)
            System.out.println("Covek pripada kategoriji zrelih ljudi");
        if (godine > 55)
            System.out.println("Covek pripada kategoriji starih ljudi");

    }

}

```

```

}

public class TestIspisivacKategorije {

    public static void main(String[] args) {

        //Ako metoda baca proveravani izuzetak, onda ona
        //MORA da se uokviri try-catch blokom. Ako metoda baca
        //neproveravani izuzetak (RuntimeException) onda ona
        //moze ali ne mora da se uokviri try-catch blokom.
        try {
            IspisivacKategorije.ispisiKategorijuVisine(-205);
        } catch (Exception e) {
            System.out.println("Greska: "+e.getMessage());
        }

        try {
            IspisivacKategorije.ispisiKategorijuGodina(27);
        } catch (Exception e) {
            System.out.println("Greska: "+e.getMessage());
        }

    }

}

```

#### Zadatak 4

Napraviti klasu **RacunException** koja predstavlja neproveravani izuzetak (nasleduje klasu RuntimeException) i ima:

- Javni konstruktor koji kao ulazni argument prima poruku greške i poziva odgovarajući konstruktor nadklase prosleđujući mu ulazni argument.

Napraviti klasu **Racun** koja ima:

- Privatni atribut broj koji predstavlja broj računa (ceo broj).
- Privatni atribut proizvod koji predstavlja naziv proizvoda koji je prodat.
- Privatni atribut iznos koji predstavlja iznos računa (npr. 234.55 din).
- Javne get i set metode za ova tri atributa. Broj računa mora biti veći od nule, iznos takođe i ne smeju se unosti null String-ovi umesto proizvoda. U slučaju unosa bilo kojih od ovih nedozvoljenih vrednosti potrebno je baciti izuzetak klase RacunException.
- Redefinisatu metodu toString klase Object. Ova metoda bi trebalo da vraća String koji sadrži vrednosti svih atributa računa kao i odgovarajući tekst uz to.

Napraviti klasu **TestRacun** koja kreira jedan objekat klase Racun, u njega unosi odgovarajuće vrednosti i ispisuje sve podatke o računu na ekranu. Pozvati metodu za unos broja tako da se, u slučaju unosa nedozvoljenih vrednosti, program ne prekida zbog izuzetka koje baca već se samo ispisuje poruka o grešci koju sadrži bačeni izuzetak. Startovati program. Pokušati unos nekih nedozvoljenih vrednosti za broj i za ostale attribute, startovati program i videti šta se dešava.

#### Rešenje:

```

//Izuzeci u Javi su zapravo klase, tako da je moguće praviti
//i svoje izuzetke. Jedino što je potrebno je da se nasledi
//klasa RuntimeException (ako je potrebno da novi izuzetak
//bude neproveravani) ili Exception (ako je potrebno da novi
//izuzetak bude proveravani) i da se napise konstruktor.
public class RacunException extends RuntimeException {

    public RacunException(String poruka) {
        super(poruka);
    }

}

```

```

public class Racun {

    private int broj;
    private String proizvod;
    private double iznos;

    public int getBroj() {
        return broj;
    }
    public void setBroj(int broj) {
        //Metoda baca izuzetak koji je malopre napravljen - RacunException.
        if (broj<=0)
            throw new RacunException("Broj racuna mora biti veci od nule");

        this.broj = broj;
    }
    public double getIznos() {
        return iznos;
    }
    public void setIznos(double iznos) {
        if (iznos<=0)
            throw new RacunException("Iznos racuna mora biti veci od nule");

        this.iznos = iznos;
    }
    public String getProizvod() {
        return proizvod;
    }
    public void setProizvod(String proizvod) {
        if (proizvod == null)
            throw new RacunException("Naziv proizvoda ne sme biti null");

        this.proizvod = proizvod;
    }

    public String toString(){
        return "Racun broj: "+broj+" Proizvod: "+proizvod+" Iznos: "+iznos;
    }
}

public class TestRacun {

    public static void main(String[] args) {

        Racun r = new Racun();

        try {
            r.setBroj(24);
        } catch (Exception e) {
            System.out.println("Greska: "+e.getMessage());
        }

        r.setProizvod("Sampon");

        r.setIznos(100.0);

        System.out.println(r);

    }
}

```

Napraviti klasu **ProizvodException** koja predstavlja proveravani izuzetak (nasleduje klasu Exception) i ima:

- Javni konstruktor koji kao ulazni argument prima poruku greške i poziva odgovarajući konstruktor nadklase prosleđujući mu ulazni argument.

Napraviti klasu **Proizvod** koja ima:

- Privatni atribut sifra koji predstavlja šifru proizvoda (ceo broj).
- Privatni atribut naziv.
- Privatni atribut cena (realan broj).
- Javne get i set metode za ove attribute. Šifra i cena moraju biti veći od nule, a naziv ne sme biti null. U slučaju unosa neke od ovih vrednosti, potrebno je baciti izuzetak klase ProizvodException.
- Redefinisani metodu toString klase Object. Ova metoda bi trebalo da vraća String koji sadrži vrednosti svih atributa proizvoda kao i odgovarajući tekst uz to.

Napraviti klasu **TestProizvod** koja kreira jedan objekat klase Proizvod, u njega unosi odgovarajuće vrednosti i ispisuje sve podatke o proizvodu na ekranu. Startovati program. Pokušati unos nekih nedozvoljenih vrednosti za šifru i za ostale attribute, startovati program i videti šta se dešava.

**Rešenje:**

```
public class ProizvodException extends Exception {

    public ProizvodException(String poruka) {
        super(poruka);
    }

}

public class Proizvod {

    private int sifra;
    private String naziv;
    private double cena;

    public double getCena() {
        return cena;
    }

    //Metoda baca izuzetak tipa ProizvodException koji je proveravan
    //pa njegovo ime pise posle reci throws u deklaraciji metode.
    public void setCena(double cena) throws ProizvodException {
        if (cena<=0)
            throw new ProizvodException("Cena mora biti veka od nule");

        this.cena = cena;
    }
    public String getNaziv() {
        return naziv;
    }
    public void setNaziv(String naziv) throws ProizvodException {
        if (naziv == null)
            throw new ProizvodException("Naziv proizvoda ne sme biti null");

        this.naziv = naziv;
    }
    public int getSifra() {
        return sifra;
    }
    public void setSifra(int sifra) throws ProizvodException {
        if (sifra<=0)
            throw new ProizvodException("Sifra mora biti veka od nule");

        this.sifra = sifra;
    }
}
```



```

    }

    public String toString(){
        return "Sifra: "+sifra+" Naziv: "+naziv+" Cena: "+cena;
    }
}

public class TestProizvod {

    public static void main(String[] args) {

        Proizvod p = new Proizvod();

        try {
            p.setSifra(123);
        } catch (ProizvodException e) {
            System.out.println("Greska: "+e.getMessage());
        }

        try {
            p.setNaziv("Pasta za zube");
        } catch (ProizvodException e) {
            System.out.println("Greska: "+e.getMessage());
        }

        try {
            p.setCena(250.0);
        } catch (ProizvodException e) {
            System.out.println("Greska: "+e.getMessage());
        }

        System.out.println(p);
    }
}

```