*EVROPSKI UNIVERZITET KALLOS TUZLA*

*TEHNIČKI FAKULTET*

*INŽENJERSKA INFORMATIKA*

**Apstraktne metode, klase, upravljanje greškama u JAVA PJ**

Mentor : **Prof.** **Dr. Adis Rahmanović** Student: Ohran Imamovic INI

Tuzla, 3. April 2020.

Sadržaj

Uvod .....................................................................................................1

Apstaktna Klasa ....................................................................................2

Primjeri Apstraktne Klase ....................................................................3 - 5

Interface u Javi.......................................................................................6

Razumjevanje Odnosa Između Klasa i Interface-a................................7

Višestruko Nasljeđivanje u Javi pomoću Inteface-a .............................8 - 9

Razlika Između Apstraktne Klase i Interface-a ....................................10

Tipovi Grešaka u Javi ...........................................................................11-15

Zaključak................................................................................................16

Literatura................................................................................................17

**UVOD**

**Java** je programski jezik i računarska platforma koju je prvi put objavio **Sun Microsystems** 1995. godine. Postoji mnogo aplikacija i web lokacija koje neće raditi ako nemate instaliranu Javu, a svaki dan ih se stvori više. Java je brza, sigurna i pouzdana.

Java programe tumači **Java Virtual Machine** ili **JVM** koji radi na više platformi. To znači da su svi Java programi **multiplatformni** i mogu se izvoditi na različitim platformama, uključujući Macintosh, Windows i Unix računare.

Java se može koristiti za kreiranje aplikacija koje se mogu izvoditi na jednom računaru ili distribuirati između servera i klijenata u mreži. Kao rezultat toga, možete ga koristiti za jednostavnu izradu **mobilnih aplikacija** ili pokretanje na **desktop aplikacijama** koje koriste različite operativne sisteme i servere, kao što su Linux ili Windows.

Java nudi veću unakrsnu funkcionalnost i prenosivost jer se programi napisani na jednoj platformi mogu izvoditi na radnim površinama, mobilnim telefonima i ugrađenim sistemima. Java je besplatna, jednostavna, objektno orijentirana, distribuirana, podržava multithreading i nudi multimedijsku i mrežnu podršku.

Sve u Javi je **objekt** koji brine i o podacima i o ponašanju. Java koristi objektno orijentisane koncepte kao što su **objekt, klasa, nasljeđivanje, inkapsulacija, polimorfizam i apstrakcija**. Java je zaštićeni programski jezik jer ne koristi eksplicitne pokazivače.

**Prednosti Java**:

* Jednostavno. Java je jednostavna za upotrebu, pisanje, kompajliranje, otklanjanje grešaka i učenje od alternativnih programskih jezika
* Objektno orijentirano. Omogućava vam da formirate standardne programe i kod za višekratnu upotrebu
* Neovisno o platformi
* Distribuirano računanje
* Sigurnost
* Dodjela memorije
* Višenamjenski

**9670 kompanija** navodno koristi Javu, ovo su neke od tih kompanija:

* Uber
* Netflix
* Google
* Instagram
* Amazon
* Spotify ...

**APSTRAKTNA KLASA**

Klasa koja je deklarisana sa ključnom riječi **abstract**, je poznata kao apstraktna klasa u javi. Ona može imati apstraktne i ne-apstraktne metode (metode sa tijelom).

Prije upoznavanja sa java apstraktnom klasom, prvo treba shvatiti pojam apstrakcije u javi.

**Apstrakcija** je proces skrivanja detalja implementacije i pokazivanja korisniku **samo funkcionalnosti**.  
Drugim riječima, korisniku se pokazuju samo važne stvari, a unutrašnji detalji ostaju skriveni. Na primjer pri slanju sms-a, mi samo kucamo tekst i šaljemo poruku, nije nam poznato interno procesiranje vezano za slanje poruke.  
Apstrakcija nam dopušta da se **fokusiramo na to što objekat radi** umjesto na kako to radi.

**Načini za postizanje apstrakcije**   
Postoje dva načina za postizanje apstrakcije u javi

1. Apstraktna klasa (0 do 100%)
2. Interfejs (100%)

**Apstraktna klasa u javi**

Klasa koja je deklarisana kao apstraktna naziva se **apstraktna klasa**. Ona mora biti naslijeđena i njen metod implementiran. Ona ne može biti instancirana.  
Apstraktna klasa je neka vrsta „**fantomske klase“**. Ona može prosljeđivati metode i varijable, ali sama nikada ne može biti instancirana, to jeste, **nije moguće kreirati objekt apstraktne klase**. U tom smislu, apstraktna klasa je **poput interfejsa**, ali za razliku od njega ona može sadržavati metode koji nisu apstraktni. Takođe može sadržavati deklaracije podataka koji nisu konstante.  
Svaka klasa koja sadrži jedan ili više apstraktnih metoda mora biti deklarisana kao apstraktna. U apstraktnim klasama (za razliku od interfejsa) **modifikator abstract** mora biti primijenjen na svaki apstraktni metod.  
Apstraktne klase se ponašaju kao „držači mjesta“ (**placeholderi**) u hijerarhiji klasa. Na primjer, apstraktna klasa može sadržavati djelimičnu deskripciju koju nasljeđuju svi njeni potomci u hijerarhiji klasa. Njena djeca, koja su mnogo određenija, popunjavaju praznine.  
**Apstraktni metod nikada ne može biti static.**

**PRIMJERI APSTAKTNE KLASE**

abstract class A{}

**Apstraktni metod**  
Metod koji je deklarisan kao apstraktan i koji nema implementaciju se naziva apstraktni metod.

Abstract void printStatus();  //nema tijela i ima abstract

**Primjer apstraktne klase koja ima apstraktni metod**  
U ovom primjeru, apstraktna klasa Bike sadrži samo jedan apstraktni metod run. Njegovu implementaciju obezbjeđuje klasa Honda.

Abstract class Bike{

  abstract void run();

}

class Honda4 extends Bike{

void run(){System.out.println(„running safely..“);}

public static void main(String args[]){

 Bike obj = new Honda4();

 obj.run();

}

}

Ispis na ekranu:

running safely..

**FACTORY METODA:** U ovom primjeru, ako kreiramo instancu klase Rectangle, biće pozvan draw() metod klase Rectangle.

Abstract class Shape{

abstract void draw();

}

//U stvarnom scenariju, implementaciju obezbjeđuje neko drugi tj. Ona je nepoznata krajnjem korisniku

class Rectangle extends Shape{

void draw(){System.out.println(„drawing rectangle“);}

}

class Circle1 extends Shape{

void draw(){System.out.println(„drawing circle“);}

}

//U stvarnom scenariju, metod poziva programer ili korisnik

class TestAbstraction1{

public static void main(String args[]){

Shape s=new Circle1();  //U stvarnom scenariju, objekt obezbjeđuje metod napr. getShape() metod

s.draw();

}

}

Ispis na ekranu:

drawing circle

**Razumijevanje stvarnog scenarija apstraktne klase**  
U ovom primjeru, Shape je apstraktna klasa, a njenu implementaciju obezbjeđuju klase Rectangle i Circle. Najčešće, mi ne znamo ništa o implementaciji klase (tj. Ona je skrivena za krajnjeg korisnika) a  objekt implementacione klase obezbjeđuje **factory metod**.  
**Factory metod** je metod koji vraća instancu klase. O factory metodu ćemo govoriti kasnije.  
U ovom primjeru, ako kreiramo instancu klase Rectangle, biće pozvan draw() metod klase Rectangle.

**Drugi primjer apstraktne klase u javi:**

abstract class Bank{

abstract int getRateOfInterest();

}

class SBI extends Bank{

int getRateOfInterest(){return 7;}

}

class PNB extends Bank{

int getRateOfInterest(){return 7;}

}

class TestBank{

public static void main(String args[]){

Bank b=new SBI();  //ako je objekt PNB, biće pozvan metod od PNB

int interest=b.getRateOfInterest();

System.out.println(„Rate of Interest is: „+interest+“ %“);

}}

Ispis na ekranu:

Rate of Interest is: 7 %

**Pravilo:** Akopostoji bar jedan apstraktni metod u klasi, ta klasa mora biti apstraktna.

Abstract void run();

  }

  Ispis na ekranu:

compile time error

Ako nasljeđujemo bilo koju apstraktnu klasu koja ima apstraktni metod, moramo obezbijediti implementaciju tog metoda ili učiniti tu klasu apstraktnom.

**Apstraktna klasa koja ima konstruktor, podatak-član, metode.**  
Apstraktna klasa može imati podatak-član, apstraktni metod, tijelo metoda, konstruktor i čak main() metod.

//primjer apstraktne klase koja ima tijelo metoda

 abstract class Bike{

   Bike(){System.out.println(„bike is created“);}

   abstract void run();

   void changeGear(){System.out.println(„gear changed“);}

 }

 class Honda extends Bike{

 void run(){System.out.println(„running safely..“);}

 }

 class TestAbstraction2{

 public static void main(String args[]){

  Bike obj = new Honda();

  obj.run();

  obj.changeGear();

 }

}

Ispis na ekranu:

 bike is created

       running safely..

       gear changed

**Još jedan stvarni scenario za apstraktnu klasu**  
Apstraktna klasa može takođe biti upotrijebljena da obezbijedi neke implementacije interfejsa. U tom slučaju, krajnji korisnik ne mora biti prinuđen da preklopi sve metode interfejsa.

Interface A{

void a();

void b();

void c();

void d();

}

abstract class B implements A{

public void c(){System.out.println(„I am C“);}

}

class M extends B{

public void a(){System.out.println(„I am a“);}

public void b(){System.out.println(„I am b“);}

public void d(){System.out.println(„I am d“);}

}

class Test5{

public static void main(String args[]){

A a=new M();

a.a();

a.b();

a.c();

a.d();

}}

Ispis na ekranu:

I am a

I am b

I am c

I am d

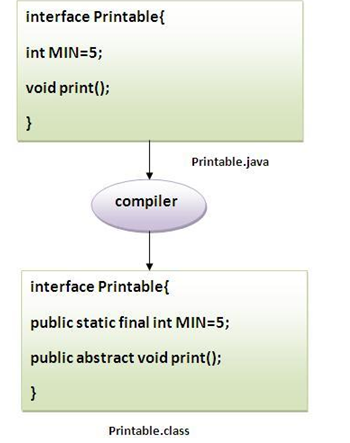
**INTERFACE U JAVI**

**Interface u Javi** je nacrt (blueprint) klase. On ima samo statičke konstante i apstraktne metode.   
Interface u javi je **mehanizam za postizanje potpune apstrakcije**. U java interface-u **mogu postojati samo apstraktni metodi bez tijela metoda**. On se koristi za **postizanje pune apstrakcije** i višestrukog nasljeđivanja u Javi.   
Java Interface takođe **predstavlja IS-A relaciju**.  
On ne može biti instanciran kao ni apstraktna klasa.

**Zašto se koristi interface?**  
Postoje uglavnom tri razloga za upotrebu interface-a. To su:

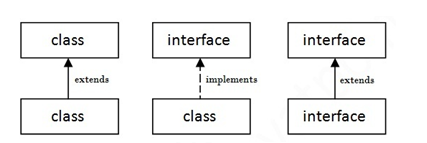
* Koristi se za postizanje pune apstrakcije.
* Pomoću interfejsa, moguće je podržati funkcionalnost višestrukog nasljeđivanja.
* Može se koristiti za postizanje tzv. labavog spajanja (loose coupling).

**Napomena:** **Java kompajler dodaje ključne riječi public i abstract prije metoda interfejsa i ključne riječi public, static i final prije podataka-članova**.  
Drugim riječima, polja interfejsa su public, static i final po default-u, a metodi su public i abstract.



**RAZUMJEVANJE ODNOSA IZMEĐU KLASA I INTERFACE-A**

Kao što je prikazano na sljedećoj slici, klasa nasljeđuje drugu klasu, interface nasljeđuje drugi interface ali **klasa implementira interface**.



**Jednostavan primjer java interface-a**  
U ovom primjeru, interface Printable ima samo jedan metod, njegova implementacija je obezbjeđena u klasi A.

interface printable{

void print();

}

class A6 implements printable{

public void print(){System.out.println("Hello");}

public static void main(String args[]){

A6 obj = new A6();

obj.print();

 }

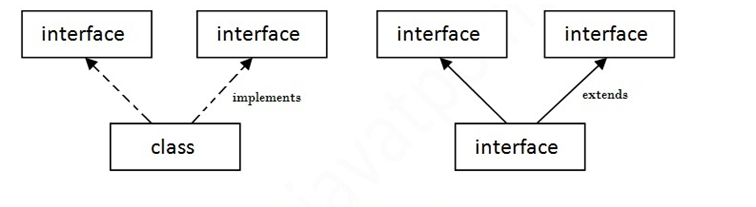
}

Ispis na ekranu:

Hello

**VIŠESTRUKO NASLJEĐIVANJE U JAVI POMOĆU INTERFACE-A**

Ako klasa implementira više interface-a, ili interface nasljeđuje više interface-a to je poznato kao višestruko nasljeđivanje (multiple inheritance).

*Višestruko nasljeđivanje*

interface Printable{

void print();

}

interface Showable{

void show();

}

class A7 implements Printable,Showable{

public void print(){System.out.println("Hello");}

public void show(){System.out.println("Welcome");}

public static void main(String args[]){

A7 obj = new A7();

obj.print();

obj.show();

 }

}

Ispis na ekranu:

Hello

Welcome

**Zašto višestruko nasljeđivanje nije podržano preko klasa u Javi ali je moguće pomoću interface-a?**  
Kao što je rečeno u poglavlju o nasljeđivanju, višestruko nasljeđivanje **nije podržano** u slučaju klasa. Međutim, **podržano je u slučaju interface-a** zato što nema dvosmislenosti jer je implementacija obezbijeđena preko implementacione klase. Na primjer:

interface Printable{

void print();

}

interface Showable{

void print();

}

class testinterface1 implements Printable,Showable{

public void print(){System.out.println("Hello");}

public static void main(String args[]){

testinterface1 obj = new testinterface1();

obj.print();

 }

}

Ispis na ekranu:

Hello

Kao što se može vidjeti u ovom primjeru, Printable i Showable interfejs imaju iste metode ali njihovu implementaciju obezbjeđuje klasa A, tako da nema dvosmislenosti.

**Nasljeđivanje interfejsa**

Klasa implementira interface ali jedan interface nasljeđuje drugi interface.

interface Printable{

void print();

}

interface Showable extends Printable{

void show();

}

class Testinterface2 implements Showable{

public void print(){System.out.println("Hello");}

public void show(){System.out.println("Welcome");}

public static void main(String args[]){

Testinterface2 obj = new Testinterface2();

obj.print();

obj.show();

 }

}

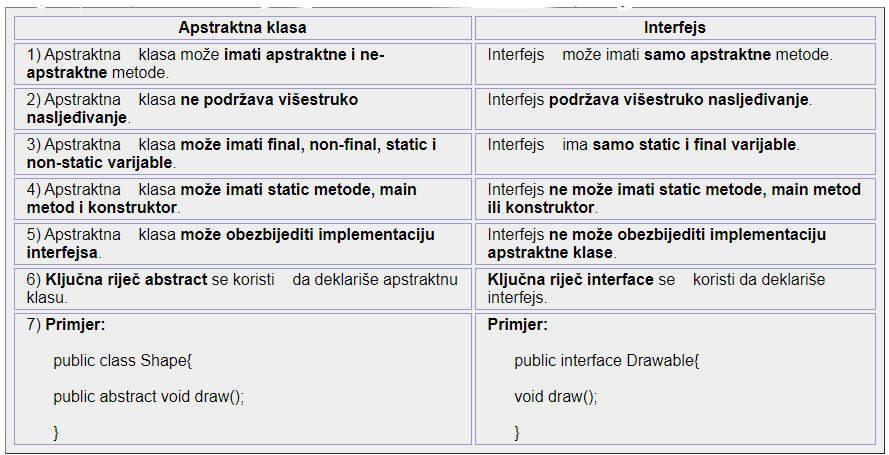
Ispis na ekranu:

Hello

Welcome

**RAZLIKA IZMEĐU APSTRAKTNE KLASE I INTERFACE-A**

**Apstraktna klasa i interface se koriste za postizanje apstrakcije** gdje možemo deklarisati apstraktne metode. Apstraktna klasa i interface se ne mogu instancirati.  
Ali postoje mnoge razlike između apstraktne klase i interface-a koje su date u sljedećoj tabeli.



Jednostavno, apstraktna klasa postiže djelimičnu apstrakciju (0 do 100%) dok interfejs postiže punu apstrakciju (100%).  
**Primjer apstraktne klase i interface-a u javi**

Pogledajmo jednostavan primjer gdje koristimo i interface i apstraktnu klasu.

|  |
| --- |
| //Kreiranje interfejsa koji ima 4 metoda  interface A{  void a(); //po default-u, public i abstract  void b();  void c();  void d();  }  //Kreiranje apstraktne klase koja obezbjeđuje implementaciju jednog metoda A interfejsa  abstract class B implements A{  public void c(){System.out.println("I am C");}  }  //Kreiranje podklase apstraktne klase, sada trebamo da obezbijedimo implementaciju ostalih //metoda  class M extends B{  public void a(){System.out.println("I am a");}  public void b(){System.out.println("I am b");}  public void d(){System.out.println("I am d");}  }  //Kreiranje test klase koja poziva metode A interfejsa  class Test5{  public static void main(String args[]){  A a=new M();  a.a();  a.b();  a.c();  a.d();  }}    Ispis na ekranu:  I am a  I am b  I am c  I am d |

**TIPOVI GREŠAKA U JAVA**

Greška je ilegalna operacija koju izvodi korisnik što rezultira nenormalnim radom programa. Programske greške često **ostaju neotkrivene** dok se program ne prevede ili izvrši. Neke greške sprečavaju program da se prevede ili izvrši. Stoga bi greške trebalo ukloniti prije sastavljanja i izvršavanja.

**Najčešće greške mogu se široko klasificirati na sljedeći način**:

Pogreška u vremenu izvođenja:

* **Greške u vremenu izvođenja** pojavljuju se ili možemo reći da su otkrivene tokom izvršavanja programa. Ponekad se otkriju kada korisnik unese nevažeće podatke ili podatke koji nisu relevantni. Greške pri izvođenju pojavljuju se kada **program ne sadrži sintaksne greške**, ali traži od računara da učini nešto što računar nije u stanju učiniti. **Tokom kompilacije**, kompajler **nema tehniku** ​​za otkrivanje ove vrste grešaka. JVM (**Java Virtual Machine**) je taj koji ga otkriva dok je program pokrenut. Da bismo riješili grešku tokom vremena izvođenja, možemo staviti naš kod greške unutar **bloka try** i uhvatiti grešku unutar **bloka catch**.

Na primjer: ako korisnik unese podatke u formatu niza kada računalo očekuje cijeli broj, doći će do greške u izvođenju.

Primjer 1: Greška u izvođenju, uzrokovana dijeljenjem sa nulom:

// Java program za demonstraciju greške u vremen izvođenja

class DivByZero {

    public static void main(String args[])

    {

        int var1 = 15;

        int var2 = 5;

        int var3 = 0;

        int ans1 = var1 / var2;

        // Ova izjava pravi grešku pri izvođenju,

        // gdje se 15 dijeli sa 0

        int ans2 = var1 / var3;

        System.out.println(

            "Division of va1"

            + " by var2 is: "

            + ans1);

        System.out.println(

            "Division of va1"

            + " by var3 is: "

            + ans2);

    }

}

Ispis:

Exception in thread "main" java.lang.ArithmeticException: / by zero

at DivByZero.main(File.java:14)

Primjer 2: Greška u izvođenju uzrokovana dodjeljivanjem / dohvatanjem vrijednosti iz niza pomoću indeksa koji je veći od veličine niza:

class RTErrorDemo {

    public static void main(String args[])

    {

        int arr[] = new int[5];

        // Array veličina je 5

        // dok ova izjava dodjeljuje

        // value 250 na 10. poziciju.

        arr[9] = 250;

        System.out.println("Value assigned! ");

    }

}

Ispis:

Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 9

at RTErrorDemo.main(File.java:10)

**Greške pri prevođenju**

Pogreške pri prevođenju su one pogreške koje sprečavaju pokretanje koda zbog **neispravne sintakse kao što je nedostatak zareza na kraju izraza ili nedostatak zagrada, klasa nije pronađena** i tako dalje. Ove greške **otkriva java kompajler** i na ekranu se prikazuje poruka o grešci tokom kompajliranja. Pogreške vremena kompajliranja ponekad se nazivaju i **sintaksne greške**. Ovu vrstu grešaka lako je uočiti i otkloniti jer ih java kompajler pronalazi za vas. **Kompajler će vam reći koji je dio koda u programu ima problema** i najbolje pogađa šta ste pogriješili. Kompajler obično označi točnu liniju gdje je greška ili ponekad liniju neposredno prije nje, međutim, ako je problem u pogrešno ugniježđenim zagradama, stvarna pogreška može biti na početku bloka. **U stvari, sintaksne greške predstavljaju gramatičke greške u upotrebi programskog jezika.**

Primjer 1: Pogrešno napisana imena varijabli ili imena metoda:

class MisspelledVar {

    public static void main(String args[])

    {

        int a = 40, b = 60;

        // Deklarisana varijabla Sum sa veliko S

        int Sum = a + b;

        // Pravilno je nazvati varijablu sum

        // sa malim slovom s ie. sum

        System.out.println(

            "Sum of variables is "

            + sum);

    }

}

Greška pri kompilaciji u Java kodu:

prog.java:14: error: cannot find symbol

+ sum);

^

symbol: variable sum

location: class MisspelledVar

1 error

Primjer 2: Nedostaju tačka i zarez

class PrintingSentence {

    public static void main(String args[])

    {

        String s = "GeeksforGeeks";

        // Nedostaje ';' na kraju

        System.out.println("Welcome to " + s)

    }

}

Greška pri kompilaciji u Java kodu:

prog.java:8: error: ';' expected

System.out.println("Welcome to " + s)

^1 error

Primjer 3: Nedostaju zagrade, uglate zagrade ili kovrčave zagrade:

class MissingParenthesis {

    public static void main(String args[])

    {

        System.out.println("Printing 1 to 5 \n");

        int i;

        // nedostatak zagrade, trebalo je biti

        // for(i=1; i<=5; i++)

        for (i = 1; i <= 5; i++ {

            System.out.println(i + "\n");

        } // for loop završetak

    } // main završetak

} // završetak klase

Greška pri kompilaciji u Java kodu:

prog.java:10: error: ')' expected

for (i = 1; i <= 5; i++ {

^ 1 error

Primjer 4: Nepravilan format izjava ili petlji za odabir:

class IncorrectLoop {

    public static void main(String args[])

    {

        System.out.println("Multiplication Table of 7");

        int a = 7, ans;

        int i;

        // Trebalo je biti

        // for(i=1; i<=10; i++)

        for (i = 1, i <= 10; i++) {

            ans = a \* i;

            System.out.println(ans + "\n");

        }

    }

}

Greška pri kompilaciji u Java kodu:

prog.java:12: error: not a statement

for (i = 1, i <= 10; i++) {

^

prog.java:12: error: ';' expected

for (i = 1, i <= 10; i++) {

^

2 errors

**Logička greška**

Logička greška je kada se vaš program kompajlira i izvrši, **ali učini pogrešnu stvar** ili vrati netačan rezultat ili nema rezultata kada bi trebao vraćati izlaz. **Te greške ne otkriva ni kompajler ni JVM**. Java sistem nema pojma šta bi vaš program trebao raditi, tako da ne pruža dodatne informacije koje će vam pomoći da pronađete grešku. Logičke greške nazivaju se i **semantičkim** greškama. **Do ovih grešaka dolazi zbog pogrešne ideje ili koncepta koji programer koristi prilikom kodiranja**. Sintaksne greške su gramatičke greške, dok su logičke greške greške koje proizlaze iz neispravnog značenja. Na primjer: ako programer slučajno doda dvije varijable kada ih je htio podijeliti, program neće dati grešku i uspješno će se izvršiti, ali sa netačnim rezultatom.

Primjer: Slučajno korištenje pogrešnog operatora na varijablama za izvođenje operacije (Korištenje operatora '/' za dobivanje modula umjesto '%'):

public class LErrorDemo {

    public static void main(String[] args)

    {

        int num = 789;

        int reversednum = 0;

        int remainder;

        while (num != 0) {

            // da bi zadržali modul % znak trebao

            // bi biti korišten umjesto/

            remainder = num / 10;

            reversednum

                = reversednum \* 10

                  + remainder;

            num /= 10;

        }

        System.out.println("Reversed number is "

                           + reversednum);

    }

}

Ispis: Reversed number is 7870

Primjer: Prikazivanje pogrešne poruke:

class IncorrectMessage {

    public static void main(String args[])

    {

        int a = 2, b = 8, c = 6;

        System.out.println(

            "Finding the largest number \n");

        if (a > b && a > c)

            System.out.println(

                a + " is the largest Number");

        else if (b > a && b > c)

            System.out.println(

                b + " is the smallest Number");

        // Ispravna poruka bi trebala biti

        // System.out.println

        //(b+" is the largest Number");

        // da bi bilo logično

        else

            System.out.println(

                c + " is the largest Number");

    }

}

Ispis: Finding the largest number

8 is the smallest Number

**ZAKLJUČAK**

**Java** je programski jezik i računarska platforma koju je prvi put objavio **Sun Microsystems** 1995. godine. Postoji mnogo aplikacija i web lokacija koje neće raditi ako nemate instaliranu Javu, a svaki dan ih se stvori više. Java je brza, sigurna i pouzdana.

Java programe tumači **Java Virtual Machine** ili **JVM** koji radi na više platformi. To znači da su svi Java programi **multiplatformni** i mogu se izvoditi na različitim platformama, uključujući Macintosh, Windows i Unix računare.

**Apstrakcija** je proces skrivanja detalja implementacije i pokazivanja korisniku **samo funkcionalnosti**.  
Drugim riječima, korisniku se pokazuju samo važne stvari, a unutrašnji detalji ostaju skriveni. Na primjer pri slanju sms-a, mi samo kucamo tekst i šaljemo poruku, nije nam poznato interno procesiranje vezano za slanje poruke.  
Apstrakcija nam dopušta da se **fokusiramo na to što objekat radi** umjesto na kako to radi.

Klasa koja je deklarisana kao apstraktna naziva se **apstraktna klasa**. Ona mora biti naslijeđena i njen metod implementiran. Ona ne može biti instancirana.  
Apstraktna klasa je neka vrsta „**fantomske klase“**. Ona može prosljeđivati metode i varijable, ali sama nikada ne može biti instancirana, to jeste, **nije moguće kreirati objekt apstraktne klase**.

**Interface u Javi** je nacrt (blueprint) klase. On ima samo statičke konstante i apstraktne metode.   
Interface u javi je **mehanizam za postizanje potpune apstrakcije**. U java interface-u **mogu postojati samo apstraktni metodi bez tijela metoda**. On se koristi za **postizanje pune apstrakcije** i višestrukog nasljeđivanja u Javi.   
Java Interface takođe **predstavlja IS-A relaciju**.  
On ne može biti instanciran kao ni apstraktna klasa.

**Greška** je ilegalna operacija koju izvodi korisnik što rezultira nenormalnim radom programa. Programske greške često **ostaju neotkrivene** dok se program ne prevede ili izvrši. Neke greške sprečavaju program da se prevede ili izvrši. Stoga bi greške trebalo ukloniti prije sastavljanja i izvršavanja

Tipovi grešaka u Javi :

* Greške u vremenu izvođenja
* Greške pri prevođenju
* Logičke greške

**LITERATURA**

<https://www.znanje.org/knjige/computer/java/ib01/300Java/31000290.htm>

[https://www.fer.unizg.hr/\_download/repository/5\_ApstraktneKlase\_Sucelja[2].pdf](https://www.fer.unizg.hr/_download/repository/5_ApstraktneKlase_Sucelja%5b2%5d.pdf)

<https://www.znanje.org/knjige/computer/java/ib01/300Java/31000310.htm?fbclid=IwAR3QVwUGTDKIVlyKLRBk7kY36c0sUF2PS8vtWY_lWp_Mq_Ll5ixHtmTxdnY>

<https://izprogramiranja.weebly.com/apstraktne_klase_i_interfejsi.html?fbclid=IwAR1euDkAlaYYYfdBZ3tKTZomjaTKMD74ngRaz5hjl7p5v5d612z233BoAQ0>

<https://www.geeksforgeeks.org/types-of-errors-in-java-with-examples/>