PRIMERI GREŠAKA I IZUZETAKA U JAVI

ERRORS

- Greška je nedozvoljen iskaz ili operacija koju izvodi programer, a koja uzrokuje da se aplikacija ili program ponaša “nenormalno” i ograničava aplikaciju da efikasno obavlja svoje zadatke
- Greške obično ostaju neotkrivene dok se program ne prevede (compile) ili izvrši (execute) i često mogu uzrokovati prekid programa ili sprečiti normalan tok prevođena ili izvršavanja
- Do pojave greške može doći iz dva razloga:
  - Ukoliko programer ne poštuje pravila pisanja nekog programskog jezika
  - Ukoliko programer izvrši operaciju koja nije predviđena zbog pogrešne ideje ili koncepta

GREŠKE U VREME PREVOĐENJA

- Vreme prevođenja (compile time) je vreme u kome se izvorni kod pretvara u izvršni kod
- Greške koje se javljaju u compile time-u su greške koje su nastale kao posledica nepoštovanja sintakse jezika
- Kompajler neće dozvoliti pokretanje programa dok se sve greške ovog tipa ne uklone iz programa. Kada se sve greške uklone iz programa, kompajler će generisati izvršnu datoteku.
- Greške u vremenu kompajliranja mogu biti:
  - Sintaksne greške
  - Semantičke greške

SINTAKSNE GREŠKE

- Problemi koji se javljaju zbog pogrešne upotrebe sintakse programskog jezika
- Greške proverava kompajler u vreme kompajliranja programa i označava deo koda u kom se greška desila, kao i tekstualni opis greške
- Tip greške koja se lako uočava i prevazilazi se povećavanjem iskustva u datom programskom jeziku
- Najčešći tipovi sintaksnih grešaka:
    A. Nedostaju tačka i zarez
    B. Nedostaju zagrade
    C. Pogrešno napisane ključne reči
    D. Upotreba nedeklarisanih promenljivih
    E. Nepravilno imenovana promenljivih ili funkcija ili klasa
    F. Klasa nije pronađena
    G. Nedostaju dvostruki navodnici u String-u

KONVENCIJA IMENOVANJA U JAVI

- Postoje određena pravila za davanje imena promenljivama, metodama i klasama u Javi:
  - Svi identifikatori treba da počnu slovom (od A do Z ili a do z), znakom valute ($) ili donjom crtom (_)
  - Nakon prvog znaka, identifikatori mogu imati bilo koju kombinaciju znakova
  - Ključne reči se ne mogu koristiti kao identifikatori
  - Identifikatori su osetljivi na mala i velika slova (case sensitive)
  - Višestruke reči u nazivima metoda i promenljivih se definišu u `camelCase` notaciji
  - Višestruke reči u nazivima klasa se definišu u `UpperCamelCase` notaciji
- Primeri dozvoljenih identifikatora:
  - ime3, $uid, _value, _3b__age, roll_no
- Primeri nedozvoljenih identifikatora:
  - 13ab, -godina

SEMANTIČKE GREŠKE

- Većina grešaka u vremenu kompajliranja su greške u opsegu i deklaraciji
- Semantička greška može nastati korišćenjem pogrešne promenljive ili korišćenjem pogrešnog operatora ili izvođenjem operacije pogrešnim redosledom
- Najčešći tipovi semantičkih grešaka:

A. Nekompatibilni tipovi operanda

B. Nedeklarisana promenljiva

C. Nepoklapanje stvarnog argumenta sa formalnim argumentom

PRIMER 1

```
class SyntaxError {
    public static void main(String[] args){
        int a = 2;
        int b = 4
        System.out.println(a + b);
    }
}
```

PRIMER 2

```
class SyntaxError {
    public static void main(String[] args){
        int a = 2;
        int b = 4;
        System.out.println(a + b);
}
```

PRIMER 3

```
class SyntaxError {
   public static void main(String[] args) {
     int a = 2;
     int b = 4;
     system.out.println(a + b);
   }
}
```

PRIMER 4

```
class 1SyntaxError {
    public static void main(String[] args) {
        int a = 2;
        int b = 4;
        System.out.println(a + b);
    }
}
```

PRIMER 5

```
class SyntaxError {
   public static void main(String[] args) {
      int a = -2;
      Scanner sc = new Scanner(System.in);
      int b = sc.nextInt();
      System.out.println(a + b);
   }
}
```

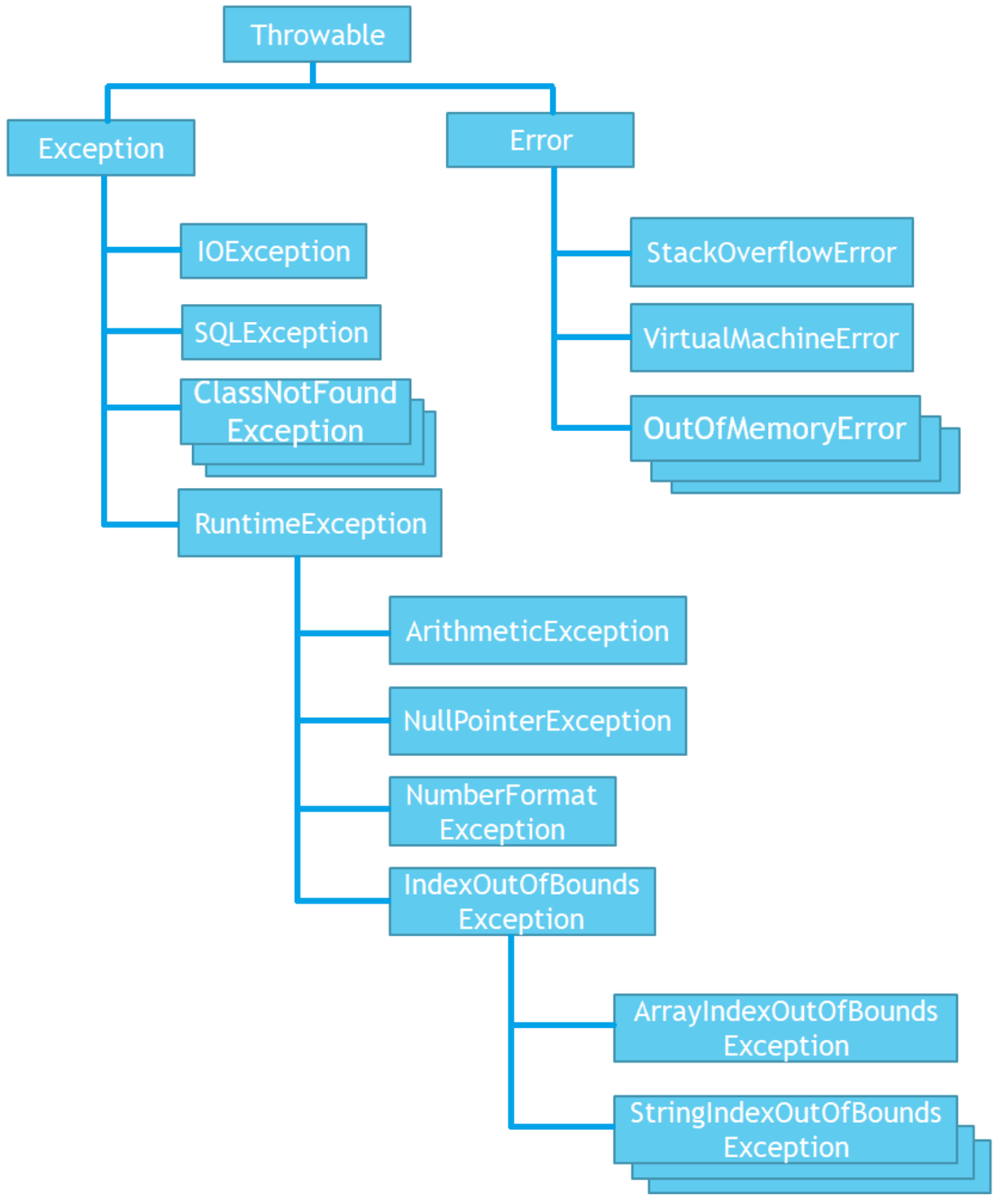
PRIMER 6

```
class SyntaxError {
    public static void main(String[] args) {
        System.out.println(Hello World!);
    }
}
```

GREŠKE U VREME IZVRŠAVANJA

- Greške tokom izvršavanja nastaju kada se program uspešno kompajlira i kreira se ".class" datoteke, međutim, program se ne izvršava kako treba. Ove greške se otkrivaju u vreme izvršavanja programa
- Java kompajler ne otkriva greške tokom izvršavanja jer u tom trenutku nema sve informacije o podacima. Java virtuelna mašina (JVM) hvata greške u toku rada kada je program pokrenut
- Ove greške u toku izvršavanja nazivaju se izuzecima (exceptions) i nenormalno prekidaju program, dajući izjavu o grešci

HIJERARHIJA IZUZUZETAKA U JAVI



TIPOVI IZUZETAKA

- Postoje tri vrste izuzetaka:
  - Checked Exception
    - Izuzeci koji se proveravaju u vreme kompajliranja (checked in compile time)
    - Ako neki kod unutar metode baca checked izuzetak, onda metoda **mora** ili da obradi izuzetak ili **mora** da specificira izuzetak koristeći ključnu reč throvs
    - Klase koje nasleđuju `Throwable` osim `RuntimeException` i `Error`
      - Dakle to su `IOException, SQLException i ClassNotFoundException`
  - Unchecked Exception
    - Proveravaju se u runtime-u (unchecked in compile time)
    - Za ovaj tip grešaka kompajler ne primorava da programera da rukuje ili specificira izuzetak
    - Klase koje nasleđuju `RuntimeException`
    - Na primer: `ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException...`
  - Error
    - Proveravaju se u run time-u, zato kažemo da se mogu svrstati u Unchecked Exceptione
    - Greške do kojih uglavnom dolazi zbog nedostatka sistemskih resursa
    - Ne mogu se uhvatiti ili handlovati

PODRAZUMEVANO RUKOVANJE IZUZECIMA

- Kada u programu unutar metode dođe do izuzetka, metoda kreira objekat poznat kao `Exception Object` i predaje ga sistemu za izvršavanje (JVM)
- Objekat izuzetka sadrži ime i opis izuzetka i trenutno stanje programa u kome je došlo do izuzetka
- Kreiranje objekta izuzetka i njegovo rukovanje u sistemu za izvršavanje naziva se bacanjem izuzetka (throwing an Exception)
- Potencijalno će postojati lista metoda koje su pozvane da bi se došlo do metode gde je došlo do izuzetka. Ova uređena lista metoda se zove stek poziva `(stack trace)`

- U programu se dešavaju sledeći koraci:

1. Run time sistem pretražuje stack trace kako bi pronašao metodu koja sadrži blok koda koji može da obradi nastali izuzetak. Blok koda se zove obrađivač izuzetaka (`Exception handler`)
2. Run time sistem počinje pretragu od metode u kojoj se dogodio izuzetak i nastavlja kroz stek poziva obrnutim redosledom od onog u kom su metode pozivane
3. Ako pronađe odgovarajući handler, onda mu prosleđuje nastali izuzetak. Odgovarajući handler znači da tip bačenog objekta izuzetka odgovara tipu objekta izuzetka koji može da obradi
4. Ako run time sistem pretraži sve metode na steku poziva i ne pronađe odgovarajući handler, onda se Exception objekat predaje podrazumevanom handleru, koji je deo run time sistema. Ovaj handler štampa informacije o izuzetku u sledećem formatu i neuobičajeno prekida program

```
Exception in thread "xxx" Name of Exception : Description
... ..... ..  // Call Stack
```

PRILAGOĐENO RUKOVANJE IZUZECIMA

- U Javi rukovanjem izuzetcima se upravlja preko pet ključnih reči: `try`, `catch`, `throw`, `throws` i `finally`
- Programske izjave za postoji mogućnost da mogu da izazovu izuzetke nalaze se u bloku `try`
- Ako dođe do izuzetka unutar `try` bloka, izvršiće se `throw` datog izuzetka

```
try {
    // block of code to monitor for errors
    // the code you think can raise an exception
} catch (ExceptionType1 exOb) {
    // exception handler for ExceptionType1
} catch (ExceptionType2 exOb) {
    // exception handler for ExceptionType2
}
// optional
finally {  // block of code to be executed after try block ends
}
```

- Programski kod takođe može uhvatiti ovaj izuzetak (koristeći `catch` blok) i rukovati njime na neki racionalan način
- Sistemski generisani izuzeci se automatski izbacuju od strane Java run time sistema. Ukoliko je potrebno ručno baciti izuzetak, koristi se ključna reč `throw`
- Svaki izuzetak koji je izbačen iz metode mora biti specificiran kao takav klauzulom `throws`
- Svaki kod koji se apsolutno mora izvršiti nakon završetka `try` bloka stavlja se u `finally` blok

PRILAGOĐENO RUKOVANJE IZUZECIMA

- U metodi može postojati više od jedne izjave koja može da izazove izuzetak, tako da ih je sve potrebno postaviti u `try` blok i obezbediti po jedan handler unutar pojedinačnih `catch` blokova
- Ako dođe do izuzetka unutar `try` bloka, izuzetkom će rukuvati handler koji je sa njim povezan. Može postojati više od jednog handler-a izuzetaka. Svaki `catch` blok je handler za izuzetak koji obrađuje izuzetak tipa naznačenog njegovim argumentom. Argument `ExceptionType` deklariše tip izuzetka koji može da obradi i mora biti ime klase koja nasleđuje klasu `Throwable`
- Za svaki `try` blok, može biti nula ili više `catch` blokova, ali samo jedan `finally` blok
- `finally` blok je opcioni. Uvek se izvršava bez obzira da li je došlo do izuzetka u `try` bloku. Ako dođe do izuzetka, on će se izvršiti nakon `try` i `catch` blokova. A ako se ne dogodi izuzetak, onda će se izvršiti nakon `try` bloka
- `finally` blok se u Javi koristi za postavljanje važnih kodova kao što je kod za čišćenje, na primer, zatvaranje datoteke ili zatvaranje veze

RUKOVANJA IZUZECIMA

```java
public class TryCatch {
    public static void main (String[] args)
    {

        // array of size 4.
        int[] arr = new int[4];
        try
        {
            int i = arr[4];
            System.out.println("Inside try block");
        }
        catch(ArrayIndexOutOfBoundsException ex)
        {
            System.out.println("Exception caught in Catch block");
        }
        System.out.println("Outside try-catch clause");
    }
}
```

RUKOVANJA IZUZECIMA

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

        // array of size 4.
        int[] arr = new int[4];

        try
        {
            int i = arr[4];
            System.out.println("Inside try block");
        }

        catch(ArrayIndexOutOfBoundsException ex)
        {
            System.out.println("Exception caught in catch block");
        }

        finally
        {
            System.out.println("finally block executed");
        }

        System.out.println("Outside try-catch-finally clause");
    }
}
```

RUKOVANJA IZUZECIMA

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

        // array of size 4.
        int[] arr = new int[4];
        try
        {
            int i = arr[4];
            System.out.println("Inside try block");
        }

        catch(NullPointerException ex)
        {
            System.out.println("Exception has been caught");
        }

        System.out.println("Outside try-catch clause");
    }
}
```

RUKOVANJA IZUZECIMA

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

        // array of size 4.
        int[] arr = new int[4];

        try
        {
            int i = arr[4];
            System.out.println("Inside try block");
        }

        catch(NullPointerException ex)
        {
            System.out.println("Exception has been caught");
        }

        finally
        {
            System.out.println("finally block executed");
        }

        System.out.println("Outside try-catch-finally clause");
    }
}
```

RUKOVANJA IZUZECIMA

```
public class TryCatchFinallyNotRaised {
    public static void main (String[] args)
    {
        try
        {
            String str = "123";

            int num = Integer.parseInt(str);
            System.out.println("try block fully executed");
        }

        catch(NumberFormatException ex)
        {
            System.out.println("catch block executed...");

        }

        finally
        {
            System.out.println("finally block executed");
        }

        System.out.println("Outside try-catch-finally clause");
    }
}
```

GREŠKE U LOGICI PROGRAMA

- Logičke greške je najteže identifikovati i ispraviti
- Najteže ih je otkriti jer ih ne identifikuje ni Java kompajler ni JVM
- Programer je u potpunosti odgovoran za njih
- Program sa logičkom greškom biće preveden i uspešno izvršen, ali neće dati očekivani rezultat.
- Testeri aplikacija mogu otkriti logičke greške kada uporede stvarni rezultat sa očekivanim rezultatom programa

DA LI POSTOJE PROBLEMI U KODU?

```
public class TestException3 {

    public static void main(String[] args) {
        try{
        bar();
        }catch(NullPointerException e){
            e.printStackTrace();
        }catch(Exception e){
            e.printStackTrace();
        }

        foo();
    }

    public static void bar(){

    }

    public static void foo() throws NullPointerException{

    }
}
```