Tačnost i robusnost

- **Osnovna filozofija Jave** je da se **loše napisan kod neće izvršavati!**
- Program je **tačan** ako uspešno realizuje zadatak za koji je projektovan
- Program je **robustan** ako je u stanju da reaguje na neočekivane situacije (kao što su nevalidni ulazni podaci) na razuman način
- Da bismo napravili **robustan sistem**, **svaka njegova komponenta mora biti robusna**

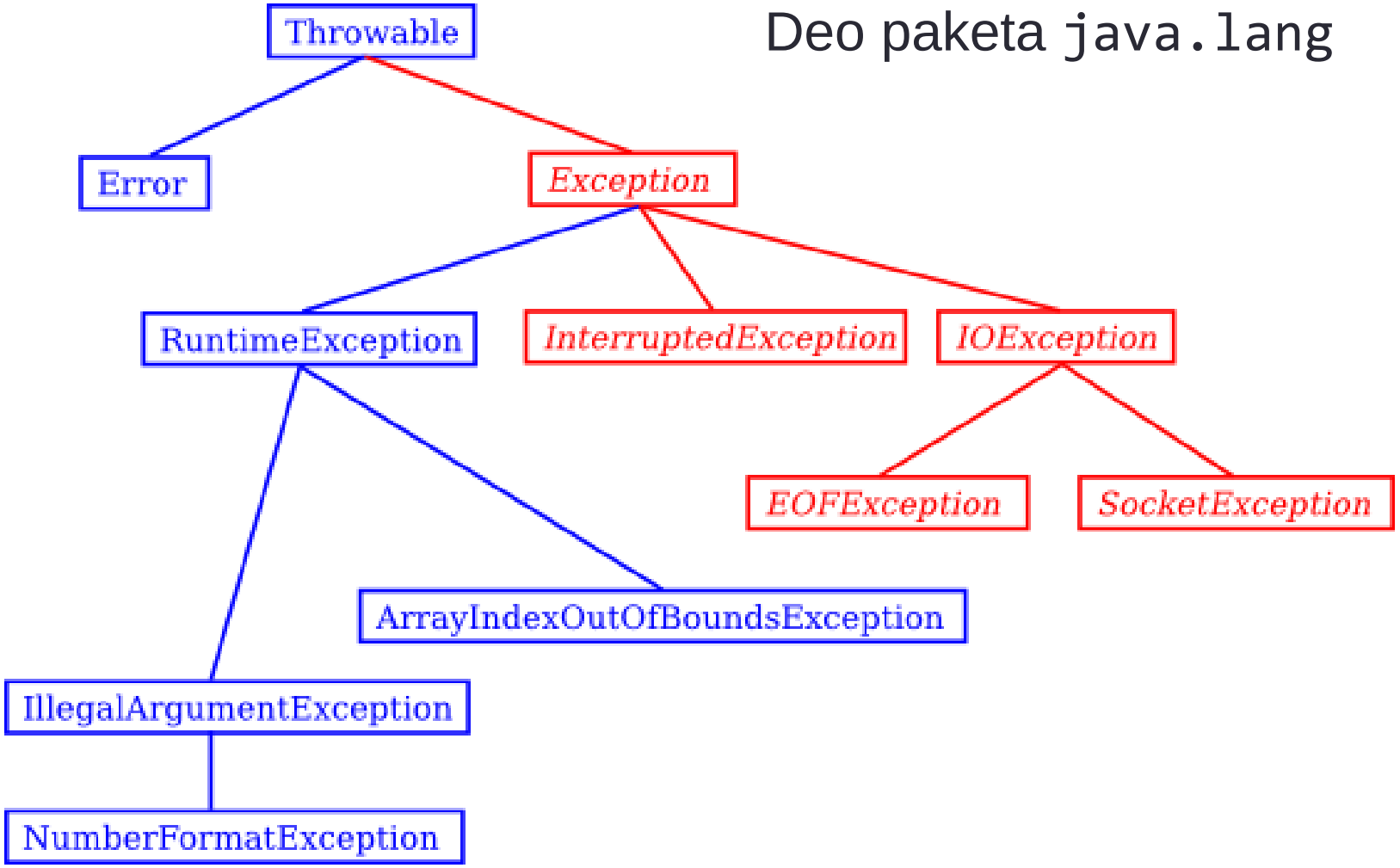
Postupanje sa greškama

- Uz pretpostavku da je kod sposoban da uoči i pronađe mesto gde se desila greška, tada se sa njom može postupati na nekoliko načina:

1. Ignorisanje greške – nije dobar način!
2. Provera pojave mogućih problema i prekid programa kada se pojavi problem
3. Provera pojave mogućih problema, hvatanje greške i pokušaj da se problem reši – pokušaj “oporavka”
4. Obrada greške pomoću izuzetaka – “bacanje” izuzetaka – poželjan način!

`Throwable` i neke njene podklase

Deo paketa `java.lang`



Obrada grešaka pomoću izuzetaka

- Izuzeci pružaju način za otkrivanje grešaka i njihovu obradu
- Za hvatanje i obradu izuzetaka postoji poseban blok
- Sintaksa bloka za hvatanje i obradu izuzetaka:

```
try {
        //ovde ide potencijalno problematican kod
} catch (Exception e) {
        //kod koji obradjuje izuzetak
}
```

- Ako je unutar `try` bloka generisan izuzetak, blok `catch` će ga obraditi.

Obrada grešaka pomoću izuzetaka

- Opšti oblik:

```
try {
            //problematican kod
} catch (Exception e) {
            //kod koji obradjuje izuzetak
            System.out.println(e.getMessage());
}
System.out.println("Izuzetak je obradjen!");
```

Primer 4.1 – hvatanje i obrada izuzetka

- Determinanta matrice jednaka je razlici proizvoda elemenata na glavnoj i sporednoj dijagonali

```
try {
        double determinanta = M[0][0]*M[1][1] - M[0][1]*M[1][0];
        ...
        System.out.println(“Determinanta M je " + determinanta);
}
catch (ArrayIndexOutOfBoundsException e) {
        System.out.println("M nije ispravne velicine!");
}
catch (NullPointerException e) {
        System.out.print("Programska greska! M ne postoji!");
}
```

- Ako se u `try` bloku generiše izuzetak, tada se on hvata u odgovarajućem `catch` bloku

Obrada grešaka pomoću izuzetaka

- Ako se izuzetak pojavi tokom izvršavanja bloka, dešava se sledeće:
  - Izvršavanje bloka se prekida
  - Proveravaju se uslovi iz klauzula `catch` (može ih biti više), da bi se utvrdilo da li za taj izuzetak postoji odgovarajući `catch` blok
  - Ako se nijedan od `catch` blokova ne tiče tog izuzetka, onda se on prosleđuje `try` bloku višeg nivoa – ako se izuzetak ne uhvati u kodu biće uhvaćen od strane sistema sa nepredvidivim ishodom!
  - Ako se pronađe odgovarajući `catch` blok, izvršavaju se naredbe iz tog bloka
  - Potom se nastavlja sa izvršavanjem programa počev od prve naredbe iza bloka `try`

Opšti oblik try – catch bloka

```
try {
 // kod koji može da izazove izuzetak
}catch(<tip1>  <ime1>){
 // kod kojim se obrađuje izuzetak 1. tipa
}
}catch(<tip2>  <ime1>){
 // kod kojim se obrađuje izuzetak 2. tipa
...[finally {blok}] // tipično ovde idu
                    // aktivnosti koje se uvek
                    // izvrsavaju, bez obzira
                    // da li se desio izuzetak
```

Čišćenje pomoću `finally`

- Često postoje delovi koda koje želimo da izvršimo bez obzira na to da li je prethodno nastao izuzetak ili ne
- To je obično slučaj u operacijama koje ne predstavljaju oporavak memorije (npr. uvek želimo da zatvorimo prethodno otvorenu datoteku)
- Dakle, u `finally` blok idu aktivnosti koje se uvek dešavaju
- Pošto Java ima sakupljač smeća, ovo naredba je neophodna kada u prvobitno stanje moramo da vratimo nešto što nije memorija – primeri: zatvaranje datoteke, raskid veze sa mrežom
- Blok finally posmatraćemo nešto kasnije u okviru primera za rad sa datotekama

Specifikacija izuzetka

- U Javi ste dužni da u specifikaciji metode navedete koje izuzetke ona može da generiše

- Za tu svrhu je rezervisana reč **`throws`**, npr.:

```
void f() throws ArithmeticException, IOException {
        //kod metode koja moze da izazove izuzetke
}
```

- Ako se u specifikaciji ne navede **`throws`**, podrazumeva se da metoda ne generiše izuzetke
- Hvatanje bilo kog tipa izuzetka vrši se pomoću osnovne klase `Exception` – **`catch`** `(Exception e) { …`

Generisanje izuzetka bez obrade

- Ima situacija kada je smisleno generisati izuzetak bez njegovog hvatanja i obrade
- Kada program otkrije neko stanje greške, ali nema razumnog načina za obradu greške, tada program može generisati izuzetak u nadi da će neki drugi deo programa da ga uhvati i obradi. U ove svrhe je rezervisan ključna reč **`throw`**
- Izuzeci mogu da se generišu pomoću uslova i **`if`** bloka u kome se, ako je ispunjen uslov, generiše izuzetak

Primer 4.2 – generisanje izuzetka

```
/**
 * Vraca veci od dva korena kvadratne jednacine
 * A*x*x + B*x + C = 0, ako ona ima korena.  Ako je A == 0 ili
 * je diskriminanta B*B - 4*A*C negativna onda se generise
 * izuzetak tipa IllegalArgumentException.
 */
static public double koren( double A, double B, double C )
                    throws IllegalArgumentException {
        if (A == 0) {
                throw new IllegalArgumentException("A ne moze biti nula!");
        }
        else {
                double disk = B*B - 4*A*C;
                if (disk < 0)
                        throw new IllegalArgumentException("Diskriminanta
                                                           manja od nule!");
                return  (-B + Math.sqrt(disk)) / (2*A);
        }
}
```

Uputstvo za korišćenje izuzetaka

- Izuzetke koristimo da:
  - Rešimo probleme i ponovo pozovemo metodu koja je prouzrokovala izuzetak
  - “Zakrpimo” grešku i nastavimo rad bez ponovnog isprobavanja metode
  - Pojednostavimo kod (ako način na koji generišemo izuzetke još više komplikuje kod, onda ga nije lako i poželjno koristiti)
  - Završimo program
  - Povećamo pouzdanost biblioteke i programa – kratkoročno ulaganje napora u pisanje koda za otklanjanje grešaka je dugoročna investicija u snagu i stabilnost aplikacije

Pretpostavke (engl. assertions)

- Pretpostavke (engl. assertions) obezbeđuju da je ispunjen neki preduslov kako bi se omogućilo dalje izvršavanje programa
- Koristi se rezervisana reč **`assert`**
- Oblici naredbe:

```
assert uslov;
assert uslov : poruka_o_gresci;
```

- Uključivanje pretpostavki u Eclipse-u: – Run As – Run Configurations... – Arguments tab – VM arguments – “-ea”
- Primer:

```
assert (fakt==1) : "Faktorijel nije inicijalizovan na 1!";
```

Primer 4.3 - pretpostavke

```
/**
 * Vraca veci od dva korena kvadratne jednacine
 * A*x*x + B*x + C = 0, ako ona ima korena.
 * Preduslovi: A != 0 i B*B – 4*A*C > 0
 */
static public double koren( double A, double B, double C )  {
          assert A != 0 : “Vodeci koeficijent kvadratne jednacine
                                ne sme biti nula!";
          double disk = B*B - 4*A*C;
          assert disk >= 0 : "Diskriminanta kvadratne jednacine
                                 ne sme biti negativna!";
          return  (-B + Math.sqrt(disk)) / (2*A);
       }
```

Beleške (engl. annotations)

- Beleške (engl. annotations) su metapodaci (podaci o podacima)
- Postaje od Jave 5 – beleške proverava kompajler kako bi osigurao da je kod u skladu sa namerama programera
- Primeri:
  - `@Override`
  - `@Deprecated`
  - `@SuppressWarnings`
- U kodu se koriste vrlo slično kao `static, final` i sl. – ako se napiše npr. `@Override` u definiciji nekog metoda, onda bi on trebalo da redefiniše istoimenu metodu iz neke nadklase – ako ovakva metoda ne postoji kompajler prijavljuje grešku!

RAD SA TOKOVIMA I DATOTEKAMA