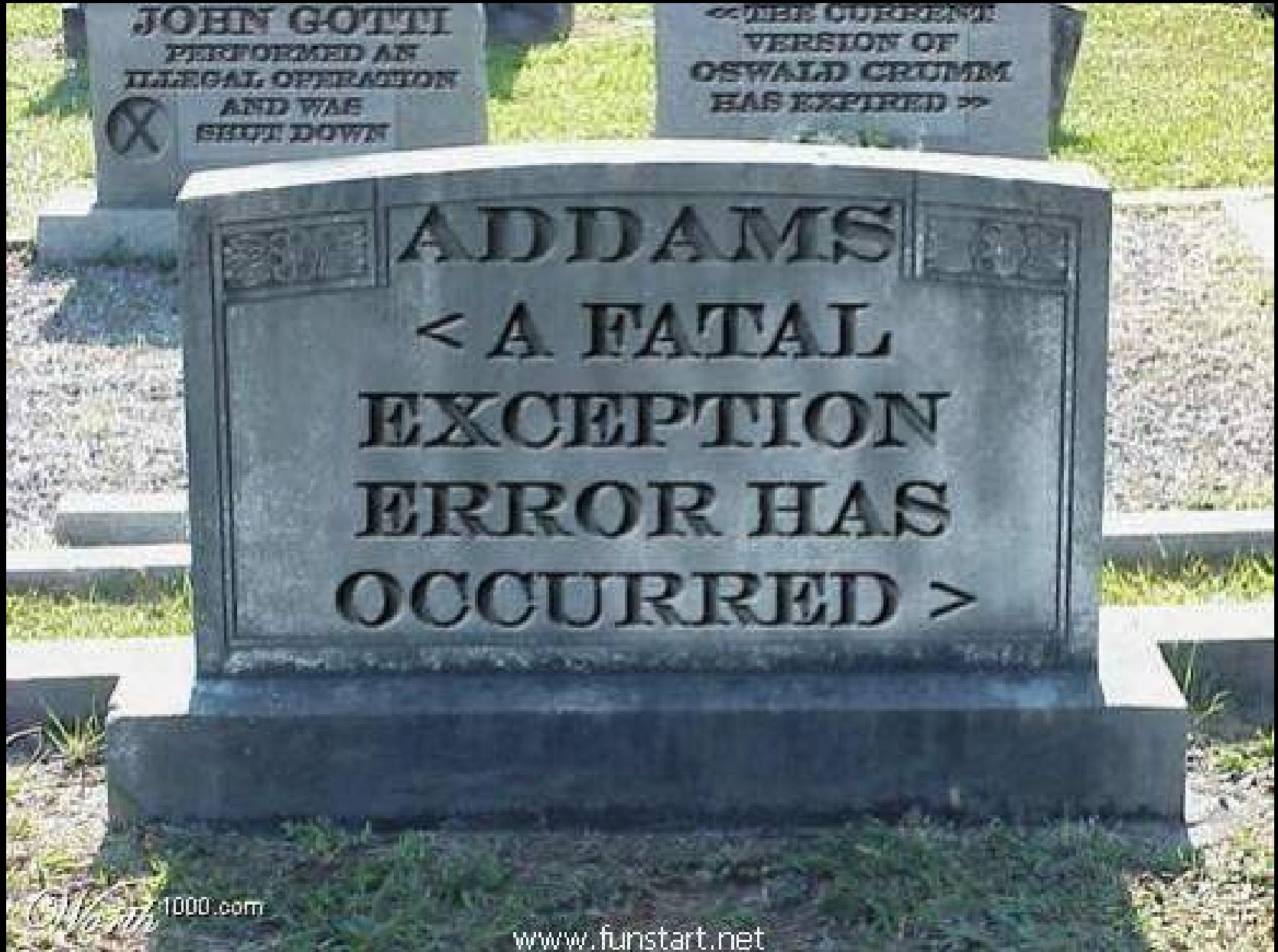


Obravnava izjem (exception handling)

Ko umre programer



Uvod

- ◆ Ne glede na to, kako dobro smo načrtovali nek program, je še vedno možnost, da lahko med izvajanjem pride do kakšnih napak.
- ◆ Dobro načrtovan program mora vsebovati kodo za obravnavo napak in drugih izjemnih pogojev.
- ◆ Ogleдали si bomo možnost obravnave izjem, ki jo nudi Java. Ta tvori objekt **Exception**, kadarkoli sreča nepričakovano napačno situacijo, kot na primer:
 - neobstoječ sistemski vir,
 - ne-numerični znak v številskem polju,
primer: `Integer.parseInt(br.readLine());`
 - manjkajočo datoteko ipd.,

Kaj je izjema

Izjema je nepričakovan dogodek (event), do katerega pride med izvajanjem programa in ki prekine normalen potek programskih ukazov .

Primeri:

- ◆ Poskus dostopa do elementa izven meja nekega polja,
- ◆ Poskus delitve z vrednostjo 0,
- ◆ Poskus dostopa do nekega URL z napačnim protokolom.

```
class test {  
    public static void main(String argv[ ]) {  
        int a=4;  
        int b=0;  
        System.out.println(a/b); // med izvajanjem dobimo ArithmeticException  
    }  
}
```

Razlika med napako in izjemo

- ◆ Naša koda lahko obravnava **izjemo** in program nadaljuje.
- ◆ Če pride do **napake**, mora program prekiniti izvajanje.

Primer

Psevdo koda funkcije, ki bere datoteko:

```
readFirstLine {  
    open the file;           // odpiranje lahko ne uspe  
    read the first line;    // branje lahko ne uspe  
    close the file;        // zapiranje lahko ne uspe  
}
```

Tradicionalna obravnava napak

```
readFirstLine {
    int errcode = 0;
    open the file;
    if (openError) {
        errcode = OPEN_ERR;
    }
    else {
        read the first line;
        if (readError) {
            errcode = READ_ERR;
        }
        close the file;
        if (closeError) {
            errcode = errcode
                and CLOSE_ERROR;
        }
    }
    return errcode; }
```

Javanska obravnava izjem

```
readFirstLine {
    try {
        open the file;
        read the first line;
        close the file;
    }
    catch (openError) {
        handle error;
    }
    catch (readError) {
        handle error;
    }
    catch (closeError) {
        handle error;
    }
}
```

Prednosti izjem v Javi (1)

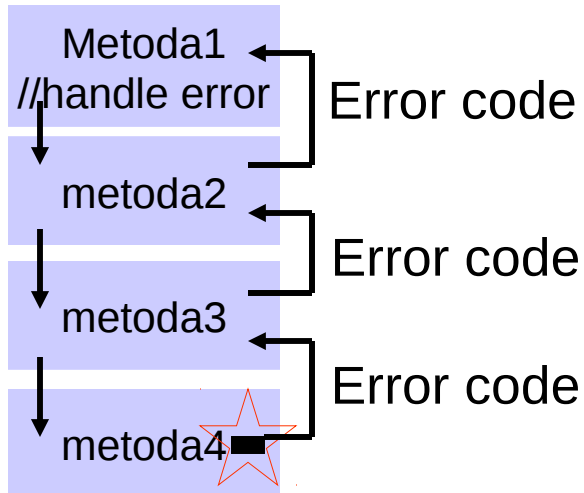
Ločena koda za obravnavo napak

- ◆ Pri tradicionalnem programiranju povzroča obravnavanje napak to, da je programska koda nejasna.
- ◆ Java loči podrobnosti obravnave nepričakovanih napak od glavne kode programa.
- ◆ Dobimo kodo, ki je jasna in je tako tudi možnost napak manjša .

Prednosti izjem v Javi (2)

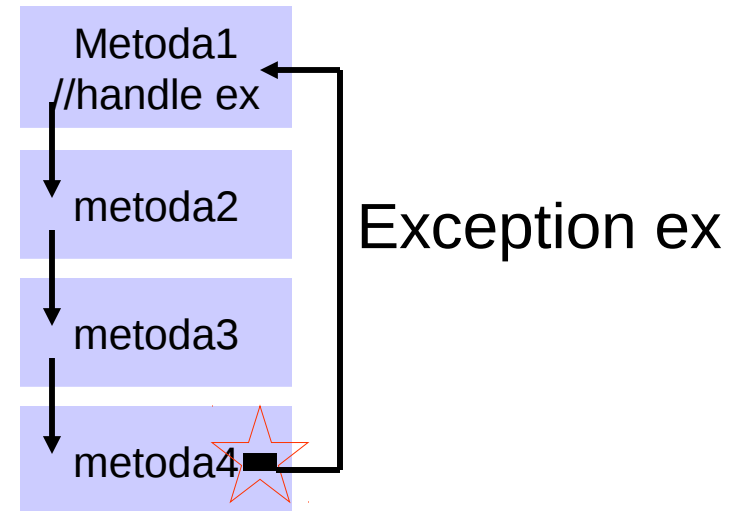
Posredovanje napak na klicni sklad

*Tradicionalna obravnava
napak*



*Vsaka metoda išče napake
in vrača kodo napake
kličeči metodi.*

Java exceptions

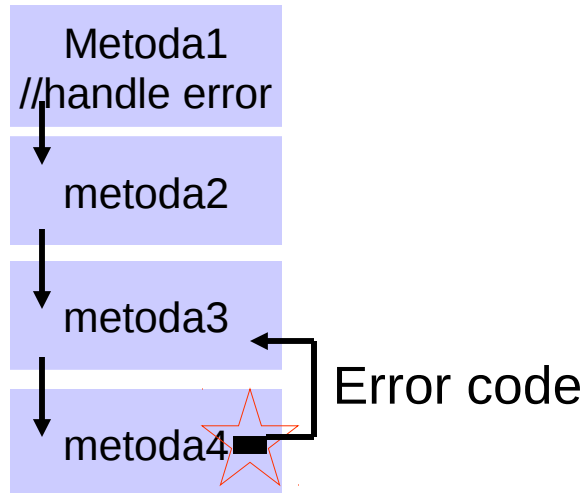


*metoda4 sproži izjemo
(exception); metoda1
jo ulovi.*

Prednosti izjem v Javi (3)

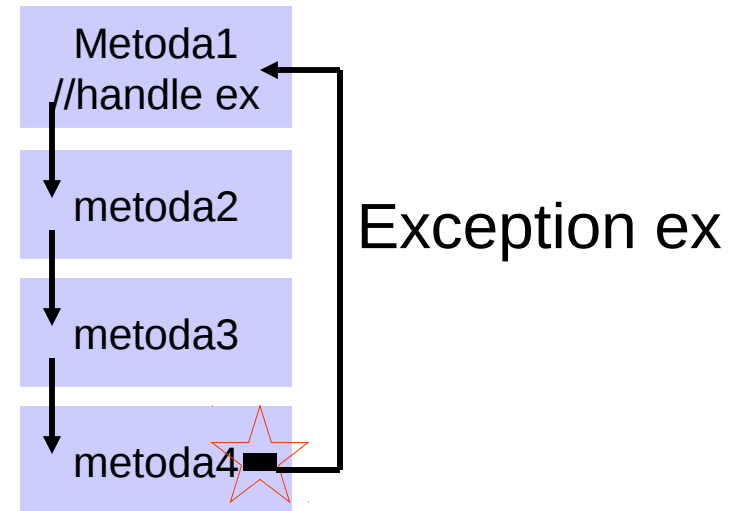
Izjeme lahko ignoriramo

*Tradicionalna obravnava
napak*



*Če metoda3 ignorira
napako, ne bo ta nikoli
obravnavana.*

Java exceptions



*Izjemo lahko ulovimo in
obravnavamo drugje*

Kaj narediti z izjemo?

Če kličemo metodo, ki lahko povzroči izjemo, imamo tri možnosti:

- 1) Izjemo ulovimo in jo obravnavamo.
- 2) Izjemo posredujemo naprej kličeči metodi.
- 3) Izjemo ulovimo in sprožimo drugačno izjemo. To, novo izjemo bomo morali obravnavati kje drugje.

Kako ujamemo in obravnavamo izjemo?

Klicana metoda bi lahko sprožila izjemo v bloku **try**.

Vsako izjemo obravnavamo v bloku **catch**.

Izvedemo končno obdelavo v bloku **finally**.

```
try {  
    // call the method  
}  
catch (exception1) {  
    // handle exception1  
}  
catch (exception2) {  
    // handle exception2  
} ...  
finally {  
    // any final processing  
}
```

Vrste izjem

Poznamo veliko vrst izjem, vse so izpeljane iz osnovnega razreda

Exception:

Exception

- ClassNotFoundException
- CloneNotSupportedException
- IllegalAccessException
- InstantiationException
- InterruptedException
- NoSuchFieldException
- NoSuchMethodException
- RuntimeException
- ArithmeticException
- ArrayStoreException
- ClassCastException
- IllegalArgumentException
- IllegalThreadStateException
- NumberFormatException
- IllegalMonitorStateException
- IllegalStateException
- IndexOutOfBoundsException
- ArrayIndexOutOfBoundsException
- StringIndexOutOfBoundsException
- NegativeArraySizeException
- NullPointerException
- SecurityException
- UnsupportedOperationException

Lovljenje izjem: try-catch

Java omogoča, da izjeme ulovimo. Če mislimo, da lahko v določenem delu kode pride do izjeme, jo postavimo v blok **try-catch**.

```
class test2 {  
    public static void main(String argv[ ]) {  
        int x=6; int y=3;  
        try { // Začetek bloka  
            for (int j=1;j<=10;j++) {  
                x=x*j/y;  
                y=y-1;  
            }  
            System.out.println("Rezultat je: " + y);  
        }  
        catch (Exception e) { // Ujamemo izjemo  
            System.out.println("Prislo je do napake!");  
        }  
    }  
}
```

Lovljenje izjem: try-catch

Če poznamo tipe izjem, ki se lahko zgodijo, lahko po bloku try postavimo **več catch blokov**:

```
try {
    for (int j=1;j<=10;j++) {
        x=x*j/y;
        y=y-1;
    }
    System.out.println("Rezultat je: " + y);
}
catch (java.lang.ArithmeticException e) {
    // Ujamemo aritm. izjemo
    System.out.println("Prislo je do aritmetične napake!");
}
catch (Exception e) {
    // Ujamemo ostale izjeme
    System.out.println("Prislo je do neke napake!");
}
```

Lovljenje izjem: try-catch-finally

Po bloku catch lahko postavimo tudi blok finally, ki se bo izvršil v vsakem primeru, ne glede na to ali je do izjeme prišlo ali ne. Izvedel se bo tudi, če pride do izjeme v bloku catch!!!

```
try {
    for (int j=1;j<=10;j++) {
        x=x*j/y;
        y=y-1;
    }
    System.out.println("Rezultat je: " + y);
}
catch (Exception e) {
    System.out.println("Prislo je do aritmetične napake!");
    y=1/0;
}
finally { // izvede se, ceprav je v catch prišlo do izjeme!
    System.out.println("To se izvede v vsakem primeru");
}
```


Stavek throws

Ni potrebno, da vsakič postavljamo try-catch bloke. Za posamezne metode lahko s stavkom **throws** povemo, da v metodi lahko pride do neke izjeme. Tako se bo izjema prenesla na metodo, ki je to metodo klicala. Vse skupaj lahko poljubno gnezdimo.

```
class racunaj {  
    // deli lahko povzroci izjemo!  
  
    static int deli(int x, int y) throws Exception {  
        return x/y;  
    }  
}  
  
class Test3 {  
    public static void main(String argv[ ]) {  
        int a=3; int b=0;  
        try {  
            racunaj.deli(a,b);  
        }  
        catch (Exception e) {  
            System.out.println("Izjema !!!");  
        }  
    }  
}
```



Izjema bo obravnavala
kličoča metoda

Stavek throw

Izjemo lahko povzročimo tudi sami s stavkom *throw*:

```
class racunaj4 { // deli lahko povzroci izjemo!  
    static int deli(int x, int y) throws Exception {  
        if (y==0) // sami sprozimo izjemo!  
            throw new Exception("Delitelj je enak nic");  
        else return x/y;  
    }  
}
```



```
class Test4 {  
    public static void main(String argv[ ]) {  
        int a=3; int b=0;  
        try {  
            racunaj4.deli(a,b);  
        }  
        catch (Exception e) {System.out.println(e.getMessage()); }  
    }  
}
```

Klicana metoda tvori izjemo