Izuzeci

Definicija

Definition: An exception is an event that occurs during the execution of a program that disrupts the normal flow of instructions. It is meant to be more general that the term “error”.

Many kinds of errors can cause exceptions

- User input errors. In addition to the inevitable typos, some users like to blaze their own trail instead of following directions. Suppose, for example, that a user asks to connect to a URL that is syntactically wrong. Your code should check the syntax, but suppose it does not. Then the network package will complain.
- Device errors. Hardware does not always do what you want it to. The printer may be turned off. A web page may be temporarily unavailable. Devices will often fail in the middle of a task. For example, a printer may run out of paper.
- Physical limitations. Disks can fill up; you can run out of available memory.
- Code errors. A method may not perform correctly. For example, it could deliver wrong answers or use other methods incorrectly. Computing an invalid array index, trying to find a nonexistent entry in a hash table, and trying to pop an empty stack are all examples of a code error.

Dealing with Errors - Ifs

One way is to use error-checking if statements:

```
if (i<0 || i>=A.length) {
... // Do something to handle the out-of-range index , i
}
else {
... // Process the ar ray element , A[i]
}
```

There are some problems with

- trying to anticipate all the possible problems can turn what would otherwise be a straightforward program into a messy tangle of if statements.
- It is difficult and sometimes impossible to anticipate all the possible things that might go wrong.
- It's not always clear what to do when an error is detected. You cannot make the decision in the method.

Dealing with Errors - Exceptions

- Exception handling works by transferring the execution of a program to an appropriate exception handler when an exception occurs.

For example, if you call a method that opens a file but the file cannot be opened, execution of that method will stop, and code that you wrote to deal with this situation will be run.

- This approach separates error detection and error handling code from normal code.
- We need a way to tell the JVM what code to execute when a certain exception happens. To do this, we use the `try` and `catch` keywords.

The `try` is used to define a block of code in which exceptions may occur. This block of code is called a guarded region (which really means "risky code goes here").

One or more catch clauses match a specific exception to a block of exception-handling code.

Ifs vs. Exceptions

- otvaranje, čitanje, pisanje i zatvaranje datoteke

```
GET A FILENAME
OPEN THE FILE
IF THERE IS NO ERROR OPENING THE FILE
        READ SOME DATA
        IF THERE IS NO ERROR READING THE DATA
                PROCESS THE DATA
                WRITE THE DATA
                IF THERE IS NO ERROR WRITING THE DATA
                        CLOSE THE FILE
                        IF THERE IS NO ERROR CLOSING FILE
                                RETURN
```

```
TRY TO DO THESE THINGS:
        GET A FILENAME
        OPEN THE FILE
        READ SOME DATA
        PROCESS THE DATA
        WRITE THE DATA
        CLOSE THE FILE
        RETURN
IF THERE WAS AN ERROR OPENING THE FILE THEN DO ...
IF THERE WAS AN ERROR READING THE DATA THEN DO ...
IF THERE WAS AN ERROR WRITING THE DATA THEN DO ...
IF THERE WAS AN ERROR CLOSING THE FILE THEN DO ...
```

```
try {
                // …
} catch (ExceptionType1 identif) {…}
catch (ExceptionType2 identif) {…}
finally  {…}
```

Handling the Exception

- When an error occurs within a Java method, the method creates an exception object that encapsulates the error information and hands it off to the runtime system.

The exception object contains information about the exception, including its type and the state of the program when the error occurred.

The runtime system is then responsible for finding some code to handle the error.

Note that the method exits immediately; it does not return its normal (or any) value.

Moreover, execution does not resume at the code that called the method; instead, the exception-handling mechanism begins its search for an exception handler that can deal with this particular error condition.

Creating an exception object and handing it to the runtime system is called throwing an exception.

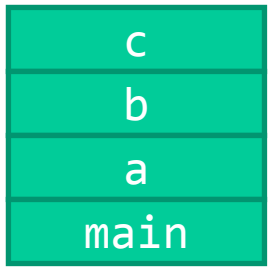
- After a method throws an exception, the runtime system leaps into action to find someone to handle the exception.

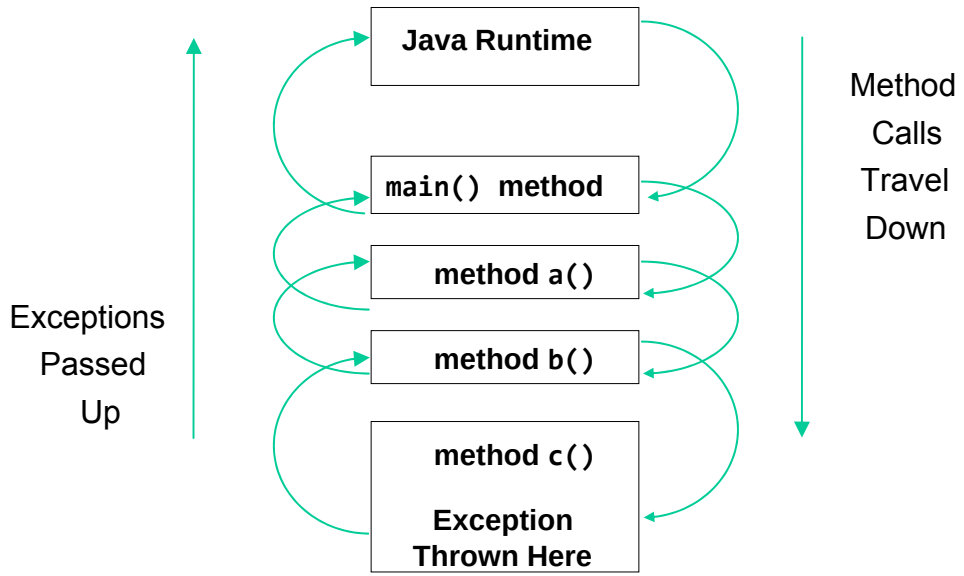
The set of possible "someones" to handle the exception is the set of methods in the call stack of the method where the error occurred. The system needs to find some method that chooses to handle the exception. i.e. some method with a catch statement.

Method stack or a call stack

- The call stack is the chain of methods that program executes to get to the current method.

If your program starts in method `main()` and `main()` calls method `a()`, which calls method `b()` that in turn calls method `c()`, the call stack consists of the following:





Finding an Exception Handler

- i.e. Find a catch statement that takes the type of this exception object as an input parameter:

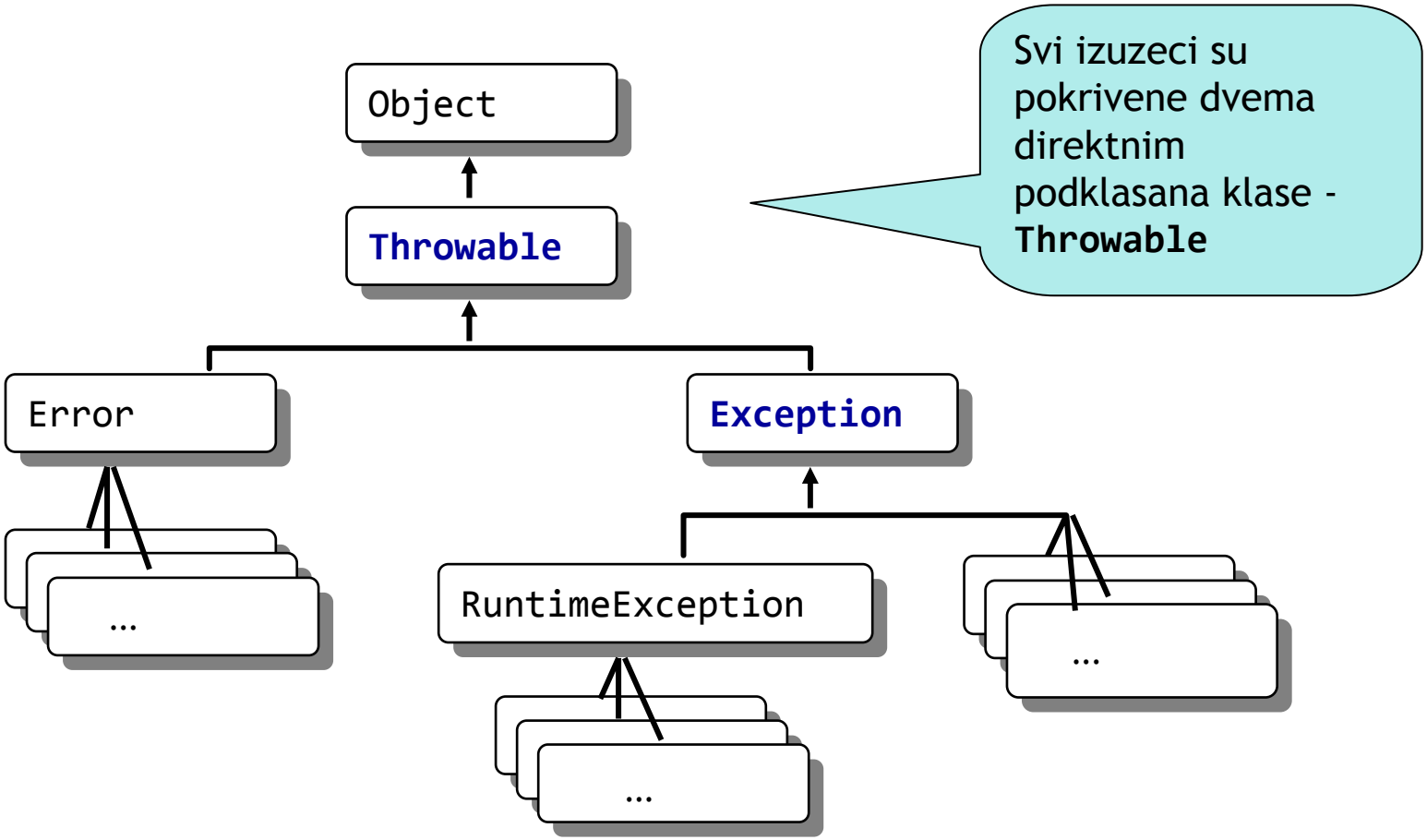
```
catch(IOException io)
```

The runtime system searches backwards through the call stack until it finds a method that contains an appropriate exception handler.

An exception handler is considered appropriate if the type of the exception thrown is the same as the type of exception handled by the handler.

If the runtime system exhaustively searches all of the methods on the call stack without finding an appropriate exception handler, the runtime system (and consequently the Java program) terminates.

Tipovi izuzetaka u Javi



Izuzeci tipa Error

- Predstavljaju izuzetke za koje se ne očekuje da ih hvata korisnik
- Tri direktne podklase klase Error:
  - ThreadDeath: baca se kada se namerno zaustavi nit u izvršavanju. Kada se ne uhvati završava nit, a ne program
  - LinkageError: ozbiljne greške unutar klasa u programu (nekompatibilnosti među klasama, pokušaj kreiranja objekta nepostojeće klase)
  - VirtualMachineError – JVM greška

Izuzeci tipa RunTimeException

- Za skoro sve izuzetke koji su obuhvaćeni podklasama klase Exception potrebno je uključiti kod za njihovu obradu ili program neće proći kompajliranje
- RunTimeException izuzeci se tretiraju drugačije. Prevodilac dozvoljava njihovo ignorisanje.

  Razlog: ovi izuzeci su najčešće pojavljuju pojavljuju kao posledica ozbiljnijih grešaka u kodu, čija obrada ne bi mogla ništa značajno promenila.

Neke njene podklase:

- `ArithmeticException`: neispravan rezultat aritmetičke operacije poput dijeljenja s nulom
- `IndexOutOfBoundsException`: indeks koji je izvan dozvoljenih granica za objekat poput niza, stringa ili vektora
- `NegativeArraySizeException`: upotreba negativnog indeksa niza
- `NullPointerException`: poziv metoda ili pristup podatku članu null objekta
- `ArrayStoreException`: pokušaj dodeljivana reference pogrešnog tipa elementu niza
- `ClassCastException`: pokušaj kastovanja objekta neodgovarajućeg tipa
- `SecurityException`: pokušaj narušavanja sigurnosnih pravila (security manager).

Šta sa izuzetkom koji metod može da baci?

1. **Obraditi**
2. **Proslediti dalje**

Proslediti dalje

- Ako se izuzetak ne hvata i rešava unutar metode, mora se deklarisati da metod može baciti izuzetak:

```
double mymethod() throws EOFException, FileNotFoundException
    {
        // kod (koristi java.io package klase)...
      // metoda ne rukuje s izuzecima koje mogu biti bačeni
    }
```

Obraditi

try/catch/finally

Obrada izuzetaka

```
try
{
        // ...
}
catch (ExceptionType1  identifier)
{
        // ...
}
catch (ExceptionType2 identifier) {
        // ...
}
finally
{
        // ...
 }
```

"prostor" u kome se hvataju izuzeci (svi, i tipa Error ili RunTimeException)

Kod koji obrađuje izuzetak. Poziva se za ExceptionType1 ili bilo koju od podklasu (sve podklase bazne klase Throwable)

Uviek se izvršava bez obzira na prethodne blokove

Višestruku `catch` blokovi

- Ako imate `catch` blokove s nekoliko tipova izuzetaka u istoj klasnoj hijearhiji, potrebno je blokove **postaviti tako da se prvo hvata izuzetak najniže podklase pa redom prema najvišoj superklasi**.

```
// neispravna sekvenca catch blokova
// neće se prevesti
try {
          // try block code
} catch(Exception e){ ... }
  catch(ArithmeticException e){ ... }
```

finally blok

- Koristi se za pospremanje (clean-up)
- Asociran s određenim try blokom (kao i catch blok)
- Može se koristiti i s try blokom koji sadrži kod koji ne baca nikakav izuzetak:
  - za kod s višestrukim break ili return elementima
  - vrednosti koje vratimo s return u finally bloku će pregaziti bilo koji return izvršen u try bloku

Ponovno bacanje izuzetka

- U slučaju da je onom koji je pozvao metod potrebna informacija o izuzetku koji je metod već obradio

```
try {
    // Code that originates an arithmetic exception
}
catch (ArithmeticException e) {
  // Deal with the exception here
  throw e;   // Rethrow the exception to the calling program
}
```

Exception objekti

- Klasa `Throwable` je bazna klasa za sve Java izuzetke i ima dva public konstruktora
  - default kontruktor
  - kontruktor koji prihvata argument tipa String (tu može biti smeštena informacija o prirodi greške)
- Exception objekat koji se prosleđuje catch bloku sadrži informacije o prirodi problema
  - Poruku (message) (koja se inicijalizuje u konstruktoru)
  - Zapis o call steku (execution stack) u trenutku kada je izuzetak kreiran (pun naziv svih pozvanih metoda, plus broj linije u kojoj se poziv dogodio)
- Throwable klasa ima public metode koje omogućavaju pristup zapisu poruka i steka:
  - getMessage() - Vraća sadržaj poruke (null za većinu predefinisanih klasa)
  - printStackTrace() - Ispis poruka i steka na standardni izlaz
  - printStackTrace(PrintStream s)
  - fillInStackTrace() - ažurira zapis steka za mesto gdje se poziv iste obavlja; koristiti se kada se ponovo baca izuzetak da bi se ažurirao zapis steka kada je ponovo bačena (korisno kod bacanja sopstvenih izuzetaka)

```
e.fillInStackTrace();
throw e;
```

Definisanje novih izuzetaka

- Dva osnovna razloga:
  - Dodavanje informacije kada se dogodi neka standardni izuzetak
  - Greška koja se događa u vašem kodu zaslužna novog tipa izuzetka
- Ako se sadržaj catch bloka često izvršava razmislite da li se sve moglo izvesti sa if-then-else
- najbolja praksa: izvesti ih iz Exception cklase -> prevodilac će pratiti gde je moguće bacanje i da li su hvatane ili prosleđene
- moguća je i upotrebaRunTimeException ili neke od njenih podklasa -> bez provere od strane prevodioca

Definisanje novih izuzetaka

```
// exception class – minimalna definicija
public class DreadfulProblemException extends Exception
    {
  // Konstruktori
    public DreadfulProblemException(){  }  // Default constructor
    public DreadfulProblemException(String s)  {
           super(s); // Call the base class constructor
      }
}
```

- Što još dodati
  - druge konstruktore
  - varijable instance za čuvanje dodatnih informacija
  - pristupne metode za varijable instance s dodatnim informacijama

Bacanje novog izuzetaka

I

```
DreadfulProblemException e = new DreadfulProblemException();
throw e;
```

II

```
DreadfulProblemException e = new DreadfulProblemException("Uh-Oh");
throw e;
// getMessage() vraća: "DreadfulProblemException:Uh-Oh"
```

III

```
throw new DreadfulProblemException("Double trouble!");
```

Zajedno

```
void method1(...)
{
  try
  {
    int x= method2(...);
  }
  catch( Exception1 e)
  {
    //handle exception
  }
  catch( Exception2 e)
  {
    //handle exception
  }

}
```

```
int method2(...) throws Exception1,
Exception2
{
  try
  {
    // try block code that may
    // throw ArithmeticException
  }
  catch(ArithmeticException e)
  {
    // Analysis code
    if(...) throw Exception1;
    else throw Exception2;
  }
}
```

Da ne tražite - `throws`

- 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. Klase koje su navedene u tabeli Javinih predefinisanih izuzetaka nasleđuju klasu RuntimeException pa pripadaju grupi neproveravanih izuzetaka.

    Ako metoda baca neki neproveravani izuzetak, poziv te metode može, ali ne mora biti uokviren “try-catch” blokom koji hvata taj izuzetak.
  - dok su sve ostale klase iz grupe proveravanih izuzetaka (npr. Exception, IOException...).

    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.
- Poenta: ako za neki metod navedete da baca (`throws`) određene izuzetke, onda će kompajler naterati korisnika tog metoda da navedene izuzetke i obradi pri pozivu, tj. da uokviri poziv `try` blokom i obradi `catch` bloku (videti primer na sledećem slajdu)

Da ne tražite - throws

```
public class TestIzuz {
    public static void main(String args[]) {
      KK k=new KK();
      k.gr();
    }
}
public class KK {
public void gr() throws Izuz{ int i=2/0; }
}
public class Izuz extends Exception{
    Izuz() {super("nemas pojma");}
}
```

Primer je namerno napravljen pomalo nerezonski.

1. za gr je rečeno da diže `Izuz` i obzirom da je checked, to znači da će morati u main-u da bude obrađen
2. gr() ovako definisan će uzrokovati `ArithmeticException` koji spada u unchecked pa ne mora biti proveren – bar što se kompajlera tiče.

Ispravite kod da prodje kompajler, a onda i da stvarno reguliše deljenje nulom.

Još ovo

- Legalno je baciti izuzetke koji su izvedeni iz izuzetaka navedenih u throws klauzuli – razlog: klasa se može koristiti polimorfno gde god se njena superklasa očekuje
- Bacanje nedeklarisanog tipa izuzetka je pogrešno:
  - bilo direktno koristeći throw ili indirektno pozivajući drugi metod
  - otkriva prevodilac i javlja grešku
- Java je **striktna u forsiranju proverenih izuzetaka** – dakle, ako se pozove metod koji navodi izuzetak u njegovoj throws klauzuli postoje 3 mogućnosti:
  - uhvatiti i obraditi izuzetak
  - uhvatiti izuzetak i preslikati ga u neki drugi izuzetak bacanjem tog drugog izuzetka
  - deklarisati izuzetak u vlastitoj throws klauzuli i proslediti izuzetak dalje (na pozivajući nivo)