

ΤΕΧΝΙΚΕΣ ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΑΦΟΥΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

Εξαιρέσεις

Εξαιρέσεις

- Στα προγράμματα μας θα πρέπει να μπορούμε να χειριστούμε περιπτώσεις που το πρόγραμμα **δεν** εξελίσσεται όπως το είχαμε προβλέψει
 - Π.χ., κάνουμε μια διαίρεση και ο παρανομαστής είναι μηδέν
 - Θέλουμε να διαβάσουμε ένα ακέραιο, αλλά η είσοδος είναι ένα String
 - Θέλουμε να διαβάσουμε από ένα αρχείο αλλά δώσαμε λάθος το όνομα.
- Για τη διαχείριση τέτοιων εξαιρετικών περιπτώσεων υπάρχουν οι **Εξαιρέσεις (Exceptions)**
 - Οι εξαιρέσεις μας επιτρέπουν να εντοπίσουμε το πρόβλημα σε ένα σημείο (**throw an Exception**) και να το χειριστούμε σε κάποιο άλλο σημείο (**handle the Exception**)
 - Οι εξαιρέσεις είναι ένα αρκετά προχωρημένο προγραμματιστικό εργαλείο.
 - Ακόμη κι αν δεν τις χρησιμοποιήσετε, εμφανίζονται σε διάφορες βιβλιοθήκες της Java, οπότε θα πρέπει να ξέρετε να τις χειρίζεστε

Ένα απλό παράδειγμα

- Ένα πρόγραμμα σχολής χορού ταιριάζει χορευτές με χορεύτριες
 - Αν οι άνδρες είναι περισσότεροι από τις γυναίκες τότε ο καθένας θα χορέψει με πάνω από μία γυναίκα
 - Αν οι γυναίκες είναι παραπάνω από τους άνδρες τότε η κάθε μία θα χορέψει με παραπάνω από έναν άνδρα.
 - Αν είναι μισοί μισοί, τότε ταιριάζονται ένας προς ένα.
- Τι γίνεται αν δεν υπάρχουν άνδρες, ή γυναίκες, ή καθόλου μαθητές?
 - Αυτό είναι μια ειδική περίπτωση για την οποία δημιουργούμε μια εξαίρεση.

Υλοποίηση χωρίς εξαιρέσεις

```
import java.util.Scanner;

public class DanceLesson
{
    public static void main(String[] args)
    {
        Scanner keyboard = new Scanner(System.in);
        System.out.println("Enter number of male and female dancers:");
        int men = keyboard.nextInt();
        int women = keyboard.nextInt();

        if (men == 0 && women == 0){
            System.out.println("Lesson is canceled. No students.");
            System.exit(0);
        }else if (men == 0){
            System.out.println("Lesson is canceled. No men.");
            System.exit(0);
        }else if (women == 0){
            System.out.println("Lesson is canceled. No women.");
            System.exit(0);
        }

        if (women >= men)
            System.out.println("Each man must dance with " +
                                women/(double)men + " women.");
        else
            System.out.println("Each woman must dance with " +
                                men/(double)women + " men.");
        System.out.println("Begin the lesson.");
    }
}
```

Μηχανισμός try-throw-catch

- Ο κώδικας που μπορεί να δημιουργήσει εξαίρεση μπαίνει σε ένα **try-block**
- Αν η εξέλιξη του κώδικα είναι προβληματική εκτελείται η εντολή **throw** η οποία «πετάει» την εξαίρεση.
- Το πέταγμα της εξαίρεσης μπορεί να γίνεται και από κάποια μέθοδο που καλείται μέσα στο **try block**
- Αν υπάρξει εξαίρεση η ροή του κώδικα μεταφέρεται στο **catch-block** το οποίο χειρίζεται τις εξαιρέσεις

```
try
{
  <Κώδικας πριν>

  <Κώδικας ο οποίος μπορεί να κάνει throw exception>

  <Κώδικας μετά>
}
catch (Exception e)
{
  <Κώδικας που χειρίζεται την εξαίρεση>
  <Χρησιμοποιεί το αντικείμενο e>
}
```

To try block

- Σύνταξη

```
try
{
    <Κώδικας που μπορεί να προκαλέσει εξαίρεση>
}
```

- Το **try block** είναι ένα **block** όπως όλα τα άλλα στην Java
 - Ότι μεταβλητή ορίζεται μέσα στο block είναι τοπική, κλπ...

Η εντολή throw

- Σύνταξη

```
throw <Αντικείμενο της κλάσης Exception (ή παράγωγης)>
```

- Η εντολή **throw** λειτουργεί ως τελεστής, και ακολουθείται από ένα αντικείμενο τύπου **Exception**, ή **παράγωγης κλάσης** της **Exception**
 - Αυτή είναι η εξαίρεση που **πετάει** ο κώδικας.
- Όταν πεταχτεί η εξαίρεση (π.χ., όταν κληθεί η **throw**) **βγαίνουμε αυτόματα εκτός** του **try block** και ο έλεγχος του προγράμματος μεταφέρεται στο αντίστοιχο **catch block**
 - Λειτουργεί αντίστοιχα με την **break** σε **switch block**.

Η κλάση Exception

- Η κλάση Exception κρατάει πληροφορίες για την εξαίρεση που δημιουργήθηκε
 - Π.χ., όταν καλούμε τον constructor `new Exception("No students. No Lesson");`
- Στο private πεδίο `message` της κλάσης Exception αποθηκεύεται το μήνυμα που δίνουμε ως όρισμα.
- Μπορούμε να δημιουργήσουμε **παράγωγες κλάσεις** της Exception και να δημιουργήσουμε **επιπλέον πεδία** για να κρατάμε περισσότερες πληροφορίες για κάποια εξαίρεση.

Το catch block

- Σύνταξη

```
catch (Exception e)
{
    <Κώδικας που χειρίζεται την εξαίρεση>
}
```

- Η παράμετρος `Exception e` δηλώνει τον **τύπο της εξαίρεσης** που χειρίζεται το block και τη μεταβλητή `e` της εξαίρεσης.
- Χρησιμοποιώντας τη μεταβλητή μπορούμε να έχουμε πρόσβαση στα **πεδία** της εξαίρεσης
 - Παράδειγμα

```
catch (Exception e)
{
    String message = e.getMessage();
    System.out.println(message);
    System.exit(0);
}
```

Επιστρέφει το String του message

Try-throw-catch

- Σύνταξη

```
try
{
    <Κώδικας πριν>
    <Κώδικας ο οποίος μπορεί να κάνει throw exception>
    <Κώδικας μετά>
}
catch (Exception e)
{
    <Κώδικας που χειρίζεται την εξαίρεση>
}
```

- Μπαίνοντας στο try block, εκτελείται ο κώδικας πριν.
- Αν υπάρχει εξαίρεση η ροή μεταφέρεται στο catch block
- Αν δεν υπάρχει εξαίρεση εκτελείται ο κώδικας μετά. Ο κώδικας του catch block δεν εκτελείται ποτέ.

Υλοποίηση με εξαιρέσεις

```
import java.util.Scanner;

public class DanceLesson2
{
    public static void main(String[] args)
    {
        Scanner keyboard = new Scanner(System.in);
        System.out.println("Enter number of male and female dancers:");
        int men = keyboard.nextInt();
        int women = keyboard.nextInt();

        try{
            if (men == 0 && women == 0)
                throw new Exception("Lesson is canceled. No students.");
            else if (men == 0)
                throw new Exception("Lesson is canceled. No men.");
            else if (women == 0)
                throw new Exception("Lesson is canceled. No women.");

            if (women >= men)
                System.out.println("Each man must dance with " +
                                    women/(double)men + " women.");
            else
                System.out.println("Each woman must dance with " +
                                    men/(double)women + " men.");
        }
        catch(Exception e){
            String message = e.getMessage( );
            System.out.println(message);
            System.exit(0);
        }
        System.out.println("Begin the lesson.");
    }
}
```

Σημείωση: Το παράδειγμα είναι ενδεικτικό. Στην πράξη ποτέ δεν θα χρησιμοποιούσατε εξαιρέσεις με αυτόν τον τρόπο και για ένα τόσο απλό πρόβλημα.

Εξειδικευμένες εξαιρέσεις

- Η κλάση Exception είναι η πιο γενική κλάση εξαίρεσης. Υπάρχουν και πιο **εξειδικευμένες κλάσεις εξαιρέσεων** που **κληρονομούν** από την Exception σε διάφορα πακέτα της Java. Π.χ.
 - FileNotFoundException
 - IOException
- Μπορούμε επίσης να ορίσουμε και **δικές μας κλάσεις εξαιρέσεων** ανάλογα με τις ανάγκες μας.
- Αυτό είναι χρήσιμο ώστε να έχουμε και **εξειδικευμένα catch blocks** όπως θα δούμε αργότερα.

Παράδειγμα

- Θέλουμε να ορίσουμε μια εξαίρεση για την περίπτωση που προσπαθούμε να διαιρέσουμε με το μηδέν
 - Η κλάση `DivisionByZeroException`
- Η κλάση μας θα κληρονομεί από την `Exception` οπότε θα έχει την μέθοδο `getMessage()` για να επιστρέφει το μήνυμα
 - Συνήθως το μόνο που χρειάζεται είναι να ορίσουμε τον constructor.

Παράδειγμα

```
public class DivisionByZeroException extends Exception
{
    public DivisionByZeroException( )
    {
        super("Division by Zero!");
    }

    public DivisionByZeroException(String message)
    {
        super(message);
    }
}
```

Η κλάση κληρονομεί και την μέθοδο `getMessage()`

```
import java.util.Scanner;

public class DivisionDemoFirstVersion
{
    public static void main(String[] args)
    {
        try
        {
            Scanner keyboard = new Scanner(System.in);

            System.out.println("Enter numerator:");
            int numerator = keyboard.nextInt();
            System.out.println("Enter denominator:");
            int denominator = keyboard.nextInt();

            if (denominator == 0)
                throw new DivisionByZeroException( );

            double quotient = numerator/(double)denominator;
            System.out.println(numerator + "/"
                               + denominator
                               + " = " + quotient);
        }
        catch(DivisionByZeroException e)
        {
            System.out.println(e.getMessage( ));
            system.Exit(0);
        }

        System.out.println("End of program.");
    }
}
```

```

import java.util.Scanner;

public class DivisionDemoFirstVersion
{
    public static void main(String[] args)
    {
        try
        {
            Scanner keyboard = new Scanner(System.in);

            System.out.println("Enter numerator:");
            int numerator = keyboard.nextInt();
            System.out.println("Enter denominator:");
            int denominator = keyboard.nextInt();

            if (denominator == 0)
                throw new DivisionByZeroException( );

            double quotient = numerator/(double)denominator;
            System.out.println(numerator + "/"
                               + denominator
                               + " = " + quotient);
        }
        catch(DivisionByZeroException e)
        {
            System.out.println(e.getMessage( ));
            secondChance( );
        }

        System.out.println("End of program.");
    }
}

```

Μπορούμε μέσα στο catch block να καλούμε μία άλλη μέθοδο


```
public static void secondChance( )
{
    Scanner keyboard = new Scanner(System.in);

    System.out.println("Try again:");
    System.out.println("Enter numerator:");
    int numerator = keyboard.nextInt();
    System.out.println("Enter denominator:");
    System.out.println("Be sure the denominator is not zero.");
    int denominator = keyboard.nextInt();

    if (denominator == 0)
    {
        System.out.println("I cannot do division by zero.");
        System.out.println("Aborting program.");
        System.exit(0);
    }

    double quotient = ((double)numerator)/denominator;
    System.out.println(numerator + "/"
        + denominator
        + " = " + quotient);
}
}
```

Ορίζοντας Exceptions

- Ορίζουμε μια νέα εξαίρεση μόνο αν υπάρχει **ανάγκη**, αλλιώς μπορούμε να χρησιμοποιήσουμε την κλάση `Exception`.
- Στη νέα κλάση ορίζουμε πάντα ένα **constructor χωρίς ορίσματα** και έναν που παίρνει το **String του μηνύματος**.
- Διατηρούμε την μέθοδο **`getMessage()`** ως έχει
 - Συνήθως δεν θα χρειαστούμε κάποια άλλη μέθοδο.

Εξαιρέσεις με επιπλέον πληροφορία

- Μια εξαίρεση συνήθως έχει ένα μήνυμα σε μορφή String. Μπορεί να έχει και **επιπλέον πληροφορία** η οποία αποθηκεύεται σε **πεδία της μεθόδου**.
- Παράδειγμα: Ζητάμε το έτος γέννησης και θέλουμε να πετάμε μια εξαίρεση αν είναι μεγαλύτερο από 2014.
 - Θα ορίσουμε το **BadNumberException**
 - Η εξαίρεση θα πρέπει να μεταφέρει **πληροφορία** για τον **αριθμό** που δόθηκε.

```
public class BadNumberException extends Exception
{
    private int badNumber;

    public BadNumberException(int number)
    {
        super("BadNumberException");
        badNumber = number;
    }

    public BadNumberException( )
    {
        super("BadNumberException");
    }

    public BadNumberException(String message)
    {
        super(message);
    }

    public int getBadNumber( )
    {
        return badNumber;
    }
}
```

```
import java.util.Scanner;

public class BadNumberExceptionDemo
{
    public static void main(String[] args)
    {
        try
        {
            Scanner keyboard = new Scanner(System.in);

            System.out.println("Enter year of birth:");
            int inputNumber = keyboard.nextInt();

            if (inputNumber > 2014)
                throw new BadNumberException(inputNumber);

            System.out.println("Thank you for entering " + inputNumber);
        }
        catch (BadNumberException e)
        {
            System.out.println(e.getBadNumber( ) + " is not valid.");
        }

        System.out.println("End of program.");
    }
}
```

Μας επιστρέφει τον αριθμό που προκάλεσε την εξαίρεση

Πολλαπλά catch blocks

- Εφόσον έχουμε πολλαπλά είδη εξαιρέσεων είναι δυνατόν ένα `try block` να **πετάει** παραπάνω από ένα τύπο **εξαίρεσης**.
- Στην περίπτωση αυτή χρειαζόμαστε και **διαφορετικά catch blocks**.

```
public class NegativeNumberException extends Exception
{
    public NegativeNumberException( )
    {
        super("Negative Number Exception!");
    }

    public NegativeNumberException(String message)
    {
        super(message) ;
    }
}
```

```
try
{
    System.out.println("How many pencils do you have?");
    int pencils = keyboard.nextInt();

    if (pencils < 0)
        throw new NegativeNumberException("pencils");

    System.out.println("How many erasers do you have?");
    int erasers = keyboard.nextInt();
    double pencilsPerEraser;

    if (erasers < 0)
        throw new NegativeNumberException("erasers");
    else if (erasers != 0)
        pencilsPerEraser = pencils/(double)erasers;
    else
        throw new DivisionByZeroException( );

    System.out.println("Each eraser must last through "
        + pencilsPerEraser + " pencils.");
}

catch(NegativeNumberException e)
{
    System.out.println("Cannot have a negative number of " + e.getMessage( ));
}
catch(DivisionByZeroException e)
{
    System.out.println("Do not make any mistakes.");
}
```


Προσοχή

- Όταν πεταχτεί μια εξαίρεση και βγούμε από ένα try block, τα **catch blocks** εξετάζονται με την σειρά που εμφανίζονται στον κώδικα.
- Θα εκτελεστεί το **πρώτο** catch block με όρισμα που ταιριάζει στο **exception** που έχει πεταχτεί.
- Για να είμαστε σίγουροι ότι θα εκτελεστεί το σωστό catch block θα πρέπει να έχουμε τις **πιο συγκεκριμένες εξαιρέσεις πρώτες** και τις **πιο γενικές μετά**.
 - Αν είναι ανάποδα, οι πιο συγκεκριμένες εξαιρέσεις δεν θα εκτελεστούν **ποτέ**.
 - Ο compiler μπορεί να σας βγάλει μήνυμα λάθους αν έχετε ήδη πιάσει μια εξαίρεση.

```
import java.util.Scanner;
```

```
public class BadNumberExceptionDemo2
{
    public static void main(String[] args)
    {
        try
        {
            Scanner keyboard = new Scanner(System.in);

            System.out.println("Enter year of birth:");
            int inputNumber = keyboard.nextInt();
            if (inputNumber <=1973)
                throw new Exception("You are too old");
            if (inputNumber > 2015)
                throw new BadNumberException(inputNumber);

            System.out.println("Thank you for entering " + inputNumber);
        }
        catch(Exception e) {
            System.out.println(e.getMessage());
        }
        catch(BadNumberException e) {
            System.out.println(e.getBadNumber( ) +" is not valid.");
        }

        System.out.println("End of program.");
    }
}
```

To Exception είναι πιο γενικό από το BadNumberException

Ο compiler θα μας χτυπήσει λάθος γιατί δεν γίνεται ποτέ να πάμε στο δεύτερο catch block

Μέθοδοι που πετάνε εξαιρέσεις

- Μέχρι τώρα είδαμε παραδείγματα όπου οι εξαιρέσεις πετιόνται και πιάνονται στον ίδιο κώδικα.
 - Αυτό δεν είναι και τόσο ρεαλιστικό σενάριο
- Το πιο σύνηθες είναι ότι την **εξαίρεση** την πετάμε σε μια μέθοδο και την **πιάνουμε** σε μία άλλη.

Μέθοδος που πετάει εξαίρεση

- Σύνταξη

```
ReturnType methodName(argument list) throws Exception  
{  
    <Κώδικας πριν>  
    <Κώδικας ο οποίος κάνει throw Exception>  
    <Κώδικας μετά>  
}
```

- Αν η μέθοδος πετάξει μια εξαίρεση τότε **σταματάει** η εκτέλεση του κώδικα **στο σημείο που πετάει την εκτέλεση**.
 - Με τον ίδιο τρόπο όπως η εντολή return

Μέθοδος που πετάει εξαίρεση

- Μία μέθοδος μπορεί να πετάει πολλές εξαιρέσεις
- Σύνταξη:

```
ReturnType methodName(argument list)
    throws Exception1, Exception2
{
    <Κώδικας πριν>
    <Κώδικας ο οποίος κάνει throw Exception1>
    <Κώδικας μετά>
    <Κώδικας ο οποίος κάνει throw Exception2>
    <Κώδικας μετά>
}
```

```

import java.util.Scanner;

public class DivisionDemoSecondVersion
{
    public static void main(String[] args)
    {
        Scanner keyboard = new Scanner(System.in);

        try
        {
            System.out.println("Enter numerator and denominator :");
            int numerator = keyboard.nextInt(), int denominator = keyboard.nextInt();

            double quotient = safeDivide(numerator, denominator);
            System.out.println(numerator + "/" + denominator + " = " + quotient);
        }
        catch (DivisionByZeroException e)
        {
            System.out.println(e.getMessage ( ));
            secondChance ( );
        }

        System.out.println("End of program.");
    }

    public static double safeDivide(int top, int bottom) throws DivisionByZeroException
    {
        if (bottom == 0)
            throw new DivisionByZeroException ( );

        return top/(double)bottom;
    }
}

```

Εφόσον έχουμε μία μέθοδο που πετάει εξαίρεση, **πρέπει** να τη βάλουμε μέσα σε try-catch block

Η εξαίρεση δημιουργείται στην **safeDivide** αλλά την πιάνουμε και την χειριζόμαστε στην main

Catch or Declare

- Μια μέθοδος η οποία **καλεί** μια άλλη μέθοδο που πετάει **εξαίρεση** έχει δύο επιλογές
 - **Catch**: Να **πιάσει** και να **χειριστεί** την εξαίρεση.
 - **Declare**: Να κάνει κι αυτή **throw** την εξαίρεση.
 - Αυτό είναι μια μορφή **μετάθεσης ευθυνών**, αφήνουμε την παραπάνω μέθοδο να χειριστεί την εξαίρεση.
- Αν δεν κάνουμε ένα από τα δύο, ο **compiler** θα παραπονεθεί.
- **Εξαίρεση**: **Runtime exceptions**
 - Κάποιες εξαιρέσεις μπορούμε απλά να τις **αφήσουμε**. Αν συμβούν το πρόγραμμα μας θα τερματίσει με λάθος
 - Π.χ., **NullPointerException**

```
import java.util.Scanner;
```

```
public class DivisionDemoSecondVersion
```

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

```
{  
    Scanner keyboard = new Scanner(System.in);
```

```
    try
```

```
{  
        System.out.println("Enter numerator, denominator :");  
        int numerator = keyboard.nextInt(); int denominator = keyboard.nextInt();
```

```
        int percentage = safePercentage(numerator, denominator);  
        System.out.println("percentage = " + percentage + "%");
```

```
    }  
    catch(DivisionByZeroException e)
```

```
{  
        System.out.println(e.getMessage( ));  
        secondChance();  
    }
```

```
}
```

Εφόσον η main δεν πετάει εξαίρεση, θα πρέπει να βάλουμε την κλήση της safePercentage μέσα σε try-catch block

Η safePercentage δεν χρειάζεται try-catch block γιατί πετάει κι αυτή την εξαίρεση της safeDivide (declare). Αλλιώς θα είχαμε compile error.

```
public static int safePercentage(int top, int bottom) throws DivisionByZeroException
```

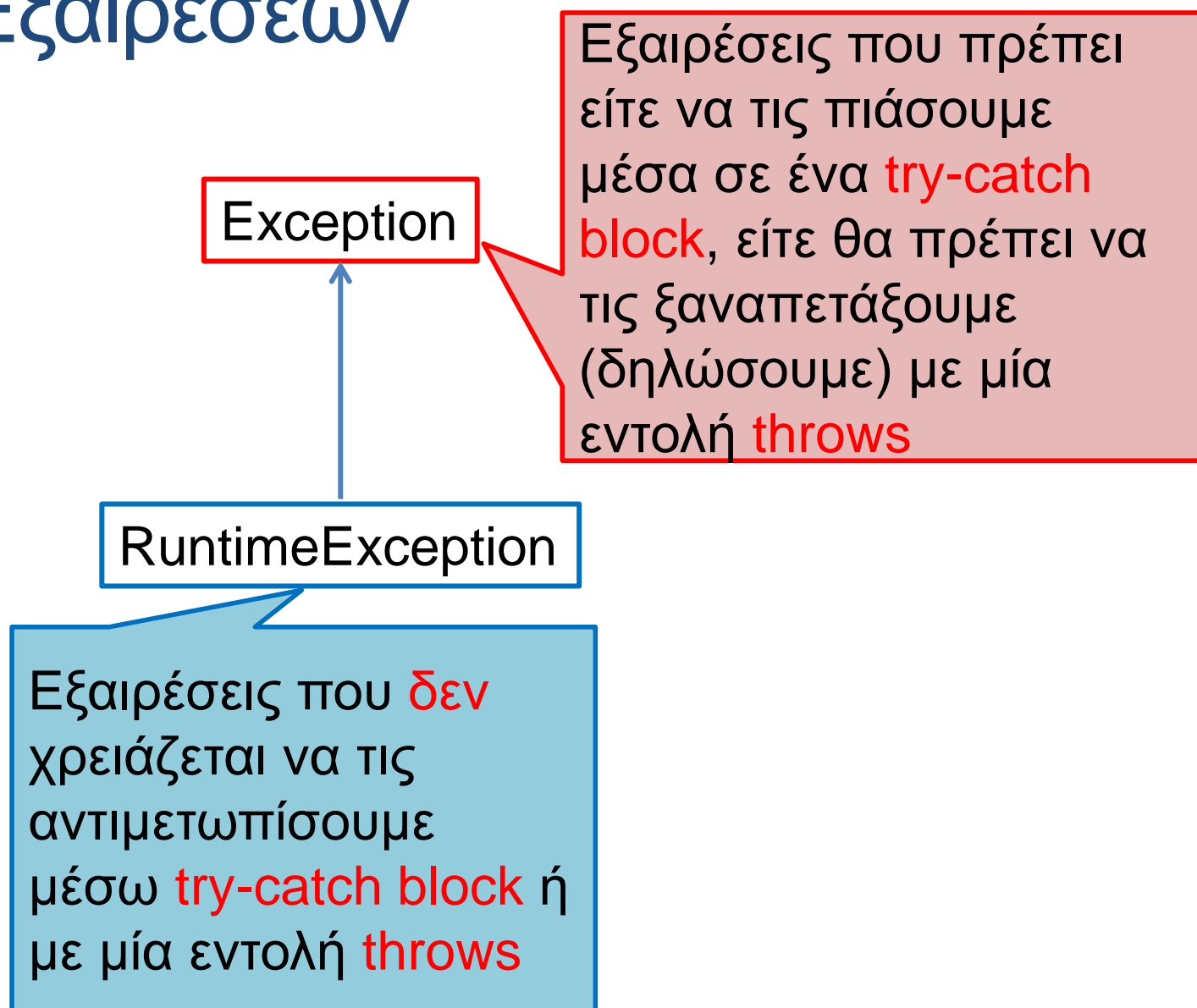
```
{  
    double ratio = safeDivide(top, bottom);  
    return (int)(ratio*100);  
}
```

```
public static double safeDivide(int top, int bottom) throws DivisionByZeroException
```

```
{  
    if (bottom == 0)  
        throw new DivisionByZeroException( );  
    return top/(double)bottom;  
}
```

```
}
```


Τύποι Εξαιρέσεων



```

import java.util.Scanner;
import java.util.InputMismatchException;

public class InputMismatchExceptionDemo
{
    public static void main(String[] args)
    {
        Scanner keyboard = new Scanner(System.in);
        int number = 0; //to keep compiler happy
        boolean done = false;

        while (!done)
        {
            try
            {
                System.out.println("Enter a whole number:");
                number = keyboard.nextInt();
                done = true;
            }
            catch (InputMismatchException e)
            {
                keyboard.nextLine();
                System.out.println("Not a correctly written whole number.");
                System.out.println("Try again.");
            }
        }

        System.out.println("You entered " + number);
    }
}

```

Αν και δεν είναι απαραίτητο μπορούμε να πιάσουμε ένα RuntimeException.

Στο παράδειγμα αυτό χρησιμοποιούμε το InputMismatchException για να δημιουργήσουμε ένα βρόχο μέχρι να δοθεί το σωστό input

Η εξαίρεση δημιουργείται από την μέθοδο nextInt()

Το InputMismatchException είναι υπάρχουσα RuntimeException της Java

```
import java.util.Scanner;
import java.util.InputMismatchException;

public class InputMismatchExceptionDemo2
{
    public static void main(String[] args)
    {
        Scanner keyboard = new Scanner(System.in);
        int number = 0; //to keep compiler happy

        while (true)
        {
            try
            {
                System.out.println("Enter a whole number:");
                number = keyboard.nextInt();
                break;
            }
            catch (InputMismatchException e)
            {
                keyboard.nextLine();
                System.out.println("Not a correctly written whole number.");
                System.out.println("Try again.");
            }
        }

        System.out.println("You entered " + number);
    }
}
```

Άλλος τρόπος να κάνουμε τον ίδιο κώδικα χρησιμοποιώντας την **break**.

Χρήση εξαιρέσεων σε βρόχους

- Μπορούμε να χρησιμοποιούμε τις εξαιρέσεις για να δημιουργήσουμε **συνθήκες σε βρόχους** όπως είδαμε παραπάνω ώστε να εξασφαλίσουμε την λειτουργία του προγράμματος όπως την θέλουμε

Χρήση Εξαιρέσεων

- Τις εξαιρέσεις θα τις δείτε περισσότερο όταν θα πρέπει να **χρησιμοποιήσετε** κάποια **βιβλιοθήκη** που έχει μεθόδους που **πετάνε εξαιρέσεις**.
- Στον δικό σας κώδικα έχει νόημα να πετάξετε μια **εξαίρεση** όταν έχετε μία μέθοδο που **δεν ξέρει** πώς να χειριστεί ένα λάθος και η απόφαση θα πρέπει να παρθεί σε κάποιο **υψηλότερο σημείο** του κώδικα που έχουμε **περισσότερες πληροφορίες**
 - Για παράδειγμα δεν είναι δουλειά της **safeDivide** να ξαναζητήσει τους αριθμούς. Αφήνει την main να το κάνει.

Προσοχή

- Η εύκολη και **τεμπέλικη** λύση για μια εξαίρεση είναι να την **πιάσουμε** και απλά να **μην κάνουμε τίποτα**, αλλά αυτό είναι **κακή προγραμματιστική τακτική**.