Iterators

- Ένα interface που μας δίνει τις λειτουργίες για να διατρέχουμε ένα Collection
  - Ιδιαίτερα χρήσιμοι αν θέλουμε να αφαιρέσουμε στοιχεία από ένα Collection.
- Μέθοδοι του Iterator<T>
  - `hasNext()`: boolean αν ο iterator έχει φτάσει στο τέλος ή όχι.
  - `T next()`: επιστρέφει την επόμενη τιμή (αναφορά όχι αντίγραφο)
  - `remove()`: αφαιρεί το στοιχείο το οποίο επέστρεψε η τελευταία next()
- Μέθοδος του Collection :
  - `Iterator iterator()`: επιστρέφει ένα iterator για μία συλλογή. Π.χ.:
    - `HashSet<String> mySet = new HashSet<String>();`
    - `Iterator<String> iter = mySet.iterator();`

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

public class WrongIteratorExample
{
   public static void main(String[] args){
        HashSet<String> mySet = new HashSet<String>();
        Scanner input = new Scanner(System.in);

        while(input.hasNext()){
           if (!mySet.contains(name)){
                mySet.add(input.next());
           }
        }

        for (String s: mySet){
           if (s.length() <= 2){
                mySet.remove(s);
           }
        }

        for (String s:mySet){
           System.out.println(s);
        }
   }
}
```

Θέλω να αφαιρέσω από το σύνολο τα Strings με λιγότερους από 2 χαρακτήρες

Αν διατρέξουμε το set με την for-each εντολή θα πάρουμε (συνήθως) **λάθος**.

Δεν μπορούμε να αλλάζουμε το Collection ενώ το διατρέχουμε!

```
import java.util.HashSet;
import java.util.Iterator;
import java.util.Scanner;

public class IteratorExample
{
    public static void main(String[] args){
        HashSet<String> mySet = new HashSet<String>();
        Scanner input = new Scanner(System.in);

        while(input.hasNext()){
            if (!mySet.contains(name)){ mySet.add(input.next()); }
        }

        Iterator<String> it = mySet.iterator();
        while (it.hasNext()){
            if (it.next().length() <= 2){
                it.remove();
            }
        }

        it = mySet.iterator();
        while (it.hasNext()){
            System.out.println(it.next());
        }
    }
}
```

Θέλω να αφαιρέσω από το σύνολο τα Strings με λιγότερους από 2 χαρακτήρες

Ο Iterator μας επιτρέπει να διατρέχουμε την συλλογή και να διαγράφουμε στοιχεία.

Ξανα-διατρέχουμε τον πίνακα. Ο iterator πρέπει να ξανα-οριστεί για να ξεκινήσει από την αρχή του συνόλου.

```java
import java.util.Map;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Scanner;

class IteratorExample2
{
    public static void main(String[] args){
        HashMap<String, Integer> myMap = new HashMap<String,Integer>();
        Scanner input = new Scanner(System.in);

        while(input.hasNext()){
            String name = input.next();
            if (!myMap.containsKey(name)){myMap.put(name,1);}
            else{ myMap.put(name,myMap.get(name)+1);}
        }

        Iterator<Map.Entry<String,Integer>> iter = myMap.entrySet().iterator();
        while(iter.hasNext()){
            if (iter.next().getValue() <=2){
                iter.remove();
            }
        }

        for(String key: myMap.keySet()){
            System.out.println(key + ":" + myMap.get(key));
        }
    }
}
```

Θέλω να αφαιρέσω από το σύνολο τα Strings με λιγότερες από 2 εμφανίσεις

Η entrySet επιστρέφει μια συλλογή από Map.entry αντικείμενα (γι αυτό πρέπει να κάνουμε import το Map) τα οποία παραμετροποιούμε με τους τύπους που κρατά το HashMap

ListIterator<T>

- Ένας Iterator ειδικά για την συλλογή List
  - Κύριο πλεονέκτημα ότι επιτρέπει διάσχιση της λίστας προς τις δύο κατευθύνσεις και αλλαγές στη λίστα ενώ την διατρέχουμε.
- Επιπλέον μέθοδοι της ListIterator
  - `hasPrevious()`: boolean αν υπάρχουν κι άλλα στοιχεία πριν από αυτό στο οποίο είμαστε.
  - `T previous()`: επιστρέφει την προηγούμενη τιμή
  - `set(T)`: Θέτει την τιμή του στοιχείου που επέστρεψε η τελευταία next()
  - `add(T)`: Προσθέτει ένα στοιχείο στη λίστα αμέσως μετά από αυτό στο οποίο βρισκόμαστε
- Μέθοδος της List :
  - `ListIterator listIterator()`: επιστρέφει ένα iterator για μία συλλογή.

```
import java.util.*;

public class ListIteratorExample
{
        public static void main(String[] args){
                ArrayList<String> array = new ArrayList<String>();
                Scanner input = new Scanner(System.in);

                while(input.hasNext()){
                        String name = input.next();
                        array.add(name);
                }

                ListIterator<String> it = array.listIterator();
                while (it.hasNext()){
                        if (it.next().equals("a")){
                                it.set("b");
                                it.add("c");
                        }
                }
                it = array.listIterator();
                while (it.hasNext()){
                        System.out.println(it.next());
                }

        }
}
```

```
import java.util.*;

public class ListIteratorExample
{
    public static void main(String[] args){
       ArrayList<String> myList = new ArrayList<String>();
       Scanner input = new Scanner(System.in);

       while(input.hasNext()){
            String name = input.next();
            myList.add(name);
       }

       myList.remove("a");

       for (String s: myList){
            System.out.println(s);
       }
    }
}
```

Θέλω να αφαιρέσω από τις εμφανίσεις του String “a” από την λίστα μου

Η κλήση της remove θα αφαιρέσει μόνο την πρώτη εμφάνιση του “a”

Πως θα τις αφαιρέσουμε όλες?

Υπενθύμιση: η remove επιστρέφει boolean αν έγινε επιτυχώς αφαίρεση (αν άλλαξε δηλαδή η λίστα).

Θέλω να αφαιρέσω από τις εμφανίσεις του String “a” από την λίστα μου

```
import java.util.*;

public class ListIteratorExample
{
    public static void main(String[] args){
       ArrayList<String> myList = new ArrayList<String>();
       Scanner input = new Scanner(System.in);

       while(input.hasNext()){
            String name = input.next();
            myList.add(name);
       }

       while(myList.remove("a"));

       for (String s: myList){
            System.out.println(s);
       }
    }
}
```

Καλεί την remove μέχρι να επιστρέψει false

Η υλοποίηση αυτή όμως **δεν είναι αποδοτική** γιατί κάθε φορά που καλούμε την remove διατρέχουμε την λίστα από την αρχή. Είναι καλύτερα να χρησιμοποιήσουμε ένα iterator.

Χρήση των συλλογών

- Οι τρεις συλλογές που περιγράψαμε είναι πάρα πολύ χρήσιμες για να κάνετε γρήγορα προγράμματα
  - Συνηθίσετε να τις χρησιμοποιείτε και μάθετε πότε βολεύει να χρησιμοποιείτε την κάθε δομή
- Το HashMap είναι ιδιαίτερα χρήσιμο γιατί μας επιτρέπει πολύ γρήγορα να κάνουμε lookup: να βρίσκουμε ένα κλειδί μέσα σε ένα σύνολο και την συσχετιζόμενη τιμή

Παραδείγματα

- Έχουμε ένα πρόγραμμα που διαχειρίζεται τους φοιτητές ενός τμήματος. Ποια συλλογή πρέπει να χρησιμοποιήσουμε αν θέλουμε να λύσουμε τα παρακάτω προβλήματα?
  1. Θέλουμε να μπορούμε να εκτυπώσουμε τις πληροφορίες για τους φοιτητές που παίρνουν ένα μάθημα.
     - `ArrayList<Student> allStudents`
  2. Θέλουμε να μπορούμε να τυπώσουμε τις πληροφορίες για ένα συγκεκριμένο φοιτητή (χρησιμοποιώντας το ΑΜ του φοιτητή)
     - `HashMap<Integer,Student> allStudents`
  3. Θέλουμε να ξέρουμε ποιοι φοιτητές έχουν ξαναπάρει το μάθημα και να μπορούμε να ανακτήσουμε αυτή την πληροφορία για κάποιο φοιτητή
     - `HashSet<Integer> repeatStudents`
     - `HashSet<Student> repeatStudents`

Αναζήτηση με ΑΜ

Αναζήτηση με αντικείμενο

Χρήση δομών

- ArrayList: όταν θέλουμε να διατρέχουμε τα αντικείμενα ή όταν θέλουμε διάταξη των αντικείμενων, και δεν θα χρειαστούμε αναζήτηση κάποιου αντικείμενου
  - Π.χ., μια κλάση Course περιέχει μια λίστα από αντικείμενα τύπου Students
    - Εφόσον μας ενδιαφέρει να τυπώνουμε μόνο.
- HashSet: όταν θέλουμε να έχουμε μια συλλογή από μοναδικά αντικείμενα και θέλουμε γρήγορη αναζήτηση για να μάθουμε αν κάποιο αντικείμενο ανήκει σε αυτή
  - Π.χ., να βρούμε αν ένας φοιτητής (ΑΜ) ανήκει στη λίστα των φοιτητών που ξαναπαίρνουν το μάθημα
  - Π.χ., να βρούμε τα μοναδικά ονόματα από μια λίστα με ονόματα με επαναλήψεις
- HashMap: Ίδια λειτουργικότητα με το HashSet αλλά μας επιτρέπει να συσχετίσουμε μια τιμή με κάθε στοιχείο του συνόλου
  - Π.χ. θέλω να ανακαλέσω γρήγορα τις πληροφορίες για ένα φοιτητή χρησιμοποιώντας το ΑΜ του
  - Το HashMap είναι πιο χρήσιμο απ' ότι ίσως θα περιμένατε

Περίπλοκες δομές

- Έχουμε μάθει τρεις βασικές δομές
  - `ArrayList`
  - `HashSet`
  - `HashMap`
- Μπορούμε να δημιουργήσουμε αντικείμενα που συνδιάζουν αυτές τις δομές
  - `HashMap<String,ArrayList<String>>`
  - `ArrayList<HashSet<String>>`
  - `HashMap<Integer,HashMap<String,String>>`

Παράδειγμα

- Θέλουμε για καθένα από τα μοναδικά Strings που διαβάζουμε να κρατάμε τις θέσεις στις οποίες εμφανίστηκαν.
  - Π.χ., αν έχουμε είσοδο “a b a c b a”, για το “a” θα τυπώσουμε τις θέσεις 0,2,5, για το “b” θα τυπώσουμε τις θέσεις 1,4 και για το “c” τη θέση 3.

```
import java.util.HashMap;
import java.util.ArrayList;
import java.util.Scanner;

class HashMapArrayListExample
{
    public static void main(String[] args){
        HashMap<String,ArrayList<Integer>> myMap =
                                            new HashMap<String,ArrayList<Integer>>();
        Scanner input = new Scanner(System.in);

        int counter = 0;
        while(input.hasNext()){
            String name = input.next();
            if (!myMap.containsKey(name)){
                myMap.put(name,new ArrayList<Integer>());
            }
            myMap.get(name).add(counter);
            counter ++;
        }

        for(String name: myMap.keySet()){
            System.out.print(name + ":");
            for (Integer i:myMap.get(name)){
                System.out.print(" "+i);
            }
            System.out.println();
        }
    }
}
```

Παράδειγμα

- Στο πρόγραμμα της γραμματείας ενός πανεπιστημίου που κρατάει πληροφορία για τους φοιτητές, θέλω γρήγορα με το ΑΜ του φοιτητή να μπορώ να βρω το βαθμό για ένα μάθημα χρησιμοποιώντας τον κωδικό του μαθήματος. Τι δομή πρέπει να χρησιμοποιήσω?

Υλοποίηση

- Χρειάζομαι ένα HashMap με κλειδί το ΑΜ του φοιτητή ώστε να μπορούμε γρήγορα να βρούμε πληροφορίες για τον φοιτητή.
  - Τι τιμές θα κρατάει το HashMap?

- Θα πρέπει να κρατάει άλλο ένα HashMap το οποίο να έχει σαν κλειδί τον κωδικό του μαθήματος και σαν τιμή τον βαθμό του φοιτητή.

Ορισμός

```
HashMap<Integer,HashMap<Integer,double>> StudentCoursesGrades;
```

Χρήση

```
StudentCoursesGrades = new HashMap<Integer,HashMap<int,double>>();
StudentCoursesGrades.put(469,new HashMap<Integer,double>());
StudentCoursesGrades.get(469).put(205,9.5);
StudentCoursesGrades.get(469).get(205);
```

Προσθέτει το βαθμό

Διαβάζει το βαθμό

Διαφορετική υλοποίηση

- Στο πρόγραμμα μου να έχω μια κλάση Student που κρατάει τις πληροφορίες για ένα φοιτητή και μία κλάση StudentRecord που κρατάει την καρτέλα του φοιτητή για το μάθημα. Πως αλλάζει η υλοποίηση?

Ορισμός

```
HashMap<Integer,Student> allStudents;
```

Ορισμός

```
class Student
{
    private int AM;
    private HashMap<Integer,StudentRecord> courses;
    ...

    pubic StudentRecord getCourseRecord(int CourseId){
        return courses.get(courseId);
    }
}
```

Ορισμός

```
class StudentRecord
{
    private double grade;
    ...

    public double getGrade{
        return grade;
    }
}
```

Χρήση

```
allStudents.get(469).getCourseRecord(205).getGrade();
```

Ορισμός

```
HashMap<Integer,Student> allStudents;
```

Ορισμός

```
class Student
{
    private int AM;
    private HashMap<Integer,StudentRecord> courses;
    ...

    pubic HashMap<Integer,StudentRecord> getCourses(){
        return courses;
    }
}
```

Διαφορετική υλοποίηση
Μπορούμε να επιστρέφουμε ένα HashMap

Ορισμός

```
class StudentRecord
{
    private double grade;
    ...

    public double getGrade{
        return grade;
    }
}
```

Χρήση

```
allStudents.get(469).getCourses().get(205).getGrade();
```

Χρονική πολυπλοκότητα

- Έχει τόσο μεγάλη σημασία τι δομή θα χρησιμοποιήσουμε? Όλες οι δομές μας δίνουν περίπου την ίδια λειτουργικότητα.
  - ΝΑΙ!

- Αν κάνουμε αναζήτηση για μια τιμή σε ένα ArrayList πρέπει να διατρέξουμε τη λίστα για να δούμε αν ένα στοιχείο ανήκει ή όχι στη λίστα.
  - Κατά μέσο όρο θα συγκρίνουμε με τα μισά στοιχεία της λίστας
  - Η χρονική πολυπλοκότητα είναι τετραγωνική ως προς τον αριθμό των στοιχείων

- Σε ένα HashSet ή HashMap αυτό γίνεται σε χρόνο σχεδόν σταθερό (ή λογαριθμικό ως προς τον αριθμό των στοιχείων)
  - Αν έχουμε πολλά στοιχεία, και κάνουμε πολλές αναζητήσεις αυτό κάνει διαφορά
  - Η χρονική πολυπλοκότητα είναι γραμμική ως προς τον αριθμό των στοιχείων.

```
import java.util.*;

class ArrayHashComparison
{
    public static void main(String[] args){
          ArrayList<Integer> array = new ArrayList<Integer>();
          for (int i =0; i < 100000; i ++){
              array.add(i);
          }
          HashSet<Integer> set = new HashSet<Integer>();
          for (int i =0; i < 100000; i ++){
              set.add(i);
          }
          ArrayList<Integer> randomNumbers = new ArrayList<Integer>();
          Random rand = new Random();
          for (int i = 0; i < 100000; i ++){
              randomNumbers.add(rand.nextInt(200000));
          }

          long startTime = System.currentTimeMillis();
          for (Integer x:randomNumbers){
              boolean b = array.contains(x);
          }
          long endTime = System.currentTimeMillis();
          long duration = (endTime - startTime);
          System.out.println("Array took "+ duration + " millisecs");

          startTime = System.currentTimeMillis();
          for (Integer x:randomNumbers){
              boolean b = set.contains(x);
          }
          endTime = System.currentTimeMillis();
          duration = (endTime - startTime);
          System.out.println("Set took "+duration + " millisecs");
    }
}
```

Με το ArrayList κάνουμε περίπου 100000*100000/2 συγκρίσεις

Με το HashSet κάνουμε περίπου 100000 συγκρίσεις

21. ΕΞΑΙΡΕΣΕΙΣ

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.");
    }
}
```

Υλοποίηση με εξαιρέσεις

- Όταν υπάρχει κάποιο πρόβλημα στην εκτέλεση του προγράμματος (π.χ., μηδενικός αριθμός από άνδρες ή γυναίκες μαθητές) το πρόγραμμα μας θα πετάει (δημιουργεί) μια εξαίρεση (throws an exception) και σταματάει την ομαλή ροή του προγράμματος.
- Σε κάποιο άλλο σημείο του προγράμματος μας πιάνουμε (χειριζόμαστε) την εξαίρεση (catch the exception) και έχουμε κώδικα που την χειρίζεται.
- Τι είναι μια εξαίρεση?
  - Η Java έχει μία κλάση Exception για αυτό το σκοπό που κρατάει πληροφορία για το τι προκάλεσε την εξαίρεση.
  - Μια εξαίρεση είναι ένα αντικείμενο της κλάσης Exception ή κάποιας παράγωγης κλάσης της Exception.

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>
}
```

Το 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
  - Έχει ένα πεδίο message το οποίο κρατάει ένα μήνυμα για το πρόβλημα και το οποίο μπορούμε να διαβάσουμε με την μέθοδο getMessage()

- 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 ν 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 2016.
  - Θα ορίσουμε το BadNumberException
  - Η εξαίρεση θα 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 > 2016)
                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("No erasers. 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.");
    }
}
```

Η εντολή throw δεν μας «στέλνει» στο σωστό catch block.
Όταν πετάξει εξαίρεση, το πρόγραμμα παίρνει τα catch blocks με την σειρά και μπαίνει στο πρώτο που ταιριάζει με την εξαίρεση που πέταξε.

Το BadNumberException «είναι και» Exception και άρα θα μπει σε αυτό το block

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

```
import java.util.Scanner;

public class BadNumberExceptionDemo3
{

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

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

Η σωστή υλοποίηση.
Πρώτα η πιο ειδική εξαίρεση και μετά η πιο γενική εξαίρεση.

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 εξαίρεση.
  - 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.

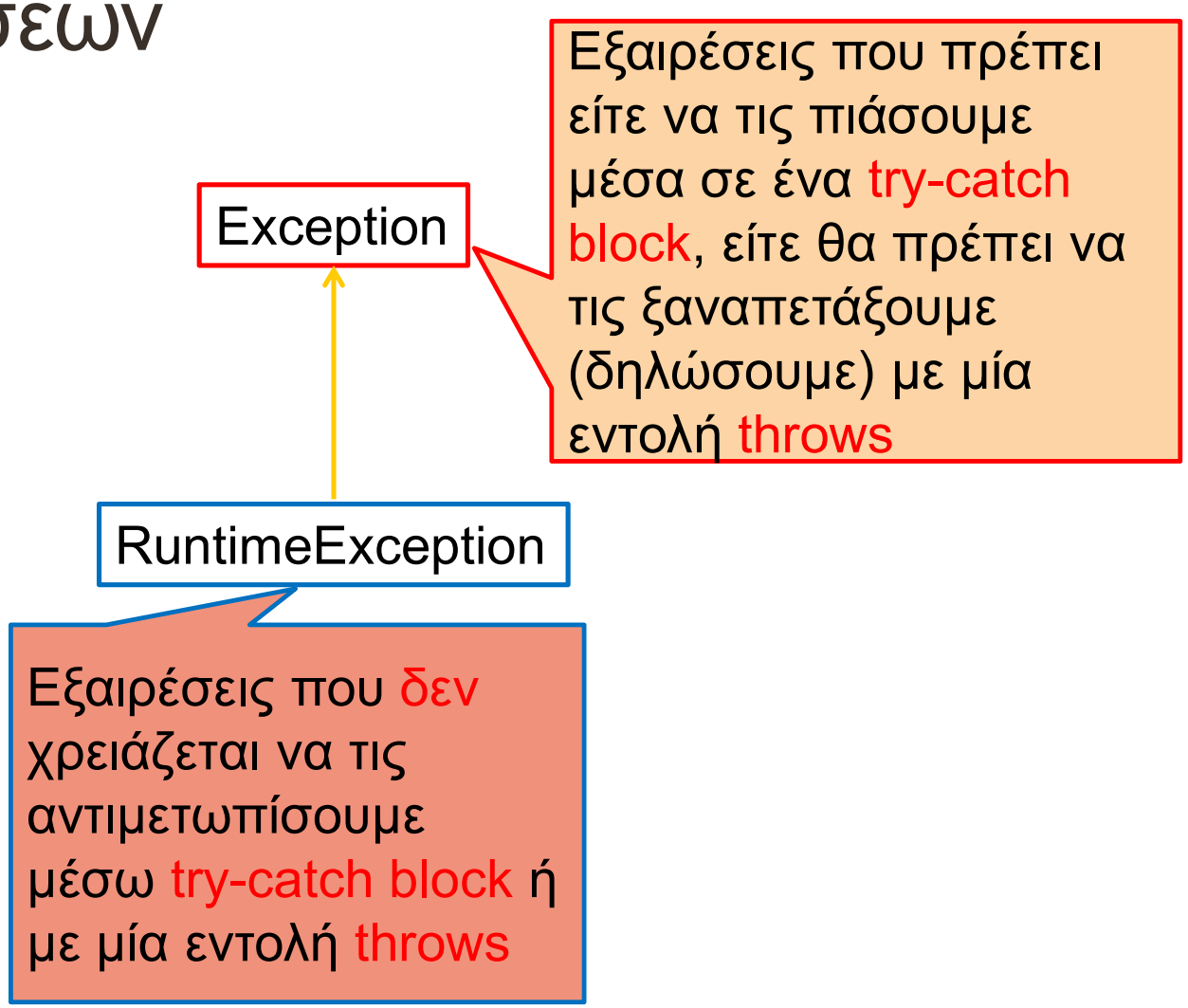
- Εξαίρεση: Runtime exceptions
  - Κάποιες εξαιρέσεις μπορούμε απλά να τις αφήσουμε. Αν συμβούν το πρόγραμμα μας θα τερματίσει με λάθος
  - Π.χ., NullPointerException

```
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, de      inator :");
           int numerator = keyboard.nextInt();     t denominator = keyboard.nextInt();

           int percentage = safePercentage(numerator, denominator);
           System.out.println("percentage  = " + percentage +"%");
        }
        catch(DivisionByZeroException e)
        {
            System.out.println(e.getMessage( ));
            secondChance();
        }
   }

   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;
   }
}
```

Εφόσον η main δεν πετάει εξαίρεση, θα πρέπει να βάλουμε την κλήση της safePercentageμέσα 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.

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.