# Διάλεξη 21: Είσοδος / Έξοδος (I/O)

**Στην ενότητα αυτή θα μελετηθούν τα εξής επιμέρους θέματα:**

- Είσοδος/Εξοδος σε αρχεία (File I/O)
- Ροές Εισόδου / Εξόδου (Stream I/O)
- I/O (IO) vs. New I/O (NIO)
- NIO vs. NIO2

**Διδάσκων: Παναγιώτης Ανδρέου**

# Εισαγωγή

- Το **σύστημα αρχείων (file system) αποθηκεύει και οργανώνει αρχεία** πάνω σε κάποια μέσα (π.χ., σκληρός δίσκος)
- Τα περισσότερα file systems αποθηκεύουν τα αρχεία σε μία **ιεραρχική (δεντρική) δομή**
- Στην **κορυφή του δέντρου (ρίζα)** υπάρχουν κάποιοι κόμβοι ρίζας
- Κάτω από κάθε κόμβο ρίζας, υπάρχουν **αρχεία και φάκελοι (directories/folders)**
- Κάθε φάκελος περιέχει αρχεία και υπο-φάκελους αναδρομικά

# Μονοπάτια (Paths)

- **Ένα μονοπάτι υποδηλώνει μία διαδρομή προς ένα αρχείο ή φάκελο**
- Κάποια λειτουργικά υποστηρίζουν **πολλαπλές ρίζες μονοπατιών** (π.χ., Windows C:\, D:\)
- Παραδείγματα μονοπατιών
  - /home/EPL233/examples (unix, linux, solaris)
  - C:\home\EPL233\examples (windows)
- Ένα μονοπάτι είναι **Απόλυτο (Absolute) ή Αναφορικό (Relative)**
- Ένα απόλυτο μονοπάτι περιλαμβάνει την διαδρομή από την ρίζα μέχρι το αρχείο/φάκελο
  Π.χ., C:\home\EPL233\examples
- Ένα αναφορικό μονοπάτι πρέπει να συνδυαστεί με ένα άλλο μονοπάτι για να έχουμε πρόσβαση στο αρχείο
  Π.χ., αν βρισκόμαστε στο φάκελο home τότε το EPL233\examples σημαίνει <τρέχον φάκελος>\<EPL233\examples>

# Αρχεία και φάκελοι

- Στην JAVA η κλάση File αντιπροσωπεύει αρχεία και φακέλους
- Παρέχει αφαιρετικότητα η οποία αντιμετωπίζει την πολυπλοκότητα των αρχείων και των μονοπατιών σε διαφορετικές μηχανές
- Το όνομα ενός αρχείου είναι ένα απλό String
- Το αντικείμενο τύπου File είναι ένα wrapper class το οποίο περιλαμβάνει το όνομα του αρχείου/φακέλου, τον φάκελο που ανήκει καθώς και πολλές άλλες πληροφορίες.
- Παραδείγματα

```
File f = new File("C:\\2012F.EPL233\\lectures");
File f = new File("C:\\2012F.EPL233\\classlist.xls");
```

# Χρήσιμες Μεθόδοι Κλάσης File

- **boolean exists(), canExecute(), canRead(), canWrite()**
  Ελέγχουν αν υπάρχει, αν μπορούμε να εκτελέσουμε, γράψουμε ή να διαβάσουμε ένα αρχείο
- **boolean createNewFile()**
  Δημιουργία ενός καινούριου αρχείου
- **boolean delete()**
  Διαγράφει ένα αρχείο
- **String getAbsolutePath(), getName(), getParent()**
  Επιστρέφουν το απόλυτο μονοπάτι, το όνομα και τον πατέρα του αρχείου
- **boolean isDirectory(), isFile()**
  Ελέγχουν αν είναι φάκελος ή αρχείο ένα File
- **long length(), getFreeSpace(), getTotalSpace(), getUsableSpace()**
  Χρήσιμες πληροφορίες για το μέγεθος του αρχείου ή φακέλου

# Χρήσιμες Μεθόδοι Κλάσης File (συν.)

- **String[] list(), list(FilenameFilter filter)**
  Επιστρέφουν ένα πίνακα με (String) με τα όλα ή συγκεκριμένα (filter) αρχεία που υπάρχουν μέσα σε ένα φάκελο
- **File[] listFiles(), listFiles(FileFilter filter)**
  Επιστρέφουν ένα πίνακα με (File) με τα όλα ή συγκεκριμένα (filter) αρχεία που υπάρχουν μέσα σε ένα φάκελο
- **boolean mkdir()**
  Δημιουργία ενός φακέλου
- **boolean renameTo(File dest)**
  Μετονομασία ενός αρχείου
- **boolean setReadable(boolean readable), setWritable(boolean writable), setReadOnly()**
  Αλλαγή διαφόρων ιδιοτήτων ενός αρχείου ή φακέλου

# Παράδειγμα: Μέγεθος Φακέλου (Directory Size)

```
import java.io.File;
import java.util.Scanner;

public class DirectorySize {
  public static void main(String[] args) {
    // Prompt the user to enter a directory or a file
    System.out.print("Enter a directory or a file: ");
    Scanner input = new Scanner(System.in);
    String directory = input.nextLine();

    // Display the size
    System.out.println(getSize(new File(directory)) + " bytes");
  }

  public static long getSize(File file) {
    long size = 0; // Store the total size of all files

    if (file.isDirectory()) {
      File[] files = file.listFiles(); // All files and subdirectories
      for (int i = 0; i < files.length; i++) {
        size += getSize(files[i]); // Recursive call
      }
    }
    else { // Base case
      size += file.length();
    }
    return size;
  }
}
```

# Ροές Εισόδου/Εξόδου (I/O Streams)

- Μία **ροή (stream)** αντιπροσωπεύει πολλά διαφορετικά είδη πηγών εισόδου ή εξόδου όπως αρχεία σε ένα δίσκο, συσκευές, άλλα προγράμματα κτλ.
- Οι ροές υποστηρίζουν πολλά είδη δεδομένων όπως απλά bytes, primitive data types, localized characters, και objects. Κάποιες ροές απλά μεταφέρουν δεδομένα, άλλες τα διαμορφώνουν και μετασχηματίζουν με διάφορους τρόπους (π.χ., binary)
- Ένα πρόγραμμα χρησιμοποιεί μία **ροή εισόδου (input stream) για να διαβάσει δεδομένα από μία πηγή**
- Ένα πρόγραμμα χρησιμοποιεί μία **ροή εξόδου (output stream) για να γράψει δεδομένα σε μία πηγή**
- **ΣΗΜΑΝΤΙΚΟ:** Πάντα να κλείνουμε μία ροή

# Byte Streams

- **Το πιο μικρό είδος δεδομένων** που μπορεί να διαβαστεί ή να γραφτεί σε ένα αρχείο είναι το byte
- Όλες οι κλάσεις ροής που διαχειρίζονται Bytes **κληρονομούν από τις κλάσεις InputStream και OutputStream.**
- **Για αρχεία**, συνήθως χρησιμοποιούνται οι κλάσεις **FileInputStream και FileOutputStream.**
- Για χαρακτήρες καλύτερα να χρησιμοποιούνται CharacterStreams

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;

public class CopyBytes {
  public static void main(String[] args)
throws IOException {

    FileInputStream in = null;
    FileOutputStream out = null;

    try {
      in = new FileInputStream("in.txt");
      out = new FileOutputStream(
                    new File("out.txt"));

      int c;
      while ((c = in.read()) != -1) {
        out.write(c);
      }
    }
    finally {
      if (in != null) { in.close(); }
      if (out != null) { out.close(); }
    }
  }
}
```

# Βασικές Ροές Εισόδου/Εξόδου (Standard I/O)

- Οι βασικές ροές εισόδου/εξόδου υποστηρίζονται από τα πλείστα OS και εξ' ορισμού διαβάζουν από το πληκτρολόγιο και τυπώνουν στην οθόνη
- Επίσης υποστηρίζουν είσοδο έξοδο σε αρχεία και μεταξύ προγραμμάτων
- **Η JAVA υποστηρίζει 3 βασικές ροές: *Standard Input (*System.in), *Standard Output(*System.out) και *Standard Error*(System.err).**
- Όλες οι 3 βασικές ροές **είναι τύπου ByteStream**
- **Ερώτηση:** Γιατί υπάρχουν δυο ροές εξόδου;
- **Απάντηση:** Για να μπορεί ο χρήστης να γράφει την ροή εξόδου σε αρχείο και παράλληλα να μπορεί να βλέπει τα errors στην οθόνη
- **Μία ακόμα σημαντική ροή εισόδου είναι η Console** που περιέχει την μέθοδο **readPassword()** που επιτρέπει απόκρυψη της εισόδου του χρήστη

# Character Streams

- **Παρόμοιες με Byte Streams**
- **Η Java αποθηκεύει χαρακτήρες σε Unicode μορφή.**
- Τα Character streams μεταφράζουν αυτόματα από την εσωτερική μορφή στην τοπική μορφή χαρακτήρων.
- Πιο εύκολο για προγράμματα τα οποία πρέπει να δουλεύουν για **πολλές γλώσσες**
- Όλες οι κλάσεις ροής που διαχειρίζονται Characters **κληρονομούν από τις κλάσεις Reader και Writer.**

```
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

public class CopyCharacters {
  public static void main(String[] args)
throws IOException {

    FileReader in = null;
    FileWriter out = null;

    try {
      in = new FileReader("in.txt");
      out = new FileWriter(
                    new File("out.txt"));

      int c;
      while ((c = in.read()) != -1) {
        out.write(c);
      }
    }
    finally {
      if (in != null) { in.close(); }
      if (out != null) { out.close(); }
    }
  }
}
```

# Buffered Streams

- Τα προηγούμενα παραδείγματα χρησιμοποιούν *unbuffered* I/O.
- Αυτό σημαίνει ότι **κάθε αίτηση read ή write request χειρίζεται από το λειτουργικό ➔ μη αποδοτικό** αφού μπορεί να σχετίζεται με πρόσβαση σε αρχεία, δίκτυο ή άλλη λειτουργία
- **Η JAVA χρησιμοποιεί ενδιάμεσα buffers (*buffered* I/O streams) για αύξηση αποδοτικότητας**
- Τα **buffered input streams διαβάζουν δεδομένα από ένα buffer** και πρόσβαση στο λειτουργικό γίνεται μόνο όταν το buffer είναι άδειο
- Συμμετρικά, τα **buffered output streams γράφουν σε ένα buffer** και πρόσβαση στο λειτουργικό γίνεται μόνο όταν το buffer είναι γεμάτο.
- **Η μετατροπή από buffered σε unbuffered γίνεται με τους κατασκευαστές** των αντίστοιχων κλάσεων

```
BufferedReader in = new BufferedReader(new FileReader("in.txt"));

BufferedWriter out = new BufferedRiter(new FileWriter("out.txt"));
```

# Είσοδος/Εξοδος ανά γραμμή

- Τις περισσότερες φορές χρειαζόμαστε να διαβάσουμε ή να γράψουμε ολόκληρες προτάσεις και όχι μόνο χαρακτήρες
- Αντικείμενα της κλάσης **BufferedReader** περιέχουν την μέθοδο **readLine()** η οποία διαβάζει μία ολόκληρη γραμμή από το αρχείο
- Παράδειγμα
  **String c = in.readLine()**

- Τα αντικείμενα της κλάσης BufferedWriter δεν περιέχουν μέθοδο για εκτύπωση ολόκληρης γραμμής σε αρχείο
- Μπορεί να χρησιμοποιηθεί η κλάση PrintWriter η οποία παρέχει μία οικογένεια από μεθόδους για εκτύπωση σε ροές (π.χ., println(…))
- Παράδειγμα
  **PrintWriter pw= new PrintWriter(new BufferedWriter(…));**
  **pw.println("this is a test");**

# Σάρωση (Scanning)

- Τα αντικείμενα τύπου Scanner (Text I/O) περιέχουν χρήσιμες συναρτήσεις για διάσπαση της εισόδου σε κομμάτια και μετά μετάφραση των κομματιών ανάλογα με τον τύπο τους
- Τα κομμάτια σε ένα αρχείο είναι διαχωρισμένα με ένα διαχωριστή (delimiter).
- Ο default delimiter το space και μπορεί να αλλάξει με την συνάρτηση useDelimiter([String|Pattern] pattern)
- Παράδειγμα δημιουργίας Scanner
  **Scanner s = new Scanner(new File ("txt.txt")));**
- Χρήσιμες Συναρτήσεις Scanner
  - next[ | Int | Double | Short | …] επιστρέφει το επόμενο String από την είσοδο και τον μετατρέπει σε συγκεκριμένο τύπο (κενό=String)
  - hasNext[ | Int | Double | Short | …] επιστρέφει true αν στην είσοδο ο συγκεκριμένος τύπος (κενό=String)

# Μορφοποίηση (Formatting)

- Τα αντικείμενα τύπου PrintStream (π.χ., System.out) περιέχουν μεθόδους για να εκτυπώνουμε μορφοποιημένη έξοδο, παρόμοια με την C.
- Περιλαμβάνουν τις μεθόδους **print** και **println** που μορφοποιούν τιμές με παρόμοιο τρόπο με την C
- Περιλαμβάνουν την μέθοδο **format** που μορφοποιεί τιμές με πολύ περισσότερες επιλογές
- Παράδειγμα:
  `System.out.format("%1$+020.10f", Math.PI);`
  - **%:** Αρχή της μορφοποίησης
  - **1$:** Ο αριθμός της παραμέτρου
  - **+0:** Flags
  - **20:** Συνολικό Μήκος συμβολοσειράς
  - **.10:** Ακρίβεια
  - **f:** Μετατροπή

# Data Streams

- Τα **Data Streams** υποστηρίζουν δυαδικό (**binary I/O**) για **αρχέγονους τύπους και Strings** αλλά **όχι για αντικείμενα**
- Όλα τα Data Streams **υλοποιούν** τις **διαπροσωπείες DataInput** και **DataOutput**
- Οι πιο γνωστές κλάσεις είναι οι **DataInputStream** και **DataOutputStream**.

```
import …
public class CopyData {

static final double[] prices ={ 19.99, 9.99, 15.99 };
static final String[] descs ={ "T-shirt", "Mug", "Pin"};

public static void main(String[] args) throws
IOException {
  DataInputStream in = null;
  DataOutputStream out = null;
  try {
    out = new DataOutputStream(new BufferedOutputStream(
            new FileOutputStream(new File("out.txt"))));
    for (int i = 0; i < prices.length; i++) {
      out.writeDouble(prices[i]);
      out.writeUTF(descs[i]); }
    out.close();

    in = new DataInputStream(new BufferedInputStream(
    new FileInputStream("out.txt")));
    double price;   String desc;
    try {
      while (true) {
        price = in.readDouble();
        desc = in.readUTF();
          System.out.format("You ordered %s at $%.2f%n",
                                              desc, price);
      }
} catch (EOFException e) { }
} finally {
if (in != null) { in.close(); }
if (out != null) { out.close(); }  }  }  }
```

# Object Streams

- Τα **Object Streams** υποστηρίζουν δυαδικό (**binary I/O**) **για αντικείμενα**
- Όλα τα Object Streams **υλοποιούν** τις **διαπροσωπείες ObjectInput** και **ObjectOutput**
- Οι μοναδικές κλάσεις που υλοποιούν τις πιο πάνω διαπροσωπείες είναι οι **ObjectInputStream** και **ObjectOutputStream**.
- Περιέχουν τις μεθόδους **readObject()** και **writeObject(Object obj)** για διάβασμα και εγγραφή αντικειμένων αντίστοιχα
- Για να μπορούν τα αντικείμενα μίας κλάσης να περαστούν και να επιστραφούν σαν παράμετροι, αυτή **πρέπει να υλοποιεί την άδεια διαπροσωπεία Serializable (αλλιώς java.io.NotSerializableException)**
- **Serialization** είναι η διαδικασία **μετατροπής ενός αντικειμένου σε ByteStream** και **Deserialization αντίστροφα**

# Παράδειγμα Object Streams

```
import …
class Point implements Serializable{
  int x; int y;
  Point(int x, int y){
    this.x=x; this.y=y;
  }
}
public class CopyObject {
  public static void main(String[] args) throws IOException {
    ObjectInputStream in = null;
    ObjectOutputStream out = null;
    try {
      Point p1 = new Point(1,2);
      Point p2 = new Point(3,4);
      out = new ObjectOutputStream(new BufferedOutputStream(new FileOutputStream(new
File("out.txt"))));
      out.writeObject(p1);
      out.writeObject(p2);
      out.close();
      in = new ObjectInputStream(new BufferedInputStream(new
FileInputStream("out.txt")));
      Point pin1 = (Point) in.readObject();
      System.out.println("x="+pin1.x + " y="+pin1.y);
      Point pin2 = (Point) in.readObject();
      System.out.println("x="+pin2.x + " y="+pin2.y);
    }
    catch(…){}
    finally {  //in.close(), out.close()
```

# Γράφοι Αντικειμένων (Object Graphs)

- **Τι συμβαίνει αν ένα serializable αντικείμενο έχει δείκτες σε άλλα serializable αντικείμενα;** Π.χ., (1) δενδροειδής δομή Node=(int data, Node left και Node right), (2) ΠΙΝΑΚΑΣ
- **Τι στιγμή που το αντικείμενο γίνεται serialize ΟΛΟΣ Ο ΓΡΑΦΟΣ γίνεται serialize!**

```
…
class Node implements Serializable{
   int x;              Node left;                    Node right;
   Node(int x){ this.x=x;}
}
public class CopyObjectsComplex {
  public static void main(String[] args) throws IOException {
  ObjectInputStream in = null; ObjectOutputStream out = null;
  try {
    Node root = new Node(2);
    Node leftNode = new Node(1);
    Node rightNode = new Node(3);
    root.left = leftNode;
    root.right=rightNode;
    out = new ObjectOutputStream(new BufferedOutputStream(new FileOutputStream(new File("out.txt"))));
    out.writeObject(root);
    out.close();
    in = new ObjectInputStream(new BufferedInputStream(new FileInputStream("out.txt")));
    Node inNode = (Node) in.readObject();
    System.out.println("x="+inNode.x + " l.x="+inNode.left.x +" r.x="+inNode.right.x);
  } …
```

# Non Serializable Objects

- **Ερώτηση:** Αν ένα αντικείμενο υλοποιεί τη διαπροσωπεία Serializable αλλά περιέχει αντικείμενα που δεν είναι serializable, μπορεί να γραφεί σε αρχείο;
- Παράδειγμα

```
class A { }

class B implements Serializable {
   int x=1;
   A a = new A();
}
```

- **Απάντηση: ΟΧΙ** (java.io.NotSerializableException) …εκτός και αν χρησιμοποιηθεί ο τροποποιητής **transient: υποδηλώνει ότι η συγκεκριμένη μεταβλητή δεν θα γίνει serialize**

- **Τι συμβαίνει αν μία μεταβλητή οριστεί σαν static; (ΣΥΖΗΤΗΣΗ!)**

# I/O vs. New I/O (NIO)

- Οι κλάσεις της JAVA IO παρέχουν δεδομένα υπό την μορφή ροών διαβάζοντας ή γράφοντας ένα byte κάθε φορά σε μία ροή. Αυτό παρουσιάζει τα εξής προβλήματα:
- **Πρόβλημα 1:** Τα **δεδομένα (bytes) δεν αποθηκεύονται** κάπου εκτός και αν ο χρήστης τα αποθηκεύσει ➔ αν θέλω να ξαναέχω πρόσβαση **πρέπει να τα ξαναδιαβάσω από τη ροή**
- **Πρόβλημα 2:** Ο χρήστης δεν μπορεί να μετακινηθεί μπρος/πίσω σε μία ροή ➔ θα πρέπει να τα αποθηκεύσει κάπου ενδιάμεσα (π.χ., buffer)
- **Πρόβλημα 3:** Οι ροές δουλεύουν με μπλοκαρίσματα (blocking IO). Αυτό σημαίνει ότι όταν καλούμε τις μεθόδους read() ή write() τότε η τρέχον διαδικασία (thread) μπλοκάρει μέχρι να υπάρχουν δεδομένα να διαβαστούν ή τα δεδομένα να γραφτούν μέχρι τέλους στη ροή ➔ η τρέχον διαδικασία δεν μπορεί να κάνει τίποτα άλλο μέχρι να τελειώσει η λειτουργία διαβάσματος ή εγγραφής

# I/O vs. New I/O (NIO) (συν.)

- Για να αντιμετωπιστούν τα προβλήματα της προηγούμενης διαφάνειας, η JAVA J2SE 1.4 εισήγαγε τις βιβλιοθήκες java.nio (JAVA New I/O) οι οποίες περιέχουν buffers και channels και έχουν τα εξής πλεονεκτήματα:
- **Πλεονέκτημα 1:** Τα **δεδομένα (bytes) αποθηκεύονται σε ενδιάμεσα buffers** ➔ αν θέλω να ξαναέχω πρόσβαση τότε τα διαβάζω από το buffer **χωρίς να πρέπει να τα ξαναδιαβάσω από τη ροή**
  - **Προσοχή:** πρέπει να γίνεται έλεγχος ότι το buffer περιέχει όλα τα δεδομένα που χρειαζόμαστε.
    Επίσης, όταν αποθηκεύουμε νέα δεδομένα στο buffer πρέπει να προσέχουμε να μην υπερκαλύψουμε υφιστάμενα δεδομένα που δεν έχουμε επεξεργαστεί.
- **Πλεονέκτημα 2:** Ο χρήστης μπορεί να μετακινηθεί μπρος/πίσω χωρίς προβλήματα σε ένα buffer ➔ Συνεπώς μπορούμε να έχουμε πρόσβαση διαβάσματος και εγγραφής ταυτόχρονα

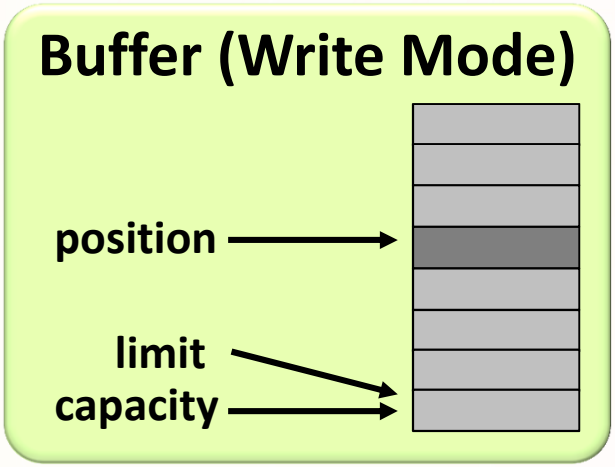
# I/O vs. New I/O (NIO) (συν.)

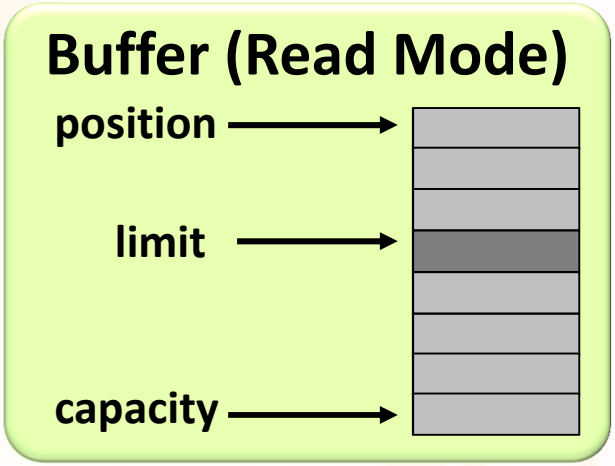
- **Πλεονέκτημα 3:** Οι κλάσεις NIO δουλεύουν με κανάλια και buffers αντί με ροές. Αν ένα νήμα (thread) ζητήσει να διαβάσει ή να γράψει σε ένα κανάλι τότε θα του επιστραφεί ότι υπάρχει στο κανάλι ➔ παράλληλα, μπορεί να προχωρήσει με άλλες διαδικασίες: **non-blocking I/O**.
- **Πλεονέκτημα 4:** Αφού ένα νήμα μπορεί κατά τη διάρκεια που διαβάζει ή γράφει σε ένα κανάλι να προχωρήσει σε άλλες ενέργειες, τότε μπορεί και να διαβάζει ή να γράφει σε άλλα κανάλια ➔ ένα νήμα μπορεί να διαχειριστεί πολλαπλά κανάλια εισόδου και εξόδου

| IO | NIO |
|---|---|
| Stream-oriented | Buffer-oriented |
| blocking | Non-blocking |

# NIO Buffers

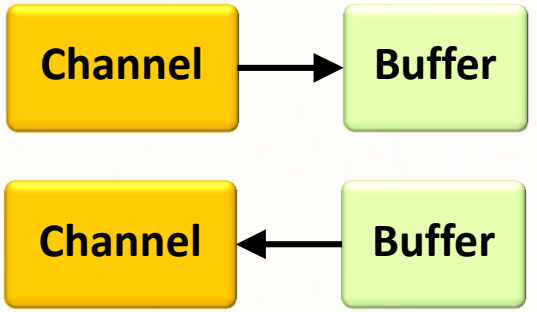
- Ένα buffer είναι ένα block μνήμης όπου μπορούμε να γράψουμε και μετά να διαβάσουμε δεδομένα
- Βασικές Ιδιότητες και Μεθόδοι ενός Buffer:
  - **capacity:** Η χωρητικότητα του (μέγεθος μνήμης)
  - **position:** Η τρέχον θέση εγγραφής/διαβάσματος (αυξάνεται κατά ένα κάθε φορά).
  - **limit:** Για εγγραφή το limit εκφράζει το πόσα bytes μπορούν να εγγραφούν σε ένα buffer (δηλ.=capacity).
    Για ανάγνωσης, πόσα bytes μπορούν να διαβαστούν
  - **flip():** Αλλαγή από read σε write mode. Συνοπτικά, θέτει το limit στο τρέχον position και το position σε 0.



# NIO Channels

- Τα Channels είναι παρόμοια με τις ροές με τις εξής διαφορές:
  - Υποστηρίζουν εγγραφή και διάβασμα ταυτόχρονα
  - Η εγγραφή/διάβασμα γίνεται ασύγχρονα
  - Η εγγραφή/διάβασμα γίνεται μέσω Buffers




- Οι πιο σημαντικές κλάσεις Channels είναι:
  - **FileChannel:** εγγραφή/διάβασμα από/προς αρχεία
  - **DatagramChannel:** εγγραφή/διάβασμα από/προς το δίκτυο με UDP
  - **SocketChannel:** εγγραφή/διάβασμα από/προς το δίκτυο με TCP
  - **ServerSocketChannel:** ακούει για εισερχόμενες συνδέσεις TCP (π.χ., όπως ένας web server). Για κάθε σύνδεση δημιουργείται ένα SocketChannel.

# Αρχεία Τυχαίας-προσπέλασης (Random-access Files)

- Η χρήση καναλιών μας επιτρέπει τυχαία προσπέλαση σε αρχείο (εναλλακτικά μπορούμε να χρησιμοποιήσουμε την κλάση RandomAccessFile αλλά δεν παρέχει τις βελτιστοποιήσεις του NIO)

- Η κλάση **ByteBuffer** επιτρέπει την δημιουργία buffers.
  - **allocateDirect()**: Δημιουργία direct buffer (native i/o)
  - **allocate()**: Δημιουργία non-direct buffer (intermediate memory buffer)
  - **wrap()**: Μετατροπή υφιστάμενου byte array σε buffer

- Η κλάση **SeekableByteChannel** επεκτείνει ένα κανάλι με την έννοια της τρέχων θέσης (current position). Χρησιμοποιείται μέσω ενός αντικειμένου **FileChannel**. Επίσης υποστηρίζει μεθόδους όπως:
  - **position** – Η τρέχων θέση
  - **position(long)** – Ανάθεση της τρέχων θέσης
  - **read(ByteBuffer)** – Διάβασμα από το κανάλι και αποθήκευση στο buffer
  - **write(ByteBuffer)** – Εγγραφή από το buffer προς το κανάλι

# Αρχεία Τυχαίας-προσπέλασης (Random-access Files)

```
import java.nio.ByteBuffer;import
java.nio.channels.FileChannel; import java.nio.file.*;

public class RandomAccessFile_NIO {
  public static void main(String[] args) {
    String s = "I was here!\n";
    byte data[] = s.getBytes();
    ByteBuffer out = ByteBuffer.wrap(data);
    ByteBuffer copy = ByteBuffer.allocate(12);

    try {
      FileChannel fc = FileChannel.open(Paths.get("out.txt"),
          StandardOpenOption.READ, StandardOpenOption.WRITE);
      // Read the first 12 bytes of the file.
      int nread;
      do {
          nread = fc.read(copy);
      } while (nread != -1 && copy.hasRemaining());

      // Write "I was here!" at the beginning of the file.
      fc.position(0);
      while (out.hasRemaining())
          fc.write(out);
      out.rewind();

      // Move to the end of the file and Copy the first 12
      // bytes.  Then write "I was here!" again.
      long length = fc.size();
      fc.position(length-1);
      copy.flip();
      while (copy.hasRemaining()) fc.write(copy);
      while (out.hasRemaining()) fc.write(out);
  } catch (java.io.IOException x) { } } }
```

**Δημιουργία 2 direct ByteBuffers μέσω των Μεθόδων wrap και allocate**

**Δημιουργία ενός SeekableByteChannel**

# NIO vs. NIO2

- Η έκδοση JDK 7 περιλαμβάνει μία καινούρια κλάση, την java.nio.file.Path, η οποία παρέχει περισσότερες δυνατότητες για ενέργειες στο filesystem

  Π.χ., `FileChannel fc = FileChannel.open(Paths.get("out.txt");`
- Σημαντική είναι η παροχή μεθόδων για επεξεργασία συμβολικών links. Π.χ., **/sys-data/ μεταφράζεται στο /admin/system/data/**
- Επίσης, παρέχει αντιγραφή μεγάλων φακέλων πιο γρήγορα με την βελτιστοποιημένη έκδοση της κλάσης java.nio.file.
- Τέλος, παρέχει και ένα καινούριο μοντέλο επεξεργασία που μεταχειρίζεται τα αρχεία τύπου zip και rar σαν file systems κάνοντας πιο εύκολη την επεξεργασίας τους.
- Παράδειγμα Χρήσης

```
URI uri = URI.create("jar:file:/codeSamples/zipfs/zipfstest.zip");
FileSystem zipfs = FileSystems.newFileSystem(uri, env))
Path externalTxtFile = Paths.get("/codeSamples/zipfs/SomeTextFile.txt");
Path pathInZipfile = zipfs.getPath("/SomeTextFile.txt");
externalTxtFile.copyTo(pathInZipfile);
```