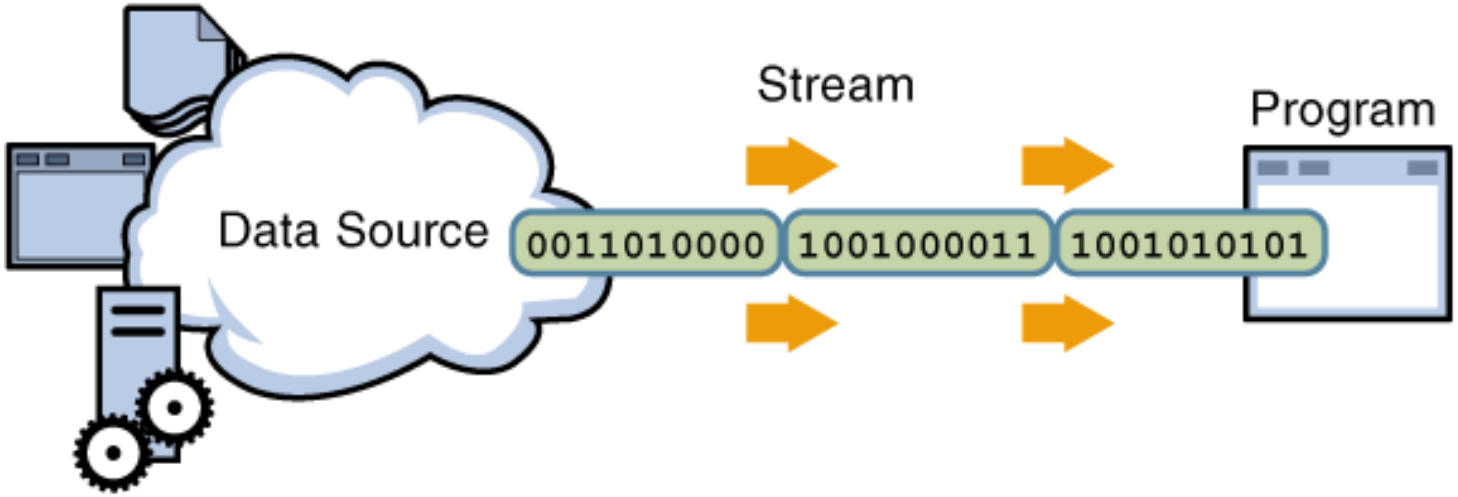
Sabancı Üniversitesi

# Advanced Java I/O & Serialization

Sabancı Üniversitesi

# Streams

- A stream is a connection to a source of data or to a destination for data



Sabancı Üniversitesi

# Useful Java I/O Classes

- **`java.io` classes**
  - `File` – useful for obtaining information about files and directories
  - ***Reader and Writer Classes (Character base I/O)***
    - `FileReader` and `FileWriter`
    - `InputStreamReader` and `OutputStreamWriter`
    - `BufferedReader` and `BufferedWriter`
  - ***Stream Classes (Byte based I/O)***
    - `FileInputStream` and `FileOutputStream`
    - `BufferedInputStream` and `BufferedOutputStream`
    - `DataInputStream` and `DataOutputStream`
    - `ObjectInputStream` and `ObjectOutputStream` (Serialization!)

Sabancı Üniversitesi

# Legacy Class `File`

- Not a stream class
- `createNewFile()` creates a new, empty file named according to the file name used during the File instantiation. It creates a new file only if a file with this name does not exist
- `delete()` deletes file or directory
- `renameTo()` renames a file
- `length()` returns the length of the file in bytes
- `exists()` tests whether the file with specified name exists
- `list()` returns an array of strings naming the files and directories in the specified directory
- `lastModified()` returns the time that the file was last modified
- `mkDir()` creates a directory

Sabancı Üniversitesi

# Reader & Writer Classes

For Character Based I/O Operations

Sabancı Üniversitesi

# FileReader & FileWriter

- Used for character based I/O operations
- Poor performance !!!
- They use default encoding (can not be changed !!!)
- If you want to change encoding use `InputStreamReader` & `OutputStreamWriter` classes

```
FileReader fr = new FileReader (pathname);
fr.read ();
```

```
FileWriter fw= new FileWriter (pathname,true);
fw.write ("test");
```

Append or create new file ?

Sabancı Üniversitesi

# InputStreamReader & OutputStreamWriter

- Used for character based I/O operations
- Encoding can be change by passing encoding type into constructor
- Poor performance !!!

```
FileInputStream fis = new FileInputStream("c:/ test.txt ");
InputStreamReader r = new InputStreamReader(fis, "ISO-8859-9");
```

```
FileOutputStream o = new FileOutputStream("c:/ test.txt ");
OutputStreamWriter osw = new OutputStreamWriter(o,"ISO-8859-9");
osw.write("şşşiiüüüğğğğççööö");
osw.flush();
```

Sabancı Üniversitesi

# BufferedReader & BufferedWriter

- Used for reading and writing big chunk of data (eg: reading lines or character arrays)
- Increases the reading & writing performance

```
FileReader fr = new FileReader("c:/test.txt");
BufferedReader rd = new BufferedReader(fr);
System.out.println(rd.readLine());
```

```
FileWriter fw = new FileWriter("c:/test.txt",true);
BufferedWriter bw = new BufferedWriter(fw);
bw.write("deneme");
```

Sabancı Üniversitesi

# Stream Classes

For byte Based I/O Operations

Sabancı Üniversitesi

# FileInputStream & FileOutputStream

- Used for byte-based I/O operations
- Methods
  - `read()` reads byte from file at a time and returns it if reaches the end of file returns **-1**

```
FileInputStream fis = new FileInputStream (pathname);
System.out.println (fis.read ());
```

  - `write()`

```
FileOutputStream fos = new FileOutputStream (pathname);
fos.write (64);
```

Eg: Binary & text file copy example

Sabancı Üniversitesi

# BufferedInputStream & BufferedOutputStream

- Used for reading and writing big chunk of data (eg: reading lines)
- Increases the performance  (lets do the test)

```
FileInputStream fi = new FileInputStream("c:/ test.txt ");
BufferedInputStream bis = new BufferedInputStream(fi);
bis.read();
```

```
FileOutputStream os = new FileOutputStream("c:/ test.txt ");
BufferedOutputStream bos = new BufferedOutputStream(os);
bos.write(12);
```

Custom buffering ??? ( fastest one )

Sabancı Üniversitesi

# DataInputStream & DataOutputStream

- Used for reading and writing primitive data types

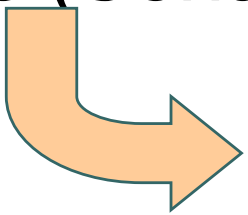
```
FileInputStream fis = new FileInputStream("c:/test.txt");
DataInputStream dis = new DataInputStream(fis);
boolean value = dis.readBoolean();  //int, char, double..etc
```

```
FileOutputStream fos = new FileOutputStream("c:/ test.txt ");
DataOutputStream dos = new DataOutputStream(fos);
dos.writeBoolean(true);
```

Sabancı Üniversitesi

# ObjectInputStream & ObjectOutputStream

- We can read and write primitive types, if we want to read and write objects (Serialization)



Sabancı Üniversitesi

# Serialization

- You can also read and write *objects* to files
- Object I/O i also known as serialization
- If an object is to be serialized:
  - The class must be declared as public
  - The class must implement Serializable
  - The class must have a default constructor
  - All fields of the class must be serializable: either primitive types or serializable objects

Sabancı Üniversitesi

# Implementing Serializable

- To “implement” an interface means to define all the methods declared by that interface, but...
- The Serializable interface does not define any methods!
- Serializable is used as flag to tell Java it needs to do extra work with this class

Sabancı Üniversitesi

# Serialization Example

Serialize

```
Student s = new Student("ahmet",12);
FileOutputStream fos = new FileOutputStream("c:/aaa.txt");
ObjectOutputStream oos = new ObjectOutputStream(fos);
oos.writeObject(s);
```

Be sure that : Student class is Serializable

Deserialize

```
FileInputStream is = new FileInputStream("c:/aaa.txt");
ObjectInputStream ois = new ObjectInputStream(is);
Student aa = (Student) ois.readObject();
System.out.println(aa.getName());
System.out.println(aa.getNo());
```

Sabancı Üniversitesi

# Transient Data

- Some of the data are not serializable or you don't want to include them into the persistent data. /For security reasons

```
public class Student implements Serializable{
   transient private String name;
   private int no;
   public Student() {
      this.name ="";
      this.no=0;
   }
   ....
   .....
}
```

Sabancı Üniversitesi

# Serial Version UID

- During serialization, a version number, serialVersionUID, is used to associate the serialized output with the class used in the serialization process.
- After deserialization, the serialVersionUID is checked to verify that the classes loaded are compatible with the object being deserialized.
- If the receiver of a serialized object has loaded classes for that object with different serialVersionUID, deserialization will result in an `InvalidClassException`.
- A serializable class can declare its own serialVersionUID by explicitly declaring a field named `serialVersionUID` as a static final and of type long: `private static long serialVersionUID = 42L;`

Sabancı Üniversitesi

# Reading&Writing Objects

```
1 public static void main(String[] args) {
2     Stock s1 = new Stock("ORCL", 100, 32.50);
3     Stock s2 = new Stock("APPL", 100, 245);
4     Stock s3 = new Stock("GOOG", 100, 54.67);
5     Portfolio p = new   Portfolio(s1, s2, s3);
6     try (FileOutputStream fos = new FileOutputStream(args[0]);
7          ObjectOutputStream out = new ObjectOutputStream(fos)) {
8         out.writeObject(p);
9     } catch (IOException i) {
10         System.out.println("Exception writing out Portfolio: " + i);
11    }
12    try (FileInputStream fis = new FileInputStream(args[0]);
13         ObjectInputStream in = new ObjectInputStream(fis)) {
14        Portfolio newP = (Portfolio)in.readObject();
15    } catch (ClassNotFoundException | IOException i) {
16        System.out.println("Exception reading in Portfolio: " + i);
17 }
```

Sabancı Üniversitesi

# Serialization Methods

- An object being serialized (and deserialized) can control the serialization of its own fields.

```
public class MyClass implements Serializable {
    // Fields
    private void writeObject(ObjectOutputStream oos) throws IOException {
        oos.defaultWriteObject();
        // Write/save additional fields
        oos.writeObject(new java.util.Date());
    }
}
```

- For example, in this class, the current time is written into the object graph.
- During deserialization, a similar method is invoked:

```
private void readObject(ObjectInputStream ois) throws ClassNotFoundException,
IOException {}
```

Sabancı Üniversitesi

# readObject: Example

```
1   public class Stock implements Serializable {
2       private static final long serialVersionUID = 100L;
3       private String symbol;
4       private int shares;
5       private double purchasePrice;
6       private transient double currPrice;
7
8       public Stock(String symbol, int shares, double purchasePrice) {
9           this.symbol = symbol;
10          this.shares = shares;
11          this.purchasePrice = purchasePrice;
12          setStockPrice();
13      }
14
15      // This method is called post-serialization
16      private void readObject(ObjectInputStream ois)
17                              throws IOException, ClassNotFoundException {
18          ois.defaultReadObject();
19          // perform other initialization
20          setStockPrice();
21      }
22  }
```

Stock `currPrice` is set by the `setStockPrice` method during creation of the Stock object, but the constructor is not called during deserialization.

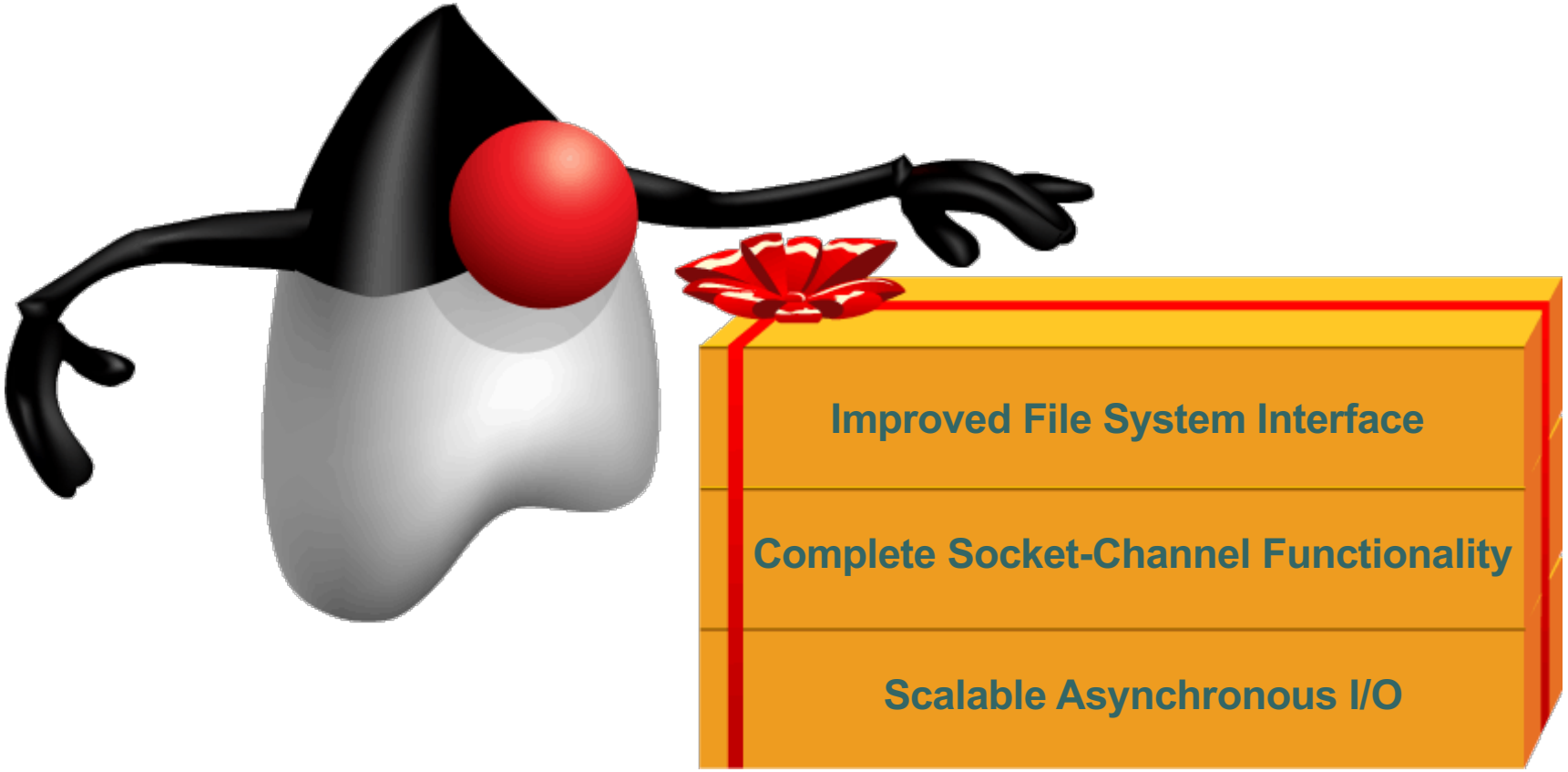
Stock `currPrice` is set after the other fields are deserialized.

Sabancı Üniversitesi

# Java File I/O (NIO.2)

Sabancı Üniversitesi

# New File I/O API (NIO.2)

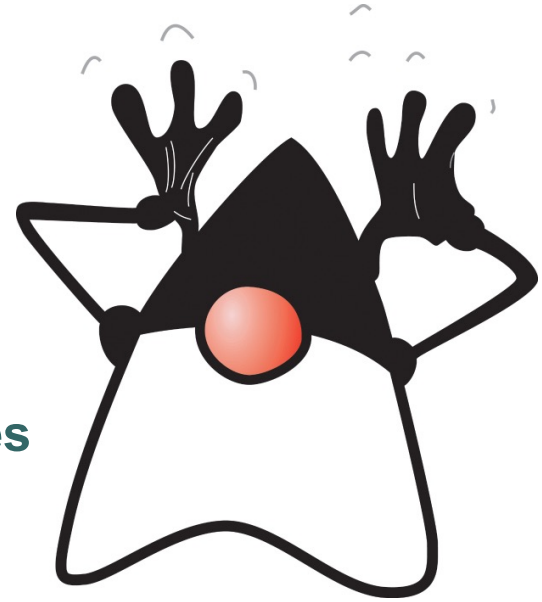


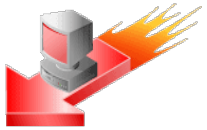
Sabancı Üniversitesi

# Limitations of `java.io.File`

Does not work well with symbolic links

Scalability issues





Performance issues

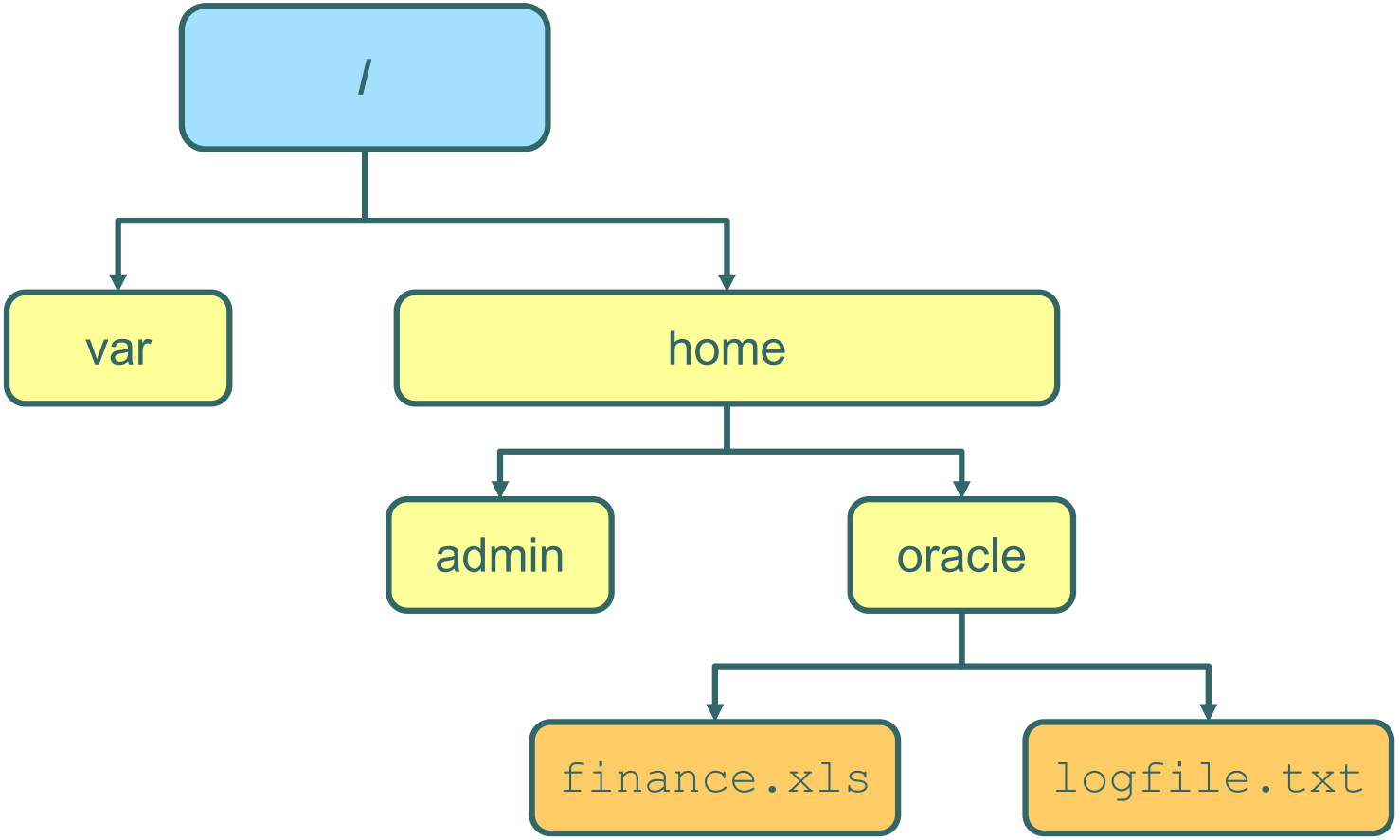
Very limited set of file attributes

Very basic file system access functionality

Sabancı Üniversitesi

# File Systems, Paths, Files

- In NIO.2, both files and directories are represented by a path, which is the relative or absolute location of the file or directory.



Sabancı Üniversitesi

# Relative Path Versus Absolute Path

- A path is either *relative* or *absolute*.
- An absolute path always contains the root element and the complete directory list required to locate the file.
- Example:

```
...
/home/peter/statusReport
...
```

- A relative path must be combined with another path in order to access a file.
- Example:

```
...
clarence/foo
...
```

Sabancı Üniversitesi

# Java NIO.2 Concepts

- Prior to JDK 7, the `java.io.File` class was the entry point for all file and directory operations. With NIO.2, there is a new package and classes:
  - `java.nio.file.Path`: Locates a file or a directory by using a system-dependent path
  - `java.nio.file.Files`: Using a `Path`, performs operations on files and directories
  - `java.nio.file.FileSystem`: Provides an interface to a file system and a factory for creating a `Path` and other objects that access a file system
  - All the methods that access the file system throw `IOException` or a subclass.

Sabancı Üniversitesi

# `Path` Interface

- The `java.nio.file.Path` interface provides the entry point for the NIO.2 file and directory manipulation.

```
FileSystem fs = FileSystems.getDefault();
Path p1 = fs.getPath (“/home/oracle/labs/resources/myFile.txt");
```

- To obtain a `Path` object, obtain an instance of the default file system, and then invoke the `getPath` method:

```
Path p1 = Paths.get(“/home/oracle/labs/resources/myFile.txt");
Path p2 = Paths.get(“/home/oracle", "labs", "resources", "myFile.txt");
```

Sabancı Üniversitesi

# `Path` Interface Features

- The `Path` interface defines the methods used to locate a file or a directory in a file system. These methods include:
  - To access the components of a path:
    - `getFileName`, `getParent`, `getRoot`, `getNameCount`
  - To operate on a path:
    - `normalize`, `toUri`, `toAbsolutePath`, `subpath`, `resolve`, `relativize`
  - To compare paths:
    - `startsWith`, `endsWith`, `equals`

Sabancı Üniversitesi

# Path: Example

```
1    public class PathTest
2        public static void main(String[] args) {
3             Path p1 = Paths.get(args[0]);
4             System.out.format("getFileName: %s%n", p1.getFileName());
5             System.out.format("getParent: %s%n", p1.getParent());
6             System.out.format("getNameCount: %d%n", p1.getNameCount());
7             System.out.format("getRoot: %s%n", p1.getRoot());
8             System.out.format("isAbsolute: %b%n", p1.isAbsolute());
9             System.out.format("toAbsolutePath: %s%n", p1.toAbsolutePath());
10           System.out.format("toURI: %s%n", p1.toUri());
11       }
12   }
```

```
java PathTest /home/oracle/file1.txt
getFileName: file1.txt
getParent: /home/oracle
getNameCount: 3
getRoot: /
isAbsolute: true
toAbsolutePath: /home/oracle/file1.txt
toURI: file:///home/oracle/file1.txt
```

Sabancı Üniversitesi

# Removing Redundancies from a `Path`

- Many file systems use "`.`" notation to denote the current directory and "`..`" to denote the parent directory.
- The following examples both include redundancies:

```
/home/./clarence/foo
/home/peter/../clarence/foo
```

- The `normalize` method removes any redundant elements, which includes any "`.`" or "`directory/..`" occurrences.
- Example:

```
Path p = Paths.get("/home/peter/../clarence/foo");
Path normalizedPath = p.normalize();
```

```
/home/clarence/foo
```

Sabancı Üniversitesi

# Creating a Subpath

- A portion of a path can be obtained by creating a subpath using the `subpath` method:

```
Path subpath(int beginIndex, int endIndex);
```

- The element returned by `endIndex` is one less that the `endIndex` valu
- Example:

home= 0
oracle = 1
Temp = 2

```
Path p1 = Paths.get ("/home/oracle/Temp/foo/bar");
Path p2 = p1.subpath (1, 3);
```

Include the element at index 2.

```
oracle/Temp
```

Sabancı Üniversitesi

# Joining Two Paths

- The `resolve` method is used to combine two paths.
- Example:

```
Path p1 = Paths.get("/home/clarence/foo");
p1.resolve("bar");     // Returns /home/clarence/foo/bar
```

- Passing an absolute path to the `resolve` method returns the passed-in path.

```
Paths.get("foo").resolve("/home/clarence"); // Returns /home/clarence
```

Sabancı Üniversitesi

# File Operations

- Checking a File or Directory
- Deleting a File or Directory
- Copying a File or Directory
- Moving a File or Directory
- Managing Metadata
- Reading, Writing, and Creating Files
- Random Access Files
- Creating and Reading Directories

Sabancı Üniversitesi

# Checking a File or Directory

A `Path` object represents the concept of a file or a directory location. Before you can access a file or directory, you should first access the file system to determine whether it exists using the following `Files` methods:

`exists(Path p, LinkOption... option)`

*Tests to see whether a file exists. By default, symbolic links are followed.*

`notExists(Path p, LinkOption... option)`

*Tests to see whether a file does not exist. By default, symbolic links are followed.*

- Example:

Optional argument

```
Path p = Paths.get(args[0]);
System.out.format("Path %s exists: %b%n", p,
                  Files.exists(p, LinkOption.NOFOLLOW_LINKS));
```

Sabancı Üniversitesi

# Checking a File or Directory

To verify that a file can be accessed, the `Files` class provides the following `boolean` methods.

- `isReadable(Path)`
- `isWritable(Path)`
- `isExecutable(Path)`

Note that these tests are not atomic with respect to other file system operations. Therefore, the results of these tests may not be reliable once the methods complete.

- The `isSameFile (Path, Path)` method tests to see whether two paths point to the same file. This is particularly useful in file systems that support symbolic links.

Sabancı Üniversitesi

# Creating Files and Directories

- Files and directories can be created using one of the following methods:

```
Files.createFile (Path dir);
Files.createDirectory (Path dir);
```

  - The `createDirectories` method can be used to create directories that do not exist, from top to bottom:

```
Files.createDirectories(Paths.get("/home/oracle/Temp/foo/bar/example"));
```

Sabancı Üniversitesi

# Deleting a File or Directory

- You can delete files, directories, or links. The `Files` class provides two methods:
  - `delete(Path)`

```
//...
Files.delete(path);
//...
```

Throws a NoSuchFileException, DirectoryNotEmptyException, or IOException

```
//...
Files.deleteIfExists(Path)
//...
```

No exception thrown

Sabancı Üniversitesi

# Copying a File or Directory

- You can copy a file or directory by using the `copy(Path, Path, CopyOption...)` method.
- When directories are copied, the files inside the directory are not copied.

StandardCopyOption parameters

```
//...
copy(Path, Path, CopyOption...)
//...
```

```
REPLACE_EXISTING
COPY_ATTRIBUTES
NOFOLLOW_LINKS
```

- Example:

```
import static java.nio.file.StandardCopyOption.*;
//...
Files.copy(source, target, REPLACE_EXISTING, NOFOLLOW_LINKS);
```

Sabancı Üniversitesi

# Moving a File or Directory

- You can move a file or directory by using the `move(Path, Path, CopyOption...)` method.
- Moving a directory will not move the contents of the directory.

StandardCopyOption parameters

```
//...
move(Path, Path, CopyOption...)
//...
```

**REPLACE_EXISTING**
**ATOMIC_MOVE**

- Example:

```
import static java.nio.file.StandardCopyOption.*;
//...
Files.move(source, target, REPLACE_EXISTING);
```

Sabancı Üniversitesi

# List the Contents of a Directory

To get a list of the files in the current directory, use the `Files.list()` method.

```
public class FileList {

  public static void main(String[] args) {

    try(Stream<Path> files = Files.list(Paths.get("."))){

      files
          .forEach(line -> System.out.println(line));

    } catch (IOException e){
      System.out.println("Message: " + e.getMessage());
    }
  }
}
```

Sabancı Üniversitesi

# Walk the Directory Structure

The `Files.walk()` method walks a directory structure recursively.

```
public class A11FileWalk {

  public static void main(String[] args) {

    try(Stream<Path> files = Files.walk(Paths.get("."))){

      files
        .forEach(line -> System.out.println(line));

    } catch (Exception e){
      System.out.println("Message: " + e.getMessage());
    }
  }
}
```

Sabancı Üniversitesi

# BufferedReader File Stream

- The new `lines()` method converts a `BufferedReader` into a stream.

```
public class BufferedRead {
  public static void main(String[] args) {
    try(BufferedReader bReader =
      new BufferedReader(new FileReader("tempest.txt"))){

      bReader.lines()
        .forEach(line ->
            System.out.println("Line: " + line));

    } catch (IOException e){
        System.out.println("Message: " + e.getMessage());
    }
  }
}
```

Sabancı Üniversitesi

# NIO File Stream

The `lines()` method can be called using NIO classes

```
public class ReadNio {

  public static void main(String[] args) {

    try(Stream<String> lines =
        Files.lines(Paths.get("tempest.txt"))){

      lines.forEach(line ->
        System.out.println("Line: " + line));

    } catch (IOException e){
      System.out.println("Error: " + e.getMessage());
    }
  }
}
```

Sabancı Üniversitesi

# Read File into `ArrayList`

Use `readAllLines()` to load a file into an `ArrayList`.

```
public class ReadAllNio {
  public static void main(String[] args) {
    Path file = Paths.get("tempest.txt");
    List<String> fileArr;

    try{

      fileArr = Files.readAllLines(file);

      fileArr.stream()
        .filter(line -> line.contains("PROSPERO"))
        .forEach(line -> System.out.println(line));

    } catch (IOException e){
      System.out.println("Message: " + e.getMessage());
    }
  }
}
```

Sabancı Üniversitesi

# Managing Metadata

| Method | Explanation |
|---|---|
| `size` | Returns the size of the specified file in bytes |
| `isDirectory` | Returns true if the specified `Path` locates a file that is a directory |
| `isRegularFile` | Returns true if the specified `Path` locates a file that is a regular file |
| `isSymbolicLink` | Returns true if the specified `Path` locates a file that is a symbolic link |
| `isHidden` | Returns true if the specified `Path` locates a file that is considered hidden by the file system |
| `getLastModifiedTime` | Returns or sets the specified file's last modified time |
| `setLastModifiedTime` | |
| `getAttribute` | Returns or sets the value of a file attribute |
| `setAttribute` | |