/ JAVA DB SCHOOL

Input/Output, Serializable, Threads

/ Input/Output, Serializable

File Class

- Allows accessing file properties, reading their content, renaming, deleting, etc.
- https://docs.oracle.com/javase/7/docs/api/java/io/File.html
- Constructor cand take a relative or an absolute path

File Class

- Example of reading one file’s properties:

```
public class Test {
    public static void main(String[] args) {
        java.io.File f = new java.io.File("data.txt");
        System.out.println(f.exists());
        System.out.println(f.length());
        System.out.println(f.canRead());
        System.out.println(f.canWrite());
        System.out.println(f.isDirectory());
        System.out.println(f.isFile());
        System.out.println(f.isAbsolute());
        System.out.println(f.isHidden());
        System.out.println(f.getAbsolutePath());
    }
}
```

Writing to a Text File

- Can be performed using the `PrintWriter` class
https://docs.oracle.com/javase/7/docs/api/java/io/PrintWriter.html

- Example:

```
public class Test {
    public static void main(String[] args) throws Exception {
        File f = new File("data.txt");
        if (f.exists()) {
            System.err.println("The files does not exist!");
            System.exit(0);
        }
        PrintWriter pw = new PrintWriter(f);
        pw.print("Hello, World!");
        pw.println(2021);
        pw.close();
    }
}
```

Reading from a Text File

- Performed using the Scanner class

```
Scanner s = new Scanner(System.in);
Scanner s = new Scanner(new File(fileName));
```

https://docs.oracle.com/javase/7/docs/api/java/util/Scanner.html

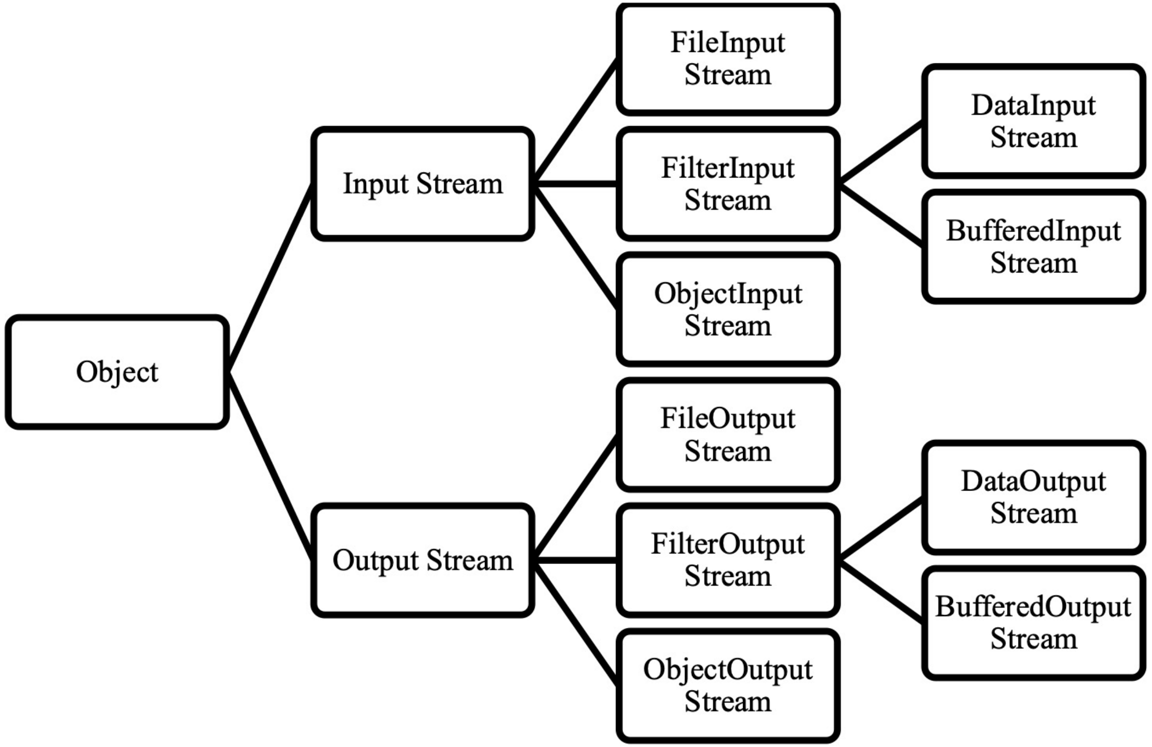
- Other classes to be used for reading text files: FileWriter, BufferedWriter, FileReader, BufferedReader

Reading and Writing to a Binary File

- Can be performed using the `InputStream` and `OutputStrem` classes
https://docs.oracle.com/javase/7/docs/api/java/io/InputStream.html
https://docs.oracle.com/javase/7/docs/api/java/io/OutputStream.html

- More exactly: FileInputStream and FileOutputStream

Input & Output Streams



FileInputStream and FileOutputStream Examples

```
public class Test {
    public static void main(String[] args) throws IOException {
        FileOutputStream out = new FileOutputStream("test.dat");
        for (int i = 1; i <= 50; i++) {
            out.write(i);
        }
        out.close();
        FileInputStream in = new FileInputStream("test.dat");
        int x;
        while ((x = in.read()) != -1) {
            System.out.print(x + " ");
        }
        in.close();
    }
}
```

Serializable Interface

- Allows saving the content and the state of an object
- `transient` marks a field not to be used in seralizing
- Examples of classes that implement Serializable: Date

```
import java.io.Serializable;
public class Employee implements Serializable {
    private String name;
    private transient Integer no;
    public Employee(String name, Integer no) {
        this.name = name;
        this.no = no;
    }
    public String toString() {
        return name;
    }
}
```

Serializable Interface

```
Employee emp = new Employee();
ObjectOutputStream os = null;
try {
    os = new ObjectOutputStream(new FileOutputStream("out.bin"));
    os.writeObject(emp);
}
...
ObjectInputStream is = new ObjectInputStream(new FileInputStream("out.bin"));
emp = (Employee) is.readObject();
...
```

RandomAccessFile

- Allows reading with a “cursor” or a file pointer

```
RandomAccessFile f = new RandomAccessFile("test.dat", "rw");
f.setLength(0);
for (int i = 1; i <= 10; i++)
        f.writeInt(i);
f.seek(0);
System.out.println(f.readInt());
f.seek(4);
System.out.println(f.readInt());
f.seek(20);
System.out.println(f.readInt());
f.seek(f.length());
f.writeInt(20);
System.out.println("Noua lungime este " + f.length());
f.close();
```

/ Threads

Multithreading

- Allows running concurrent (parallel) tasks in a program

- Thread = execution flow of a task from the beginning to its end

- Threads in Java

1. Extend Thread class
2. Implement Runnable in a class and… make a Thread object using this class

Implements Runnable

- Classes needs to implement run method

```
class Employee implements Runnable {

    int no;

    public Employee(int no) {
        this.no = no;
    }

    @Override
    public void run() {
        System.out.println("Employee " + this.no + " arrived at work!");
    }
}
```

Implements Runnable

- Classes needs to implement run method

```
public class Main {
    public static void main(String[] args) {
        // write your code here
        Thread[] employees = new Thread[30];
        for (int i = 0; i < 30; i++) {
            Employee employee = new Employee(i);
            employees[i] = new Thread(employee);
            employees[i].start();
        }
    }
}
```

Extends Thread

- Classes needs to override run method

```
class Employee extends Thread {

    int no;

    public Employee(int no) {
        this.no = no;
    }

    @Override
    public void run() {
        System.out.println("Employee " + this.no + " arrived at work!");
    }
}
```

Threads Join

- Waits for the specified thread to finish

```
try {
    for (int i = 1; i <= 30; i++) {
        employees[i].join();
    }
} catch (InterruptedException ex) {
}
```

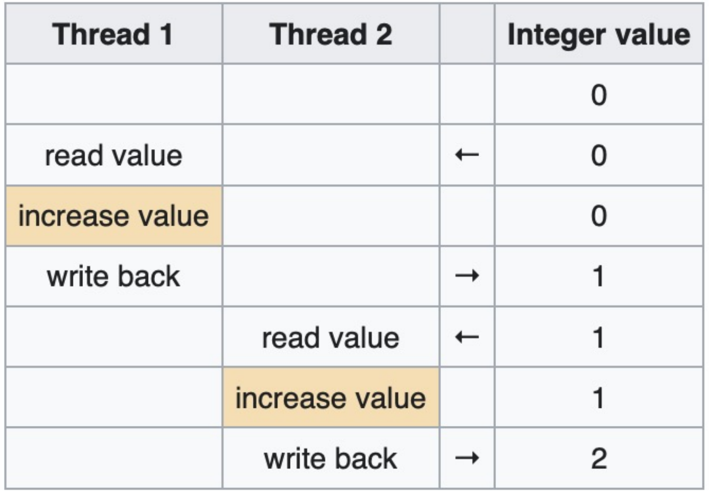
Threads Pool

- Consists of many threads that must be run in parallel

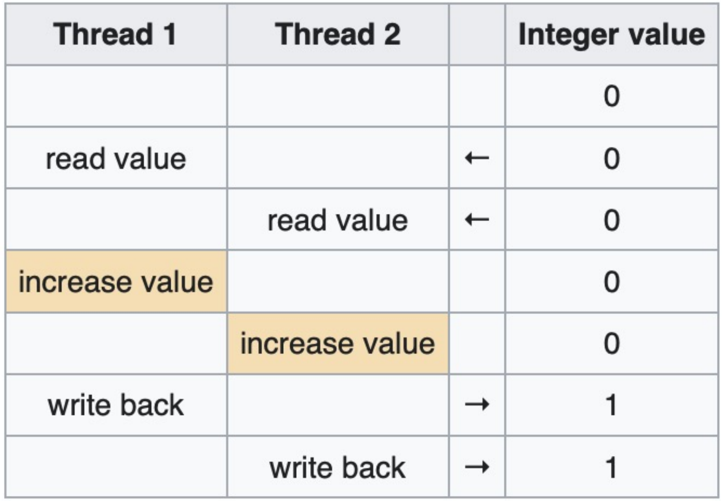
```
public class ExecutorTest {
    public static void main(String[] args) {
        ExecutorService executor = Executors.newFixedThreadPool(3);
        executor.execute(new Employee(1));
        executor.execute(new Employee(2));
        executor.execute(new Employee(3));
        executor.shutdown();
    }
}
```

Race Conditions

- Can happen when two threads try to modify a resource at the same given time



Thread 1	Thread 2		Integer value
			0
read value		←	0
increase value			0
write back		→	1
	read value	←	1
	increase value		1
	write back	→	2



Thread 1	Thread 2		Integer value
			0
read value		←	0
	read value	←	0
increase value			0
	increase value		0
write back		→	1
	write back	→	1

Synchronized

- Synchronized methods and blocks limit one thread to enter a specified area at a given time

```
public synchronized void method1() {
    // code ...
}
public void method2() {
    synchronized (this) {
        // code ...
    }
}
```

Locks

- Allows a programmer to control synchronized areas using the lock/unlock facility
- In Java, ReentrantLock is one class of this kind

```
private static Lock lock = new ReentrantLock();
private int deposit = 0;

public void addMoney(int amount) {
    lock.lock();
    deposit += amount;
    lock.unlock();
}
```

Semaphores

- Similar with Locks, but allows more than one thread to enter a synchronized area
- Can be instantiated with a number of *permits*

```
private static Semaphore semaphore = new Semaphore(4);

 public void intersectionCross() {
     try {
         semaphore.acquire();
         System.out.println("One more car crosses the intersection");
         semaphore.release();
     } catch (InterruptedException exception) {
         exception.printStackTrace();
     }
 }
```

Synchronized Collections

- Java Collections contain a series of synchronized methods, which are synchronized by themselves

https://docs.oracle.com/javase/7/docs/api/java/util/Collections.html

- synchronizedList
- synchronizedMap
- synchronizedSet

/ Practice, practice, practice

Input/Output, Serializable

Define a *Student* register. It contains multiple *Students*, each student having a `name` and a `grade`. Requirements:

- Create a method that saves an array of students into a binary file
- Create a method that loads an array of students from a binary file
- Test the class in a main method

Intersection

- Write a program that simulates an intersection. There must be two directions: Nord-South and West-East
- When cars pass through one direction, the other cars pass through the other direction
- Implement this using two semaphores

/ Q&A

MOBILE ACADEMY