CST-239 Activity 4 Guide

Contents

Part 1: Reading and Writing Text Files

Overview

Goal and Directions:

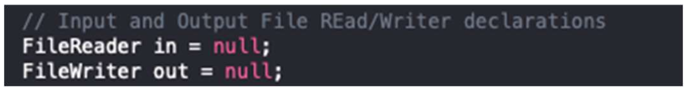
In this activity, you will learn how to read and write text files using non buffered and buffered File I/O classes, then learn how to parse a String into tokens using the String split() method, and finally, you will learn how to handle exceptions using various approaches. Complete the following tasks for this activity:

Execution

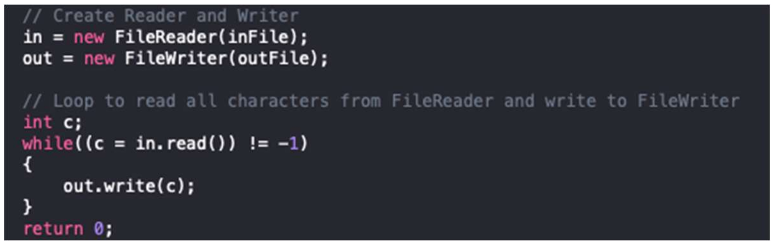
Part 1a:

1. Create a new project named *topic4-1a*.
2. Create a new Java Class named *FilePlay* in the *app* package with a *main*() method.
3. Right click on the root of the project and select the File → New File menu options. Name the file *InUsers.txt*.
    a. Open the *InUsers.txt* with a text editor, such as Notepad, and enter 5 user records where the first field is a person's first name, the second field is their last name, and the third field is their age.
    b. Separate each field with a pipe delimiter (|).
    c. Put each user record on a newline in the file.
4. Create a private static method *copyFile()* that returns an integer and takes 2 arguments. The first argument should be a String for the input filename and the second argument a String for the output filename.
5. Call the *copyFile()* from the *main()* method using an input filename of *InFile.txt* and an output filename of *OutFile.txt*.
6. Implement the *copyFile()* method:
    a. Declare 2 method scoped variables of type *FileReader* and *FileWriter* and initialize their values to null.

b. Create instances of the *FileReader* and *FileWriter* surrounding the code with appropriate try catch blocks that catch the *FileNotFoundException* and the IOException.
c. Declare a local method scope variable *c* of type int.
d. Call the *FileRead read*() method until a -1 is returned, saving the return value from the *read()* method in variable *c*.
e. Call the *FileWriter write*() method for each character read (variable *c*).
f. Return a 0 if all characters are copied without error, a -1 for a File Not Found, and -2 for an I/O Error.



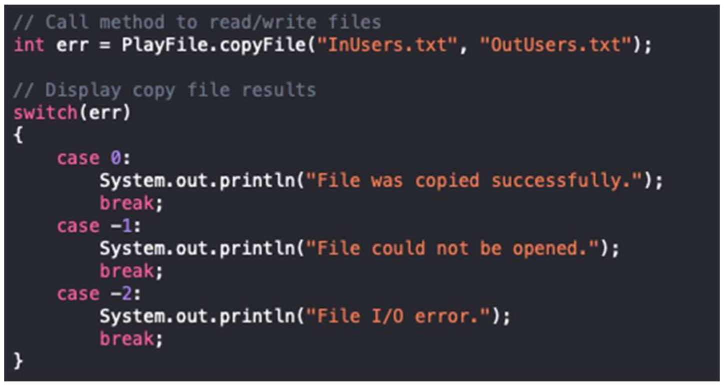
```
// Input and Output File REad/Writer declarations
FileReader in = null;
FileWriter out = null;
```



```
// Create Reader and Writer
in = new FileReader(inFile);
out = new FileWriter(outFile);

// Loop to read all characters from FileReader and write to FileWriter
int c;
while((c = in.read()) != -1)
{
    out.write(c);
}
return 0;
```

7. In the *main*() method, display a separate and appropriate error message for each error code (0, -1, -2) using a switch statement from the error code returned from *copyFile*().



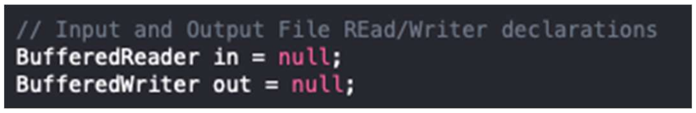
```
// Call method to read/write files
int err = PlayFile.copyFile("InUsers.txt", "OutUsers.txt");

// Display copy file results
switch(err)
{
    case 0:
        System.out.println("File was copied successfully.");
        break;
    case -1:
        System.out.println("File could not be opened.");
        break;
    case -2:
        System.out.println("File I/O error.");
        break;
}
```

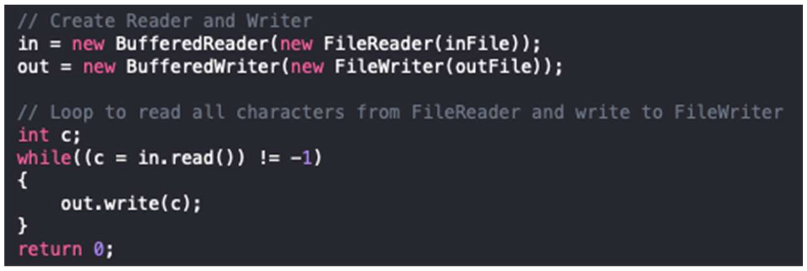
8. Run the application (refresh the project by right clicking on the project and selecting the Refresh menu option).
9. Take a screenshot of the final console output and the resulting output file (*OutFile.txt*) contents.
10. Change the input filename to a file that does not exist.
11. Run the application.
12. Take a screenshot of the final console output that includes the exception stack trace and error message.

Part 1b:

1. Create a new project named *topic4-1b.*
2. Copy all the code from Part 1a into this new project.
3. Update the application to use a *BufferedReader* and *BufferedWriter* (using the *BufferedReader* and *BufferedWriter* constructors that take a *FileReader* and *FileWriter*).



```
// Input and Output File REad/Writer declarations
BufferedReader in = null;
BufferedWriter out = null;
```



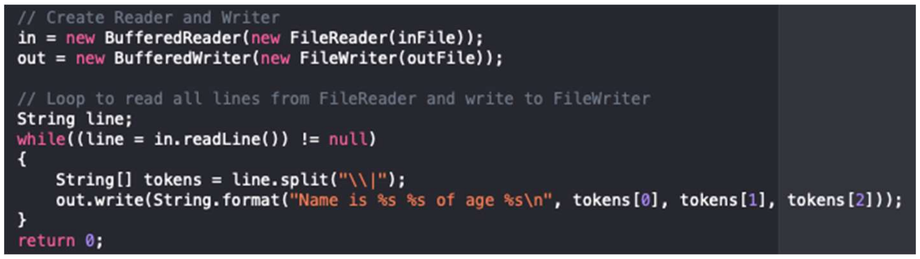
```
// Create Reader and Writer
in = new BufferedReader(new FileReader(inFile));
out = new BufferedWriter(new FileWriter(outFile));

// Loop to read all characters from FileReader and write to FileWriter
int c;
while((c = in.read()) != -1)
{
    out.write(c);
}
return 0;
```

4. Run the application (refresh the project by right clicking on the project and selecting the Refresh menu option).
5. Take a screenshot of the final console output and the resulting output file (*OutFile.txt*) contents.

Part 1c:

1. Create a new project named *topic4-1c.*
1. Copy all the code from Part 1b into this new project.
2. Change the read and write logic to the following:
    a. Read a line from the file using the *readLine*() method.
    b. Use the String *split*() method with a regex delimiter of a pipe (“\\|”).
    c. Write each of the tokens to the output file *OutFile.txt* using a *String.format("Name is %s %s of age %s\n")* method.



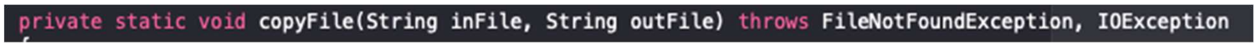
```
// Create Reader and Writer
in = new BufferedReader(new FileReader(inFile));
out = new BufferedWriter(new FileWriter(outFile));

// Loop to read all lines from FileReader and write to FileWriter
String line;
while((line = in.readLine()) != null)
{
    String[] tokens = line.split("\\|");
    out.write(String.format("Name is %s %s of age %s\n", tokens[0], tokens[1], tokens[2]));
}
return 0;
```

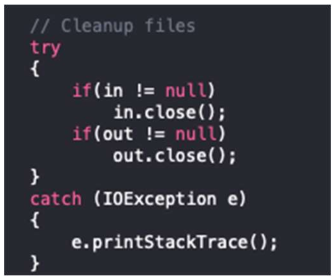
3. Run the application (if there are errors, refresh the project by right clicking on the project and selecting the Refresh menu option).
4. Take a screenshot of the final console output and the resulting output file (*OutFile.txt*) contents.

Part 1d:

1. Create a new project named *topic4-1d.*
2. Copy all the code from Part 1c into this new project.
3. Remove the exception handling from *copyFile*() method and use the *throws* clause in the *copyFile*() method signature to throw the *FileNotFoundException* and the *IOException*. Make sure to add the proper cleanup logic to close all the files.
4. Change the method signature of *copyFile*() to return void.

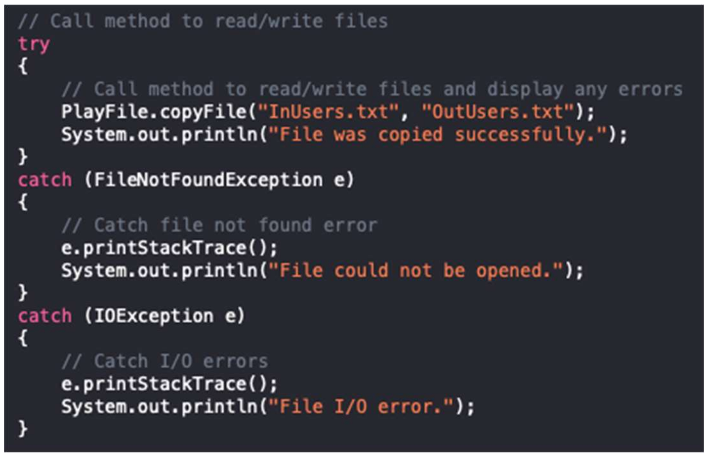


```java
private static void copyFile(String inFile, String outFile) throws FileNotFoundException, IOException
```



```java
// Cleanup files
try
{
    if(in != null)
        in.close();
    if(out != null)
        out.close();
}
catch (IOException e)
{
    e.printStackTrace();
}
```

5. Surround the call to the *copyFile*() method in the *main*() method with the appropriate try catch blocks and with the proper error message display. Remove the error switch statement.



```java
// Call method to read/write files
try
{
    // Call method to read/write files and display any errors
    PlayFile.copyFile("InUsers.txt", "OutUsers.txt");
    System.out.println("File was copied successfully.");
}
catch (FileNotFoundException e)
{
    // Catch file not found error
    e.printStackTrace();
    System.out.println("File could not be opened.");
}
catch (IOException e)
{
    // Catch I/O errors
    e.printStackTrace();
    System.out.println("File I/O error.");
}
```

6. Run the application (if there are errors, refresh the project by right clicking on the project and selecting the Refresh menu option).
7. Take a screenshot of the final console output and the resulting output file (*OutFile.txt*) contents.
8. Generate the JavaDoc for all classes.

Deliverables:

The following need to be submitted as this part of the activity:

- ✓ a. All screenshots of application in operation.
- ✓ b. ZIP file of the code in the project folder. Include the JavaDoc generated for the project.

Part 2: Reading and Writing JSON Files

Overview

Goal and Directions:

In this activity, you will learn how to read, write, and serialize JSON files to and from Java objects.

There are a number of libraries available to work with JSON data. One of these is an open source library called Jackson. Read the following tutorials:

- http://tutorials.jenkov.com/java-json/jackson-objectmapper.html
- https://www.tutorialspoint.com/jackson/index.htm
- https://www.baeldung.com/jackson-inheritance

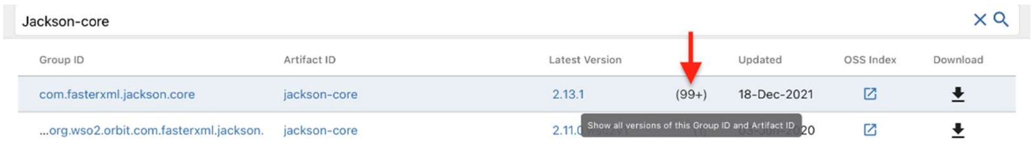
JSON is often hard to read (because returned JSON is stripped of all the linefeeds). To work with JSON so that it is readable, you can use a website such as JSON Formatter & Validator (at https://jsonformatter.curiousconcept.com/). Simply copy your JSON into the site and click the Process button.

Complete the following tasks for this activity:

Execution

1. Read the tutorial:
   - Go to http://tutorials.jenkov.com/java-json/jackson-objectmapper.html.
   - Complete the following parts of the tutorial:
     a. Jackson DataBind (and the Jackson Installation)
     b. How Jackson ObjectMapper (and the Jackson Annotations)
     c. Jackson Annotations
     d. Read Object From JSON String
     e. Read Object From JSON Reader
     f. Read Object From JSON File
     g. Read Object From JSON InputStream
     h. Read Object Array From JSON Array String
     i. Read Object List From JSON Array String
     j. Write JSON From Objects
2. Create a new project named *topic4-2*.
3. Add the Jackson JSON libraries to the new project by using the following steps:
   - To get the Jackson Library, go to https://search.maven.org/, in search bar enter jackson-core, click the search icon, and under the latest version of the library click

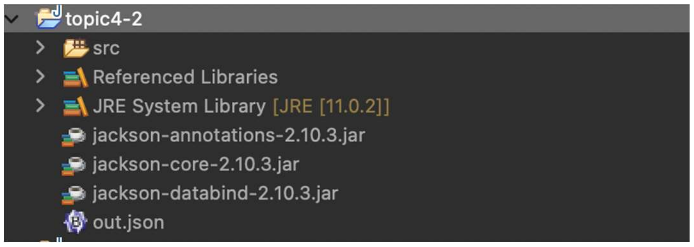
the link to the right of the latest version, and download the 2.10.5 JAR file, repeat and download the JAR files for jackson-annotations-2.10.5 and jackson-databind-2.10.3.


Jackson-core

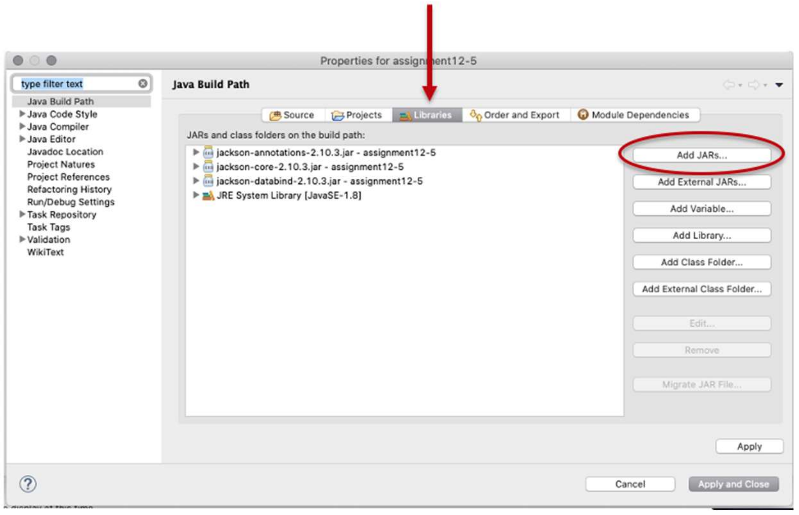
Group ID	Artifact ID	Latest Version		Updated	OSS Index	Download
com.fasterxml.jackson.core	jackson-core	2.13.1	(99+)	18-Dec-2021		
...org.wso2.orbit.com.fasterxml.jackson.	jackson-core	2.11.[illegible]		[illegible]20		

Show all versions of this Group ID and Artifact ID

- Copy jackson-core-2.10.5.jar, jackson-annotations-2.10.5.jar, and jackson-databind-2.10.3.jar to the root of your project as shown in the figure below:


```
topic4-2
  > src
  > Referenced Libraries
  > JRE System Library [JRE [11.0.2]]
    jackson-annotations-2.10.3.jar
    jackson-core-2.10.3.jar
    jackson-databind-2.10.3.jar
    out.json
```

- Right click on the project, select Build Path → Configure Build Path, under the Libraries tab, click the Add JARS button, and select the jackson-core-2.10.5.jar, jackson-annotations-2.10.5.jar, and jackson-databind-2.10.3.jar files copied to your project, click Apply and Close button.

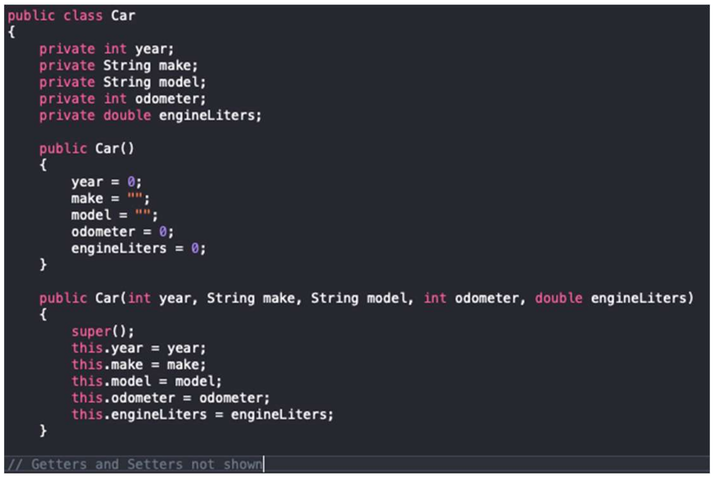

Properties for assignment12-5

Java Build Path

JARs and class folders on the build path:

```
▶ jackson-annotations-2.10.3.jar - assignment12-5
▶ jackson-core-2.10.3.jar - assignment12-5
▶ jackson-databind-2.10.3.jar - assignment12-5
▶ JRE System Library [JavaSE-1.8]
```

4. Create a new Java Class named *Car* in the *app* package with a *main*() method with the private properties of *year*, *make*, *model*, *odometer*, and *engineLiters*, with a default and non-default constructor, and with getter and setter methods for each property.



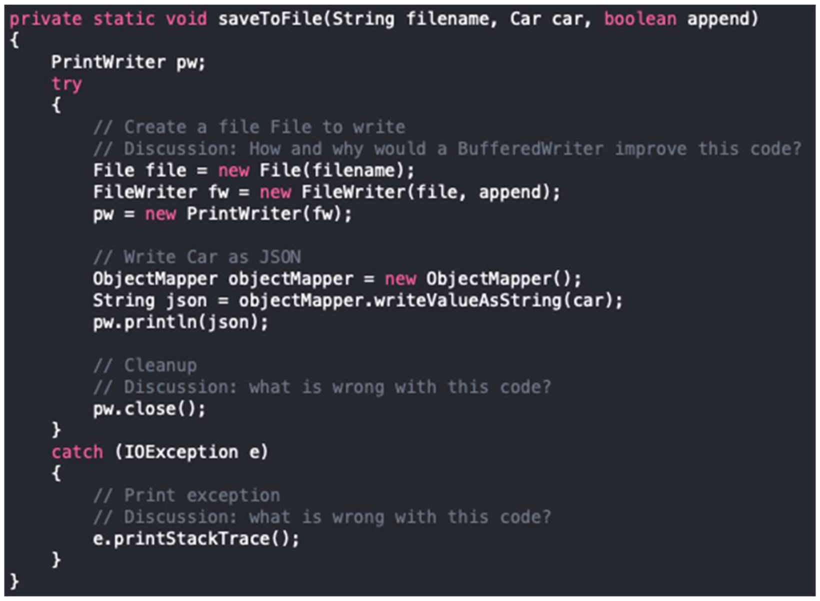
```
public class Car
{
    private int year;
    private String make;
    private String model;
    private int odometer;
    private double engineLiters;

    public Car()
    {
        year = 0;
        make = "";
        model = "";
        odometer = 0;
        engineLiters = 0;
    }

    public Car(int year, String make, String model, int odometer, double engineLiters)
    {
        super();
        this.year = year;
        this.make = make;
        this.model = model;
        this.odometer = odometer;
        this.engineLiters = engineLiters;
    }

// Getters and Setters not shown
```

5. Create a new Java Class named *DemoJsonFiles* in the *app* package with a *main*() method.
6. Create a new method named *saveToFile()* in the *DemoJsonFiles* class that will serialize a Car Object to a JSON file using the Jackson library:

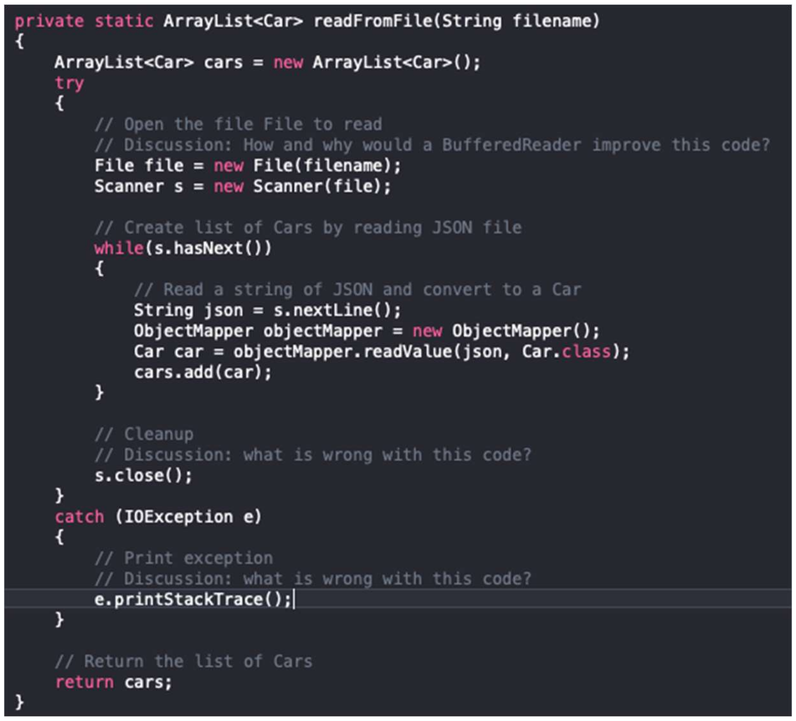


```
private static void saveToFile(String filename, Car car, boolean append)
{
    PrintWriter pw;
    try
    {
        // Create a file File to write
        // Discussion: How and why would a BufferedWriter improve this code?
        File file = new File(filename);
        FileWriter fw = new FileWriter(file, append);
        pw = new PrintWriter(fw);

        // Write Car as JSON
        ObjectMapper objectMapper = new ObjectMapper();
        String json = objectMapper.writeValueAsString(car);
        pw.println(json);

        // Cleanup
        // Discussion: what is wrong with this code?
        pw.close();
    }
    catch (IOException e)
    {
        // Print exception
        // Discussion: what is wrong with this code?
        e.printStackTrace();
    }
}
```

7. Create a new method named *readFromFile()* in the *DemoJsonFiles* class to read JSON from a file and serialize it to a Car Object using the Jackson library:



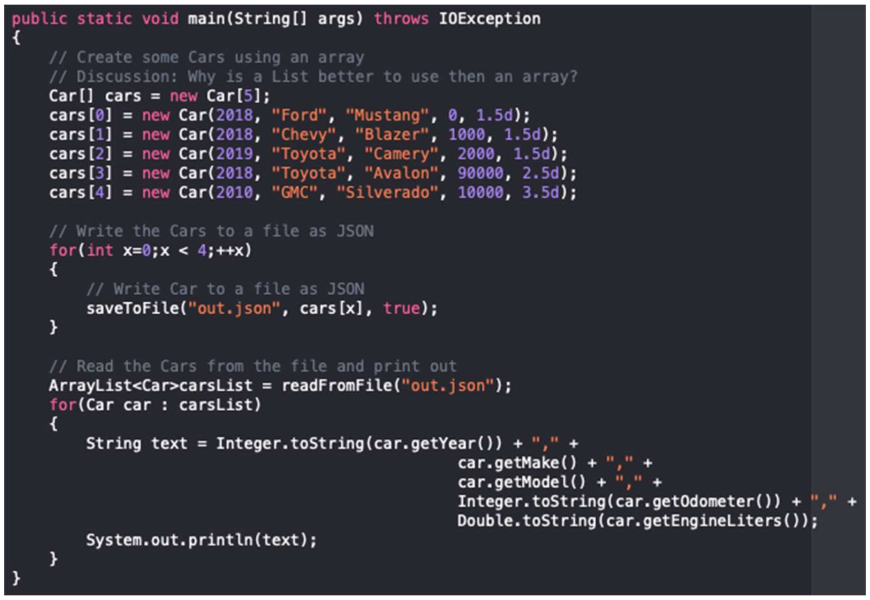
```java
private static ArrayList<Car> readFromFile(String filename)
{
    ArrayList<Car> cars = new ArrayList<Car>();
    try
    {
        // Open the file File to read
        // Discussion: How and why would a BufferedReader improve this code?
        File file = new File(filename);
        Scanner s = new Scanner(file);

        // Create list of Cars by reading JSON file
        while(s.hasNext())
        {
            // Read a string of JSON and convert to a Car
            String json = s.nextLine();
            ObjectMapper objectMapper = new ObjectMapper();
            Car car = objectMapper.readValue(json, Car.class);
            cars.add(car);
        }

        // Cleanup
        // Discussion: what is wrong with this code?
        s.close();
    }
    catch (IOException e)
    {
        // Print exception
        // Discussion: what is wrong with this code?
        e.printStackTrace();
    }

    // Return the list of Cars
    return cars;
}
```

8. Implement the *main()* method in the *DemoJsonFiles* class that will convert an array of Cars to JSON, then read the JSON back, converting it back to an array of Cars and displaying the Cars back to the console.
9. Run the application (if there are errors, refresh the project by right clicking on the project and selecting the Refresh menu option). Inspect the JSON file that was generated by the code with any text editor and beautify it using JSON Pretty Print. Take a screenshot of the final JSON file and a screenshot of the console output after the application has been run.



```
public static void main(String[] args) throws IOException
{
    // Create some Cars using an array
    // Discussion: Why is a List better to use then an array?
    Car[] cars = new Car[5];
    cars[0] = new Car(2018, "Ford", "Mustang", 0, 1.5d);
    cars[1] = new Car(2018, "Chevy", "Blazer", 1000, 1.5d);
    cars[2] = new Car(2019, "Toyota", "Camery", 2000, 1.5d);
    cars[3] = new Car(2018, "Toyota", "Avalon", 90000, 2.5d);
    cars[4] = new Car(2010, "GMC", "Silverado", 10000, 3.5d);

    // Write the Cars to a file as JSON
    for(int x=0;x < 4;++x)
    {
        // Write Car to a file as JSON
        saveToFile("out.json", cars[x], true);
    }

    // Read the Cars from the file and print out
    ArrayList<Car>carsList = readFromFile("out.json");
    for(Car car : carsList)
    {
        String text = Integer.toString(car.getYear()) + "," +
                                                car.getMake() + "," +
                                                car.getModel() + "," +
                                                Integer.toString(car.getOdometer()) + "," +
                                                Double.toString(car.getEngineLiters());
        System.out.println(text);
    }
}
```

Deliverables:

The following need to be submitted as this part of the activity:

a. ✓ All screenshots of application in operation.
b. ✓ ZIP file of the code in the project folder. Include the JavaDoc generated for the project.

Research Questions

1. Research Questions: For traditional ground students. in a Microsoft Word document, answer the following questions:
    a. Explain the difference between (compile time) checked and (runtime) unchecked exceptions. When would you design a checked exception? When would you design an unchecked exception? Summarize your answers and rationale in 300 words.
    b. What is pass by reference and pass by value mean in the Java programming language? Does Java pass by reference or pass by value? Provide code examples to show both cases. Summarize your answers and explanation for how your code examples work in 300 words.

Final Activity Submission

1. In a Microsoft Word document, complete the following for the Activity Report:
    a. Cover sheet with the name of this assignment, date, and your name.
    b. Section with a title that contains all the diagrams, screenshots, and theory of operation write-ups.
    c. Zip file with all code and generated JavaDoc documentation files.
    d. Section with a title that contains the answers to the Research Questions (traditional ground students only).
2. Submit the Activity Report and zip file of the code and documentation to the Learning Management System (LMS).