Collections Tutorial

In this tutorial, we will practice Java Collections. Create a new maven project and put this in your pom.xml. Make sure you have Java 16 setup.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
	<modelVersion>4.0.0</modelVersion>
	<groupId>tutorial</groupId>
	<artifactId>SwingTutorial</artifactId>
	<version>0.0.1-SNAPSHOT</version>
	<build>
		<plugins>
			<plugin>
				<artifactId>maven-surefire-plugin</artifactId>
				<version>2.22.2</version>
			</plugin>
		</plugins>
	</build>
	<dependencies>
		<dependency>
			<groupId>org.junit.jupiter</groupId>
			<artifactId>junit-jupiter</artifactId>
			<version>5.7.1</version>
			<scope>test</scope>
		</dependency>
	</dependencies>

	<properties>
		<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
		<maven.compiler.source>16</maven.compiler.source>
		<maven.compiler.target>16</maven.compiler.target>
		<java.version>16</java.version>
	</properties>
</project>
```

To make it easy, we will load the CSV file from here
https://sample-videos.com/download-sample-csv.php#google_vignette
"https://sample-videos.com/csv/Sample-Spreadsheet-1000-rows.csv"

Look over the data to get some sense of it. It seems like some products are associated with price, territory, and seller agent.

TASK 1: load CSV file

If you do not know, then google it. We need a URL, and we need to process it line by line. We can get the URL as an input stream and adapt it to a Reader. We want it fast, so we add buffering. Also, URL resource might have a specific encoding:

```
public final void loadData() {
	URL url = null;
	BufferedReader in = null;
	try {
		url = new URL("https://sample-videos.com/csv/Sample-Spreadsheet-1000-rows.csv");
		// this works on Cerny's  try UTF-8 if messy chars
		in = new BufferedReader(new InputStreamReader(url.openStream(),
									StandardCharsets.ISO_8859_1));

		String inputLine;
		while ((inputLine = in.readLine()) != null) {
			System.out.println(inputLine);
		}
	} catch (MalformedURLException e2) {
		e2.printStackTrace();
	} catch (IOException e1) {
		e1.printStackTrace();
	} finally {
		try {
			if(in != null) {
				in.close();
			}
		} catch (IOException e) {
			e.printStackTrace();
		}
	}
}
```

Here is a method that can load the data and process them line by line. Remember to deal with exceptions and close the stream! The encoding StandardCharsets.*ISO_8859_1* worked for me. Try to see if it works for you or apply a different one if it does not.

TASK 2: make this function reusable.
The easiest way to make something reusable is to make a template. Using OOP, we can use inheritance. So pay attention to the above fragment and note the line System.out.println(inputLine); Since we will replace this with a method to process the line by line. This method will be virtual and abstract like this:

```
protected abstract void processLine(String tokens);
```

So we can replace

System.out.println(inputLine); → processLine(inputLine);

Then we place this all into a class that will be abstract and meant to be extended.

```
public abstract class URLLoader {
```

So next, we can easily extend this fragment.

TASK 3: Apply template

Try to create a new class extending from URLLoader to tell us how many lines were loaded.

Before you look below, try to do it yourself.

```
public class MyListDemo extends URLLoader {

	protected List<String> list = new ArrayList<>();

	@Override
	protected void processLine(String tokens) {
		list.add(tokens);
	}


	public static void main(String[] args) {
		MyListDemo myListDemo = new MyListDemo();
		myListDemo.loadData();
		System.out.println(myListDemo.list.size());
	}

}
```

TASK 4: Apply filter

Let's filter only one territory in the data CSV.

First, we need to fragment the data to recognize columns, but how?
This will do it `inputLine.split(",");`
But we can also change the process line to deal with String[] array. Next, we can update our list to be `List<List<String>> list.` It is populated as this: `list.add(Arrays.asList(tokens));`

Once we have that, we can add a condition for the territory.

```
public class MyListDemo2 extends URLLoader {

	protected List<List<String>> list = new ArrayList<>();

	@Override
	protected void processLine(String[] tokens) {
		if(tokens[7].equalsIgnoreCase("British Columbia")) {
			list.add(Arrays.asList(tokens));
		}
```

```
	}

	public static void main(String[] args) {
		MyListDemo2 myListDemo = new MyListDemo2();
		myListDemo.loadData();

		System.out.println(myListDemo.list.size());
	}

}
```

Easy, right?

TASK 5: Export our results to XLS

Seems like a task beyond our options. No, just add this to your pom under dependencies.

```
	<dependencies>
		<dependency>
			<groupId>org.apache.poi</groupId>
			<artifactId>poi</artifactId>
			<version>3.9</version>
		</dependency>
		<dependency>
			<groupId>org.apache.poi</groupId>
			<artifactId>poi-ooxml</artifactId>
			<version>3.9</version>
		</dependency>
		..
	</dependencies>
```

Now we added the XLS library. It is easy:

```
protected void createXLS() {
	try {
		XSSFWorkbook workbook = new XSSFWorkbook();
		XSSFSheet sheet = workbook.createSheet("sheet1");// creating a blank sheet
		int rownum = 0;
		for (List<String> line : list) {
			Row row = sheet.createRow(rownum++);
			createList(line, row);
		}
		FileOutputStream out = new FileOutputStream(new File("NewFile3.xlsx")); // file name with path
		workbook.write(out);
		out.close();
	} catch (Exception e) {
		e.printStackTrace();
	}
}
```

This creates a new sheet and calls createList(line, row); to add lines there. Then it makes a new file there. out.close() would be better in the final section, but for a demo, it is ok.

To populate row cells, we operate with Cell

```
private void createList(List<String> line, Row row) {
		Cell cell = row.createCell(0);
		cell.setCellValue(line.get(0));

		cell = row.createCell(1);
		cell.setCellValue(line.get(1));

		cell = row.createCell(2);
		cell.setCellValue(line.get(2));

		cell = row.createCell(3);
		cell.setCellValue(line.get(3));

		cell = row.createCell(4);
		cell.setCellValue(line.get(5));

		cell = row.createCell(5);
		cell.setCellValue(line.get(7));

		cell = row.createCell(6);
		cell.setCellValue(line.get(8));

	}
```

Nothing hard to get.. Finally call it:

```
public static void main(String[] args) {
		MyListDemo3 myListDemo = new MyListDemo3();
		myListDemo.loadData();

		System.out.println(myListDemo.list.size());
		myListDemo.createXLS();

	}
```

Run and check the workspace (remember to refresh your project F5 or right-click and refresh)

Task 5: Make our C code-like design to OOD

Let's extract Class for the data we operate within the CSV.

```
package edu.baylor.cs;

public class Product {
	private Long id;
	private String name;
	private String agentName;
```

```
	private Long agentId;
	private Double price;
	private String territory;
	private String category;

	public Long getId() {
		return id;
	}
	public void setId(Long id) {
		this.id = id;
	}
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
	public String getAgentName() {
		return agentName;
	}
	public void setAgentName(String agentName) {
		this.agentName = agentName;
	}
	public Long getAgentId() {
		return agentId;
	}
	public void setAgentId(Long agentId) {
		this.agentId = agentId;
	}
	public Double getPrice() {
		return price;
	}
	public void setPrice(Double price) {
		this.price = price;
	}
	public String getTerritory() {
		return territory;
	}
	public void setTerritory(String territory) {
		this.territory = territory;
	}
	public String getCategory() {
		return category;
	}
	public void setCategory(String category) {
		this.category = category;
	}

}
```

Next, apply this in:

```
@Override
```

```
protected void processLine(String[] tokens) {
        if (tokens[7].equalsIgnoreCase("British Columbia")) {

                Product product = new Product();
                product.setId(Long.parseLong(tokens[0]));
                product.setName(tokens[1]);
                product.setAgentName(tokens[2]);
                product.setAgentId(Long.parseLong(tokens[3]));
                product.setPrice(Double.parseDouble(tokens[5]));
                product.setTerritory(tokens[7]);
                product.setCategory(tokens[8]);

                list.add(product);
        }

}
```

And

```
private void createList(Product product, Row row) {
        Cell cell = row.createCell(0);
        cell.setCellValue(product.getId());

        cell = row.createCell(1);
        cell.setCellValue(product.getName());

        cell = row.createCell(2);
        cell.setCellValue(product.getAgentName());

        cell = row.createCell(3);
        cell.setCellValue(product.getAgentId());

        cell = row.createCell(4);
        cell.setCellValue(product.getTerritory());

        cell = row.createCell(5);
        cell.setCellValue(product.getCategory());

}
```

Task 6: How many unique product names are there?

It is a good question. How do we do it? Think a minute.

We can check the list we are inserting into if the product name exists. But is it so expensive! How about a Set?

```
protected Set<String> resultSet = new HashSet<>();
```

We still load the list of things, but add one more call to our main applySearch()

```
public static void main(String[] args) {
		MySetDemo5 myListDemo = new MySetDemo5();
		myListDemo.loadData();

		myListDemo.applySearch();

		System.out.println(myListDemo.list.size());
		myListDemo.createXLS();
		System.out.println(myListDemo.resultSet.size());

}
```

Here we use the set and make Product.class -> productName mapping reduction.

```
for (Product product : list) {
	resultSet.add(product.getName());
}
```

As you see in the main, we print set size.

Why did we use HashSet? Well doesn't really matter here, but it should be your default option as it is fast.

Task 7: CSV cell trouble?

What if the cell contains something like this:
example "Eldon Base for stackable storage shelf, platinum"
This should be a cell but
inputLine.split(",");
It is not working well.

So we need a better split by a RegExp; you can ask Google to do it, but here is what I found.

```
inputLine.split(",(?=(?:[^\"]*\"[^\"]*\")*[^\"]*$)", -1);
```

It works well, but still, it leaves the quotes there! And worse, some quotes are part of the cell text. Here: "Acme Design Line 8"" Stainless Steel Bent Scissors w/Champagne Handles, 3-1/8"" Cut"

So how to do it? Replace? ReplaceAll? One takes a sequence, and the other takes RegExp.

```
protected String[] split(String inputLine) {
		String[] tokens = inputLine.split(",(?=(?:[^\"]*\"[^\"]*\")*[^\"]*$)", -1);
		for (int i = 0; i < tokens.length; i++) {
			if (tokens[i].contains("\"")) {
				tokens[i] = tokens[i].replaceAll("\"\"", "\"");
				tokens[i] = tokens[i].replaceAll("^\"", ""); // beginning
				tokens[i] = tokens[i].replaceAll("\"$", ""); // end
```

```
                }
            }
            return tokens;
        }
```

Then update the loadData() with
processLine(split(inputLine));

All works as before, but we get correct data values.

Task 8: Which products are exactly once on the list?

This is a great question, we know all products, and we all know unique names, but it has nothing to do with products that are there once. How do you approach it?

How about we check if the unique set already contains it? If so, it is a duplicate. Then we take all products and remove those we know are duplicates. Will that work? Yes, what a trick!

```
protected List<Product> list = new ArrayList<>();
protected Set<String> all = new HashSet<>();
protected Set<String> duplicates = new HashSet<>();
protected Set<String> oneOccurence = null;
```

Then use the check on uniqueness: if(!all.add(name)) - Check the documentation on what this means, so you understand it!

```
private void applySearch() {
            for (Product product : list) {
                String name = product.getName();
                if (!all.add(name)) { // Check documentation on what this means
                    duplicates.add(name);
                }
            }
        }
```

When printing results, we can do this to remove duplicates from all

```
oneOccurence = new HashSet<>(all);
oneOccurence.removeAll(duplicates);
for (String result : oneOccurence) {
        Row row = sheet.createRow(rownum++);
        Cell cell = row.createCell(0);
        cell.setCellValue(result);
}
```

And print all statistics at the main end:

```
System.out.println("Size all: " + myListDemo.list.size());
System.out.println("Size unique: " + myListDemo.all.size());
```

```
System.out.println("Size duplicates: " + myListDemo.duplicates.size());
System.out.println("Size one occurence: " + myListDemo.oneOccurence.size());
```

Task 9: Sort the product names

One way to do it is explicit sort.

```
List<String> sorted = new ArrayList<>(oneOccurence);
                        Collections.sort(sorted);
```

Another is implicit sort ;)

```
oneOccurence = new TreeSet<>(all);
oneOccurence.removeAll(duplicates);
```

The TreeSet here is meant to do the sort for us. Since it is a String, we do not need to make our own comparator!

Task 10: How many times are the products repeated?

It seems like our set could contain a pair of values of product name and count, indicating repetition count.

Wait a minute, that is what Map does!

```
protected List<Product> list = new ArrayList<>();
protected Map<String, Integer> map = new HashMap<>();
```

And replace the add method with put and basic init logic.

```
private void applySearch() {
        for (Product product : list) {
                String name = product.getName();
                Integer count = map.get(name);
                if (count == null) {
                        count = 0;
                }
                map.put(name, ++count);
        }
}
```

That is it, though to print it, we need to do

```
for (Entry<String,Integer> entry : map.entrySet()) {
        System.out.println(entry.getKey() +" "+entry.getValue());
}
```

Task 11: Sort how many times the products are repeated by the count!

So we need to sort? How about TreeMap? It sorts! But it sorts keys, not values :|

We need to sort by value and keep it that way! So some sort of linked list? Linked map?

How about this? We convert the map to a list entry, sort it by value, and make new LinkedHashMap

```
public static LinkedHashMap<String, Integer> sortByValue1(Map<String, Integer> map) {
		List<Entry<String, Integer>> list = new ArrayList<>(map.entrySet());
		list.sort(Entry.comparingByValue());

		LinkedHashMap<String, Integer> result = new LinkedHashMap<>();
		for (Entry<String, Integer> entry : list) {
			result.put(entry.getKey(), entry.getValue());
		}

		return result;
	}
```

It works awesome, but we want a reverse order! So we need our own comparator.

```
public static LinkedHashMap<String, Integer> sortByValue(Map<String, Integer> map) {
		List<Entry<String, Integer>> list = new ArrayList<>(map.entrySet());
		list.sort(Entry.comparingByValue());
		list.sort(new Comparator<Map.Entry<String, Integer>>() {

			@Override
			public int compare(Entry<String, Integer> o1, Entry<String, Integer> o2) {
				if(o2.getValue().equals(o1.getValue())) {
					return o1.getKey().compareTo(o2.getKey());
				} else {

					return o2.getValue().compareTo(o1.getValue());
				}
			}
		});


		LinkedHashMap<String, Integer> result = new LinkedHashMap<>();
		for (Entry<String, Integer> entry : list) {
			result.put(entry.getKey(), entry.getValue());
		}

		return result;
	}
```

So that is it!

To print it, just use:

```
			LinkedHashMap<String, Integer> sortedMap = sortByValue(map);
			for (Entry<String, Integer> result : sortedMap.entrySet()) {
```

```
                    Row row = sheet.createRow(rownum++);
                    Cell cell = row.createCell(0);
                    cell.setCellValue(result.getKey());
                    cell = row.createCell(1);
                    cell.setCellValue(result.getValue());

                }
```

Next, try it!

Task 12: Use generics!

Would you know how to convert sortByValue into a generic? Try it, please, on your own, and look below once you are done or lost.

```
public static <K extends Comparable<? super K>, V extends Comparable<? super V>> LinkedHashMap<K, V>
sortByValue(Map<K, V> map) {
        List<Entry<K, V>> list = new ArrayList<>(map.entrySet());
//      list.sort(Entry.comparingByValue());
        list.sort(new Comparator<Map.Entry<K, V>>() {

            @Override
            public int compare(Entry<K, V> o1, Entry<K, V> o2) {
                if(o2.getValue().equals(o1.getValue())) {
                    return o1.getKey().compareTo(o2.getKey());
                } else {

                return o2.getValue().compareTo(o1.getValue());
                }
            }
        });


        LinkedHashMap<K, V> result = new LinkedHashMap<>();
        for (Entry<K, V> entry : list) {
            result.put(entry.getKey(), entry.getValue());
        }

        return result;
    }
```

Task 12: Group by territory

We want to group all results by a territory such as

British Columbia	Product 1
	Product 2
	Product 3

With streams, it is a piece of cake!

We need a list and map.

```
protected List<Product> list = new ArrayList<>();
protected Map<String, List<Product>> map = null;
```

With streams, we use group by and collect.

```
private void applySearch() {
	map = list.stream().collect(Collectors.groupingBy(Product::getTerritory));
}
```

To print, we do a little trick with the first row.

```
		for (Entry<String, List<Product>> result : map.entrySet()) {
			Row row = sheet.createRow(rownum++);
			Cell cell = row.createCell(0);
			cell.setCellValue(result.getKey());
			boolean skipLine = true;
			for (Product product : result.getValue()) {
				if (skipLine) {
					skipLine = false;
				} else {
					row = sheet.createRow(rownum++);
				}
				cell = row.createCell(1);
				cell.setCellValue(product.getName());
				cell = row.createCell(2);
				cell.setCellValue(product.getTerritory());
			}

		}
```

Task 12: Group by sorted by product name

We can extend the stream with sort fed by lambda comparator:

```
private void applySearch() {
		// list.stream().sorted((o1, o2)->o1.setName().compareTo(o2.setName()));
		map = list.stream()
			.sorted((o1, o2)->o1.getName().compareTo(o2.getName()))
			.collect(Collectors.groupingBy(Product::getTerritory));
	}
```

Task 13: Territory with summed costs of products

With streams, easy to accomplish

```
map = list.stream()
	.sorted((o1, o2)->o1.getTerritory().compareTo(o2.getTerritory()))
	.collect(Collectors.groupingBy(Product::getTerritory,Collectors.summingDouble(Product::getPrice)));
```

And simple print

```
for (Entry<String, Double> result : map.entrySet()) {
        Row row = sheet.createRow(rownum++);
        Cell cell = row.createCell(0);
        cell.setCellValue(result.getKey());
        cell = row.createCell(1);
        cell.setCellValue(result.getValue());
}
```

That is all! You should be more confident now about how to use Collections.