Advanced Programming

Lab 5

Streams of bytes / characters

- **Data** may be represented either:
  - Binary (pdf, png, mp3, etc.) or
  - Text (txt, xml, json, etc.)
- **I/O Streams** are responsible with reading/writing data from/in external files.
  - InputStream, OutputStream → **bytes (8 bits)**
  - Reader, Writer → **characters (16 bits)**
- Depending on their job, streams are:
  - **Primitive**: FileReader, StringWriter, etc.
  - **Decorators**: BufferedReader, ObjectInputStream, etc.

The *Main* Class

```
public class Main {

  public static void main(String args[]) {
    Main app = new Main();
    app.testRepo();
    app.testLoadView();
  }

  private void testRepo() {
    var repo = new Repository("c:/documents");

    var service = new RepositoryService();
    service.print(repo);
    service.export(repo, "c:/repository.json");


    var doc = repo.findDocument("...");
    service.view(doc);

    ...
  }
}
```

Create separate classes for model and logic.

You may want to specify a key (id) for a document.

Using *record* classes

```
public record Person (int id, String name)  {
}
```

```
var p = new Person(1001, "Popescu"); //generated constructor

System.out.println(p); //toString implementation

System.out.println(p.name()); //accesor methods
```

Immutable by Default

Automatic Field Accessors

Compact Syntax

Transparent Implementation of toString(), equals(), and hashCode()

Support for Additional Methods

Final Semantics

Compiler-Generated Constructors

The *Repository* Class

```
public class Repository {

    private String directory;
    private Map<Person, List<Document>> documents = new HashMap<>();

    public Repository(String directory) {
        this.directory = directory;
        loadDocuments();

}

    private void loadDocuments() {
        // Read all sub-directories
        // c:/documents/Popescu_1001, ...

        // Read all files in the sub-directories
        // diploma_bac.pdf, copie_buletin.png, ...
    }
    ...

    }
```

java.nio.Files.walk()

The *Service* Class

using JSON serialization

```
public class RepositoryService {

  public void export(Repository repo, String path)
                                             throws IOException {

    ObjectMapper objectMapper = new ObjectMapper();
    objectMapper.writeValue(
        new File(path),
        repo);
  }

  public Repository read(String path)
                                              throws InvalidCatalogException {
    ObjectMapper objectMapper = new ObjectMapper();
    Catalog catalog = objectMapper.readValue(
        new File(path),
        Repository.class);

  }


}
```

```
<dependency>
        <groupId>com.fasterxml.jackson.core</groupId>
        <artifactId>jackson-databind</artifactId>
        <version>2.13.2</version>
</dependency>
```

Custom Exceptions

```
public class InvalidRepositoryException extends Exception {

    public InvalidRepositoryException(Exception ex) {
        super("Invalid repository.", ex);
    }

}
```

Or you can extend *RuntimeException.*
Checked vs. Uncecked

The Algorithm

- We want to enumerate all **maximal** cliques in a graph
- Determining a **maximum** clique is NP-hard.
- The **Bron–Kerbosch** algorithm is an algorithm for finding all maximal cliques in an undirected graph. A clique in a graph is a subset of vertices where every pair of vertices is connected by an edge. A maximal clique is a clique that cannot be extended by including one more adjacent vertex without violating the clique property.