The Art Gallery Time-Capsule

In this week, you are both an **artist** and a **software archivist**.

You will:

- Generate a random piece of art using math and logic
- Sign it with the exact time it was created
- Seal it into a **time-capsule file** using Java serialization
- Reload it later and **prove that it survived**

What You're Building

Your program will produce **two files**:

1. `signature.txt`
   A text-based artist's signature containing a formatted timestamp.

2. `artwork.ser`
   A serialized Java object that stores the "art" (randomly generated shapes).

A **graphical viewer** will later load these files and display the artwork visually.

Step 1: The Artist's Signature

Every artwork needs a signature.

Your Tasks:

1. **Capture the Current Time**
   - Use `LocalDateTime.now()` from the `java.time` package.

2. **Format It Artistically**
   - Use `DateTimeFormatter`
   - Your format **must include**:
     - The **day name** (e.g., Monday)
     - The **AM/PM marker**

Example:

```
signature.txt
1   === ARTIST'S SIGNATURE ===
2   Art Created On: Sunday, January 25, 2026 - 09:19:38 PM
3   Artist: Java Time-Capsule Generator
4   ========================
```

3. **Write It to a File**
    - File name: **`signature.txt`**
    - Use:
        - `FileWriter`
        - `BufferedWriter`

4. **Important Rules**
    - All file operations **must be inside a `try` block**
    - You **must close** your writer when finished

Buffered writers store data in memory first. If you don't close them, your file may be empty, even if your program ran without errors.

Step 2: Designing the Canvas

Now let's make an object that can survive beyond program execution.

Your Tasks:

1. **Create a Class**
    - Name it: **`ArtCanvas`**

2. **Make It Serializable**
    - Implement `java.io.Serializable`

3. **Store the Artwork**
    - Use an `ArrayList` to store your shape data (e.g., coordinates or descriptions)

Example:

```java
class ArtCanvas implements Serializable {

    // Store coordinate data
    private ArrayList<String> coordinates;

    public ArtCanvas() {
        this.coordinates = new ArrayList<>();
    }

    public void addCoordinate(String coordinate) {
        coordinates.add(coordinate);
    }

    public ArrayList<String> getCoordinates() {
        return coordinates;
    }
}
```

Quick note: What Is Serialization?

Serialization converts a Java object into raw bytes so it can be:

- Saved to disk
- Sent over a network
- Reloaded later exactly as it was

Think of it as **freezing an object in time**. In this case, you will be storing the object of the ArtCanvas class itself.

Step 3: Painting with Logic

Now for the fun part, creating the art itself.

Your Tasks:

1. **Generate Random Values**
    - Use `java.util.Random` (not `Math.random()`)

2. **Create Coordinates**
    - Generate random X and Y values
    - Ensure they are **positive**
        - Use `Math.abs()`

3. **Work with Shapes**
    - Create circular shapes using:
        - Radius
        - Area formula using `Math.PI`

4. **Clean the Data**
    - Use:
        - `Math.round()` to make values neat and readable

5. **Store Each Shape**
    - Add each generated shape to your `ArtCanvas` object

Step 4: Sealing the Time-Capsule

Your artwork is complete—now seal it for the future.

Your Tasks:

1. **Serialize the Artwork**
    - File name: **`artwork.ser`**
    - Use:
        - `FileOutputStream`
        - `ObjectOutputStream`
2. **Verify the Save**
    - Immediately read the file back using:
        - `FileInputStream`
        - `ObjectInputStream`
3. **Confirm Success**
    - Print:
        - Number of shapes loaded
        - A few sample entries
4. **Close All Streams**
    - Just like text files, object streams **must be closed**

Quick note: Why verify?

Serialization can fail silently if:

- The class isn't serializable
- The file is corrupted
- Streams aren't closed properly

Bonus: Viewing the Artwork

You'll find **`GraphicalArtViewer.java`** in the repository.

What It Does:

- Reads `signature.txt`
- Deserializes `artwork.ser`
- Parses shape data
- Displays the artwork using Java Swing

Important:
The viewer **will not work** unless your program correctly generates both files.