AP Computer Science A - Unit 7 Final Project

Creative Portfolio & Recursive Art Generator

Estimated Time: 6-8 hours
Due Date: [Insert Date]
Points: 100 points

🎯 Project Overview

You will create a **Creative Portfolio Management System** that showcases digital artwork generated through recursive algorithms. This project combines all Unit 7 concepts: project planning, object-oriented design, method overloading/overriding, private methods, and recursion.

Your system will:

- Generate beautiful recursive art patterns (fractals, trees, spirals)
- Manage a portfolio of created artworks
- Allow users to save, load, and categorize their creations
- Provide detailed analytics about the portfolio

📋 Project Requirements

Core Components (Must Implement All)

1. **Project Planning Documentation** (10 points)

Create a comprehensive project plan including:

- **Project Backlog:** List of 15+ prioritized tasks
- **Benchmarks:** 4 major milestones with specific deliverables
- **User Stories:** 8+ user stories following the format: "As a [user type], I want [goal] so that [benefit]"

2. **ArtWork Class Hierarchy** (25 points)

```java
// Base class
public abstract class ArtWork {
    // Instance variables, constructors, methods
}

// Subclasses (implement at least 3)
public class FractalArt extends ArtWork { }
public class RecursiveTree extends ArtWork { }
public class SpiralArt extends ArtWork { }
```

Requirements:

- Abstract base class with at least 5 instance variables
- 3+ concrete subclasses representing different art types
- Method overriding for `display()`, `calculateComplexity()`, and `getDescription()`
- Overloaded constructors (at least 2 per class)
- Private helper methods for calculations

3. **Portfolio Management System** (25 points)

```java
public class ArtPortfolio {
    // Manage collection of artworks
}
```

Features:

- Add/remove artworks (method overloading for different ways to add)
- Search functionality (by type, date, complexity, etc.)
- Sort artworks using object references as parameters
- Generate portfolio statistics
- Save/load portfolio data

4. **Recursive Art Generation** (25 points)

Implement **at least 3** recursive algorithms:

Option A: Fractal Tree

```java
public void drawTree(Graphics g, int x, int y, double angle, int depth) {
    // Base case: stop when depth reaches 0
    // Recursive case: draw branch and call recursively for sub-branches
}
```

Option B: Sierpinski Triangle

```java
public void drawSierpinski(Graphics g, int x1, int y1, int x2, int y2, int x3, int y3,
    // Base case: draw triangle when depth is small enough
    // Recursive case: divide into smaller triangles
}
```

Option C: Spiral Patterns

```java
public void drawSpiral(Graphics g, int centerX, int centerY, double radius, double ang
    // Base case: stop when radius is too small
    // Recursive case: draw current segment and recurse
}
```

5. **User Interface & Interaction** (10 points)

- Text-based menu system for user interaction
- Clear display of generated art (ASCII art or simple graphics)
- User input validation and error handling
- Help system explaining how to use the program

6. **Code Quality & Documentation** (5 points)

- Proper JavaDoc comments for all public methods
- Clear variable names and code organization
- Follows Java naming conventions
- Includes intellectual property considerations

📖 Detailed Specifications

ArtWork Class Hierarchy

Base ArtWork Class:

java

```java
public abstract class ArtWork {
    protected String title;
    protected String artist;
    protected Date creationDate;
    protected int complexityLevel;
    protected String description;

    // Abstract methods to be overridden
    public abstract void display();
    public abstract int calculateComplexity();
    public abstract String getArtType();

    // Concrete methods
    public boolean equals(Object other) { /* Override properly */ }
    public String toString() { /* Override properly */ }
}
```

Example Subclass Implementation:

java

```java
public class FractalArt extends ArtWork {
    private int iterations;
    private double scaleFactor;
    private String fractalType; // "mandelbrot", "julia", "tree", etc.

    // Multiple constructors (overloading)
    public FractalArt(String title, String artist) { }
    public FractalArt(String title, String artist, int iterations) { }
    public FractalArt(String title, String artist, int iterations, String type) { }

    @Override
    public void display() {
        // Use recursion to generate and display the fractal
    }

    @Override
    public int calculateComplexity() {
        return calculateFractalComplexity(iterations); // Private helper method
    }

    // Private helper methods
    private int calculateFractalComplexity(int depth) {
        // Recursive calculation of complexity
    }

    private void generateFractal(int x, int y, int depth) {
        // Recursive fractal generation
    }
}
```

Portfolio Management System

java

```java
public class ArtPortfolio {
    private ArrayList<ArtWork> artworks;
    private String portfolioName;
    private static int totalPortfolios = 0; // Static variable

    // Method overloading for adding artworks
    public void addArtwork(ArtWork art) { }
    public void addArtwork(String title, String artist, String type) { }
    public void addArtwork(ArtWork[] artArray) { }

    // Object references as parameters
    public void sortByComplexity(ArrayList<ArtWork> artList) {
        // Sort the actual list, not a copy
    }

    public ArtWork findMostComplex(ArrayList<ArtWork> artList) {
        // Find artwork with highest complexity using recursion
    }

    // Static methods
    public static int getTotalPortfolios() { return totalPortfolios; }

    // Portfolio analytics
    public void generateReport() {
        // Use private helper methods to calculate statistics
    }

    private double calculateAverageComplexity() { }
    private Map<String, Integer> getArtTypeDistribution() { }
}
```

Required Recursive Algorithms

1. Recursive Tree (Minimum Requirement):

- Start with a trunk
- At each branch point, create 2-3 sub-branches at different angles
- Each sub-branch is shorter than its parent
- Stop when branches become too small (base case)
- Allow user to specify depth and branching factor

2. Choose One Additional:

- **Fractal Spiral:** Recursive spiral that gets smaller at each iteration
- **Koch Snowflake:** Classic fractal with recursive edge replacement
- **Dragon Curve:** Recursive curve that folds on itself
- **Custom Fractal:** Your own creative recursive pattern

3. Recursive Helper Functions:

- `calculateTotalComplexity()` - recursively sum complexity of all artworks
- `findArtworkByTitle()` - recursive search through portfolio
- `generateArtworkID()` - recursive ID generation system

🎨 Creative Extensions (Bonus Points)

Level 1 Extensions (+5 points each)

- **Color Variations:** Add color properties and recursive color blending
- **Animation Simulation:** Show "frames" of recursive generation step-by-step
- **Export Functionality:** Save artwork descriptions to text files
- **Portfolio Themes:** Group artworks by themes with special display

Level 2 Extensions (+10 points each)

- **Recursive Maze Generator:** Create mazes using recursive backtracking
- **L-System Plants:** Implement Lindenmayer systems for plant-like structures
- **Interactive Parameters:** Allow users to modify recursive parameters in real-time
- **Portfolio Sharing:** Import/export portfolios between users

Level 3 Extensions (+15 points each)

- **Genetic Art Evolution:** Use recursion to "evolve" artwork parameters
- **3D Recursive Structures:** Extend to three-dimensional recursive art
- **Performance Analytics:** Measure and display recursive algorithm efficiency
- **AI Art Critique:** Simulate an AI that analyzes and comments on artwork

---

📅 Project Timeline & Benchmarks

Week 1: Planning & Foundation

Benchmark 1: Project Setup Complete

- [ ] Complete project planning documentation
- [ ] Set up base ArtWork class structure
- [ ] Implement basic Portfolio class
- [ ] Create simple menu system

Week 2: Core Implementation

Benchmark 2: Basic Functionality Working

- [ ] Complete ArtWork subclasses with proper inheritance
- [ ] Implement at least one recursive art algorithm
- [ ] Add/remove artwork functionality working
- [ ] Basic portfolio display working

Week 3: Advanced Features

Benchmark 3: Full Feature Set Complete

- [ ] All 3 recursive algorithms implemented
- [ ] Portfolio search and sort functions working
- [ ] Proper method overloading throughout system
- [ ] Object equality and proper toString() methods

Week 4: Polish & Testing

Benchmark 4: Production Ready

- [ ] Complete user interface with error handling
- [ ] All documentation and JavaDoc comments
- [ ] Testing completed with sample data
- [ ] Creative extensions implemented

📊 Grading Rubric

Component	Excellent (A)	Good (B)	Satisfactory (C)	Needs Work (D/F)
Project Planning (10 pts)	Comprehensive plan with detailed user stories, clear backlog, measurable benchmarks	Good planning with most elements present	Basic planning with some missing elements	Minimal or missing planning
Class Design (25 pts)	Excellent inheritance hierarchy, proper overriding, clean design	Good OOP design with minor issues	Basic inheritance with some problems	Poor design or major issues
Portfolio System (25 pts)	Full-featured system with all requirements plus extras	Most features working well	Basic functionality present	Minimal functionality
Recursion (25 pts)	3+ complex recursive algorithms working perfectly	2-3 algorithms with good implementation	1-2 basic recursive functions	Poor or missing recursion
UI & Polish (10 pts)	Intuitive interface, excellent user experience	Good interface with minor issues	Basic but functional interface	Poor or confusing interface
Code Quality (5 pts)	Excellent documentation, style, and organization	Good coding practices	Adequate code quality	Poor coding practices

Bonus Points: Up to 30 additional points for creative extensions

---

🚀 Getting Started Guide

Step 1: Create Your Project Plan

1. Open a document and create your project backlog
2. Write user stories from different perspectives:
   - "As an artist, I want to generate fractal trees so that I can explore natural patterns"
   - "As a portfolio manager, I want to sort artworks by complexity so that I can organize exhibitions"
3. Define your 4 benchmarks with specific, measurable goals

Step 2: Set Up Your Class Structure

```java
// Start with this basic structure
public abstract class ArtWork {
    // Your base class implementation
}

public class ArtPortfolio {
    private ArrayList<ArtWork> artworks = new ArrayList<>();
    // Your portfolio implementation
}

public class ArtGenerator {
    public static void main(String[] args) {
        // Your main program loop
    }
}
```

Step 3: Implement Your First Recursive Algorithm

Start with a simple recursive tree:

```java
public void drawTree(int x, int y, int length, double angle, int depth) {
    if (depth == 0) return; // Base case

    // Calculate end point of current branch
    int endX = x + (int)(length * Math.cos(angle));
    int endY = y + (int)(length * Math.sin(angle));

    // Draw current branch (or add to description)
    System.out.println("Branch from (" + x + "," + y + ") to (" + endX + "," + endY +

    // Recursive calls for sub-branches
    drawTree(endX, endY, length*0.7, angle - 0.3, depth - 1);
    drawTree(endX, endY, length*0.7, angle + 0.3, depth - 1);
}
```

Step 4: Build Incrementally

- Test each component as you build it
- Use your benchmarks to track progress

- Ask for help when needed
- Document your code as you write it

💡 Tips for Success

Planning Tips

- **Break it down:** Don't try to build everything at once
- **Start simple:** Get basic functionality working first
- **Test frequently:** Make sure each piece works before adding complexity
- **Document as you go:** Don't leave all documentation for the end

Programming Tips

- **Trace your recursion:** Draw out the recursive calls on paper
- **Use meaningful names:** Your variable and method names should be descriptive
- **Handle edge cases:** What happens with invalid input?
- **Keep methods focused:** Each method should do one thing well

Creative Tips

- **Research real fractals:** Look up examples online for inspiration
- **Experiment with parameters:** Small changes can create dramatically different results
- **Think about user experience:** How can you make your program enjoyable to use?
- **Consider performance:** Some recursive algorithms can be slow with deep recursion

---

📤 Submission Requirements

What to Submit:

1. **All Java source files** (.java files)
2. **Project planning document** (PDF or Word)
3. **README file** explaining how to run your program
4. **Sample output** showing your program in action
5. **Reflection essay** (500 words) discussing:
   - Challenges you faced
   - How you used each Unit 7 concept

- What you learned about recursive algorithms
- Ideas for future improvements

File Organization:

```
YourName_Unit7_Project/
├── src/
│   ├── ArtWork.java
│   ├── FractalArt.java
│   ├── RecursiveTree.java
│   ├── SpiralArt.java
│   ├── ArtPortfolio.java
│   └── ArtGenerator.java
├── docs/
│   ├── ProjectPlan.pdf
│   ├── README.md
│   └── SampleOutput.txt
└── reflection/
    └── ProjectReflection.pdf
```

❓ Frequently Asked Questions

Q: How complex should my recursive algorithms be? A: Focus on correctness first, then complexity. A simple but correctly implemented recursive tree is better than a complex algorithm that doesn't work.

Q: Can I use graphics libraries for the art display? A: You can, but it's not required. ASCII art or text descriptions of the artwork are perfectly acceptable.

Q: How do I handle the "intellectual property" requirement? A: Include comments about any algorithms you research online. Cite your sources and explain what you learned vs. what you implemented yourself.

Q: What if I can't finish all the extensions? A: Focus on the core requirements first. Extensions are bonus points - it's better to have a solid core implementation than a partially working system with attempted extensions.

Q: How do I test my recursive methods? A: Start with small values! Test with depth 1, then 2, then 3. Trace through the recursive calls on paper to make sure your logic is correct.

🏆 Example Project Ideas for Inspiration

"The Fractal Forest"

- Generate recursive trees with different seasons (colors, leaf patterns)
- Portfolio organized by "forest sections"
- Recursive weather patterns affecting tree growth

"Geometric Dreams"

- Focus on mathematical fractals (Mandelbrot, Julia sets)
- Portfolio analytics show mathematical properties
- Recursive zoom functionality

"Nature's Patterns"

- Combine multiple natural recursive patterns
- Lightning bolts, river systems, leaf veins
- Portfolio tracks "natural inspiration" themes

"Abstract Expressionism"

- Recursive color blending and shape generation
- Portfolio organized by artistic movements
- Recursive composition analysis

Remember: This project is your chance to showcase everything you've learned in Unit 7. Take your time, plan carefully, and don't hesitate to ask questions. Good luck creating something amazing!

"The best way to learn recursion is to understand recursion. The best way to understand recursion is to learn recursion." - Anonymous Computer Scientist