

Welcome to

CS 351

Design of Large Programs

Instructor: **Joel Castellanos**

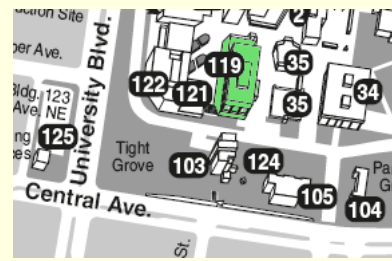
e-mail: joel@unm.edu

Web: <http://cs.unm.edu/~joel/>

Office: Farris Engineering
Center (FEC) room 321

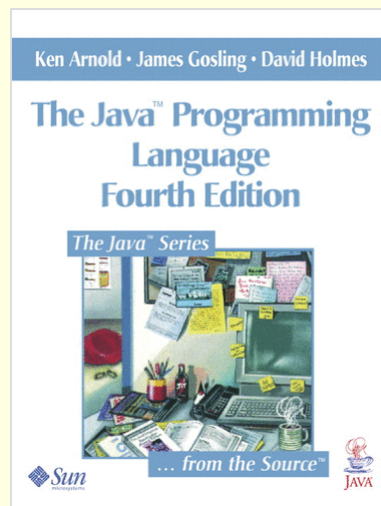
Lab Instructor: **David Godinez**

e-mail: dgodinez@cs.unm.edu



1/23/2009

Course Resources



Class website:

<http://cs.unm.edu/~joel/cs351/>

- Syllabus
- Projects
- Lecture Notes
- Supplemental readings

WebCT:

- Assignment Drop-box
- Assignment discussions (blogs)
- Grades

2

Course Description

- This class is about designing big software, where big refers to projects with a scope too large to be handled by
 - any one person (communication with collaborators)
 - at any one time (communication with "ghosts" of past and "apparitions" of future)
- This course primarily deals with software design, time management, and strategies for completing complex coding tasks.
- Java and Eclipse Integrated Development Environment (IDE)

3

Java Rocks



- Advantages for Large Programs
- JUnit Testing
- JavaDocs
- Object-Oriented
- Platform-independent
- Secure
- Robust (lot of emphasis on early checking for possible errors)
- Multithreaded
- Well suited for Automated Optimization



4

Java Performance - wikipedia

Programs written in Java have had a reputation for being slower and requiring more memory than those written in natively compiled languages such as C or C++.

However, Java programs' execution speed has improved significantly due to introduction of Just-In Time compilation (in 1997/1998 for Java 1.1), addition of language features supporting better code analysis, and optimizations in the Java Virtual Machine itself (such as HotSpot becoming the default for Sun JVM in 2000).

5

Micro benchmarks vs. Scientific Code

C/C++ (small)

- Hand optimized, Pointers are quick
- Native floating-point operations.
- No error checking
- Quick startup

Java (big)

- C-style pointers make optimization hard in languages that support them (particular fine grain parallelism)
- Adaptive optimization is impossible in fully compiled code.
- Auto *Garbage Collection* beats programmed GC.

6

Course Grading

- 50% Programming Projects. Three individual, one group.
- 20% Exams (midterm and final)
- 15% laboratory quizzes and class programming assignments.
- 15% Lecture quizzes (i-clicker), group project presentations, and group project presentation evaluations.

Late projects/assignments will receive a 5% per day penalty.

7

Working Together but do not Cheat

- Working together and helping one another on all projects is highly encouraged. This includes discussion of project:
 - *specification,*
 - *algorithms,*
 - *data structures,*
 - *and test cases.*
- Do *not* share code.
- It is considered cheating to leave your code (paper or electronic copies) where others can find it. You responsible for the security of your intellectual property.

8

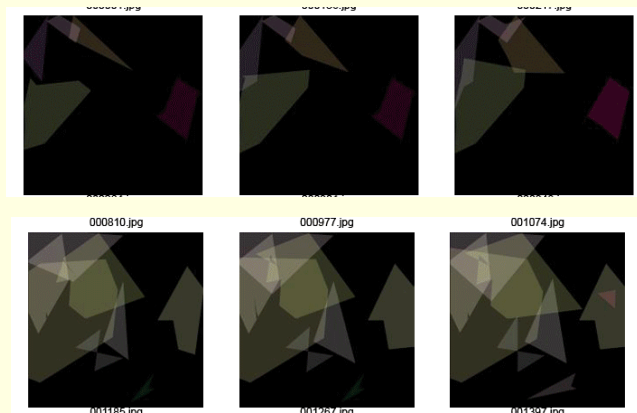
Project 1: Scientific Modeling

- This project is the implementation of a science based model.
- Externally visible classes must follow a strict interface specification document
- Must compile and run with a pre-existing (instructor written) Graphical User Interface (GUI).
- JUnit testing must be used.
- Three week project with milestones.

9

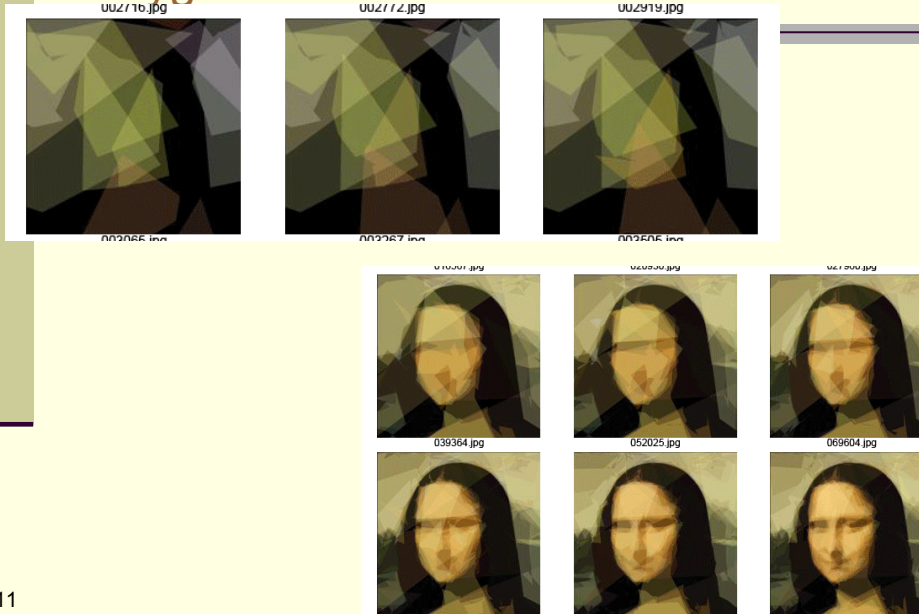
Project 2: Genetic Algorithm applied to Image Processing

Could you paint a replica of the Mona Lisa using only 50 semi transparent polygons?



10

Ploygons



11

Project 3: Agent Based Simulation

- Group project
- Six weeks (all semester)
- Very specific case of a common real-world problem. The topic will be such that without detailed design, success will be nigh impossible. The project will be many facetted including:
 - Research
 - Design.
 - Data collection.
 - Data representation.
 - Objects with complex and well defined interactions.
 - Graphical user input.
 - Graphical display of data, model dynamics and results.
 - Unit testing
 - Integration testing (does your code do what you say it does?)
 - Model Validation (predictability)

12