C Sc 335 Course Overview

Object-Oriented Programming and Design

© Rick Mercer
Major Topics in C Sc 335

1. Java
2. Object-Oriented Programming
3. Object-Oriented Design
4. Technology
5. Object-Oriented Principles
6. Software Development
7. Team Project
1. Java

- Classes and Interfaces
- Exceptions, Streams, Persistence
- Graphical Components
- Event-driven programming
  - Make something happen on a click, mouse motion, window close, checkbox....
- Socket Networking
- Concurrency with Java Threads
2. Object Oriented Programming

- Encapsulation / Modularity
  - keeping data and behavior together

- Inheritance
  - Capture common data and behavior in a class, then let other classes extend it

- Polymorphism
  - via interfaces and inheritance
3. Object-Oriented Design

- Design Guidelines such as
  - Assign a responsibility to the object that has the necessary information, high cohesion, low coupling

- Object-Oriented Design Patterns such as
  - Iterator
  - Strategy
  - Adaptor
  - Decorator
  - Composite
  - Mediator
  - Command
  - Observer
3. **OO Design** continued

- Responsibility Driven Design (RDD)
- Unified Modeling Language (UML)
- Test Driven Design (TDD)
- Refactoring
  - Improving the design of existing code without changing its meaning—make it more readable and maintainable, a few examples:
    - Rename, Extract method, Exit method as soon as possible, Change method signature
4. Technology

- Professional IDE: Eclipse
- Concurrent Versioning System (CVS)
- Use existing frameworks
  - Java's Collection Framework
  - javax.swing, javax.awt
  - java.io
  - java.net
5. Object-Oriented Principles

- The Single Responsibility Principle
- The Open–Closed Principle
- The Dependency Inversion Principle
- The Liskov Substitution Principle
- Favor composition over inheritance
- Encapsulate what varies
- Program to interfaces, not implementations
6. Software Development

- We'll use a mash up of Agile techniques
  - Test Driven Development (TDD)
  - Short iterations
  - Coding standard and collective code ownership
  - Pair programming
  - Frequent build updates
  - Sustainable pace
  - Estimating and planning
  - Retrospectives
7. *Team Project*

- Great project choices has each person designing and developing 50-80 hours each over the final six weeks
  - Can get high marks with 4x50 hours
- Teams of four
- Some rough estimates
  - A few interfaces
  - 15-25 classes
  - 3,500 to 6,500 lines of code (LoC)
No Text Book to buy

- There is no one textbook for this class
- There will be readings of online content, some views of videos and
- Selected readings are from Safari Books Online
  
  
  - You need to be at a UofA computer or establish a Virtual Private Network (VPN) connection on your machine, UofA has a Cisco solution for free
    
    [https://vpn.arizona.edu](https://vpn.arizona.edu)
  
  - Gain access to thousands of technical books
Goals

- Understand and use the fundamentals of object-oriented programming: encapsulation, polymorphism, and inheritance
- Understand the relationships between objects, classes, and interfaces
- Build complex systems where one has 15 or more classes that you develop in a team of four
Goals (continued)

- Learn to work on teams
- Use good practices of programming to develop good object-oriented software
- Become comfortable with event-driven programming and graphical user interfaces
- Use tools of object-oriented software development
  - Design Patterns, the Unified Modeling Language (UML), unit testing (JUnit), a professional IDE (Eclipse), frameworks, Agile techniques
Goals continued

- Value TDD and see how it helps design and provide confidence in correctness
- Write clean code
- Be able to make intelligent design decisions
- Build a project that is better than the sum of the parts (team project is greater than what 1 person can do in the same number of person hours)
- Have some fun
Object-Oriented Technology:

- Outline
  - Consider a few ways in which data is protected from careless modification
  - Mention the key features object-oriented style of software development
  - Consider why object-oriented technology is important
Chapter 1: Beating the Software Crisis

- We continue to become more and more dependent on information
  - Their ability to manage data decreases
  - The problem is the software, not the hardware

- The Software crisis
  - How often is software delivered on time, under budget, and does what it’s supposed to?
  - The software is not flexible enough to handle rapid changes
How Software is Constructed

- Wanted:
  - robust large-scale applications that evolve with the corporation
- It isn’t easy!
- Modular Programming
  - Break large-scale problems into smaller components that are constructed independently
  - Programs were viewed as a collection of procedures, each containing a sequence of instructions
Modular Programming

- Subroutine (1960s)
  - Provided a natural division of labor
  - Could be reused in other programs
  - Took decades for acceptance (like OO)

- Structured Programming and Design (1970s)
  - It was considered a good idea to program with a limited set of control structures (no go to statements, single returns from functions)
    - sequence, selection, repetition, recursion
  - Program design was at the level of subroutines
    - functional decomposition
Functional Decomposition

- main
  - readData
  - mainMenu
  - saveData
    - addRecord
    - editRecord
    - deleteRecord
A Problem with Structured Design

- Structured programming has a serious limitation:
  - It’s rarely possible to anticipate the design of a completed system before it’s implemented
  - The larger the system, the more restructuring takes place
And What About the Data?

- Software development had focused on the modularization of code,
  - the data was either moved around between functions via argument/parameter associations
  - or the data was global
    - works okay for smaller programs or for big programs when there aren't too many global variables
    - Not good when variables number in the hundreds
Don’t use Global Variables

- Sharing data (global variables) is a violation of modular programming
- This makes all modules dependent on one another
  - this is dangerous
Information (Data) Hiding

- An improvement:
  - Give each subroutine its own local data
  - This data can only be “touched” by that single subroutine
  - Subroutines can be designed, implemented, and maintained independently

- Other necessary data is passed amongst the procedures via argument/parameter associations.
Use functions

- Localize data inside the functions
- This makes functions more independent of one another
  - Local Data
The Structured Design Approach

- The procedural style of programming, using structured design builds systems one subroutine at a time
  - This approach doesn’t work well in large systems
  - The result is defective software; difficult to maintain

- There is a better way, the OO way:
  - A 10,000 statement program (in 1959) becomes structured with 1,000 functions (in 1989) where there are 10 statements in each function, which then becomes (in 1999), a program with 100 classes, with 10 functions each, 10 statements each

“The one thing we missed was putting the data and functions together”
Larry Constantine, author of Structured Design
Object-Oriented Style of Design and Programming

Object-Oriented Programming

- Encapsulation
  - Hide details in a class, provide methods

- Polymorphism
  - Same name, different behavior, based on type

- Inheritance
  - Capture common attributes and behaviors in a base class and extend it for different types
Object-Oriented Technology

- OOT began with Simula 67
  - developed in Norway
  - acronym for simulation language
- Why this “new” language?
  - to build accurate models of complex working systems
- The modularization occurs at the physical object level (not at a procedural level)
Simula 67 was designed for system simulation (in Norway by Kristen Nygaard and Ole-Johan Dahl)
- Caller and called subprogram-equal relationship
- First notion of objects including class-instance distinctions
• Nygaard and Dahl invented inheritance
• Won the 2002 Turing Award
• Were to accept it at OOPSLA 11-2002
Alan Kay developed **Smalltalk (1980)**
- The first commercial OO programming language
- Won 2005 Turing award

In 1968 Kay created a very interesting concept—the Dynabook. He wanted to make **A Personal Computer For Children Of All Ages**—a thin portable computer, highly dynamic device that weighed no more than two pounds.

**SPD / OOPD**

- **Structured programming and design**
  - Systems are modeled as a collection of functions and procedures that pass data all over the place

- **Object-oriented programming and design**
  - Systems are modeled as a collection of interacting objects where the data is encapsulated with the methods that need and operate on those values
    - Repeating “The one thing we missed was putting the data and functions together”
      - Larry Constantine *Structured Programming*
Objects

- An object is:
  - a software “package” that contains a collection of related methods and data
  - an excellent software module
  - an instance of a class

- We understand an object through:
  - the values the object stores (state)
  - the services the object provides (behavior)
Modeling an Automated Vehicle

- Consider how we would model the control system for an automated guided vehicle (AGV):
  - Behaviors:
    - move from one location to another
    - loading and unloading contents
  - Must maintain information about
    - its inherent characteristic: pallet size, lifting capacity, maximum speed, ...
    - its current state: contents, location, velocity, ...
One instance of a vehicle

- Every object has
  - Identity: A way to distinguish objects
  - Attributes: values stored in computer memory
  - Behavior: The service the objects provide
Messages

- Real-world objects exhibit many effects on each other.
- These interactions are represented in software as messages (a.k.a. method or function calls).
- A message has these three things:
  - *sender*: the initiator of the message
  - *receiver*: the recipient of the message
  - *arguments*: data required by the receiver
Example messages

- Smalltalk examples
  
  *sender is the object sending the message--sender is the context from where the message is sent (a method)*
  
  ```smalltalk
  vehicle104 moveTo:binB7
  myAccount withdraw:100.00
  ```

- Java examples
  
  ```java
  vehicle104.moveTo(binB7);
  myAccount.withdraw(100.00);
  ```

- An object-oriented program has objects interacting with other objects by sending messages
Encapsulation via Classes

- We often need many of the same objects within a program many
  - Numbers, Strings, BankAccounts, Employees …
- We need an efficient way to redefine the same thing many times
  - and hide the details from other objects
- The class defines the behaviors and attributes
- Each object (instance) has its own state — the set of values for that instance.
One Class can Generates Many Objects

- One class makes many instances (objects)
  - Each object has its own values and identity
Polymorphism

- A single name can represent different behaviors based on the type of the object
- Polymorphism is possible
  - via Java interfaces (or pure virtual C++ classes)
    - compareTo
    - List
  - via inheritance
    - toString
    - equals
**Inheriting Class Information**

- Imagine systems that need several classes of similar automated Vehicle, or BankAccount, or Employee objects
- New classes can be derived from existing ones through the inheritance
  - The "top" class is called the superclass (or base class)
  - The new class is called a subclass (or derived class)
Inheritance

- The superclass should pull together the common characteristics
- Subclasses inherit methods & attributes
- Derived classes add data and methods
- Derived classes can override some methods to give the method new meaning
  - In Java, you often override these methods of class Object:
    - `toString` equals
Inheritance Hierarchies

- Classes can be extended to any degree
- Each new derived class accumulates data and methods of the classes it extends
- Much human knowledge is structured in this manner (insects, inventory items, lendables)
- We use software models of real world objects
- The base class captures common data and behavior
Two Small Class Hierarchies

In UML

BankAccount
  ├── SavingsAccount
  │      ├── PalletAGV
  │      └── RollAGV
  └── CheckingAccount
      └── BusinessAccount

Software needed to control these AVs
More Recent OOPLs

- C became C++ in the mid 80s
  - AT&T (Bjarne Stroustrup)
  - Added classes to the popular C language
  - Hybrid -- both procedural and object-oriented
- Ada became OO in 95 (first ISO standard)
- Java started in the mid 90s
  - Looked like C++
- C# (Microsoft)
  - Looked a lot like Java
  - `int`, `double`, `char` look like objects now
    - SmallTalk had already done this in the 80s
- Java and C# are similar
  - Both added generics `<Type>` for example
OO Analysis and Design

- New methods and technologies for analysis, design, and testing have been created to deal with larger systems
  - Use cases help analyze and capture requirements
  - Responsibility-Driven Design helps determine which objects are needed and what they must do
  - Test Driven Development aids in implementation, design, and maintenance of software
  - Unified Modeling Language (UML) used to design and document large systems
The OO approach

- More natural—models real world, not computers
- More flexible
- More maintainable
  - People can understand it better
  - touch as few classes as possible when enhancing or fixing
- More reliable – can perform unit testing
- Can build reusable modules (classes)
- Extensible: can add functionality and offer alternatives
Promises of the OO Approach

- OOT offers
  - Techniques for creating flexible, natural software modules.
  - Systems that are much easier to adapt to new demands
  - Reuse shortens the development life cycle
  - Systems are more understandable
    - easier to remember 50 real objects rather than 500 functions