College of San Mateo
Official Course Outline

1. **COURSE ID:** CIS 255 **TITLE:** (CS1) Programming Methods: Java **C-ID:** COMP 122
   Units: 4.0 units **Hours/Semester:** 48.0-54.0 Lecture hours; 48.0-54.0 Lab hours; and 96.0-108.0 Homework hours
   Method of Grading: Grade Option (Letter Grade or Pass/No Pass)
   Prerequisite: MATH 120, and CIS 254

2. **COURSE DESIGNATION:**
   Degree Credit
   Transfer credit: CSU; UC
   AA/AS Degree Requirements:
   CSM - COMPETENCY REQUIREMENTS: C1 Math/Quantitative Reasoning Basic Competency
   CSM - GENERAL EDUCATION REQUIREMENTS: E2b. Communication and Analytical Thinking

3. **COURSE DESCRIPTIONS:**
   Catalog Description:
   Object-oriented programming methodology for both computer science majors and computer professionals. Systematic approach to design, construction, and management of computer programs; emphasizing program documentation, testing, debugging, maintenance and software reuse. Also includes evolution of programming languages and emergence of paradigms, UML, virtual machines, exception handling, sorting and searching algorithms, recursion, inheritance, polymorphism, fundamental graphics, and computer ethics. This course conforms to the ACM CS1 standards. A materials fee in the amount shown in the Schedule of Classes is payable upon registration.

4. **STUDENT LEARNING OUTCOME(S) (SLO'S):**
   Upon successful completion of this course, a student will meet the following outcomes:
   1. Demonstrate knowledge and understanding of programming paradigms and the principal object-oriented programming concepts.
   2. Design, implement, and use classes, interfaces, and methods, employing standard naming conventions and advanced features including exception handling, 1/O, GUIs, and event handling.
   3. Employ object-oriented methodology to design and effectively implement medium-sized computer programs using simple Unified Modeling Language (UML) notation.
   4. Decompose a real-world problem and apply strategies for the reuse of existing components with inheritance and polymorphism.
   5. Describe the concept of recursion, and implement, test, and debug simple recursive methods.
   6. Explain and employ basic sorting and searching algorithms.
   7. Use and create standard API documents to understand and document the use of classes and methods.
   8. Demonstrate an understanding of professional codes of ethics, such as ACM and IEEE.

5. **SPECIFIC INSTRUCTIONAL OBJECTIVES:**
   Upon successful completion of this course, a student will be able to:
   1. Demonstrate knowledge and understanding of programming paradigms and the principal object-oriented programming concepts.
   2. Design, implement, and use classes, interfaces, and methods, employing standard naming conventions and advanced features including exception handling, I/O, GUIs, and event handling.
   3. Employ object-oriented methodology to design and effectively implement medium-sized computer programs using simple Unified Modeling Language (UML) notation.
   4. Decompose a real-world problem and apply strategies for the reuse of existing components with inheritance and polymorphism.
   5. Describe the concept of recursion, and implement, test, and debug simple recursive methods.
   6. Explain and employ basic sorting and searching algorithms.
   7. Use and create standard API documents to understand and document the use of classes and methods.
   8. Demonstrate an understanding of professional codes of ethics, such as ACM and IEEE.

6. **COURSE CONTENT:**
   Lecture Content:
   1. Introduction

- A. History and evolution of programming languages
- B. Procedural languages
- C. Programming paradigms
- D. Compilers and interpreters
- E. Object-oriented programming
    - a. Object-oriented methodology
    - b. Object-oriented design
    - c. Software tools
- F. Machine-level representation of data

2. Object-Oriented Design
    - A. Concept of design patterns
    - B. Use of API's
    - C. Modeling tools
        - a. UML
    - D. Virtual Machines
    - E. Encapsulation
    - F. Structure decomposition
3. Fundamental Data and Data Structures
    - A. Built-in
        - a. Primitive data types
        - b. Arrays
        - c. Vectors/ArrayLists
        - d. Strings/StringBuilders
        - e. Generic classes
    - B. Variable scope and binding
    - C. Abstract Data Types
    - D. Wrapper classes
    - E. Dynamic data structures
    - F. Simple text files
4. Methods and Control Structures
    - A. Arguments and parameters
    - B. Return types
    - C. Access modifiers
    - D. Method overloading
    - E. Loop control variables
5. Fundamental Computing Algorithms
    - A. Sorting
    - B. Searching
    - C. Recursive algorithms
6. Event-Driven Programming
    - A. Event-handling interfaces/classes
    - B. Event propagation
    - C. Exception handling
7. Fundamental Techniques in Graphics
    - A. Graphical User Interfaces
        - a. Swing components
    - B. Graphics and Graphics2D classes
8. Software Engineering Issues
    - A. Tools
    - B. Processes
    - C. Requirements and specifications
    - D. Design and testing
    - E. Refactoring
9. Inheritance
    - A. Superclasses/subclasses
    - B. Method overriding
    - C. Abstract classes
    - D. Interfaces
    - E. Polymorphism
        - a. Dynamic Binding

10. Computer Ethics
    A. Association for Computing Machinery (ACM)
    B. IEEE

Lab Content:

Students will write weekly short programs (25-175 lines of code) as well as 8-10 projects (200-600 lines of code) using the following:

1. Basic Program Design
    A. Methods
    B. Generic Methods
    C. Recursion
2. Fundamental Data Structures and Data Persistence
    A. Arrays
    B. Vectors/ArrayLists
    C. Strings/StringBuilders
    D. Enumerations
    E. Files and File I/O
3. Fundamental Computing Algorithms
    A. Sorting and searching
    B. Inheritance
    C. Abstract classes
    D. Interfaces
    E. Polymorphism
    F. Dynamic binding
4. Event-Driven Programming
    A. Event-handling interfaces/classes
    B. Event propagation
    C. Exception handling
    D. Graphical User Interfaces
    E. Swing components
    F. Graphics and Graphics2D classes

7. **REPRESENTATIVE METHODS OF INSTRUCTION:**

Typical methods of instruction may include:

A. Lecture
B. Lab
C. Activity
D. Directed Study
E. Critique
F. Discussion
G. Observation and Demonstration
H. Other (Specify): Lecture will be used to introduce new topics; Teacher will model problem-solving techniques. Class will solve a problem together, each person contributing a potential "next step". Students will participate in short in-class projects (in teacher-organized small groups) to ensure that students experiment with the new topics in realistic problem settings; Teacher will invite questions AND ANSWERS from students, generating discussion about areas of misunderstanding; Teacher will create and manage an internet conference for discussion of course topics; and students will work in small groups on programming assignments.

8. **REPRESENTATIVE ASSIGNMENTS**

Representative assignments in this course may include, but are not limited to the following:

Writing Assignments:

The primary writing opportunity for students in this course is documentation supporting their lab and programming projects. This includes both technical documentation and end-user documentation. The technical documentation describes the problem to be solved, the scope of the project, an overview of the solution, and any limitations of the solution. User documentation will be provided to the client of any reusable code.

Reading Assignments:

Weekly textbook readings.

Other Outside Assignments:

Weekly exercises from the textbook and lab/programming assignments comprise the majority of the

assignments. The lab and programming assignments support learning outcomes. In addition, students will create several substantial programs consisting of 500-600 lines of code.

9. **REPRESENTATIVE METHODS OF EVALUATION**
   Representative methods of evaluation may include:
   A. Exams/Tests
   B. Group Projects
   C. Homework
   D. Lab Activities
   E. Projects
   F. Quizzes
   G. Written examination
   H. Bi-weekly quizzes (short answer-from textbook material) to provide feedback to students and teacher; assessment of student contributions during class discussion and project time; individual programming assignments; Midterm and Final exams (short answer, general problem solving (similar to in-class work), short program segments (similar to programming assignments); Assessment of group participation on course projects, including peer assessment of participation and contribution to the group effort.

10. **REPRESENTATIVE TEXT(S):**
    Possible textbooks include:
    A. Deitel. *Java: How to Program*, 11th ed. Pearson, 2017
    B. Liang. *Introduction to Java Programming and Data Structures, Comprehensive*, 11th ed. Pearson, 2017
    C. Eck. *Introduction to Programming Using Java*, 8th ed. http://math.hws.edu/eck/cs124/javanotes8/, 2018

Origination Date: November 2020
Curriculum Committee Approval Date: January 2021
Effective Term: Fall 2021
Course Originator: Mounjed Moussalem