UNIVERSITY OF THE FRASER VALLEY

ORIGINAL COURSE IMPLEMENTATION DATE:	September 2003
REVISED COURSE IMPLEMENTATION DATE:	January 2025
COURSE TO BE REVIEWED (six years after UEC approval)**:**	September 2030

Course outline form version: 28/10/2022

OFFICIAL UNDERGRADUATE COURSE OUTLINE FORM

Note: The University reserves the right to amend course outlines as needed without notice.

Course Code and Number: COMP 155	**Number of Credits:** 4 Course credit policy (105)
Course Full Title: Object-oriented Programming **Course Short Title:** Object-oriented Programming	
Faculty: Faculty of Business and Computing	**Department/School:** Computing

Calendar Description:

Continues the systematic study of programming, focusing on object-oriented programming and design. Introduces large and complex collaborations of objects and covers important skills to design modern-day software systems. Students will reflect on the practice of tómiyeqw and Stó:lō worldview of community practice in relation to the development of object-oriented software.

Prerequisites (or NONE):	C+ or better in one of COMP 150 or COMP 152.
Corequisites (if applicable, or NONE):	None.
Pre/corequisites (if applicable, or NONE):	None.

Antirequisite Courses *(Cannot be taken for additional credit.)*

Former course code/number:

Cross-listed with:

Equivalent course(s):

(If offered in the previous five years, antirequisite course(s) will be included in the calendar description as a note that students with credit for the antirequisite course(s) cannot take this course for further credit.)

Typical Structure of Instructional Hours

Lecture/seminar	45
Supervised laboratory hours (computer lab)	15
Total hours	**60**

Scheduled Laboratory Hours

Labs to be scheduled independent of lecture hours: [✓ No] Yes

Course Details

Special Topics course: **No**

(If yes, the course will be offered under different letter designations representing different topics.)

Directed Study course: **No**

(See policy 207 for more information.)

Grading System: **Letter grades**

Delivery Mode: **May be offered in multiple delivery modes**

Expected frequency: **Every semester**

Maximum enrolment (for information only)**: 35**

Prior Learning Assessment and Recognition (PLAR)

PLAR is available for this course.

Transfer Credit *(See* ***bctransferguide.ca****.)*

Transfer credit already exists: **Yes**

Submit outline for (re)articulation: **No**

(If yes, fill in transfer credit form.)

Department approval	**Date of meeting:**	November 17, 2023
Faculty Council approval	**Date of meeting:**	February 9, 2024
Undergraduate Education Committee (UEC) approval	**Date of meeting:**	September 27, 2024

Learning Outcomes *(These should contribute to students' ability to meet program outcomes and thus Institutional Learning Outcomes.)*

Upon successful completion of this course, students will be able to:

- Describe the fundamentals of object-oriented programming (abstraction, inheritance, encapsulation, and polymorphism).
- Reflect on the practice of tómiyeqw in relation to object-oriented programming fundamentals.
- Develop classes that are maintainable and reusable.
- Document programs to improve readability and maintainability of source codes.
- Express class relationships using current modeling techniques (e.g., UML).
- Apply a library of classes (e.g., Swing, JDBC, I/O, JUnit) to develop object-oriented software systems.
- Reflect on the traditional Stó:lō worldview in relations to Unit testing.
- Write a program of collaborating objects that solves problems like simulations, data organization or structure, game development, accounting problems, etc.

Recommended Evaluation Methods and Weighting *(Evaluation should align to learning outcomes.)*

Final exam:	35%	Quizzes/tests:	30%	Holistic assessment:	10%
Assignments:	10%	Lab work:	15%		%

Details:

NOTE: The following sections may vary by instructor. Please see course syllabus available from the instructor.

Typical Instructional Methods *(Guest lecturers, presentations, online instruction, field trips, etc.)*

The course will be delivered in lecture-lab format, with numerous in-class demonstrations and hands-on activities. The lab portion gives students and the instructor the ability to view and interact with current projects.

Additionally, the following is a list of considerations that are encouraged for this course:

1. Established software development principles and practices (such as inheritance, encapsulation, polymorphism) will be emphasized in alignment with the practice of tómiyeqw in the Stó:lō culture which represents the connection between the past, present, and the future.
2. Students are introduced to unit testing to emphasize that a healthy program is a collaboration of healthy objects (a unit of reusable entity). This aligns with the Stó:lō worldview of any dysfunction on the reserve hurts not just a family but the whole village – everyone feels it.
3. Illustrative examples, case studies, assignments, and projects are encouraged to be drawn from diverse sectors of the economy and students encouraged to choose projects that are influenced or inspired by their lived experiences and passion.
4. Assignments and projects will encourage solving problems within the Indigenous communities.

Texts and Resource Materials *(Include online resources and Indigenous knowledge sources. Open Educational Resources (OER) should be included whenever possible. If more space is required, use the Supplemental Texts and Resource Materials form.)*

	Type	Author or description	Title and publication/access details	Year
1.	Textbook	Koffman, E.B. and Wolfgang, P.A.T.	Data Structures: Abstraction and Design Using Java; 4th Edition; John Wiley & Sons, Inc.	2021
2.	Indigenous knowledge		Group workshop to learn about Tomiyeqw and Stó:lō worldview of community practice for methodology	
3.				

Course Content and Topics

- Background on tómiyeqw and Stó:lō worldview of community practice
- Classes and interfaces
- Abstractions and encapsulation
- Information/data hiding
- Inheritance and polymorphism
- Exception handling
- Generics
- Introduction to modeling (with UML - e.g., class and object diagrams)
- Packages (structuring subsystems)
- Collections and Iterators
- Graphical User Interfaces (GUIs)
- File processing
- Variables and methods (both class and static forms)