CPSC 111 Course Learning Goals

By the end of the course students can…	1. Write and modify code to "express understanding" of basic programming constructs (including sequential exeuction, conditional execution, iteration, arrays, methods/parameter passing, object-orientation and inheritance principles).	2. Read and hand-execute (trace) provided code to "express understanding" of basic programming constructs and memory models (including sequential exeuction, conditional execution, iteration, arrays, methods/parameter passing, object-orientation and inheritance principles).	3. Write code to solve moderately-difficult problems (moderately difficult will be defined through example in an appendix)	4. Recognize, create, and manipulate various models of programs including memory tracing and UML class diagrams	5. Explain Java language features (e.g. classes, visibility, fields, and methods) which support OO design principles such as modularity, encapsulation, abstraction and inheritance.	6. Explain the major components of a computing system and how a program compiles and executes to a non-computer scientist.
Computing Systems (2)						A, B
Programming Language Basics (4)	C, D	C				E, (F?)
Classes and Objects (3)	G, H	G, H, I			I	
Conditionals (3)	J, K, L	J, K, L	(J?, K?), L			
Designing and Defining Classes (4/2)	M, N, O, (Q?)	(O?), R	N, Q	O, R	M, P	
Iteration (3)	S, U	T	S, U	T		
Arrays (3,1)	V, X		V, W, X			
Sorting (2)		Z		(Z?)		AA* (not done by everyone who teaches course)
Advanced Class Design (3)	AB1, AC1, AC3	AB4, AC4, AC5	AD		AB2, AB3, AC2, (AC4?), (AC5?)	
Graphics (2)	AE2, AF				AE1	

Topic	ID	Assessed in?	Goals
			Students can…
Computing Systems (2)	A	M1	define and give real world examples of key components of the computer (input, output, processor, memory).
	B		can distinguish and describe how layers of abstraction are supported in computing problem solving through algorithms, programming languages, assembly, and computer hardware.
Programming Language Basics (4)	C	M1, M2, F, L, A	apply with basic competence simple programming constructs such as sequential execution, variable typing and declaration, naming, algebraic operations, operation precedence.
	D	M1, M2, F, L, A	create programs which translate explicit English problem statements (an algorithm) into short series of sequential Java instructions.
	E		describe the multiple ways in which a natural language paragraph can be interpreted and contrast to the single way an algorithm can be interpreted.
	F		explain why a particular numeric type can only represent numbers in a particular range.
Classes and Objects (3)	G	M1, M2, F	define the relationship between classes and objects.
	H	M1, M2, F, L, A	read and write code utilizing the API of key Java classes (e.g. String, Scanner).
			explain how control flow and data pass on a method call.
	I		identify specific standard methods like accessors and mutators and describe why these operations are needed for non-primitive data types.
Conditionals (3)	J	M2, F, L, A	hand-trace and create programs which use if-statement conditionals to model behavior of input-driven programs.
	K	M2, F, L, A	utilize Boolean expressions, relational, and logical operators to control conditional execution.
	L	M2, F, L, A	utilize block statements, short-circuit evaluation(?), and nested ifs to create code to solve problems in Java.
Designing and Defining Classes (4/2)	M	M1, M2, F, L, A	create a simple class (with instance variables, accessors and setters) utilizing basic components of OO design (including encapsulation, visibility modifiers, and overloading) to model a real world entity (including it's actions and state).
	N	M2, F, L, A	use that class in a simple program.
	O	M2, F,	apply their understanding of references and objects by writing standard constructors and drawing diagrams of memory after an object is constructed.
	P		explain how encapsulation (as implemented with visibility modifiers) supports data integrity and good interface design.
	Q	M2, F, L, A	apply with more expert competence simple programming constructs such as sequential execution, variable typing and declaration, naming, algebraic operations, operation precedence.
	R		describe how reference objects differ from primitive variables and describe problem solving scenarios which are best supported by each.
Iteration (3)	S	M2, F, L, A	solve problems by creating code where repeated actions are controlled with looping structures (for and while loops).
	T	M2, F, L, A	identify and debug a loop that never stops (an infinite loop).
	U	M2, F, L, A	solve problems which requires a loop within a loop where the inner loop iteration does not depend on the outer loop iterator (e.g. to draw a rectangle of stars).
			solve problems which requires a loop within a loop where the inner loop iteration does depend on the outer loop iterator (e.g. to draw a triangle of stars).
Arrays (3)	V	M2, F, L, A	solve problems with collections of same-type data using arrays (including primitive type collections (e.g a collection of class grades) and collections of objects (e.g. a collection of String names or a deck of cards).
	W	M2, F, L, A	apply with more expert competence branching, looping, and nested loops through practice solving problems using arrays and 2-D arrays.
	X	M2, F, L, A	solve problems by creating code which require the creation and use of 2-D arrays (e.g. graphics and averaging scores of students and other data that can be stored in matrix form).
Sorting (2)	Z	F	identify a simple sorting algorithm.
	AA	F	explain that a simple sorting algorithm can be analyzed through simple techniques such as comparison counting and that different sorting algorithms can have different execution time costs and that the number of elements sorted is important in making these analyses.
Advanced Class Design (10)	AB1	M2, F, L, A	create codes which require the use of advanced class syntax and semantics including static methods and variables, scoping, primitive and non-primitive parameter passing.
	AB2		explain the difference between static and non-static fields and give an example of when each should be used.
	AB3		explain the difference between static and non-static methods and give an example of when each should be used.
	AB4		given a piece of code, identify the scope of a variable (locals, class-level, or global).
	AC1	F, L, (A)	create codes which require the use of advanced OO concepts such as inheritance, class hierarchy, and polymorphism.
	AC2		explain how inheritance is a form of code re-use that can be valuable in large systems.
	AC3		given a parent class and a specification for a subclass, implement the subclass, including method overriding, calls to the super class constructor and calls to the super class's version of the overwritten method.
	AC4		explain what happens when polymorphic assignment happens.
	AC5		explain what happens when a polymorphic method call is made.
	AD	F, L, (A)	apply with more expert competence class design and usage through practice with programs implementing inheritance, class hierarchy and polymorphism.
Graphics (3)	AE1		explain how graphics applications use inheritance and interfaces.
	AE2		create codes which require the use of basic graphical user interface APIs in Java.
	AF		create codes which utilize an event-driven execution model.