CPSC 211 Course Learning Goals

| After this class students can… | Move from personal software development methodologies to professional standards and practices (e.g. create programs that interact with their environment (files etc.) and human users according to standard professional norms). | Given an API, write code that conforms to the API to perform a given task. | Identify and evaluate trade-offs in design and implementation decisions for systems of an intermediate size. | Read and write programs in Java using advanced features | Extend their mental model of computation from that developed in CPSC111 | Work with an existing codebase, including reading and understanding given code, and augment its functionality. [Happens only with assignments] |
|---|---|---|---|---|---|---|
| **Programming by contract** | A1, A2, A3, A4 | | | | | |
| **Exception handling** | B1, B5 | | B1, B6 | B2, B3, B4, B5 | | |
| **Streams, I/O** | C3 | | | C2, C3 | C1 | |
| **Testing** | D1, D2, D3 | | | D4 | | |
| **Software Design** | E2, E3, E4, E5, E6 | | E1, E7, E8, E10 | E9 | | |
| **Java Collections Framework** | | F3, F8, F11, F15, F18, F19 | F1, F2, F4, F12, F16, F20 | F3, F6, F7, F10, F13, F17, F21 | | |
| **Graphical User Interfaces** | G1 | | G1 | G2, G3, G5, G6 | G4 | |
| **Multi-threaded programming** | | H6 | | H4, H5, H6 | H1, H2, H3 | |
| **Recursion** | | | I5 | I1, I4, I6 | I2, I3 | |
| **Implementing basic collection classes** | | | | J1, J2, J3 | | |

| Topic | ID | Assessed in? | Students can: |
|---|---|---|---|
| **Programming by contract** | A1 | | write client code that adheres to the contract specified for a class using invariants, preconditions and postconditions |
| | A2 | | implement a class given a contract specified by invariants, preconditions and postconditions |
| | A3 | | describe the benefits of programming by contract for client and developer |
| | A4 | | use assertions appropriately in code |
| **Exception handling** | B1 | | incorporate exception handling into the design of a method's contract |
| | B2 | | trace code that makes use of exception handling |
| | B3 | | write code to throw, catch or propagate an exception |
| | B4 | | write code that uses a finally block |
| | B5 | | write code to define a new exception class |
| | B6 | | compare and contrast checked and unchecked exceptions |
| **Streams, I/O** | C1 | | describe stream abstraction used in Java for byte and character input/output |
| | C2 | | write programs that use streams to read and write data |
| | C3 | | incorporate data persistence in a program using Java's serialization mechanism |
| **Testing** | D1 | | compare and contrast blackbox and whitebox testing (at the level of what each type of testing provides) |
| | D2 | | use blackbox testing with equivalence classes to test a method and from that a suite of test cases |
| | D3 | | describe how unit testing is applied to a class (describe a hierarchy of tests that you could apply) |
| | D4 | | write a suite of tests to apply unit testing to a class using JUnit (putting the above into practice with a particular tool) |
| **Software Design** | E1 | | describe the basic design principles of low coupling and high cohesion |
| | E2 | | design a software system (expressed in UML) from a given specification that adheres to basic design principles (lc and hc) |
| | E3 | | interpret UML class diagrams to identify relationships between classes |
| | E4 | | draw a UML class diagram to represent the design of a software system |
| | E5 | | describe the Liskov Substitution Principle |
| | E6 | | explain whether or not a given design adheres to the LSP |
| | E7 | | incorporate inheritance into the design of software systems so that the LSP is respected |
| | E8 | | compare and contrast the use of inheritance and delegation |
| | E9 | | use delegation and interfaces to realize multiple inheritance in design (e.g. to support the implementation of multiple types) |
| | E10 | | identify elements of a given design that violate the basic design principles of low coupling, high cohesion, the LSP |
| **Java Collections Framework** | F1 | | use big-O notation to categorize an algorithm as constant, linear, quadratic or logarithmic time |
| | F2 | | given two or more algorithms, rank them in terms of their time efficiency |
| | F3 | | program to the generic List interface including read and use the List API (e.g. use Lists in ways similar to arrays) |
| | F4 | | compare and contrast ArrayList and LinkedList implementations of the List interface |
| | F6 | | compare and contrast assignment with various generic collections under specific subclass scenarios |
| | F7 | | use wildcards appropriately in generic type parameters to enable assignment in sub and super class scenarios |
| | F8 | | program to the generic Iterator and ListIterator interfaces including reading and using the APIs |
| | F10 | | read and write code that uses a for-each loop to iterate over a collection |
| | F11 | | program to the generic Set and SortedSet interfaces inclduign read and use the API |
| | F12 | | compare and contrast the HashSet and TreeSet classes (benefits of using each, basic run time analysis) |
| | F13 | | design and implement a class in such a way that it can be used with the Java collections framework (overrides equals in HashCode, implement the generic Comparable and Comparator interfaces to account for multiple sorting criteria) |
| | F15 | | program to the generic Map and SortedMap interfaces by reading and using the API |
| | F16 | | compare and contrast HashMap and TreeMap classes (benefits of using each, basic run time analysis) |
| | F17 | | write code (solve problems) that uses the generic algorithms provided in the Collections class |
| | F18 | | program to the generic Queue interface |
| | F19 | | program to the API of the generic Stack class |
| | F20 | | identify (in words or through code) appropriate types for collections of data needed in a given software system |
| | F21 | | write code that implements unidirectional, bidirectional, 1-1 and 1-many associations |
| **Graphical User Interfaces** | G1 | | describe basic principles of good user interface design (user interface hall of shame) |
| | G2 | | use layout managers to produce a well designed GUI |
| | G3 | | write code to produce a well designed GUI that includes frames, panels, menus and buttons |
| | G4 | | describe the event driven model |
| | G5 | | describe and apply scoping rules that apply to the use of inner classes |
| | G6 | | write code that uses inner classes (including anonymous inner classes) to handle events raised by GUI elements |
| **Multi-threaded programming** | H1 | | Describe the multi-threaded programming model including thread scheduler, thread priority, and time slices. |
| | H2 | | describe the various states that a Java thread can achieve and the events that lead to transition from one state to another |
| | H3 | | define the terms deadlock, race condition and critical section |
| | H4 | | identify possible legal traces of a multithreaded program |
| | H5 | | identify deadlock and race conditions in a multithreaded program |
| | H6 | | write a thread-safe class using Lock and Condition objects |
| **Recursion** | I1 | | trace code that uses recursion to determine what the code does |
| | I2 | | draw a recursion tree corresponding to a recursive method call |
| | I3 | | draw a stack trace of code that uses single and multi-branch recursion |
| | I4 | | write recursive methods |
| | I5 | | compare and contrast iterative and recursive solutions to a problem |
| | I6 | | replace a recursive implementation of a method with an iterative solution that uses a stack to model the run-time stack |
| **Implementing basic collection classes** | J1 | | write code to perform search, insertion and removal operations on a singly or doubly linked list |
| | J2 | | implement a class (e.g., list, stack or queue) that stores data in a linked list |
| | J3 | | implement a class (e.g., list, stack or queue) that stores data in an array |