MID-TERM STUDY GUIDE

You're Halfway Home - Congratulations!

You have learned a lot since class began and you have become a better programmer as a result. The Mid-Term Exam is a chance for you to show off what you have learned.

Here's a list of things to make sure you understand as you prepare for the exam:

.NET and C#

- C# is used to create programs that run inside of the .NET Environment and can (optionally) interact with other .NET compatible programs easily.
- C# is a pure object-oriented programming environment where ALL data types are objects.
- Many of C#'s simpler data types are just .NET data types masquerading as basic C# types - i.e., "int" is the same as .NET's "Int32"
- C# programs are compiled into "Intermediate Language (IL)" - a low-level language that executes inside the .NET CLR (Common Language Runtime).
- The .NET CLR is a "virtual" computer, implemented in software, which can run programs from any .NET language - including C#. Because it is implemented in software, the CLR can be made to run on different microprocessors and different Operating Systems - thereby allowing standard C# programs to run on different hardware without changes.
- Compiled C# code is stored in an "Assembly" file which typically has a name ending in "EXE" (for programs) or "DLL" (for libraries).
- The Stack is a region of memory where local variables and arguments live.
- Stack memory is automatically freed up when the corresponding variables go out of scope.
- The Heap is a (bigger) region of memory where objects live. Anything created by "new" lives on the heap.
- Periodically, the CLR's Garbage Collector discards objects that are no longer used from the Heap.
- The "using" statement lets you access classes from other class libraries in your program.
- Immutable objects cannot change after they have been created. Strings are immutable.

- Whenever you are assigning something of one type to a variable of a different type and information could be lost in the conversion, you have to include a cast.
- Use a "for" loop if you are counting the number of times through the loop. Use a "foreach" loop if you doing something to all the items in an array/collection.
- Use a "do/while" loop if you have to go through the loop at least once, otherwise use "while/do" loop.
- In C#, all "case" sections in a "switch" statement must end with a "break" or "return" statement.
- A method's name, return type, parameters and attributes are called its "signature."
- Method arguments can be passed "by reference" or "by value." If passed by reference, the original variable can be modified by the method it was passed to. If passed by value, a copy of the variable's data is sent.
- By default, all arguments are passed "by value." However, remember that a variable-that-represents-an-object actually contains a pointer to the location on the heap where the actual object lives. So when a variable-that-represents-an-object is passed into a method, the method actually gets a ***copy of the pointer***. That copy points to the same object on the heap as the original variable and, if the object allows it, the method can use that copy of the reference to modify the object's properties and/or call its methods. And so, effectively, if you pass a variable-that-represents-an-object to a method, you are *effectively* passing it "by reference."
- If you have several methods with the same name but different parameter lists, those methods are called "overloaded" methods.
- Optional parameters have default values that are used if those parameters are not included in the method call. Optional parameters must be placed at the end of a method's parameter list.
- Enums let you create a type with a set of fixed, named values like Colors.
- If you want to store null in a type that doesn't normally take null as a value (like an int), add a question mark to the end of the type. (e.g., "int?")

Object-Oriented Programming

- In OO Programming, an Object is a combination of CODE *and* DATA.
- A Class is a TEMPLATE for creating objects.
- Objects are instantiated (created) by using the "new" operator on a class. Memory is allocated from the heap and then the class' constructor is called so that it can initialize that memory.
- The "static" modifier causes just ONE version of that item to be created when the program runs.

- Encapsulation protects an object's internal data from arbitrary changes. It does that by using access modifiers like "private", "protected" and "public."
- Objects that just store and retrieve data are often called "Plain Old Java Objects" (POJOs). Data inside of a POJO is private. POJOs can add "Getter" and "Setter" methods to allow selective access to their data.
- C# supports .NET Properties which are essentially encapsulated fields with automatically generated Getters and Setters.
- Encapsulation, Inheritance and Polymorphism are often called "The Three Pillars of OO Programming."
- POJOs are perfect examples of data encapsulation.
- Inheritance lets you create a tree-like hierarchy of related classes. The lower classes are said to "inherit from" the upper classes.
- A subclass automatically gets ("inherits") all of the non-private member variables and methods that all of its superclasses have.
- C# does not support multiple inheritance. Subclasses can only have one superclass.
- If a method can be overridden, add the "virtual" keyword to its method signature.
- If a method in a subclass is overriding a method in a superclass, add the "override" keyword to its method signature.
- A "sealed" class cannot be used for inheritance. You can not inherit from a sealed class.
- An "abstract" class is essentially an "incomplete class" because it has one or more incomplete "abstract" methods.
- Abstract classes cannot be instantiated. Abstract classes must have a subclass inherit from them and implement their abstract methods.
- Polymorphism lets you call the same method on two different related objects. The system determines at runtime which method to actually call based on the type of the object calling the method.
- System.Object is the superclass of everything in C#. It's methods includes: ToString(), Equals(), and MemberwiseClone() which means that everything in C# has those methods.
- When a run-time error occurs, the system creates an Exception object and "throws" it to the inner-most in-scope "catch" block it can find.
- If an exception isn't caught by any catch block inside the program, the program crashes and a stack trace appears on the screen.
- The code inside of a "finally" block is always executed regardless of whether an exception occurred or not.

- Interfaces are essentially "100% abstract" classes - all methods inside an interface are un-implemented.
- You can associate multiple interfaces with a class as long as that class implements all of the methods in each of those interfaces.
- Interfaces typically add a common feature or characteristic to a class. Examples include IComparable and IEnumerable.
- Generics allow programmers to specify the type that a method or collection will accept. This adds "type safety" because the compiler can check for incorrect assignments.
- Data Structures are different ways of storing related data items in memory.
- Common Data Structures include Arrays, Stacks, Queues, Lists, Linked Lists, Maps (Dictionaries), and Hashtables.
- Stacks are "Last-In-First-Out" (LIFO) and Queues are "First-In-First-Out" (FIFO).
- C#'s Generic Collections are flexible data structures that work with most types of C# objects.
- List is the most commonly used C# Collection.
- All C# collections implement the ICollection and IEnumerable interfaces which means they can all be used with "foreach".

Programming Best Practices

- Always strive to write self-documenting, readable, maintainable code by choosing identifier names carefully.
- "Premature optimization is the root of all evil."
- Try to use Incremental Implementation: first, get a simplified version of your program to run, then make it a little better, then see if it still runs, then repeat the process.
- Try to separate UI-specific code from the rest of your program because the UI can (and will) change.
- Methods can be used to break complex tasks down into more manageable chunks. Methods can also be used to increase code reuse.
- Well-designed methods can take multiple inputs, but should only return one output. Well-designed methods only do one "thing."
- When designing an OO program, examine a written description of the program looking for important nouns and verbs. Nouns will often become classes or fields. Verbs will often become methods in those classes.
- When using noun-verb analysis, be sure to convert passive verbs to active verbs to help determine which class the method should belong to.

- Most classes should include their own version of ToString() to help with UI and/or debugging.
- Favor immutability in your classes.
- Always test the edge cases in your code. Make sure your program can handle any input the universe can throw at it without crashing.
- POJOs should throw exceptions if invalid data is passed into them. Those exceptions should not be caught because they indicate that there is a bug in the code that is using the POJO class.
- POJOs should never have UI code inside of them.
- A constructor job is to initialize a class' member variables - and nothing else. Constructors should be compact, efficient, and bullet-proof.
- Prefer "private" to non-private. Only make something less private if you are sure you need to.
- Inheritance hierarchies are difficult to update without breaking/rewriting lots of code that depends on it.
- Code that interacts with outside systems (disk files, the network, etc.) should be inside a try/catch block in case those calls fail.
- Only catch exceptions that you can constructively handle.