What’s New In C#

C# Genealogy

C# And .Net
- Languages like C# are not isolated entities
- They interoperate in two ways:
  - By being part of a system written in more than one language
  - By accessing services and operating on a distributed environment
- Requires support from run time: .Net and the Common Language Runtime

The Simple Stuff
- Most of C# is pretty similar to languages you are used to:
  - Declarations
  - Expressions
  - Assignment and control statements
- Other elements are quite similar:
  - Classes
  - Functions

Major Topics To Discuss
- Memory system and pointers
- Execution time environment
- Threads
- Exceptions
- Type system
- Identifier scope system
- Interfaces

Memory Layout
C# Memory Management

- Static vs. dynamic
- Dynamic storage—stack and heap
- Stack (Dynamic):
  - Managed algorithmically by implementation of function calls
- Heap (Dynamic):
  - Mostly managed by system
  - Provision for management by programmer

C# Memory Management

- Allocation using new
- Deallocation by Garbage Collection
- Garbage collection:
  - Track objects that are accessible
  - Free storage associated with objects that are inaccessible
  - Garbage collector is a system provided service that runs periodically
  - Deals with fragmentation

Garbage Collector Pros & Cons

- Pros:
  - Programmer does not have to implement
  - Memory management done right
- Cons:
  - No guarantee when it runs, hence no control
  - Takes processor resources
  - Does not delete storage if it is still reachable even if you don’t want it...
  - Memory leaks can (and do) still occur

Some Specifics of C#

- Object destruction via Object.Finalize:
  - Inherited from Object type
  - Override to destroy object as desired
- Garbage collector available via GC class:
  - Runs via separate thread
  - Various methods available for access
- Pointers—yes, they are provided:
  - Syntax like C++, code marked unsafe
  - Objects managed by GC or user—pinned

Traditional Compilation

- Source Program
- Compiler
- Object Code Libraries
- Linkage Editor
- Object Program
- Loader
- Binary Program

More Flexible Compilation

- Source Program
- Compiler
- MSIL Program
- Interpreter
- JIT Compiler
- Target (Run-time Support)

Machine instructions for multiple targets
Concurrency
- Threads vs. processes/tasks
- C# supports *threads*

C# Threads
- *System.Threading* namespace
- Facilities include:
  - Thread creation, destruction
  - Child thread management, e.g. `join()`
  - Thread scheduling, priority, timing
- Synchronization:
  - Monitors
  - Semaphores (mutex class)
  - Lock—serialization of statement block

Exceptions
- Why do we need exceptions?
- How should they be made available in programming languages?
- What benefits do they provide?
- What problems could they cause for us?

One Thing Is For Sure...
- Exceptions are NOT for *dealing* with errors
- They are a mechanism for changing the flow of control from sequential to a branch if certain conditions exist
- They always indicate expected circumstances. Otherwise they could not possibly be generated

Exceptions in C#?
- *Throw* raises an exception
- *Catch* defines a block that handles the exception
- Etc.

Type System
- Type should be consistent:
  - Predefined and user-defined
- All C# types derive from `System.Object`
- Single rooted hierarchy
- Provides four standard methods:
  - `bool Equals`
  - `int GetHashCode`
  - `Type GetType`
  - `String ToString`
  - These don't necessarily mean what you think
Types Of Types

- Value types and reference types
  - Value types:
    - Program variables have a value
    - Space allocated on stack
  - Reference types:
    - Program variable is just a reference
    - Allocated space on stack
    - Reference is a "type-safe" pointer
    - Data space allocated on heap

Value vs. Reference

- Note the "special" status of primitive types
  - System.Int32 myInt = 42;
  - System.String myStr = "Hello World";
  - Circle c;
  - c = new Circle(...);

Boxing And Unboxing

- Conversion between value variable and reference variable
  - System.Int32 myInt = 42;
  - object o = myInt;
  - ymInt = (int)o;

The Role Of A Type System

- What do we need from a type system?
  - The ability to create new "things" that are more useful (usually more abstract) than the basic machine resources
  - To allow us to build mechanical checks to stop us from hurting ourselves
- The notion of type is artificial

Types across languages:
- Consistency
- Compatibility

Type safety on steroids:
- Checking for meaningful statements
- Add "speed" to "distance"?
- Assembly language vs. C vs Java and C# vs Ada

Type needs to support application semantics