Data Types-

* store and use data
* is type-safe
* 2 types
  + Value types- stack, directly, int, float, long, enum, struct
    - Static
    - Fixed size (overflow exception)
    - Easy to get
  + Reference types-heap, using class, interface, string, array, object, delegate.
    - Dynamic
    - Find to get
    - Delegate- used to encapsulate methods, only declare,
* byte (unassigned)
  + only positive
  + Size- 8 bits (1 byte)
  + Range- 27  - 27-1
  + System.Byte
* sbyte (signed)
  + negative and positive
  + Size- 8 bits (1 byte)
  + Range- 0 - 28-1
  + System.SByte
* char
  + Size- 16 bits (2 byte)
  + Range- U+0000 to U+ffff
  + System.Char
* Short
  + Positive and negative
  + Size- 16 bits (2 byte)
  + Range- 215  - 215-1
  + System.Int16
* ushort (unassigned)
  + only positive
  + Size- 16 bits (2 byte)
  + Range- 0 - 216-1
  + System.UInt16
* int
  + Positive and negative
  + Size- 32 bits (4 byte)
  + Range- 231  - 231-1
  + System.Int32
* uint (unassigned)
  + only positive
  + Size- 32 bits (4 byte)
  + Range- 0 - 232-1
  + System.UInt32
* long
  + Positive and negative
  + Size- 64 bits (8 byte)
  + Range- 263  - 263-1
  + System.Int64
* ulong (unassigned)
  + only Positive
  + Size- 64 bits (8 byte)
  + Range- 0 - 266-1
  + System.UInt64
* bool (true and false)
  + Size- 4bits (1 byte)
  + System.Boolean
* decimal
  + Size- 128 bits (16 byte)
  + System.Decimal
* float
  + Size- 32 bits (4 byte)
  + System.Single
* double
  + Size- 64 bits (8 bytes)
  + System.Double

Enumeration-

* Named integers constants
* default starts with 0
* enum- keyword
* enum -name- {-const list}

Void-

* method with no return type
* cannot be used as input parameter

var

* used to declare implicit variable
* specifies type based on the initialized value

var vs dynamic

* var- needs to be value assigned at the time of declaration and type is decided at compile time
* dynamic- need not assign value at time of declaration and type is decided at run time.

Object

* can store any data type
* but unboxing requires original type (overhead)
* preferred with no info

Boxing and Unboxing

* value to object- boxing
  + int a=10; object o=a;
* object to value- unboxing
  + int a= (int)o

Struct vs class

* class- reference type, struct- value type
* class- constructor, struct- no constructor.
* class- inheritance, struct- no inheritance

Searching

* finding particular item
* 2 types
  + Internal searching- in main or primary memory
  + External searching- in external or secondary memory
* Popular search methods-
  + Linear or sequential search- one by one
  + Binary search- by dividing the list and using LOW, HIGH and MID.

OOPS

* Application- collection of programs
* Program elements-
  + Identity- unique for program (name)
  + Variable- memory allocated to identity
  + Method- block of instruction with identity
* Class- OOPs uses classes and objects. Class contains variables and methods.
* Objects- Instance of a class. Instance variables get memory inside object. (one class many objects)
* Variables- info of class (object). 4 types-
  + Static variables- common info of all objects. Using class name.
  + Instance variables- specific to object. Using object reference.
  + Method parameters- i/p of method. Directly given to method
  + Local variables- info inside method. Local access, only inside method.
* Method-logic to perform task. 4 types
  + No i/p - no o/p
  + With i/p - no o/p
  + With i/p - with o/p
  + No i/p – with o/p
* Static method - static keyword, using class name
* Instance method- no keyword, using object reference
* Static variables- inside class and outside method, access using class name
* Getter and Setter Methods
  + set() to set value to variable , get() to get value
  + static variables- static get set, instance variables- instance get set
* Default value
  + int – 0
  + double – 0
  + char – blank
  + bool – false
  + String – Null (blank)
* Instance members-
  + Non static, related to object, invoked object-address
  + Constructor- method with class name and no return type.
  + Instance method- no static keyword, invoked object-reference
  + Instance variable- Inside class and outside method, memory inside object
* this- Default object reference variable. Holds address of current obj. access instance methods and constructors
* Access Modifiers- set permissions to access class and its members
  + Public- accessed by same or different assembly
  + Private- in same class or struct
  + Protected- in same class or derived class
  + Internal- only in same assembly.
  + Protected internal- same assembly and derived class
  + Private protected- derived class in same assembly
* Encapsulation- protecting data within class
  + POCO- Plain Old CLR Object (common language runtime)
  + Class is public, variables private, methods public.
* Constructor Chaining- invoking constructor from another constructor.
* Inheritance- new class reusing other class members
  + Single Inheritance – one to one
  + Multilevel Inheritance- one to one to one
  + Hierarchical Inheritance- another to one and another to one
  + Multiple Inheritance- one to two, cannot be done using classes, but done via interface
* this- to invoke instance members
* base- to invoke instance of parent class
* Method overriding- child and parent has same method.
  + Runtime polymorphism- same name and same signature in parent and child
  + virtual in base and override in derived
  + base()- invoke parent constructor. (with parameters?)
* Polymorphism- same class different methods
  + Compile time polymorphism- overloading, same method with different signatures
* Object up-casting- address of child in parent type reference variable.
* Down casting- child address from child reference from parent type.
* Sealed- cannot be modified, class not inherited, method not overridden, variable cannot be modified
* Abstraction- hide implementation show functionality
* Abstract method- keyword, no body. Concrete method (with body)
* Abstract class- need child class, override abstract method. Through child abstract parent class accessed
* Interfaces- only abstract methods, public abstract by default, Implemented by overriding