Advanced Programming 2

DR. ELIAHU KHALASTCHI

2016

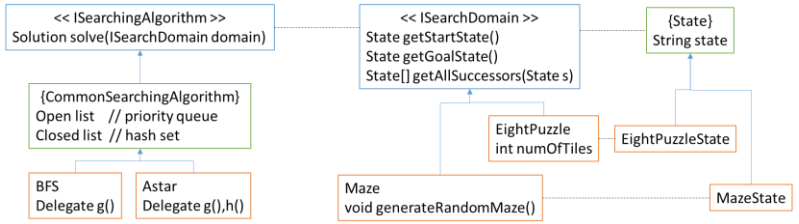
Advanced Programming 2

- This semester we are going to build a
  - ***Desktop***, ***Web***, *and* ***Mobile*** *applications*
  - Our project includes all the important elements used in the industry

- We will better understand
  - object orientation
  - concurrent programming

- We learn how to transform an algorithmic ***pseudocode*** to an ***Object Oriented Program***

```
Best First Search:
OPEN = [initial state]   // a priority queue of states to be evaluated
CLOSED = []              // a set of states already evaluated
while OPEN is not empty
do
    1. n ← dequeue(OPEN) // Remove the best node from OPEN
    2. add(n,CLOSED)     // so we won't check n again
    3. If n is the goal state,
          backtrace path to n (through recorded parents) and return path.
    4. Create n's successors.
    5. For each successor s do:
        a. If s is not in CLOSED and s is not in OPEN:
              update that we came to s from n
              add(s,OPEN)
        b. Otherwise, if this new path is better than previous one
              i.  If it is not in OPEN add it to OPEN.
              ii. Otherwise, adjust its priority in OPEN done
```



Syllabus & Course Overview

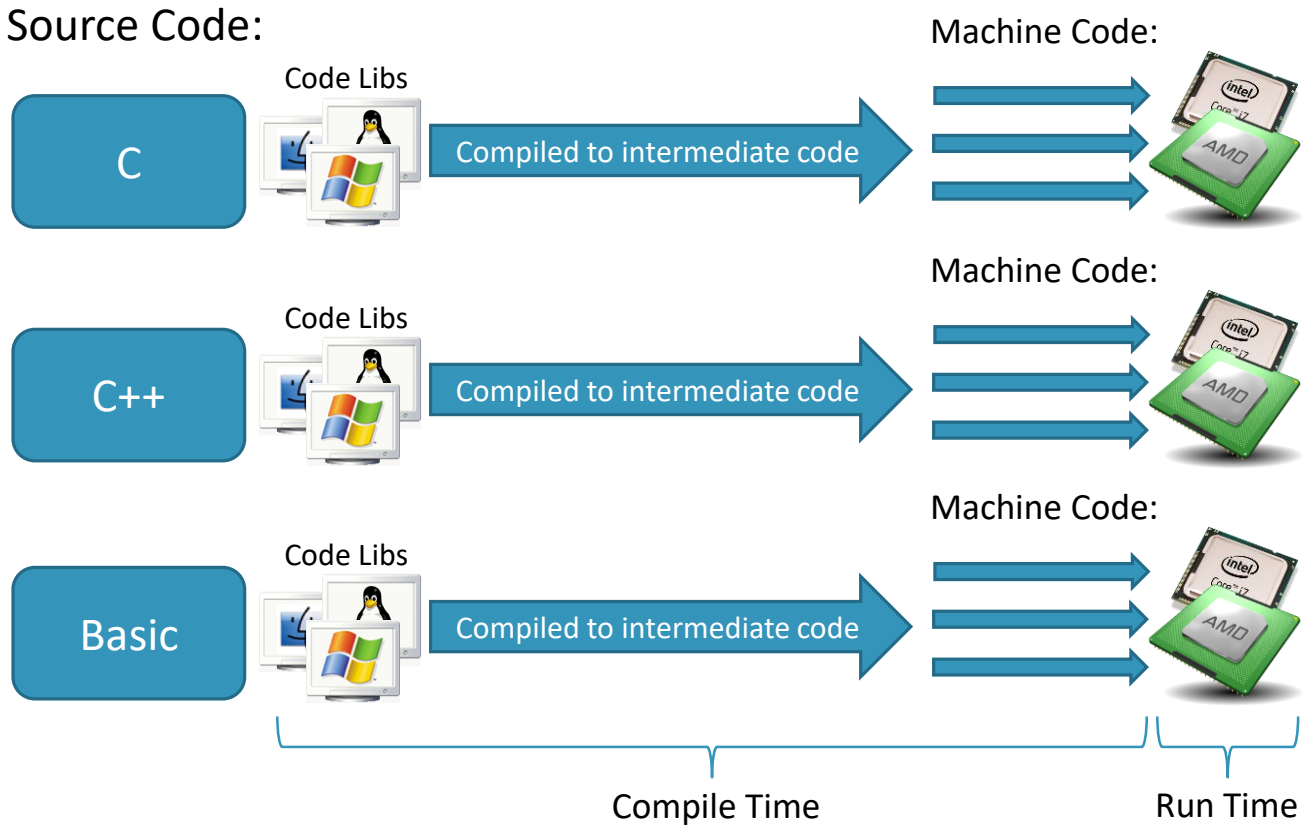
Today's Agenda…

- .Net framework
- Unique to C#…
  - Data Types
  - Parameter Passing
  - Properties
  - String Interns
  - Operator Overloading
  - Delegates
  - Events

The .NET Framework

AN INTRODUCTION

Traditional Architecture – using compilers



- Really hard to make a program portable
  - Recompile with new system / user libraries for a specific compiler
  - Harder to reuse code written in other languages
- Must implement your own infrastructure
  - Memory management
  - Threading
- Or be highly dependent on the operating system services

JVM architecture – multiplatform design

code

```
public class HelloWorldApp {
    public static void main(String[] args) {
        System.out.println("hello world!");
    }
}
```

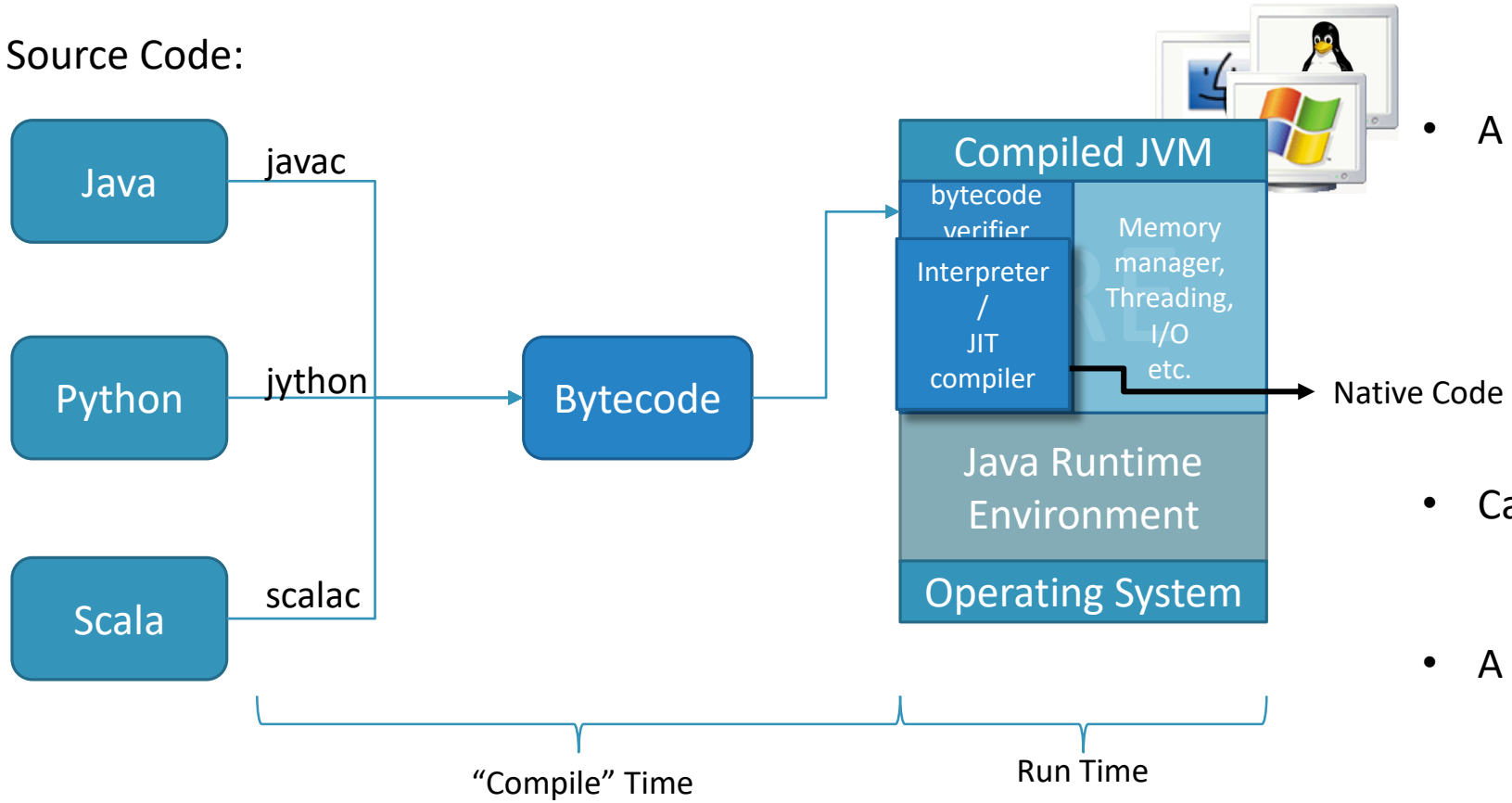
javac: Compiler → ByteCode

```
.class public HelloWorldApp
.super java/lang/Object

.method public static main : ([Ljava/lang/String;)V
    .limit stack 10
    .limit locals 10

    getstatic java/lang/System out Ljava/io/PrintStream;
    ldc “hello world!"
    invokevirtual java/io/PrintStream println (Ljava/lang/Object;)V
    return
.end method
```

JVM: Interpreter | Interpreter | Interpreter

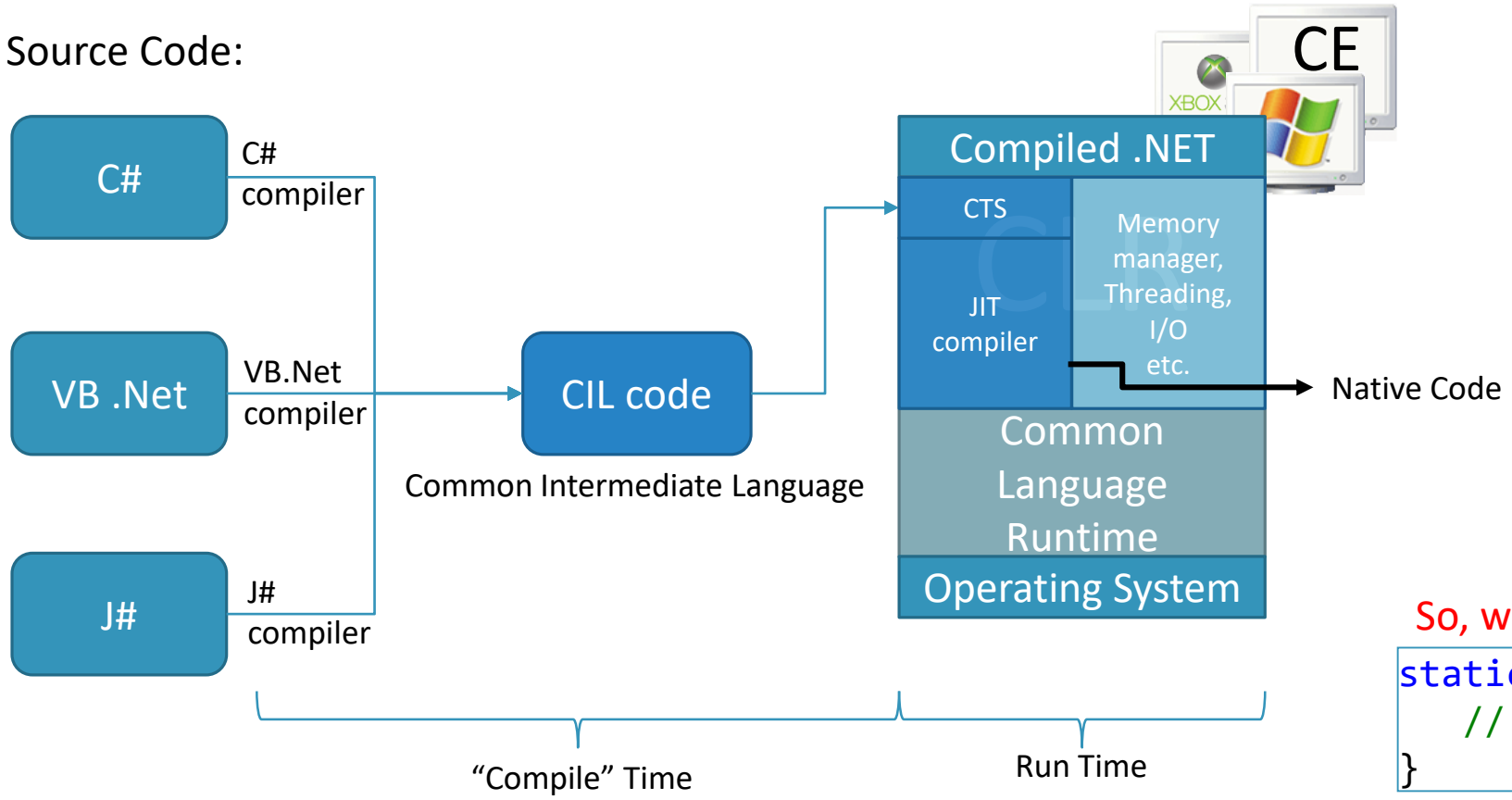
- The JVM has an interpreter
- But it does not interpret High-Level code
- **This is way too slow…**

The JVM Architecture – multiplatform



- A multiplatform architecture
  - No need to change source code to run on a different system as long there is a JVM on it
- Can load classes at runtime
  - Regardless of their source code
- A managed environment
  - E.g. a garbage collection

The .NET Architecture



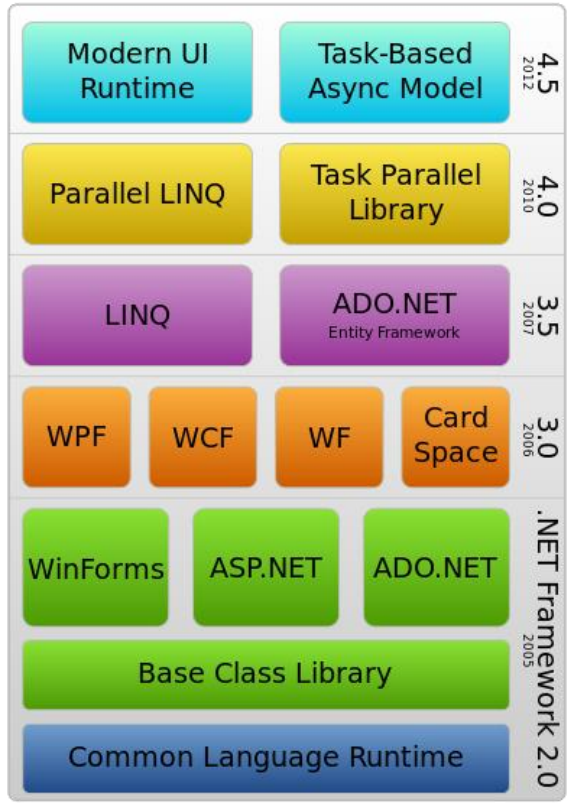
So, why the Main() method is static?

```
static int Main(string[] args) {
   //...  return 0;
}
```

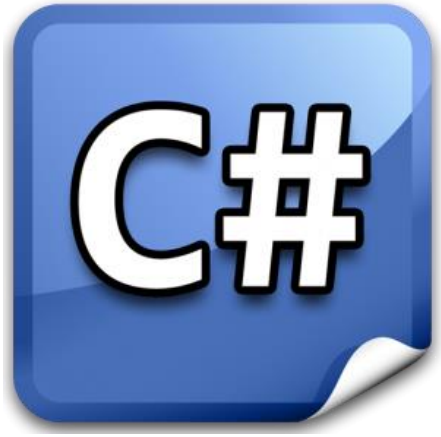
The .Net Framework Class Library

- Console Applications
- Windows Forms Applications
- Windows Presentation Foundation (WPF)
- Windows Communication Foundation (WCF)
- Windows Workflow Foundation (WF)
- ADO.NET – for work with data bases

- ASP.NET – for web development



The .NET Framework Stack



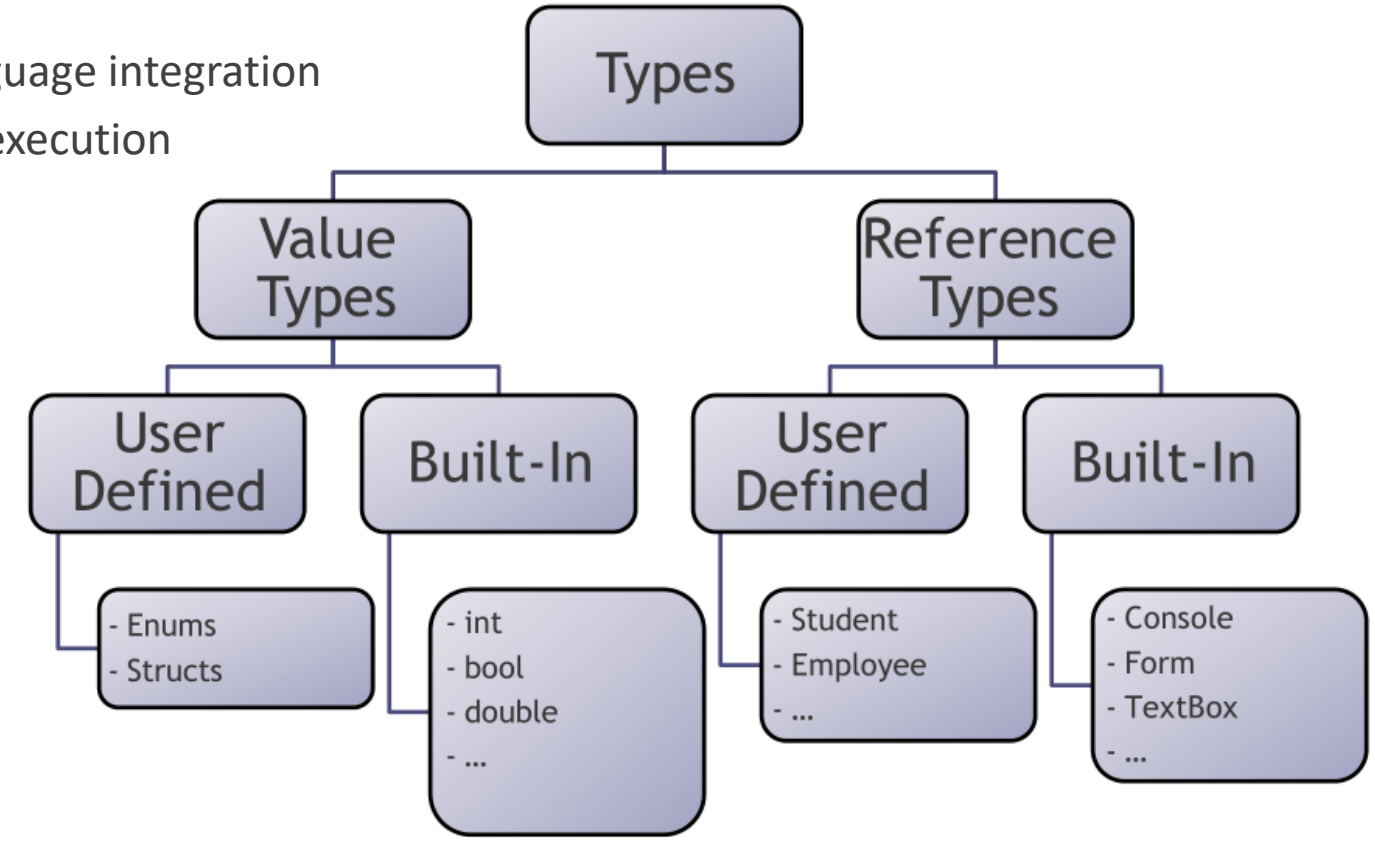
Introduction to C#

FOR JAVA DEVELOPERS

Data Types

- CTS – Common Type System
  - a framework that helps enable cross-language integration
  - type safety, and high performance code execution

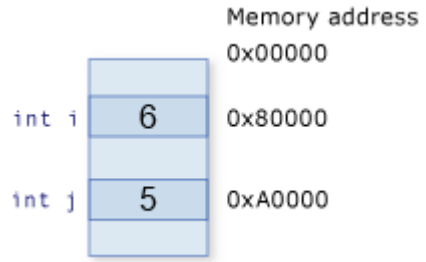
- Structures
- Enumerations
- Classes
- Interfaces
- Delegates



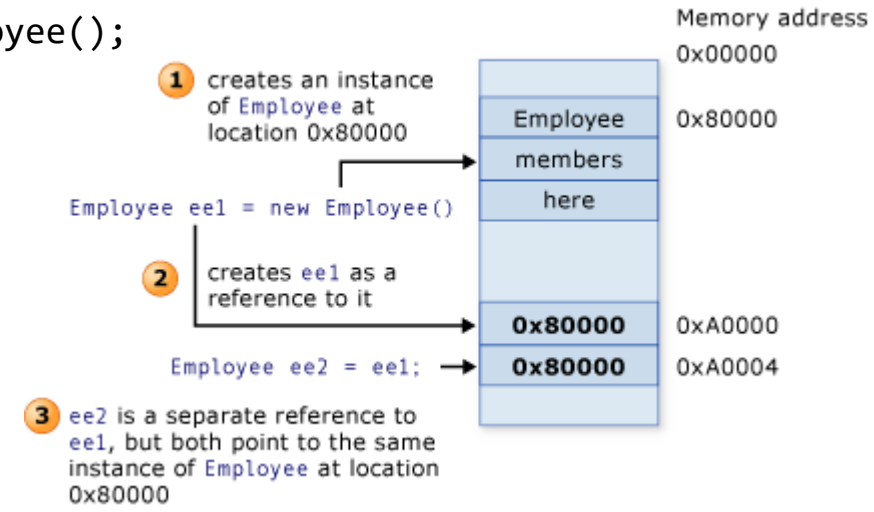
Value Types vs. Reference Types

- Value types
  - Are the built-in primitive data types, such as char, int, as well as user-defined types declared with struct
- Reference types
  - Classes and other complex data types that are constructed from the primitive types
  - Variables of such types do not contain an instance of the type, but merely a reference to an instance

```
int i = 5;
int j = i;
i = 6;
Console.WriteLine(i);    // 6
Console.WriteLine(j);    // 5
```




```
Employee ee1 = new Employee();
Employee ee2 = ee1;
```



Built-in Data Types

Short Name	.NET Class	Type	Width	Range (bits)
byte	Byte	Unsigned integer	8	0 to 255
sbyte	SByte	Signed integer	8	-128 to 127
int	Int32	Signed integer	32	-2,147,483,648 to 2,147,483,647
uint	UInt32	Unsigned integer	32	0 to 4294967295
short	Int16	Signed integer	16	-32,768 to 32,767
ushort	UInt16	Unsigned integer	16	0 to 65535
long	Int64	Signed integer	64	-9223372036854775808 to 9223372036854775807
ulong	UInt64	Unsigned integer	64	0 to 18446744073709551615
float	Single	Single-precision floating point type	32	-3.402823e38 to 3.402823e38
double	Double	Double-precision floating point type	64	-1.79769313486232e308 to 1.79769313486232e308
char	Char	A single Unicode character	16	Unicode symbols used in text
bool	Boolean	Logical Boolean type	8	True or false
object	Object	Base type of all other types		
string	String	A sequence of characters		
decimal	Decimal	Precise fractional or integral type that can represent decimal numbers with 29 significant digits	128	±1.0 × 10e−28 to ±7.9 × 10e28

- Java has primitive types and wrapper classes e.g.
  - int – Integer
  - double – Double
- Primitive types in C# are Objects!
  - int is an alias for System.Int32
  - double is an alias for System.Double

```
static void Main()
{
   int i = 10;
   object o = i;
   System.Console.WriteLine(o.ToString());
}
```

A little About Arrays…

- Arrays in C# are actually references to **objects** that contain an array (like in Java)
  - Elements of the same type placed continuously in the memory, accessed by an integer index
  - This array object has methods and properties

```
int[] arr1D;
int x = Console.Read();
arr1D = new int[x]; // arrays are initialized at runtime

object o = arr1D;   // arrays are objects

int[,] arr2D = new int[10, 20]; // a 2D array
int[,,] arr3D = new int[10, 20,30]; // a 3D array

Console.WriteLine(arr3D.GetLength(0));  // 10
Console.WriteLine(arr3D.GetLength(1));  // 20
Console.WriteLine(arr3D.GetLength(2));  // 30

Console.WriteLine(arr3D.Length);  // 10*20*30 = 6000
```

```
int[] arr = { 1, 2, 3 };
int[] arr2;
arr2 = new int[3] { 1, 2, 3 };

int[,] arr3 ={
     {1,2,3},
     {4,5,6}
};
Console.WriteLine(arr3.Rank); // 2

int[][] jaggedArray = new int[2][];
jaggedArray[0] = new int[4];
jaggedArray[1] = new int[3];
```

Parameter Passing

C# VS. JAVA VS. C++

Parameter Passing in C#

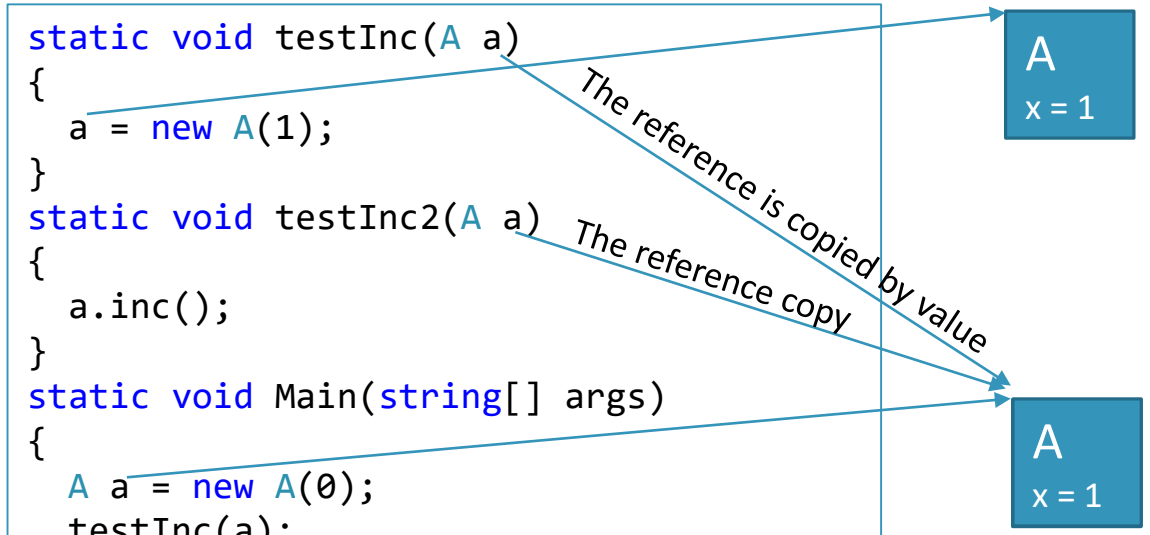
- In Java the parameters are passed “by value”
- In C# the **default** parameter passing is also “by value”

- Like in Java, when a **reference type** is passed **by value**
  - The object’s **address** is passed **by value**
  - Therefore, the local variable still points to the passed object
  - And can manipulate the object’s data
  - However, calling new will only change the address of the local variable
  - And will not change the passed object

Reference Type Passed By Value Example

```
class A
{
  private int x;
  public A(int x)
  {
    this.x = x;
  }
  public int getX() { return x; }
  public void inc()
  {
    x++;
  }

  public A(A a)   // copy Ctor
  {
    Console.WriteLine("I was never invoked!");
    x = a.x;
  }
}
```

```
static void testInc(A a)
{
  a = new A(1);
}
static void testInc2(A a)
{
  a.inc();
}
static void Main(string[] args)
{
  A a = new A(0);
  testInc(a);
  Console.WriteLine(a.getX());    // 0
  testInc2(a);
  Console.WriteLine(a.getX());    // 1
  Console.ReadKey();
}
```



By Value Example – C++ equivalent

```cpp
void testInc(A* a)
{
  a = new A(1);
}
void testInc2(A* a)
{
  a->inc();
}
void main()
{
  A* a = new A(0);
  testInc(a);
  cout<< a->getX() << endl;     // 0
  testInc2(a);
  cout<< a->getX() << endl;     // 1
}
```

C++

```cpp
static void testInc(A a)
{
  a = new A(1);
}
static void testInc2(A a)
{
  a.inc();
}
static void Main(string[] args)
{
  A a = new A(0);
  testInc(a);
  Console.WriteLine(a.getX());    // 0
  testInc2(a);
  Console.WriteLine(a.getX());    // 1
}
```

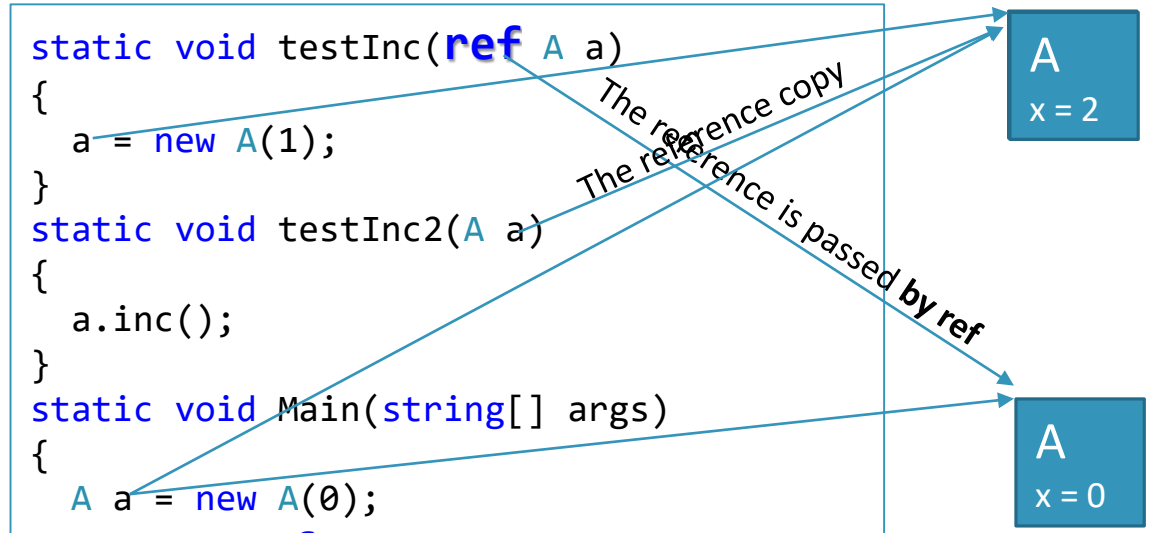
C#

Reference Type Passed By Reference

```cpp
void testInc(A* & a)
{
  a = new A(1);
}
void testInc2(A* a)
{
  a->inc();
}
void main()
{
  A* a = new A(0);
  testInc(a);
  cout<< a.getX() << endl;     // 1
  testInc2(a);
  cout<< a.getX() << endl;     // 2
}
```

C++

```cpp
static void testInc(ref A a)
{
  a = new A(1);
}
static void testInc2(A a)
{
  a.inc();
}
static void Main(string[] args)
{
  A a = new A(0);
  testInc(ref a);
  Console.WriteLine(a.getX());     // 1
  testInc2(a);
  Console.WriteLine(a.getX());     // 2
}
```



C#

Parameter Passing in C#

- In C# we can also pass parameters we want to change or even initialize their values
- We can use the **ref** keyword again, but this special case gets the **out** keyword
- What is the difference?
  - The variable does not have to be initialized
  - If it is initialized then its value is ignored
  - The variable must be initialized inside the method

```
static void initialize(out int x)
{ // we must initialize x somewhere in this method
     x = 0;
}
static void increment(ref int x)
{ // not a good idea to use "out". why?
     x++;
}
static void Main(string[] args)
{
     int x;
     initialize(out x);
     Console.WriteLine(x);    // 0
     increment(ref x);
     Console.WriteLine(x);    // 1
     Console.ReadKey();
}
```

Quiz – what is the output?

```
static void initialize(out int[] x, int length, int value)
{
     x = new int[length];
     for (int i = 0; i < x.Length; i++)
     {
          x[i] = value;
     }
}
static void change1(ref int[] x)
{
     initialize(out x, 5, 0);
}
static void change2(int[] x)
{
     x[0] = 1;
}
static void change3(int[] x)
{
     initialize(out x, 3, 0);
}
```

```
static void Main(string[] args)
{
     int[] x=null;
     change1(ref x);
     change2(x);
     change3(x);
     foreach (int i in x)
          Console.Write(i+",");

     Console.WriteLine();
     Console.ReadKey();
}
```

Properties

Property

- It is a good idea for **data members** to be **private**
- Public setters & getters can provide **managed access** to these private data members
- Yet, it would be nicer to access data members rather than activate a method…

- d is a private data member
- Dist is a public ***property***
  - It has a setter and a getter
  - (can use IF sentences)
  - They manipulate d

- Outside the class, Dist is used as a public data member

```
class Distance
{
        private int d;
        public int Dist
        {
            get
            {
                return d;
            }
            set
            {
                if (value >= 0)
                    d = value;
            }
        }
}
class Program
{
        static void Main() {
            Distance d = new Distance();
            d.Dist = 100;
        }
}
```

String Interns

C# VS. JAVA

String type Java vs. C#

- Like in Java, Strings are immutable in C#
  - meaning that the values of the strings cannot be changed once the strings have been created
  - Methods that appear to change the string actually return a new string, leaving the original unchanged
- The == and != operators are implemented for strings in C#

```java
String s1=new String("hello");
String s2=new String("hello");
System.out.println(s1==s2);// false
System.out.println(s1.equals(s2));// true
```

Java

```csharp
char[] hello = "hello".ToCharArray();
String s1 = new String(hello);
String s2 = new String(hello);
Console.WriteLine(s1 == s2);    // true
Console.WriteLine(s1.Equals(s2));   // true
```

C#

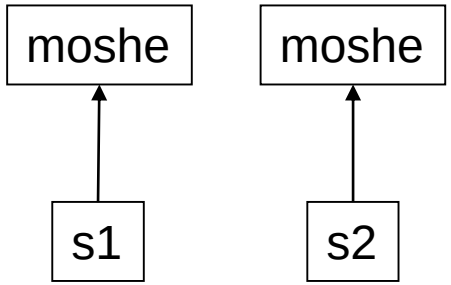
- Like in Java, we must use StringBuilder for concatenating strings efficiently

Comparing Strings

- Might take O(n)…
- But if two string objects would have the same reference then
  - Comparing them would take O(1)
  - Regardless of the String size
- Java has a unique mechanism called a String Pool
- it functions as a hash table of all the constant strings
  - E.g., “this is a constant string object in Java”
- The intern() method will create if necessary, and return a reference to the string in the pool
- Thus, we can compare two interns with the == operator

Comparing Strings in Java

- Strings are obviously Objects
- Do not use "==" to compare strings
- Use the "equals" function



For strings with n characters, ***.equals*** takes O(n), can it be done in O(1)?

```
String S1=new String("moshe");
String S2=new String("moshe");
if(S1==S2)
  System.out.println("equal");
else
  System.out.println("not equal");


if(S1.equals(S2))
  System.out.println("equal");
else
  System.out.println("not equal");
```

Comparing Strings

- The String Class has the method “intern”
  - The string that invoked the method is saved in a special pool (hash table)
  - The method returns the string from the pool

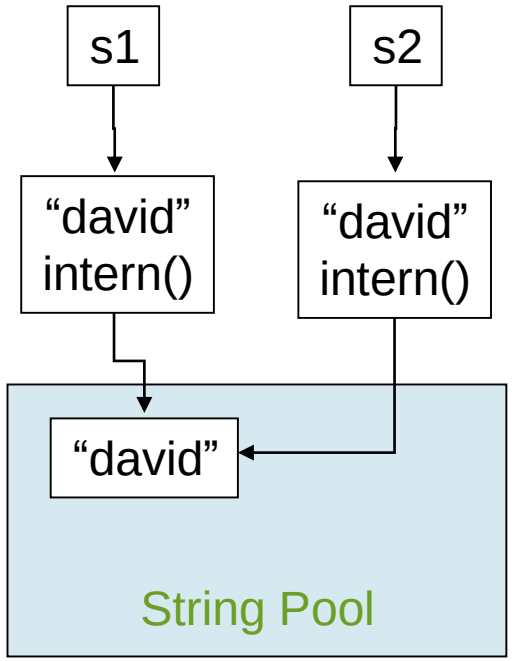
```
String s1=new String("david");
String s2=new String(s1);

if(s1==s2)
      System.out.println("equal");
else
      System.out.println("not equal");
```

O(n)

```
if(s1.intern()==s2.intern())
      System.out.println("equal");
else
      System.out.println("not equal");
```

O(1)

```
if(s1.intern()==s2.intern())
      System.out.println("equal");
else
      System.out.println("not equal");
```



Comparing Strings

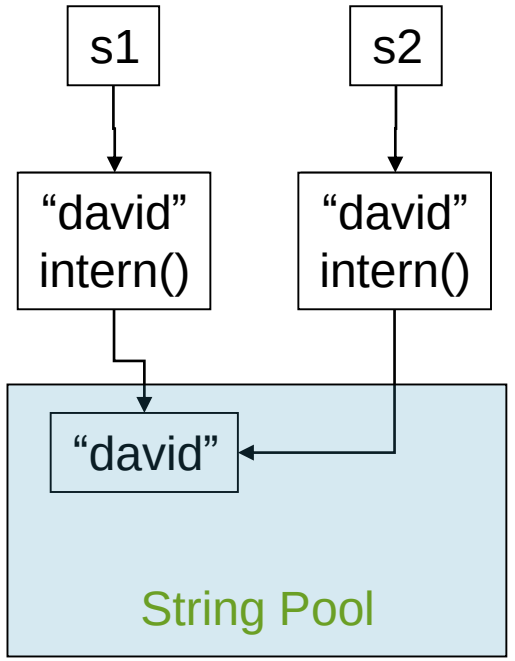
- The String Class has the method “intern”
  - The string that invoked the method is saved in a special pool (hash table)
  - The method returns the string from the pool

```
String s1=new String(input);    // “david”
String s2=new String(s1);

if(s1==s2)
      System.out.println(“equal");
else
      System.out.println(“not equal");

if(s1.intern()==s2.intern())
      System.out.println(“equal");
else
      System.out.println(“not equal");
```

O(n)

```
if(s1.intern()==s2.intern())
      System.out.println(“equal");
else
      System.out.println(“not equal");
```

O(1)



Intern in C#

- A similar mechanism exists in C#
  - It is called the “intern pool”
  - With each string creation the pool is checked for storing the string and a reference is kept
  - The Intern() is a static method of the String class that returns this reference

```
string s1 = "MyTest";
string s2 = new StringBuilder().Append("My").Append("Test").ToString();
string s3 = String.Intern(s2);

Console.WriteLine(s2==s1); // operator overloading for Equals – O(n)
Console.WriteLine((Object)s2==(Object)s1); // Different references -> false
Console.WriteLine((Object)s3==(Object)s1); // The same reference -> true O(1)
```

Operator Overloading

Operator Overloading

- Like C++, C# allows you to overload operators for use on your own classes
- This makes it possible for a user-defined data type to look as natural as a fundamental data type
- To overload an operator, write operator followed by the symbol for the operator to be overloaded
- All operator overloads are **static** methods of the class (not like C++)

- The full list of operators that can be overloaded is
  - Unary operators: **+**, **-**, **!**, **~**, **++**, **--**, **true**, **false**
  - Binary operators: **+**, **-**, *****, **/**, **%**, **&**, **|**, **^**, **<<**, **>>**, **==**, **!=**, **>**, **<**, **>=**, **<=**

Operator Overloading Example

```
public class ComplexNumber
{
        private int real;
        private int imaginary;

        public ComplexNumber(): this(0, 0)  // constructor
        {
        }
        public ComplexNumber(int r, int i)  // constructor
        {
            real = r;
            imaginary = i;
        }

        // Override ToString() to display a complex number in the traditional format:
        public override string ToString()
        {
            return (System.String.Format("{0} + {1}i", real, imaginary));
        }
```

```
// Overloading '+' operator:
public static ComplexNumber operator +(ComplexNumber a, ComplexNumber b)
{
      return new ComplexNumber(a.real + b.real, a.imaginary + b.imaginary);
}

// Overloading '-' operator:
public static ComplexNumber operator -(ComplexNumber a, ComplexNumber b)
{
      return new ComplexNumber(a.real - b.real, a.imaginary - b.imaginary);
}
```

```
ComplexNumber a = new ComplexNumber(10, 12);
ComplexNumber b = new ComplexNumber(8, 9);
ComplexNumber sum = a + b;
ComplexNumber difference = a - b;
```

Delegates & Events

Delegate variable vs. Event variable

- public void f(){…};
- public void g(){…};
- public **delegate** void myFunc();

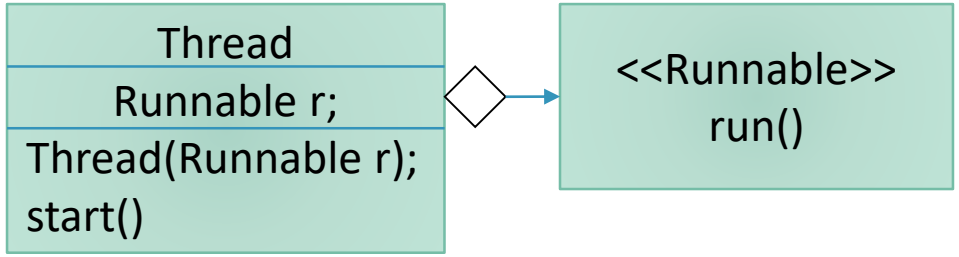
```
myFunc x;
x=f;
x(); // activate f()
x=g;
x(); // activate g()
```

```
event myFunc x;
x+=f;
x+=g;
x(); // activate f() and g()
x-=f;
x(); // activate only g()
```

We can easily pass delegates as parameters

Java: complex object injection

```
new Thread(
  new Runnable() {
    @Override
    public void run() {
      while(!stop){
        x++;
      }
    }
  }
).start();
```



We can easily pass delegates as parameters

C#: simple delegate injection

```
new Thread(
 delegate() {
  while(!stop){
    x++;
  }
 }
).Start();
```

Thread
delegate void Runnable();
Thread(Runnable r); Start()

An Alarm Clock Example

- Let's say we want to create an alarm clock
  - It should run in the background (a thread)
  - We want to be notified each second and apply different things
- We can start by creating a class that creates a thread that samples the time each second:

```
class AlarmClock {
    public Boolean stop;
    public void start()  {
        Thread t=new Thread(delegate() {
                while (!stop) {
                    String time = DateTime.Now.ToString("HH:mm:ss tt");
                    Console.WriteLine(time);
                    Thread.Sleep(1000);
                }
            });
        t.Start();
    }
}
```

An Alarm Clock Example

- We can define a **delegate** that defines what actions should be taken
  - `public delegate void whatToDo(String time);`

- We can define an **event** that is associated with the delegate
  - `public event whatToDo customEvent;`

- We can **raise the event**
  - `customEvent(time);`
  - `We'll do it every second inside the clock's loop`
  - `It will activated every subscribed delegate`

An Alarm Clock Example

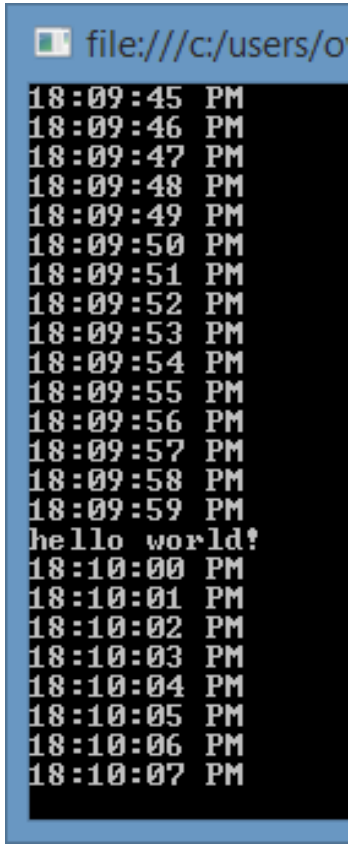
```
class AlarmClock {
    public Boolean stop;

    public delegate void whatToDo(String time);
    public event whatToDo customEvent;

    public void start()  {
        new Thread(delegate() {
                while (!stop) {
                    String time = DateTime.Now.ToString("HH:mm:ss tt");

                    customEvent(time);

                    Thread.Sleep(1000);
                }
            }).Start();
    }
}
```

An Alarm Clock Example

- Now we can use the event's **+=** operator to assign as many delegates as we wish
- The **-=** operator removes delegates from the event

```
static void Main(string[] args) {
    AlarmClock ac = new AlarmClock();

    ac.customEvent += delegate(String time) {
        if (time.Equals("18:10:00 PM")) {
            Console.WriteLine("hello world!");
        }
    };
    ac.customEvent += delegate(String time) {
        Console.WriteLine(time);
    };

    ac.start();
    Thread.Sleep(3*60*1000);
    ac.stop = true;
    Console.ReadKey();
}
```

We have added
2 event handlers



Interfaces

IN C#

Interfaces in C#

- **C++** have abstract classes that can be used as “interfaces”
  - All methods are pure virtual
  - No data members or CTORs
  - Multiple inheritance allows to implement many “interfaces”
- **Java** interfaces – contain only signatures of methods
- a **C#** interface allows:
  - Signatures of methods
  - Properties
  - Events

```
public delegate void WhatToDo();

public interface AlarmColck
{
    event WhatToDo raiseAlarm;
    string TimeSetting {
        set;
        get;
    }
    void start();
    void stop();
}
```