CLI and CLR

Antonio Cisternino
Giuseppe Attardi

Introduction

- Java made popular Virtual Machines (JVM)
- Execution environments are a generalization of virtual machines
- They provide a set of common runtime services for high level programming languages
- They incorporate valuable programming techniques developed in the last 30 years of PL research

Java, a brief history

- Java was designed by James Gosling as a language for embedded systems (e.g. washing machines)
- The original name of the language was Oak (renamed to Java for copyright reasons)
- Sun Microsystems applied for a tender to supply Java-based SetTop boxes for video-on-demand, but lost to Silicon Graphics
- Gosling team was to be dismantled, but came up with Web browser implemented in Java (HotJava), that could be extended with Java applets
- Mark Adreessen and Gosling discussed about the possibility of integrating Java in Netscape and the JVM was incorporated in Netscape browser
- Through the wide distribution of Netscape Navigator, Java became used in research and achieved popularity
- Nonetheless its strength and weakness often derive from its original design goals

Microsoft CLI

Programming in different languages is like composing pieces in different keys, particularly if you work at the keyboard. If you have learned or written pieces in many keys, each key will have its own special emotional aura. Also, certain kinds of figurations “lie in the hand” in one key but are awkward in another. So you are channeled by your choice of key. In some ways, even enharmonic keys, such as C-sharp and D-flat, are quite distinct in feeling. This shows how a notational system can play a significant role in shaping the final product.

(*Gödel, Escher, Bach: an eternal golden braid*, Hofstadter, 1980, Chapter X)

CLR and JVM

Note that the essential traits of the execution environment are similar, though there are relevant difference in the design

- Secure
- Portable
- Automatic MM (GC)
- Type safety
- Dynamic loading
- Class Library
- OOP

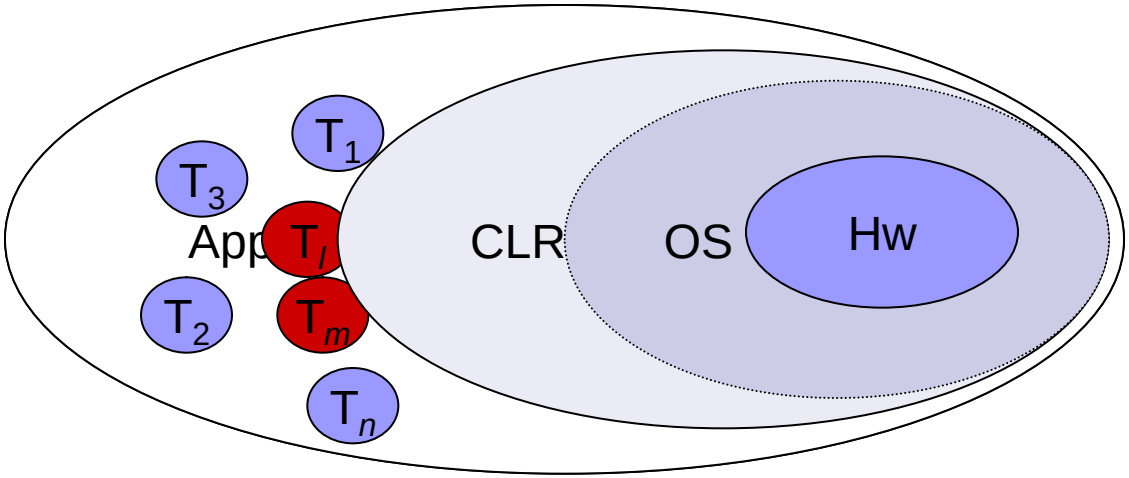
- Mix-in inheritance

CLI has been standardized (ECMA and ISO) and is a superset of Java.
We will refer mainly to CLR, pointing out feature missing from the JVM.

A new layer to the onion

Runtime exposes a superset of OS Services through the BCL

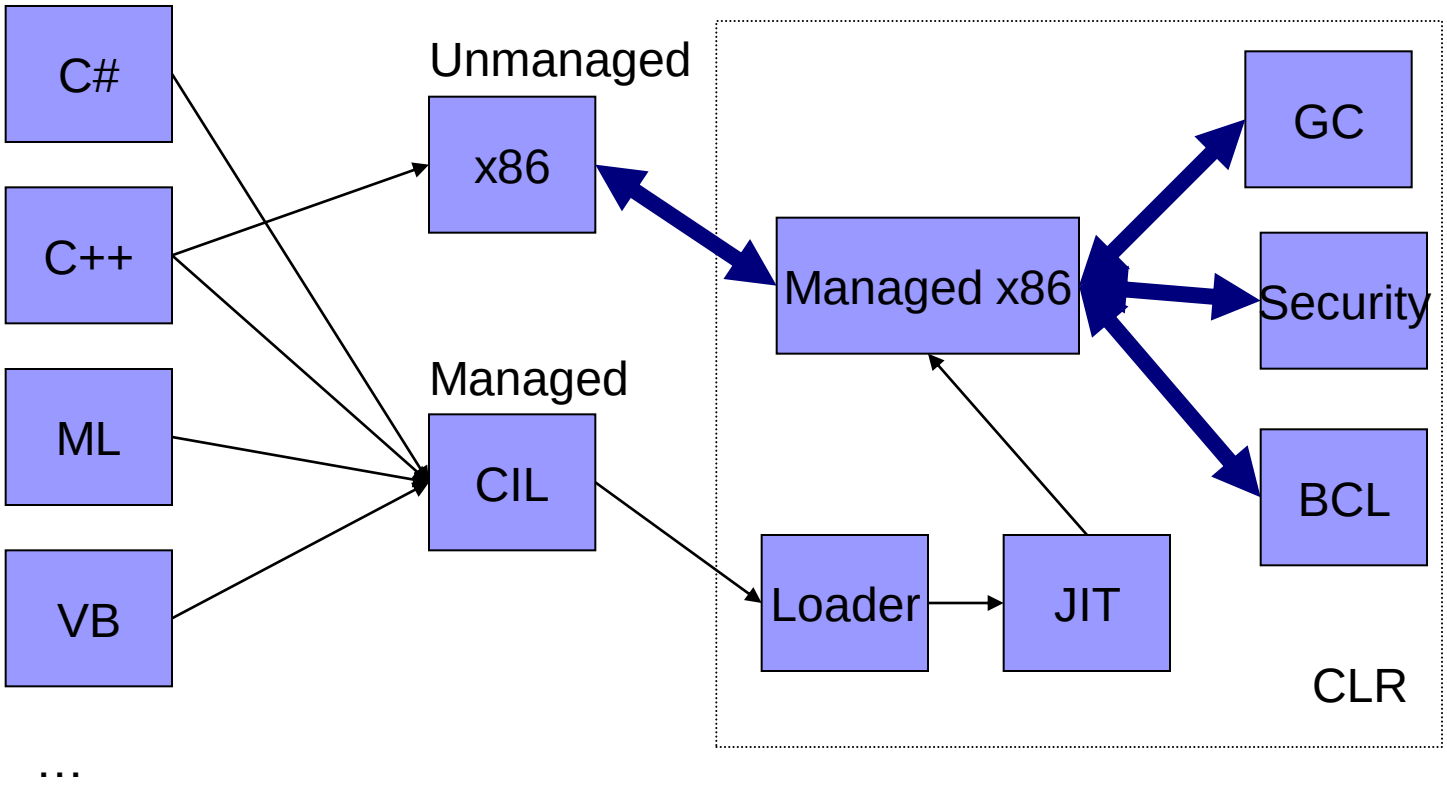
Runtime mediates access between the application and OS



Applications are group of types interacting together

Different runtimes implements in a different way LP abstractions such types: interoperability is complex

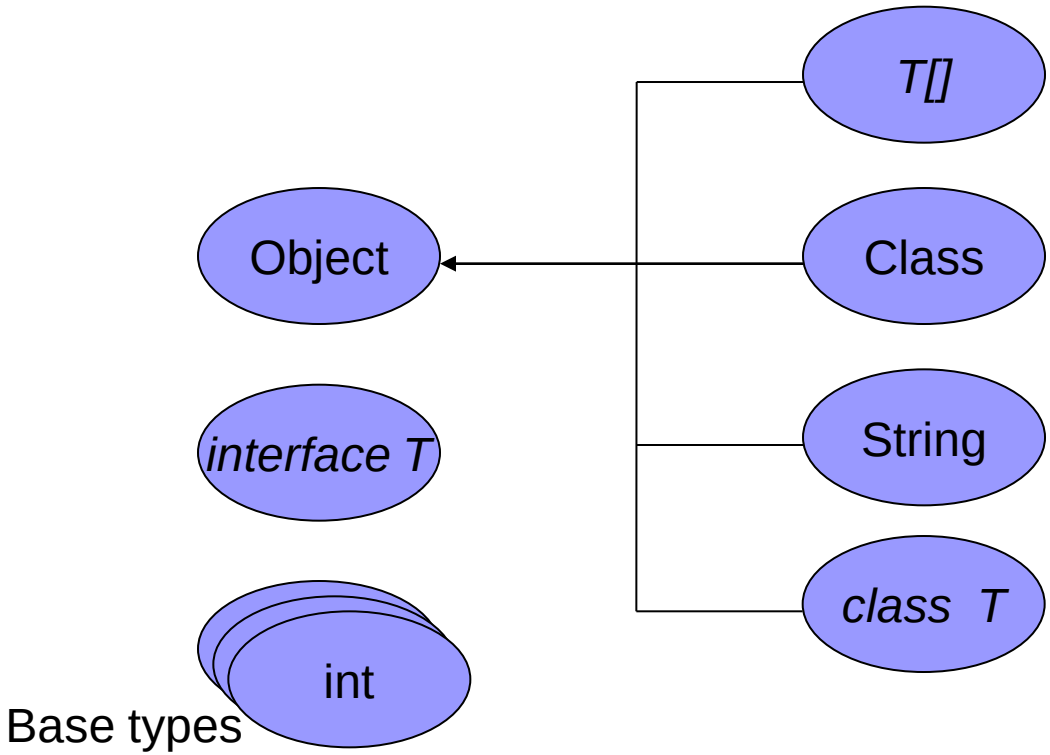
How CLR works



Type system

- Execution environments such as CLR and JVM are *data oriented*
- A type is the unit of code managed by the runtime: loading, code, state and permissions are defined in terms of types
- Applications are set of types that interact together
- One type exposes a static method (Main) which is the entry point of the application: it loads the needed types and creates the appropriate instances

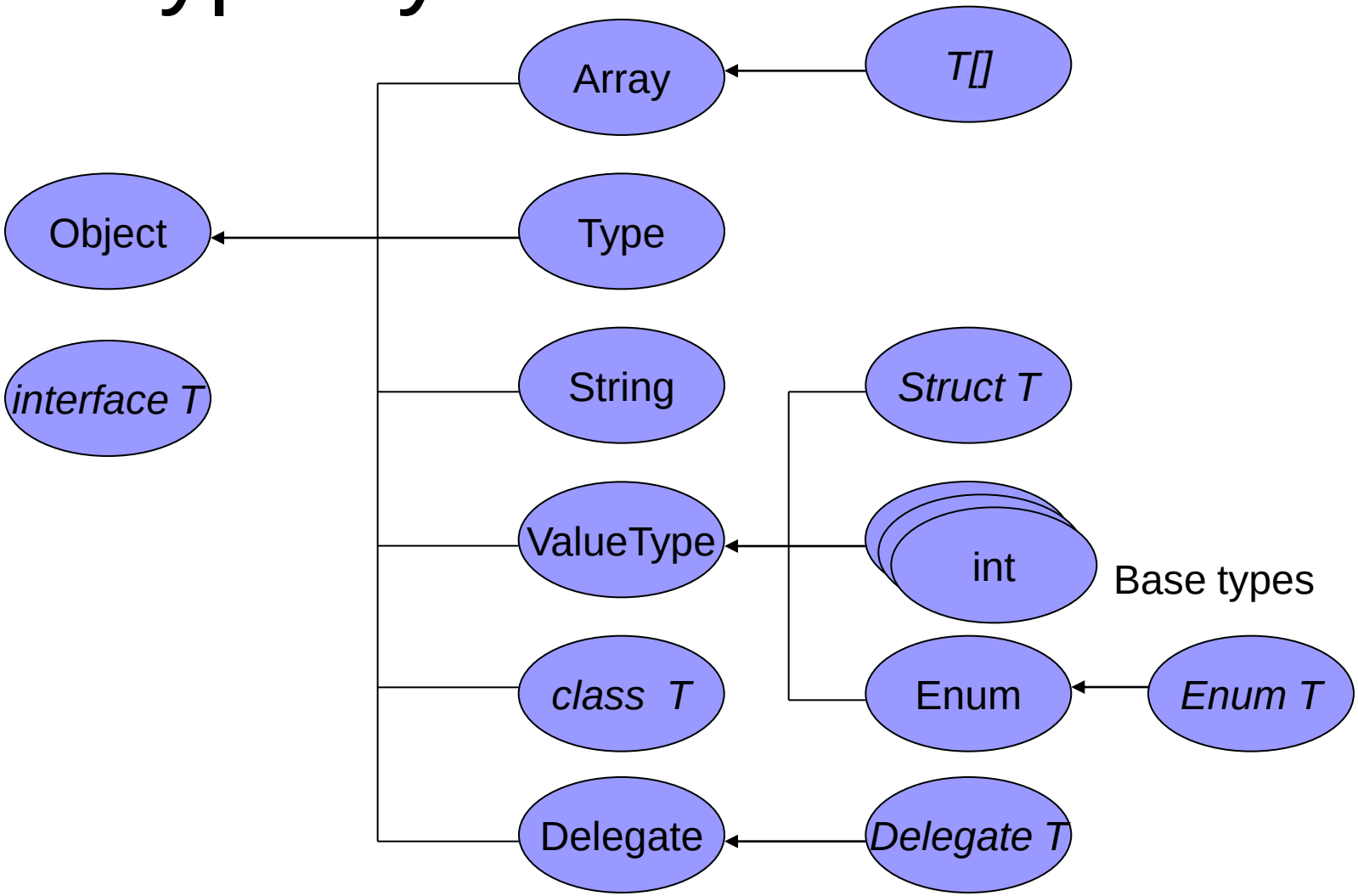
Java type system



Java type system

- There are base types: numbers, Object, String and Class (which is the entry-point for reflection)
- Type constructors are:
  - Array
  - Class
- The number types are unrelated to Object with respect to inheritance relation
- This applies to interfaces too, but objects that implements interfaces are always inherited from object
- Java type system is far simpler than the one of CLR

CLR type system



CLR Type System

- Common rooted: even numbers inherits from Object
- There are more type constructors:
  - Enum: constants
  - Struct: like class but without inheritance and stack allocation
  - Delegate: type that describes a set of methods with common signature
- Value types (numbers and structs) inherits from object. Still are not references and aren't stored on the heap
- The trick is that when a value type should be upcasted to Object it is *boxed* in a wrapper on the heap
- The opposite operation is called *unboxing*

Delegate types

- A delegate is a type that describes a set of callable methods
- Example, static method:

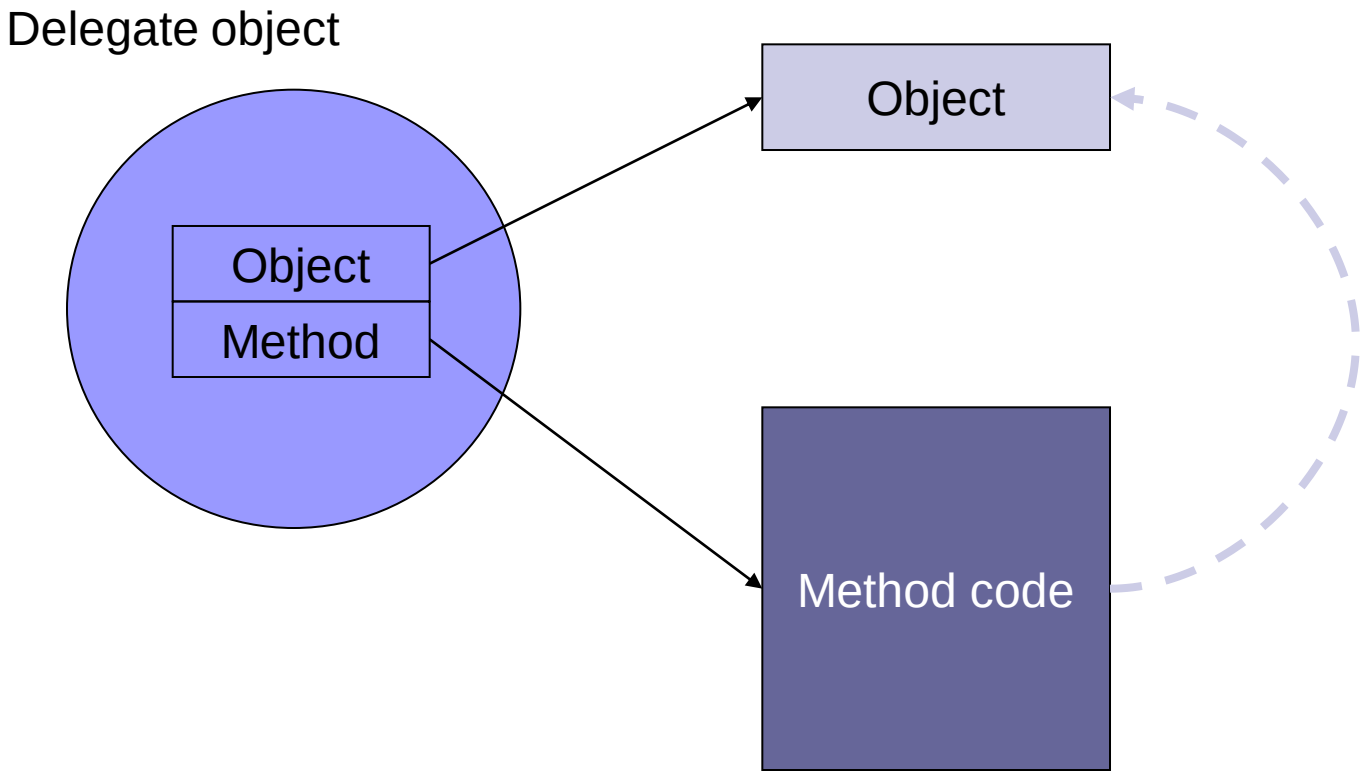
```
class Foo {
 delegate int MyFun(int i, int j);
 static int Add(int i, int j) { return i + j; }
 static void Main(string[] args) {
  MyFun f = new MyFun(Foo.Add);
  Console.WriteLine(f(2, 3));
 }
}
```

Is it a function pointer?

NOOOOOOOOOOOOOOOOOOOOOO

- A delegate is more than a pointer! It is a special object
- To understand what a delegate really is try to answer to: “How a delegate can invoke an instance method?”
- An instance method must be invoked on an object! We may use a pair (object, method)

CLR delegates



Delegates as types

- A delegate type allows building delegate objects on methods with a specified signature
- The type exposes an Invoke method with the appropriate signature at CLR level
- C# provides a special syntax for declaring delegates (not class like)
- The pair is built using the *new* operator and the pair is specified using an invocation-like syntax

Delegates like closures?

- In functional programming it is possible to define a function that refers to external variables
- The behavior of the function depends on those external values and may change
- Closures are used in functional programming to close open terms in functions
- Delegates are *not* equivalent to closures although they are a pair (env, func): the environment should be of the same type to which the method belongs

Functional programming in C#?

- Delegates allow representing static and instance methods as values
- Those values can be passed as arguments
- Methods become first class values
- Introduce elements of FP style in the mainstream, cleaner event model (call-backs can be naturally expressed as delegates)

Example: Mapping

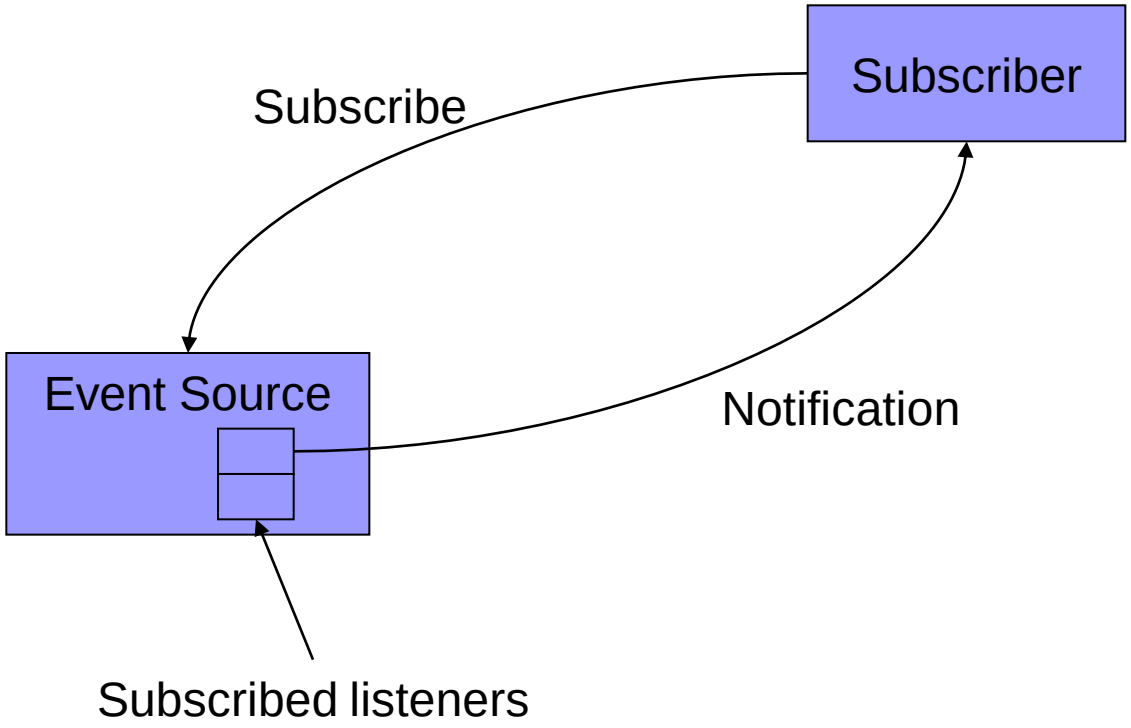
- Performing a mapping on an array:

```
delegate int MyFun(int);
int[] ApplyInt(MyFun f, int[] a) {
  int[] r = new int[a.Length];
  for (int i = 0;i < a.Length;i++)
    r[i] = f(a[i]);
  return r;
}
```

Events using delegates?

- Event systems are built on the notion of notification (call-back)
- A method invocation can be seen as a notification
- In GUI frameworks such as MFC and Java 1.0.2 they were based on virtual methods
- Java 1.1 introduces delegation event model:
  - There are source of events
  - There are listeners that ask sources for notifications
  - Event fires: a method is invoked for each subscriber

Delegation Event Model



Delegate event model in Java

- Which method should call the event source to notify the event?
- In Java there are no delegates and interfaces are used instead (XXXListener)
- The listener must implement an interface and the source provides a method for (un)subscription.
- A vector of subscribed listeners is kept by the event source

Delegates to handle events

- Delegates allow connecting event sources to listeners independent of the types involved
- In C# a delegate object can be used to specify which method must be invoked when an event is fired
- One approach could be to store an array of delegates in the source to represents subscribers
- A component (not necessarily the listener) builds a delegate on the listener and subscribes to an event

Multicast delegates

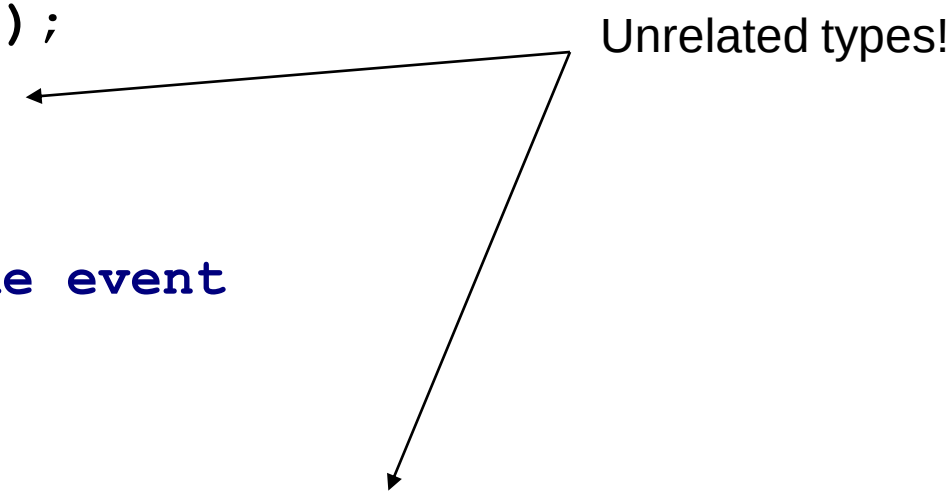
- Event notification is in general one-to-many
- CLR provides multicast delegates to support notification to many listeners
- A multicast delegate is a kind of delegate that holds inside a list of ‘delegate objects’
- Multicast delegates keep track of subscriptions to event sources reducing the burden of replicating the code

Multicast delegates: Example

```
delegate void Event();
class EventSource {
 public Event evt;
  …
  evt(); // fires the event
  …
}

class Foo { public void MyMethod() {} }

// Elsewhere in the program!
EventSource src = new EventSource();
Foo f = new Foo();
src.evt += new Event(f.MyMethod);
```



C# and delegates

- In C# there is no way to choose between single and multicast delegates
- The compiler always generates multicast delegates
- In principle JIT could get rid of possible inefficiencies
- Delegates represent a novel programming pattern

Event keyword

- C# introduces the *event* keyword to control access to a delegate member.
- If a delegate field of a class is labeled with event then outside code will be able to use only += and -= operators on it
- Listener would not be allowed to affect the subscribers list in other ways
- Event infrastructures can be easily implemented by means of this keyword and delegates

Event delegates: Example

```
delegate void Event();
class EventSource {
 public event Event evt;
 …
  evt(); // fires the event
 …
}

class Foo { public void MyMethod() {} }

// Elsewhere in the program!
EventSource src = new EventSource();
Foo f = new Foo();
src.evt += new Event(f.MyMethod);
src.evt = null; // ERROR!
```