Delegates and Events

Objectives

- **Introduce delegates and events**
  - defining delegate types
  - invoking through delegates
  - registering multiple targets
- **Discuss design benefits of using delegates**

State

- **Objects typically maintain state**
  - state changes over time

store state →
change state →

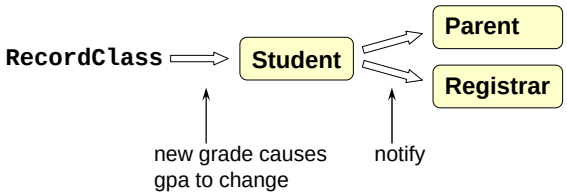
```
class Student
{
  string name;
  double gpa;
  int    units;

  public void RecordClass(int grade)
  {
    gpa = (gpa * units + grade) / (units + 1);

    units++;
  }
  ...
}
```

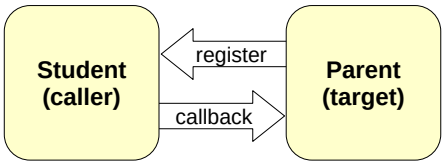

Notification

- **May want to notify interested parties of state change**
  - notification widely used throughout .NET framework
  - user interface event handling most common example



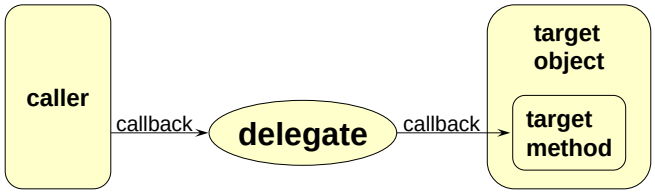
Pattern

- **Notification typically involves *registration* and *callback***
  - target registers with caller
  - caller calls back target when state changes
  - pattern also called *publish/subscribe*



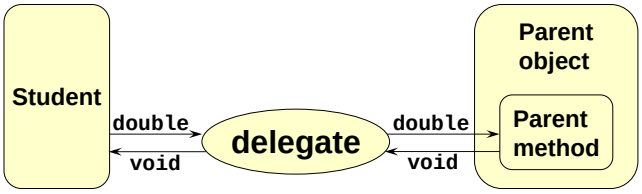
Delegates

- **.NET Framework uses *delegates* to implement callbacks**
  - intermediary between caller and target
  - declaration defines callback method signature
  - instance stores object reference and method token



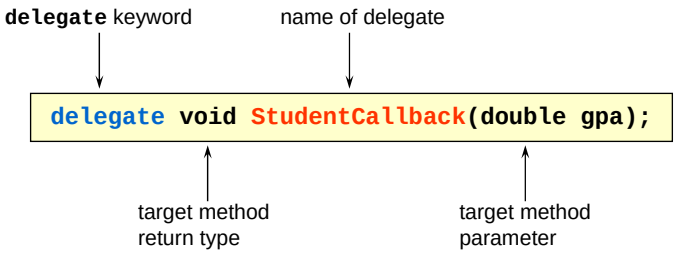
Information flow

- **Caller and target need to agree on information flow**
  - data passed through delegate



Delegate definition

- **Define delegate with `delegate` keyword**
  - syntax similar to method declaration without body
  - delegate name placed where method name usually goes



Delegate as type

- **Delegate name is type name**
  - can declare references
  - can create objects

define delegate →

```
delegate void StudentCallback(double gpa);
```

```
StudentCallback a = new StudentCallback(...);
```

reference object

Target use of delegate

- **Target defines method with signature specified by delegate**
  - parameters and return type must match
  - method name not constrained

delegate defines required signature →

```
delegate void StudentCallback(double gpa);
```

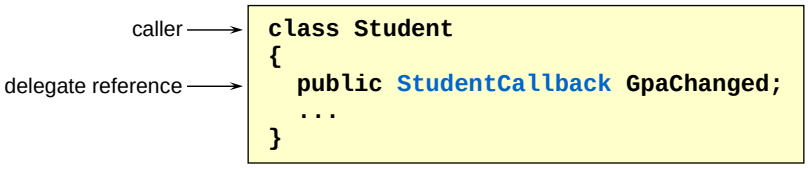
target →

method signature matches delegate →

```
class Parent
{
  public void Report(double gpa)
  {
    ...
  }
}
```

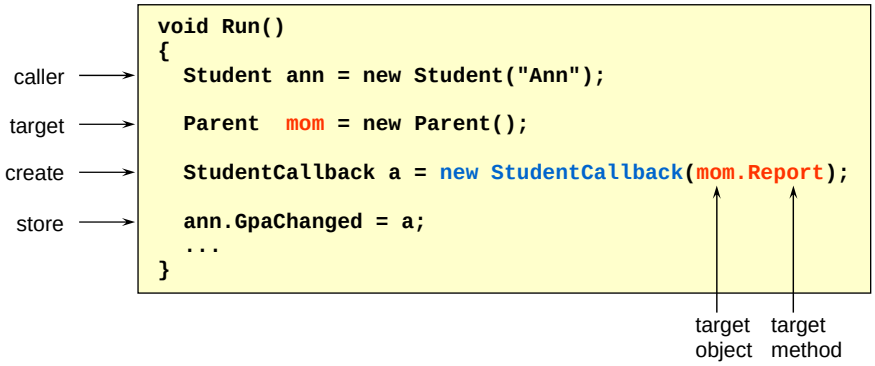
Caller use of delegate

- **Caller typically defines delegate reference**



Registration

- **Create delegate object and store in caller to register**
  - pass target object and method to delegate constructor



Invocation

- **Caller invokes callback indirectly through delegate**
  - uses method call syntax on delegate
  - delegate calls target method on target object

invoke callback through delegate →

```
class Student
{
  public StudentCallback GpaChanged;

  public void RecordClass(int grade)
  {
    // update gpa
    ...
    GpaChanged(gpa);
  }
}
```

callback takes **double** argument

Summary of delegate use

define delegate →

target method →

caller stores delegate →

caller invokes delegate →

```
delegate void StudentCallback(double gpa);

class Parent
{
  public void Report(double gpa) { ... }
}

class Student
{
  public StudentCallback GpaChanged;

  public void RecordClass(int grade)
  {
    // update gpa
    ...
    GpaChanged(gpa);
  }
}
```

create and install delegate →

```
Student ann = new Student("Ann");
Parent  mom = new Parent();

ann.GpaChanged = new StudentCallback(mom.Report);
ann.RecordClass(4); // 4 == 'A'
```

Null reference

- **Delegate is reference type**
  - defaults to **`null`** when used as field
  - typical to guard invocation

test before call →

```
class Student
{
  public StudentCallback GpaChanged;

  public void RecordClass(int grade)
  {
    // update gpa
    ...
    if (GpaChanged != null)
      GpaChanged(gpa);
  }
```


Static methods

- **Static method may be target of delegate**
  - use class name and method name when creating delegate
  - no object specified since no **`this`** object associated with call

static method →

```
class Registrar
{
  public static void Log(double gpa)
  {
    ...
  }
}
```

register →

```
void Run()
{
  Student ann = new Student("Ann");

  ann.GpaChanged = new StudentCallback(Registrar.Log);
  ...
}
```

Multiple targets

- **Can combine delegates using operator+= or operator+**
  - creates *invocation list* of delegates
  - all targets called when delegate invoked
  - targets called in order added
  - use of **+=** ok even when left-hand-side is **`null`**

targets →

```
Parent mom = new Parent();
Parent dad = new Parent();

Student ann = new Student("Ann");

ann.GpaChanged += new StudentCallback(mom.Report);
ann.GpaChanged += new StudentCallback(dad.Report);
...
```

first →
second →

Remove delegate

- **Can remove delegate from invocation list**
  - use operator **-=** or operator **-**
  - identity of target object/method determines which is removed

```
Parent mom = new Parent();
Parent dad = new Parent();

Student ann = new Student("Ann");

ann.GpaChanged += new StudentCallback(mom.Report);
ann.GpaChanged += new StudentCallback(dad.Report);
...
ann.GpaChanged -= new StudentCallback(dad.Report);
...
```

add →
remove →

Public delegate

- **Not common to have public delegate field**
  - allows assignment: could overwrite existing registrants
  - allows external invocation: decision should be made internally

public delegate →

```
class Student
{
  public StudentCallback GpaChanged;
  ...
}
```

```
Parent  mom = new Parent();
Parent  dad = new Parent();
Student ann = new Student("Ann");
...
ann.GpaChanged = new StudentCallback(mom.Report);
ann.GpaChanged = new StudentCallback(dad.Report);
...
ann.GpaChanged(4.0);
...
```

overwrite **mom** handler →

invoke →

Events

- **Events give private data/public accessor pattern for delegates**
  - created by applying **event** keyword to delegate
  - external code can uses **+=** and **-=**
  - no external assignment or invocation

event →

```
class Student
{
  public event StudentCallback GpaChanged;
  ...
}
```

```
Parent  mom = new Parent();
Student ann = new Student("Ann");

ann.GpaChanged += new StudentCallback(mom.Report);

ann.GpaChanged  = new StudentCallback(mom.Report);
ann.GpaChanged(4.0);
...
```

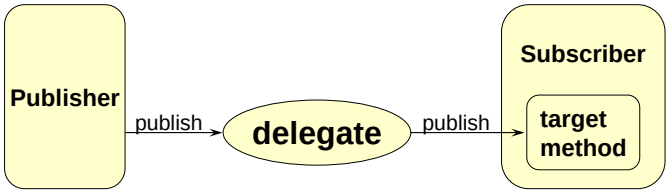
ok to use **+=** →

error to use **=** →

error to invoke →

Design

- **Delegates provide clean way to code publish/subscribe**
  - publisher independent of type of subscriber
  - publisher independent of target method name
  - subscriber constrained only by callback method signature
  - promotes loosely coupled designs



Summary

- **Delegates are objects that invoke methods on other objects**
  - useful to implement callbacks
- **Events add semantic/syntactic layer to delegates**
  - enforce clean and safe registration/deregistration protocols