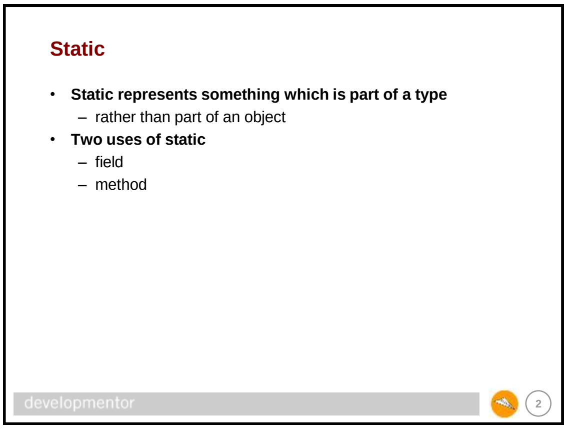
5: Static

12/01/2003

Static

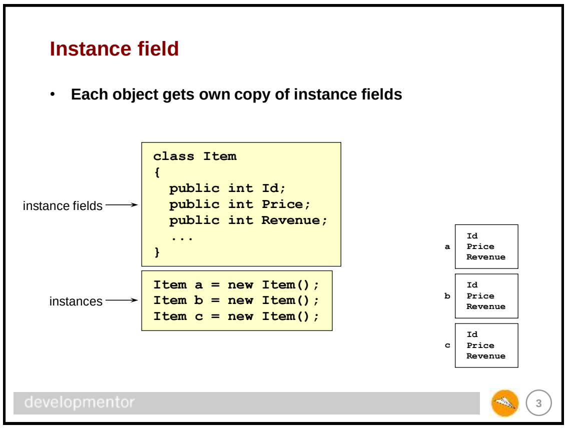
(modified)

developmentor



Static

- **Static represents something which is part of a type**
  - rather than part of an object
- **Two uses of static**
  - field
  - method



Instance field

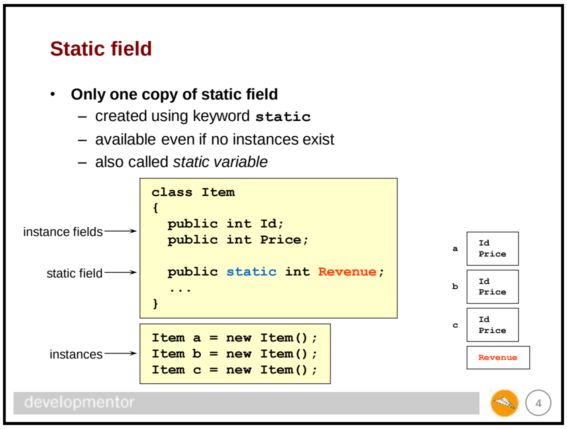
- **Each object gets own copy of instance fields**

instance fields

```
class Item
{
  public int Id;
  public int Price;
  public int Revenue;
  ...
}
```

instances

```
Item a = new Item();
Item b = new Item();
Item c = new Item();
```



Static field

- **Only one copy of static field**
  - created using keyword **`static`**
  - available even if no instances exist
  - also called *static variable*

instance fields

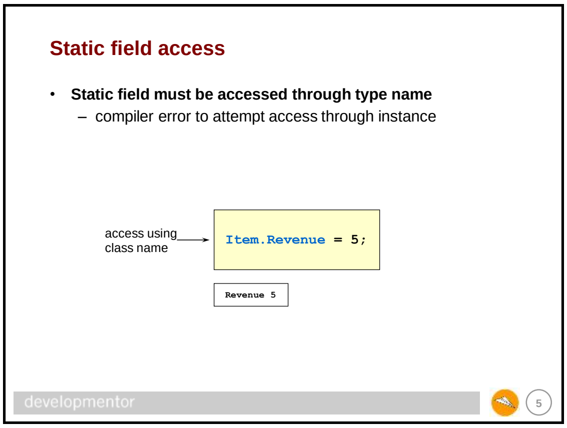
static field

```
class Item
{
  public int Id;
  public int Price;

  public static int Revenue;
  ...
}
```

instances

```
Item a = new Item();
Item b = new Item();
Item c = new Item();
```

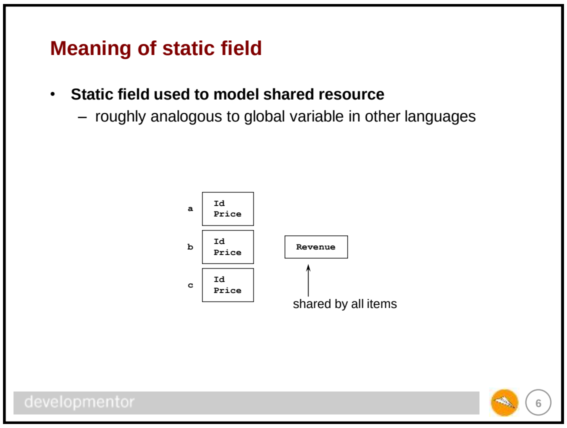


Static field access

- **Static field must be accessed through type name**
  - compiler error to attempt access through instance

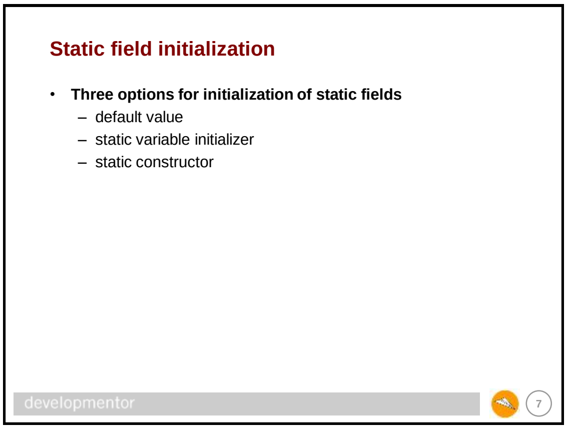
access using class name

```
Item.Revenue = 5;
```



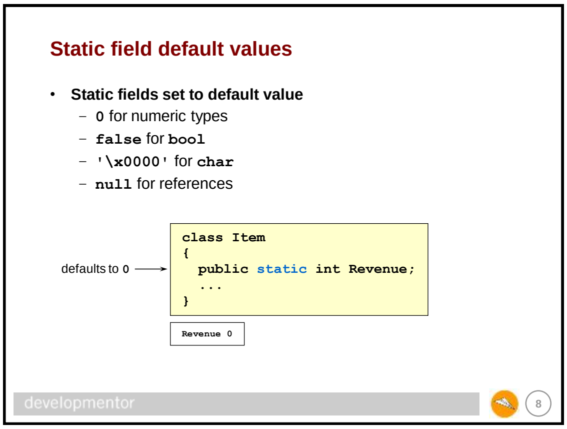
Meaning of static field

- **Static field used to model shared resource**
  - roughly analogous to global variable in other languages



Static field initialization

- **Three options for initialization of static fields**
  - default value
  - static variable initializer
  - static constructor



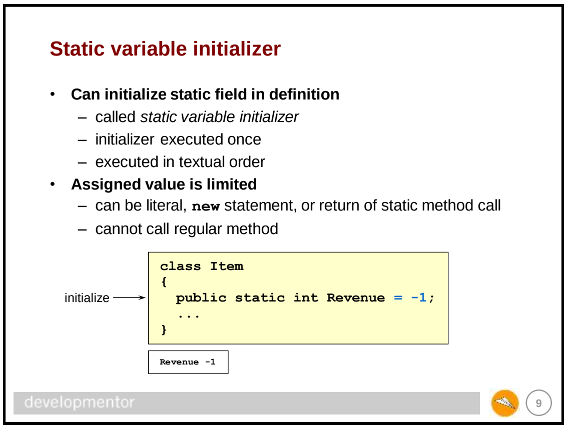
Static field default values

- **Static fields set to default value**
  - `0` for numeric types
  - **`false`** for **`bool`**
  - `'\x0000'` for **`char`**
  - **`null`** for references

defaults to 0

```
class Item
{
  public static int Revenue;
  ...
}
```

Revenue 0



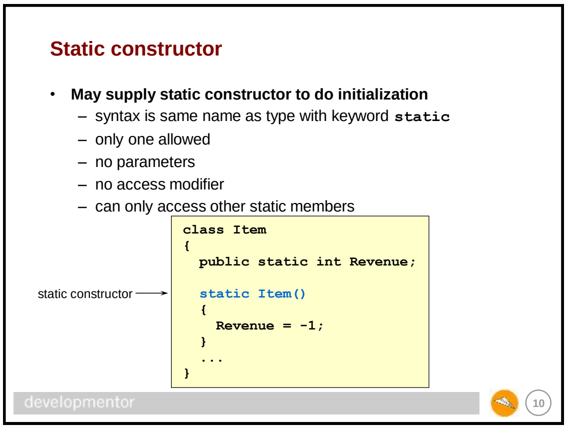
Static variable initializer

- **Can initialize static field in definition**
  - called *static variable initializer*
  - initializer executed once
  - executed in textual order
- **Assigned value is limited**
  - can be literal, **`new`** statement, or return of static method call
  - cannot call regular method

initialize

```
class Item
{
  public static int Revenue = -1;
  ...
}
```

Revenue -1



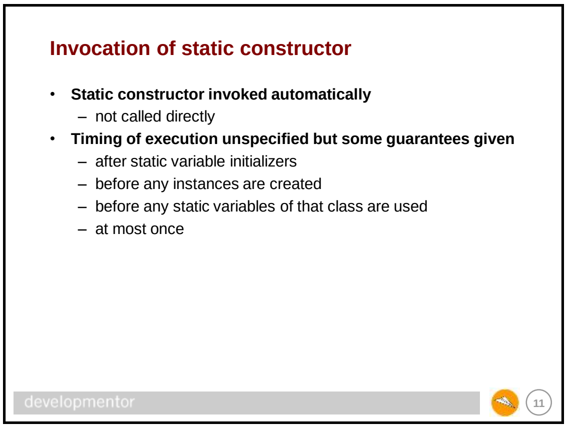
Static constructor

- **May supply static constructor to do initialization**
  - syntax is same name as type with keyword **`static`**
  - only one allowed
  - no parameters
  - no access modifier
  - can only access other static members

static constructor

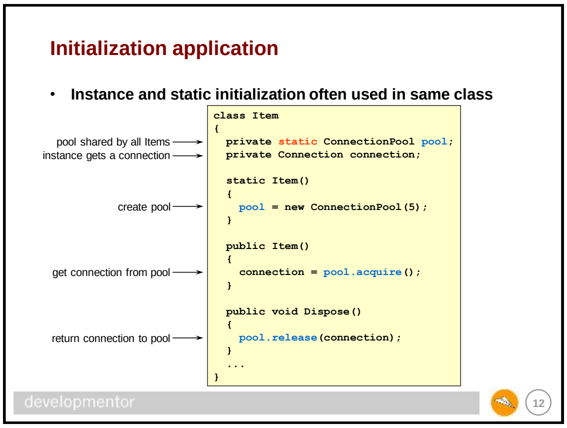
```
class Item
{
  public static int Revenue;

  static Item()
  {
    Revenue = -1;
  }
  ...
}
```



Invocation of static constructor

- **Static constructor invoked automatically**
  - not called directly
- **Timing of execution unspecified but some guarantees given**
  - after static variable initializers
  - before any instances are created
  - before any static variables of that class are used
  - at most once



Initialization application

- **Instance and static initialization often used in same class**

pool shared by all Items

instance gets a connection

create pool

get connection from pool

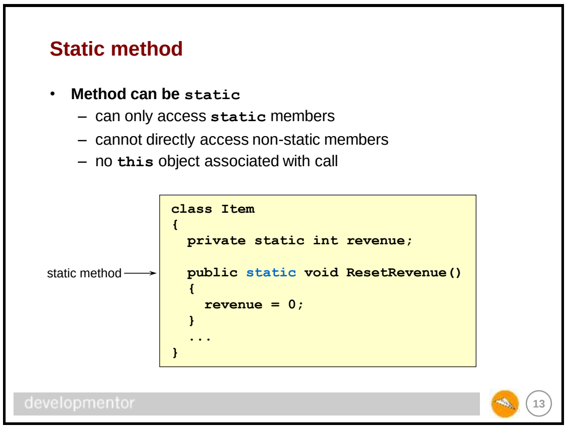
return connection to pool

```
class Item
{
  private static ConnectionPool pool;
  private Connection connection;

  static Item()
  {
    pool = new ConnectionPool(5);
  }

  public Item()
  {
    connection = pool.acquire();
  }

  public void Dispose()
  {
    pool.release(connection);
  }
  ...
}
```



Static method

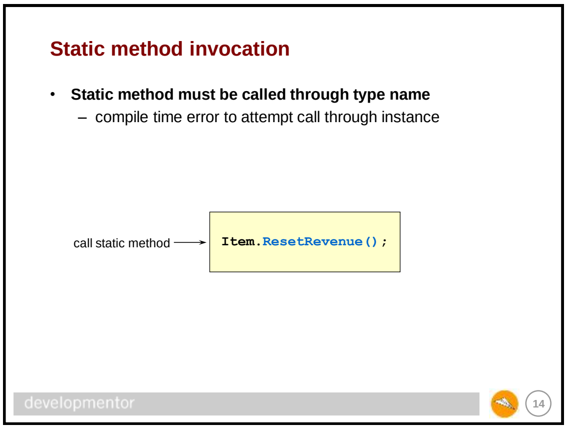
- **Method can be `static`**
  - can only access **`static`** members
  - cannot directly access non-static members
  - no **`this`** object associated with call

static method →

```
class Item
{
  private static int revenue;

  public static void ResetRevenue()
  {
    revenue = 0;
  }
  ...
}
```

developmentor
13



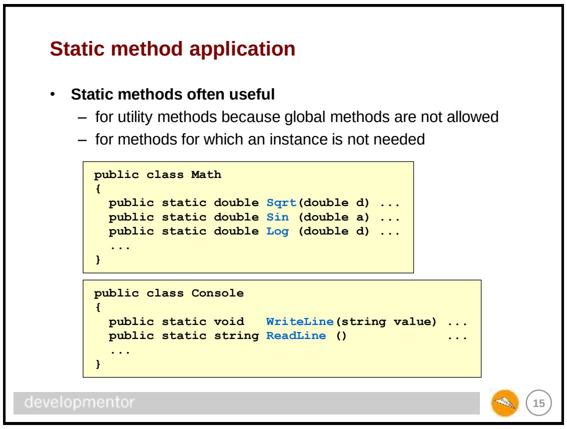
Static method invocation

- **Static method must be called through type name**
  - compile time error to attempt call through instance

call static method →

```
Item.ResetRevenue();
```

developmentor
14



Static method application

- **Static methods often useful**
  - for utility methods because global methods are not allowed
  - for methods for which an instance is not needed

```
public class Math
{
  public static double Sqrt(double d) ...
  public static double Sin (double a) ...
  public static double Log (double d) ...
  ...
}
```

```
public class Console
{
  public static void   WriteLine(string value) ...
  public static string ReadLine ()             ...
  ...
}
```