# COMP1400 Week 9

Monday, 17 September 12

# Grouping data

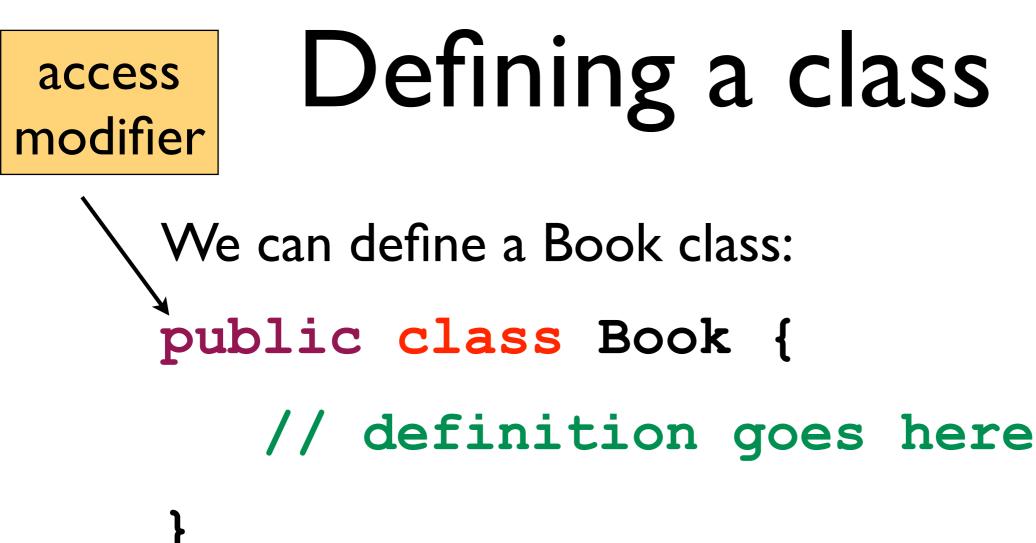
Often we want to group a collection of data together as a single object.

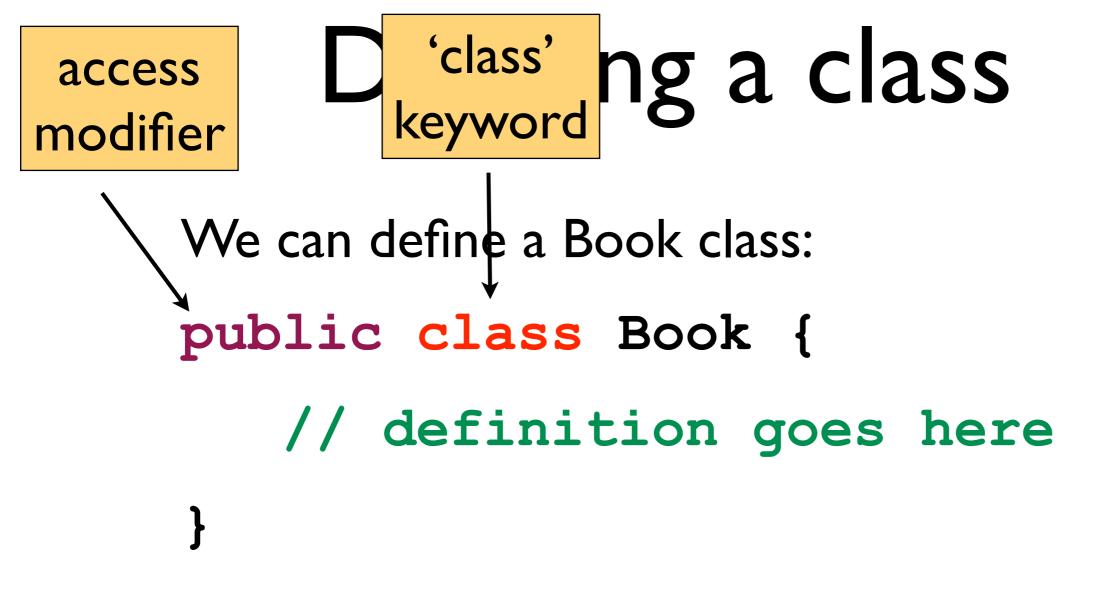
E.g. A book may have the data:

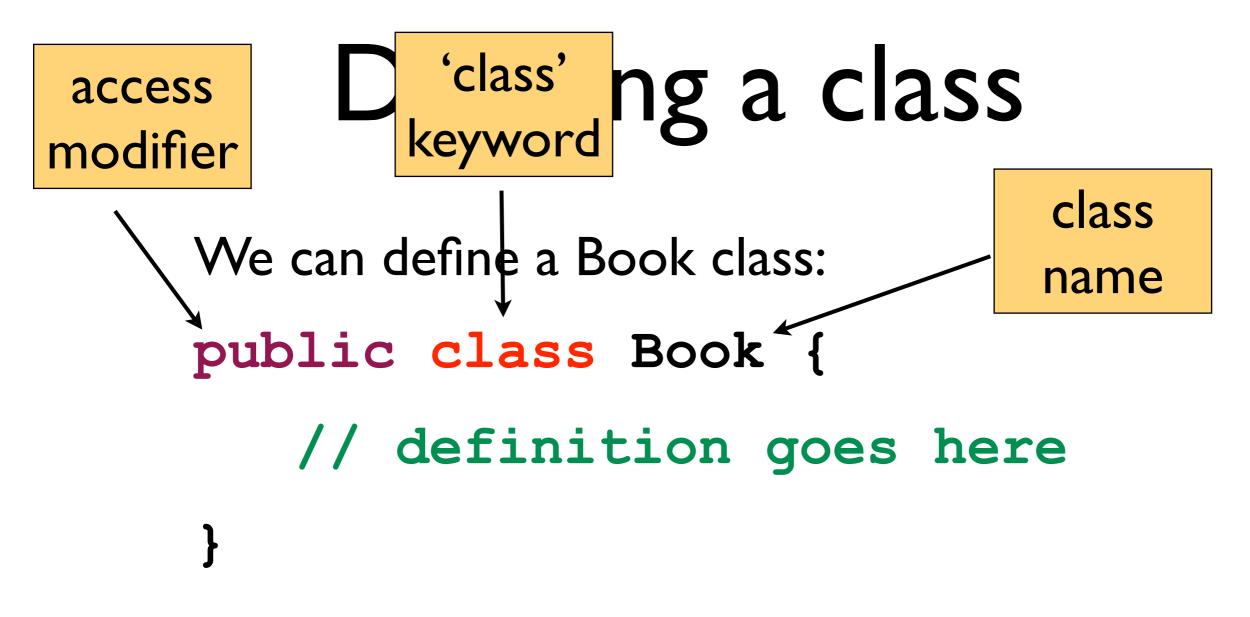
- Author
- Title
- Year published

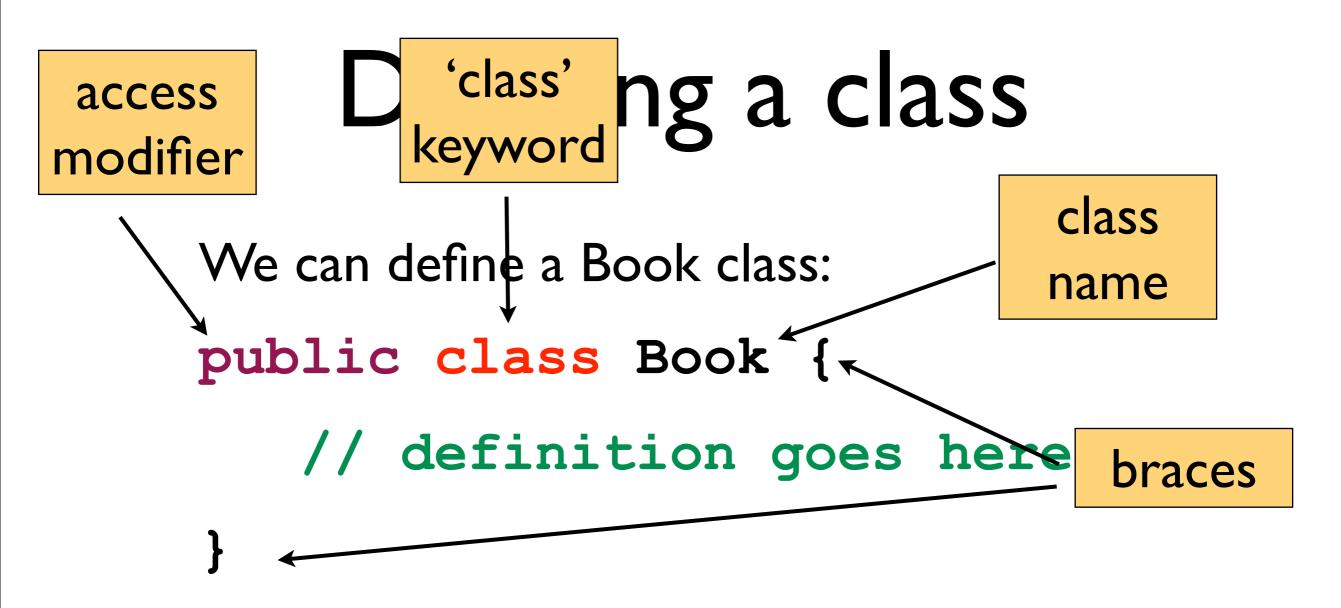
# Defining a class

We can define a Book class:
public class Book {
 // definition goes here
}









#### Fields

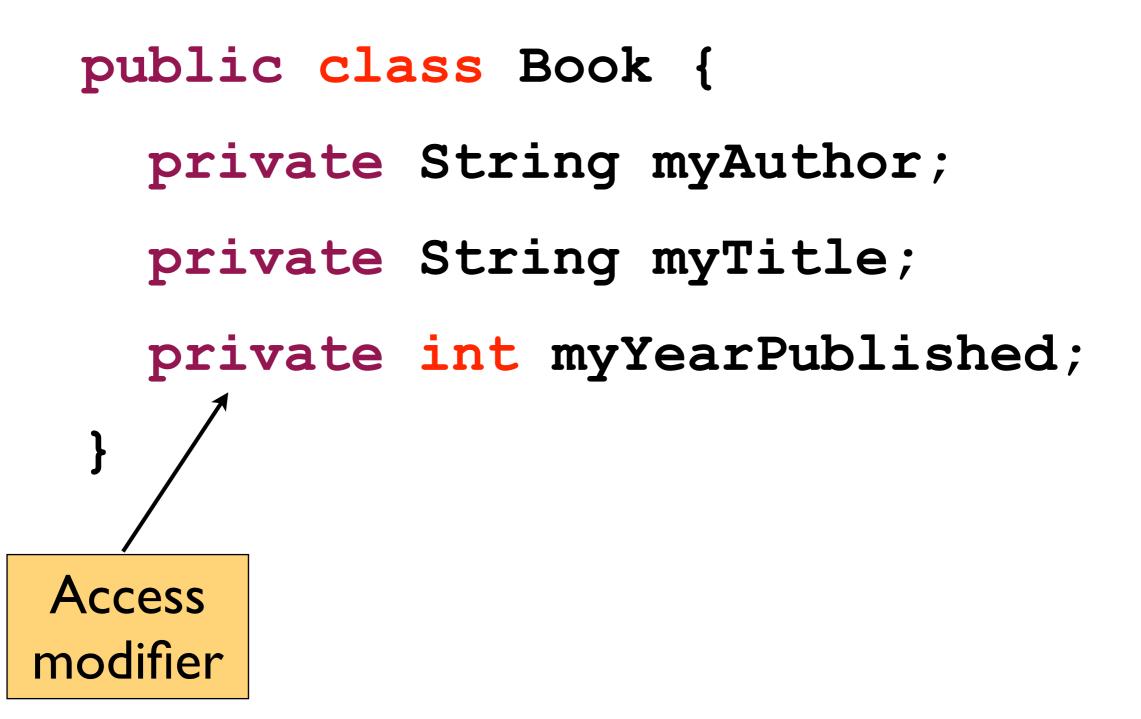
To store the data we create fields in the class.

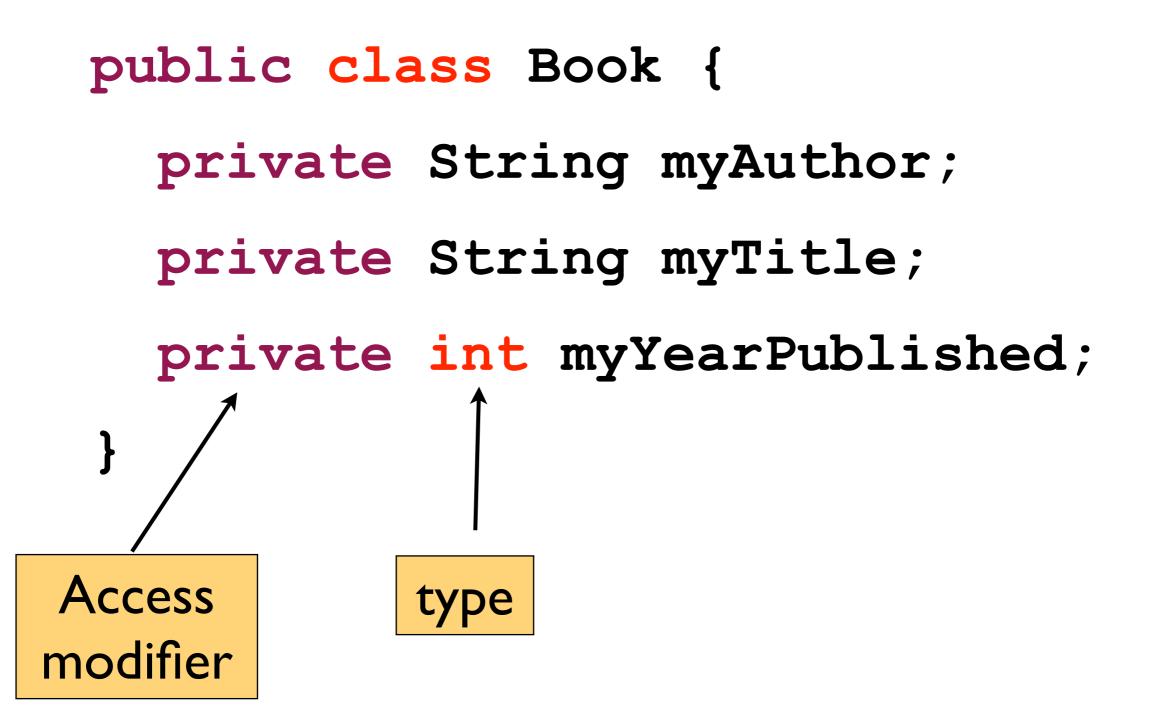
Fields are variables that belong to the object.

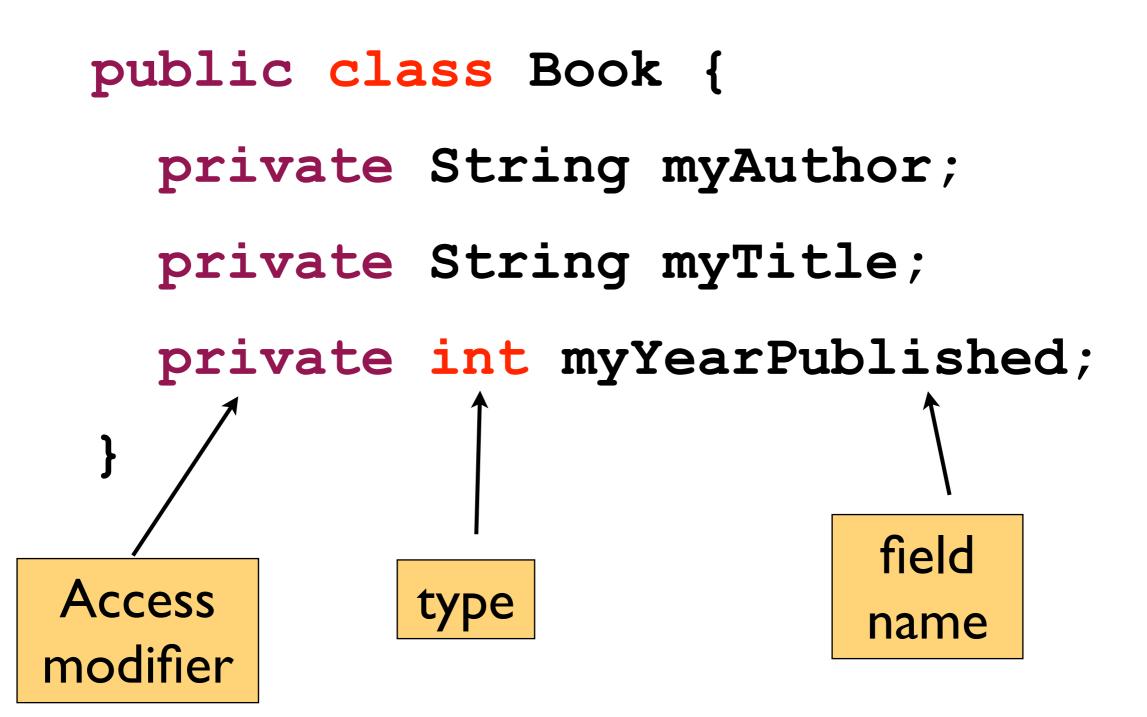
Fields can be read or written like any other variable. They are accessible anywhere within the defining class.

public class Book {
 private String myAuthor;
 private String myTitle;
 private int myYearPublished;

}







## Access modifiers

We give classes, fields and methods an access modifier to show who can use them.

Public classes, fields and methods are accessible by any object.

Private classes, fields and methods are only accessible by the defining object.

We usually make fields private and classes public.

# Style

Fields are variables so we give them meaningful bumpyCaps names starting with a lowercase letter.

It is good practice to give your fields a **distinctive naming convention** to distinguish them from normal variables.

I call all my fields "my ..."

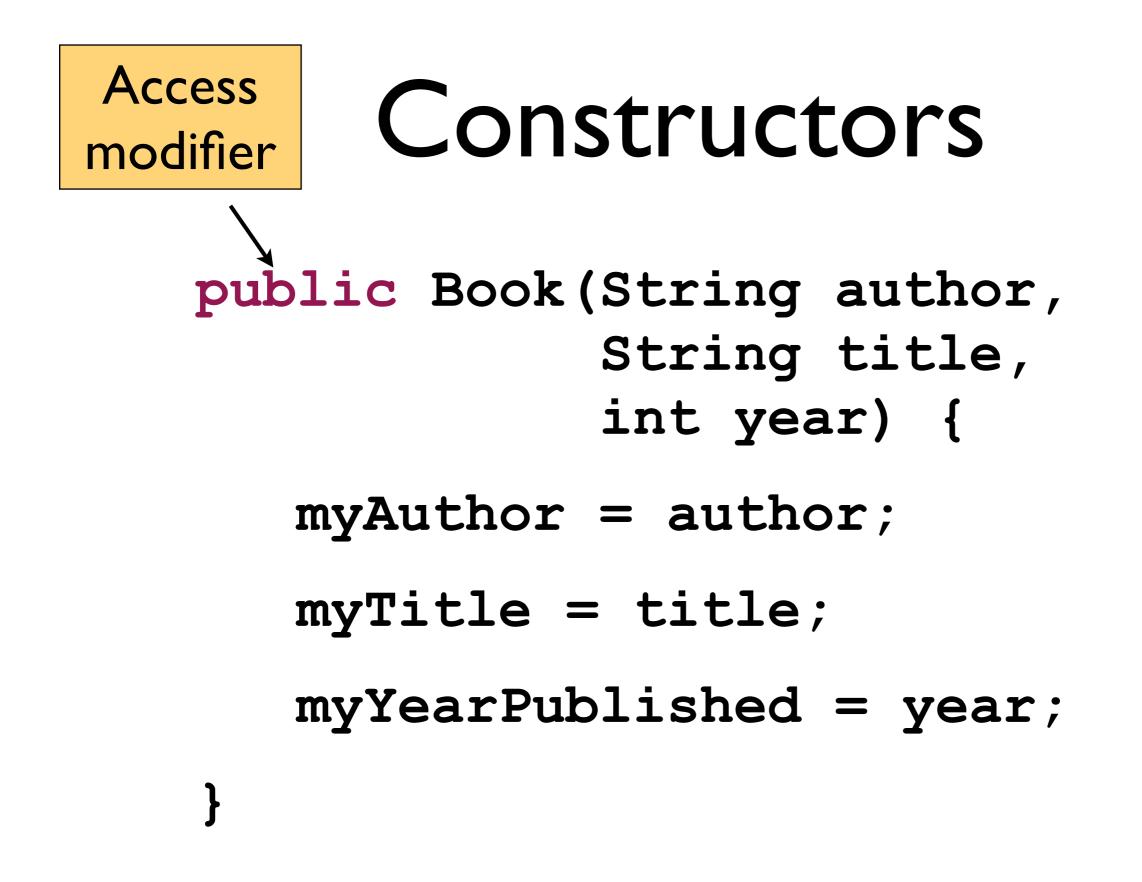
#### Constructors

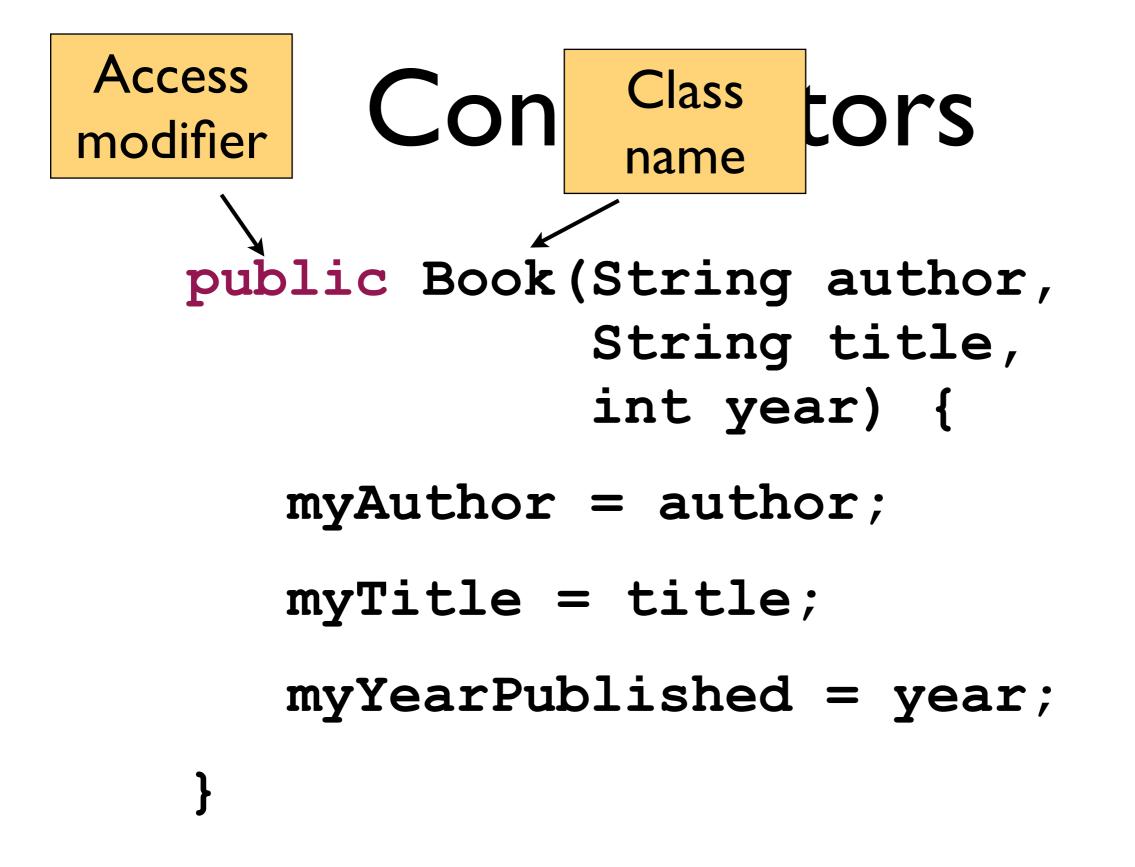
We need to initialise the fields when the object is created.

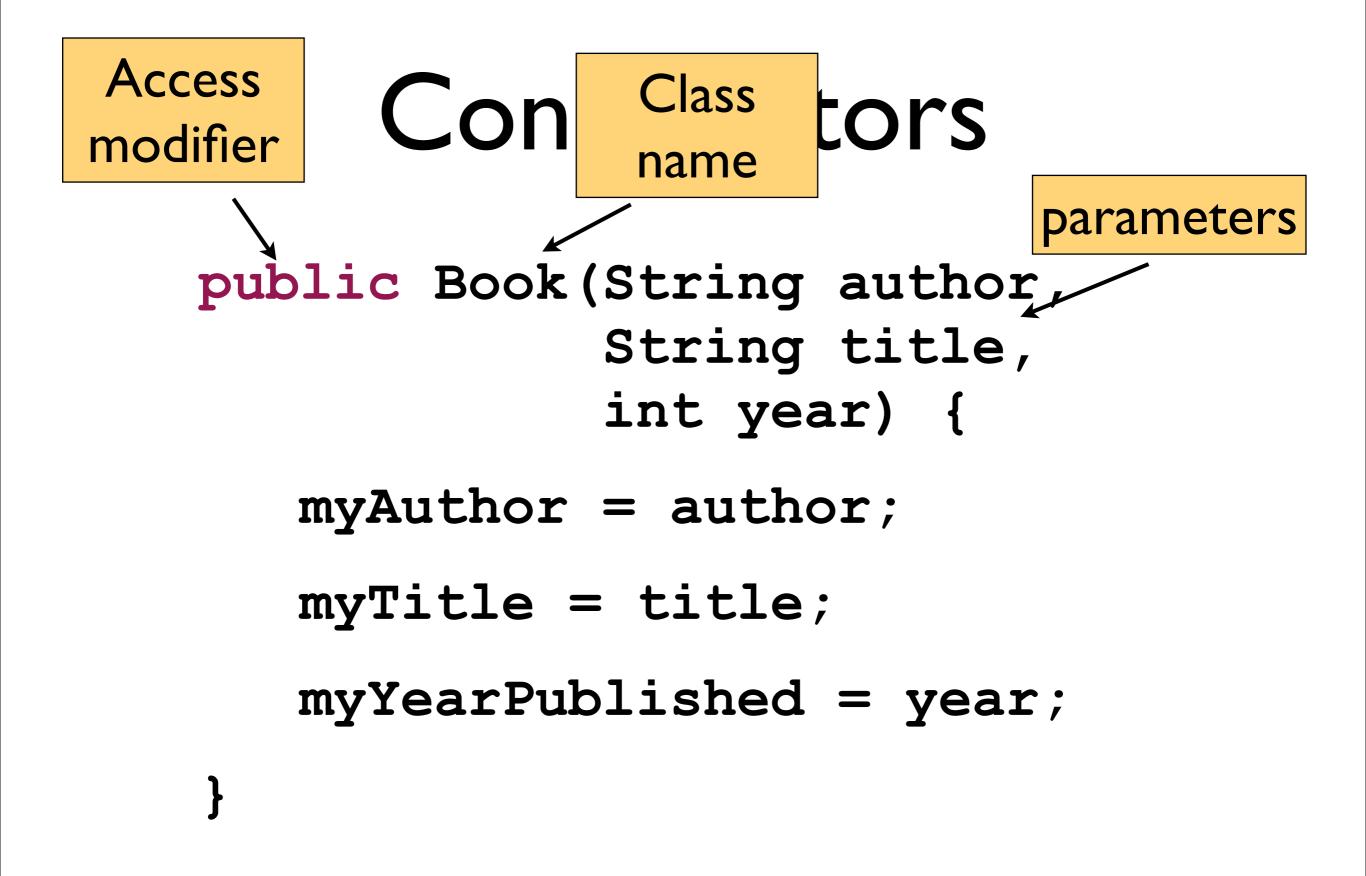
This is the job of the constructor.

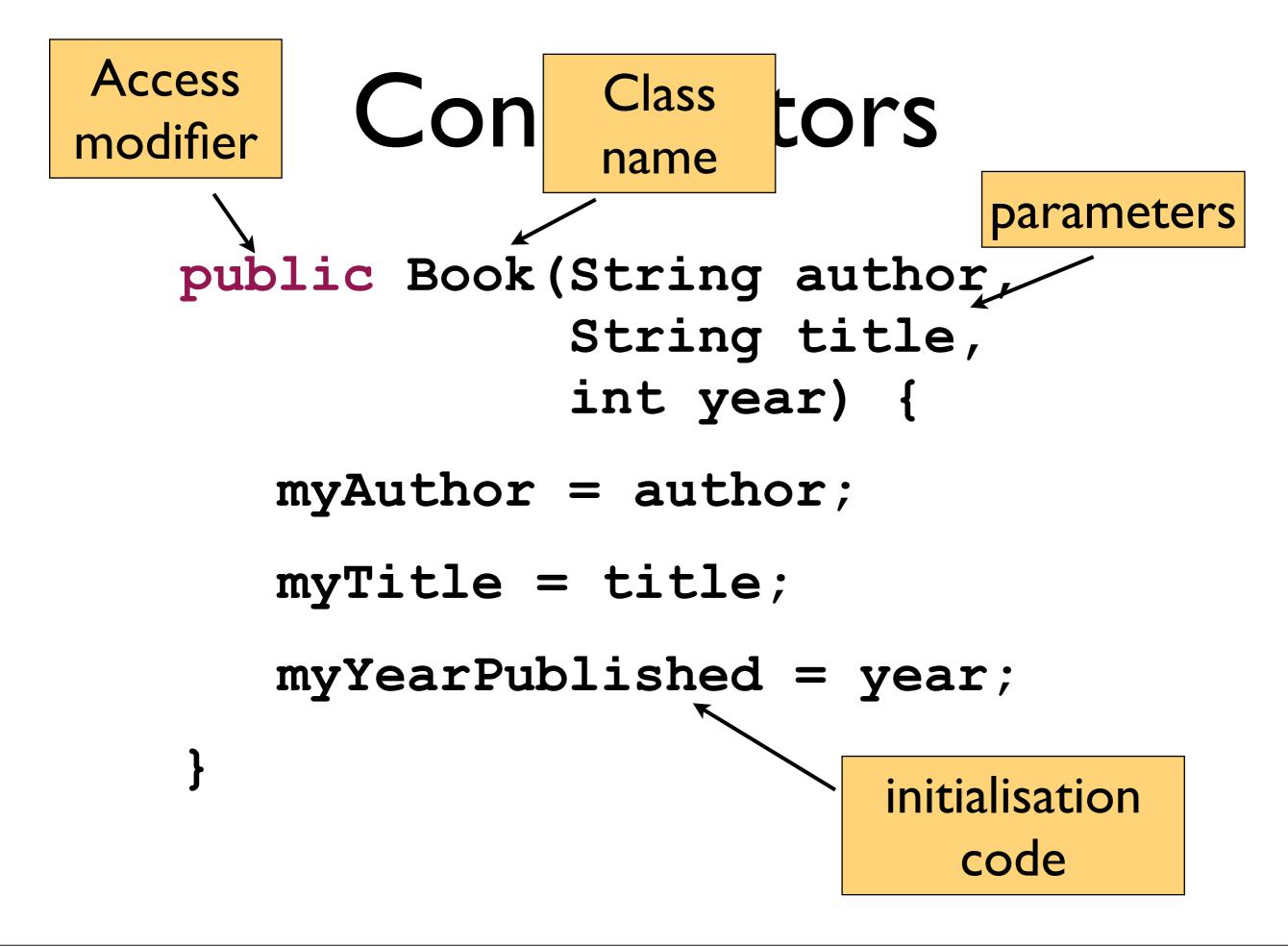
#### Constructors

- - myAuthor = author; myTitle = title; myYearPublished = year;









# Calling a constructor

We can now call this constructor with the 'new' keyword as we've seen before:

Book alice =

```
new Book(
   "Lewis Carroll",
   "Alice in Wonderland",
   1788);
```

## Multiple constructors

We can have multiple constructors with different parameters on the same class.

Java chooses whichever one matches the parameters given when it is called.

This is useful for providing default values for some fields.

## Multiple constructors

A second Book constructor with a default value for author:

```
myAuthor = "Anon.";
myTitle = title;
myYearPublished = year;
```

### Multiple constructors

// use first constructor Book nineteen84 =new Book("George Orwell", "1984", 1949); // use second constructor Book beowulf =new Book("Beowulf", 900);

## Accessor methods

Private fields can only be accessed within the object. This provides encapsulation.

We write accessor methods to allow other objects to read private fields (but not change them).

### Accessor methods

public String getAuthor() {
 return myAuthor;
}
public int getYearPublished() {
 return myYearPublished;
}

# Calling methods

We call these methods in the same way as we've seen before:

String title =
 beowulf.getTitle();
// returns "Beowulf"
String who = beowulf.getAuthor():
// returns "Anon."

#### Static

The "static" keyword is used to indicate methods and fields that belong to the class as a whole, not to individual instances (objects).

#### Static

See example in BlueJ...

#### Constants

Often you want to use certain constant values in your code.

It is better to define these values as named constants.

We use the 'final' keyword to indicate a variable is a constant. It cannot change.

We write constant names in a different style - all uppercase with underscores.

#### Constants

```
public static final
   String ANONYMOUS = "Anon.";
public Book (String title,
            int year) {
   myAuthor = ANONYMOUS;
   myTitle = title;
   myYearPublished = year;
```

# Magic numbers

Named constants avoid the "magic number" problem.

Magic numbers are values in our code with no obvious meaning.

They should be avoided. Use constants to give them meaningful names.

# Magic numbers (bad)

Random rng = new Random(499);

int dice = new int[10];

for (int i = 0; i < 10; i++) {

dice[i] = rng.nextInt(6) + 1;

}

# Magic numbers (bad)

Random rng = new Random(499);

int dice = new int[10];

for (int i = 0; i < 10; i++) {

dice[i] = rng.nextInt(6) + 1;

}

# Constants (good)

Random rng = new Random (SEED); int dice = new int[NUM DICE]; for (int i = 0;i < NUM DICE; i++) { data[i] = rnq.nextInt(NUM SIDES) + 1;