Acknowledgements

- Much of this material drawn from
  - R. S. Laramee

- Particularly for the design lectures.

Previously in this Course...

- On the first day of classes we discussed an implementation of Facebook...
- So we put it all in `public static void main`, right?
- Okay, so bad idea...
- But up to now you might be able to get away with it.
- Because your programs are small.
  - Not very many lines of code.
    - Doesn't mean that it's not hard.
    - Small programs can be very conceptually complex.
  - The key thing is the *volume* of code is manageable.
- Now, lets *really* begin to scale up...

How to make things like Facebook

- We are now going to tackle large scale code.
    - Maybe not of this size but closer to this size.
- Can you see why we can't put all this into `main`?.

Design, Abstraction, and Large Software

- Sad Truth:
  - Understanding every line of code in Facebook source is extremely hard.
  - You may be able to do it, but I know that I can't.

- Fortunate Truth:
  - You can divide the problem into smaller components.
  - Precisely define how these bits work.
  - At a higher level you only have interaction between components.
  - But you can still get the details of the components if needed.

An Engineering Tradition

http://en.wikipedia.org/wiki/File:2006_Chevrolet_Impala_SS_LS4_engine.jpg

- Cars don't have all functionalities lumped together!
- Everything is divided into components.
    - Don't need to understand pistons to fix your windshield wipers.
- Why should it be different for software?

A Reminder of the Process

- Requirements Specification:
  - Precisely what software should do,
  - and its limitations.
- Design (we are here):
  - System of how software fulfils requirements.
  - Description public behaviour of components.
  - Communications between components.
- Implementation
- System integration and testing
- Maintenance

History of Software Abstraction

- Top-down design / structured programming:
  - Abstracts the algorithmic process into stages.
  - We deal with functions & composition for design.
  - Strongly related to mathematical functions.
- Abstract Data Types (ADTs)
  - Specify some data to be stored.
  - Specify precise operations on that data.
  - Abstract away implementation details.
  - Sometimes a collection of classes.
    - Called a subsystem.

Operations and Data: Object-Oriented Design

- We always have data and operations on the data
- What data is stored by component: attributes
- What operations are executed on the data: methods
- Group of methods for an object is its behaviour
- Can treat the data and inner workings as private information
- Duplication is our enemy
  - Data should never be duplicated
  - Functionality of a component should also never be duplicated.

An Approach to Object-Oriented Design

1. Determine your system requirements precisely.
2. Identify classes needed to implement this functionality.
    - Classes are the components of the design.
3. Identify responsibilities of the class.
    - The data it is responsible for (attributes).
    - The operations it can execute on that data (methods).
    - The public methods define how the class interfaces with the world.

Encapsulation

Encapsulation:

- Groups data and operations together.
- And hides details from the rest of the system.
  - public: What the class does for outside world.
  - private: Inner workings of the class.
- Classes interact through public methods.
  - This interaction brings about system behaviour.

Main Idea:

- When looking inside the class, we see its private implementation.
- Standing outside the class, we only see public methods.

Software Reuse

- Major goal of good design is reusable software.
  - Object-oriented helps with this goal.
- Applications: complete program meeting specifications.
  - Design goals: maintainability and extensibility.
  - Classes help organise the code – achieves this goal.
- Software component: Reusable code
  - Design goals: generalisability maximum reusability.
  - Black box code allows users to only have to see public methods.

What is a Class?

- A component that has a role to play in our system:
  - In a car... pistons, brakes, and windshield wipers.
- It lives up to it's responsibilities:
  - It does what it says on the tin.
  - It does not do any more or any less.
- It is a precisely defined unit of the system.

- A class consists of:
  - some data that it stores.
  - some operations that it can execute on the data.

Classes vs Instances of Classes

- A class is a blueprint.
    - A description of a data type and operations on it.
- An instance of a class or object is:
    - An actual instance of the class where all attributes given values.
    - A piece of memory where the data specified by the class has values.

Analogy

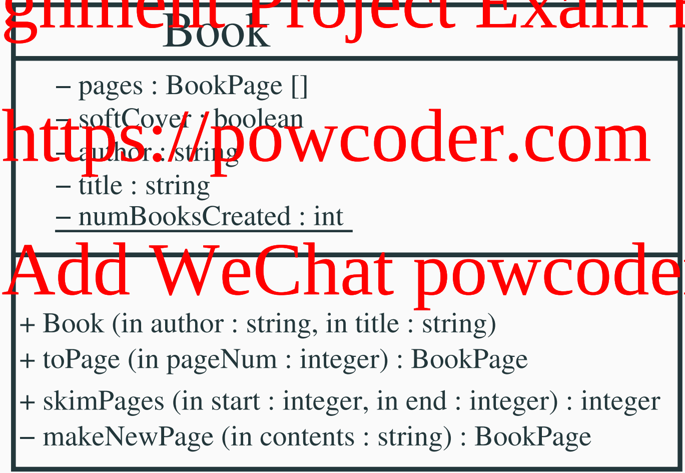
- Example of a Class:
  - `Book`
- Instances of the Class `Book`:
  - "The Hobbit", "Java for Everyone", . . .

Getting Practical: UML Class Diagrams

- Last lecture we introduced UML and the 13 diagram types.
- Now, we're going to learn how to draw Class Diagrams.
- Class Diagrams specify the structure of a class.
    - It does not say how the class works.

UML Diagram of a Class

- A class is represented by a rectangle with 3 areas.
- The class name is always in the first row.



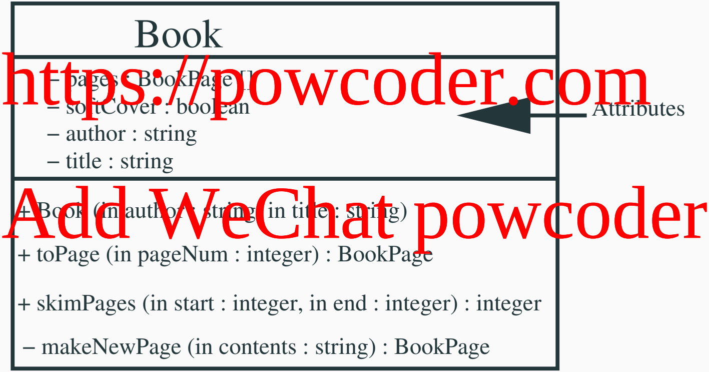
Book

– pages : BookPage []
– softCover : boolean
– author : string
– title : string
– numBooksCreated : int

+ Book (in author : string, in title : string)
+ toPage (in pageNum : integer) : BookPage
+ skimPages (in start : integer, in end : integer) : integer
– makeNewPage (in contents : string) : BookPage

Attributes

- The second row states the attributes.
- Instances of the class store their own data in the attributes.



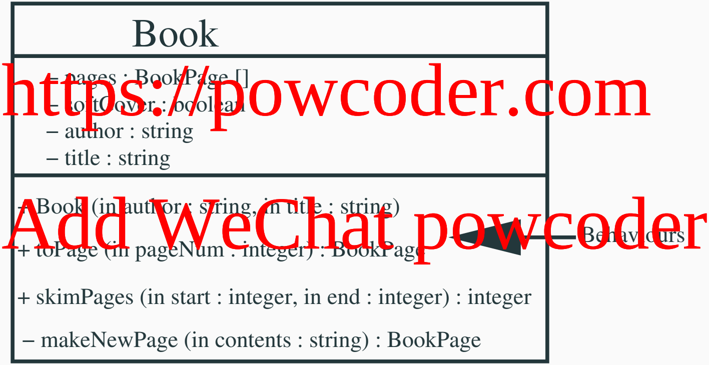
Book

– pages : BookPage []
– softCover : boolean
– author : string
– title : string

+ Book (in author : string, in title : string)
+ toPage (in pageNum : integer) : BookPage
+ skimPages (in start : integer, in end : integer) : integer
– makeNewPage (in contents : string) : BookPage

Operations

- The final row states the operations.
- The operations allow instances to manipulate their data.



Book

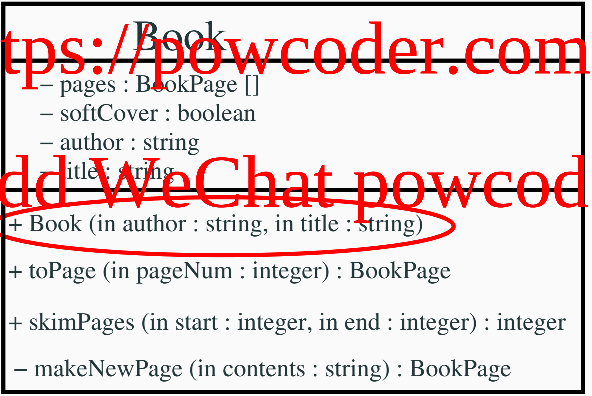
– pages : BookPage []
– softCover : boolean
– author : string
– title : string

+ Book (in author : string, in title : string)
+ toPage (in pageNum : integer) : BookPage
+ skimPages (in start : integer, in end : integer) : integer
– makeNewPage (in contents : string) : BookPage

- Note: “in” and “out” specify input/output parameters (these can be omitted).

Constructors

- Behaviour used to construct a new instance of a class.
- Initialises values stored by the instance.
- In UML, it is the only operation without a return type.



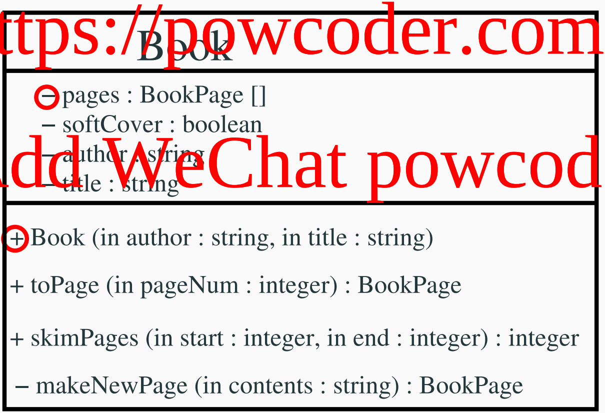
Book

– pages : BookPage []
– softCover : boolean
– author : string
– title : string

+ Book (in author : string, in title : string)
+ toPage (in pageNum : integer) : BookPage
+ skimPages (in start : integer, in end : integer) : integer
– makeNewPage (in contents : string) : BookPage

Visibility

- Attributes and operations have visibilities:
- Public indicated with +, accessible to other classes.
- Private indicated with −, accessible only to this class.
- Protected indicated with #, accessible to subclasses.
- Package indicated with ~, accessible to classes in package.



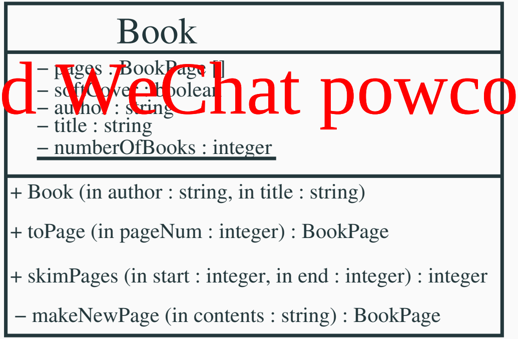
Book

− pages : BookPage []
− softCover : boolean
− author : string
− title : string

+ Book (in author : string, in title : string)
+ toPage (in pageNum : integer) : BookPage
+ skimPages (in start : integer, in end : integer) : integer
− makeNewPage (in contents : string) : BookPage

Static vs Non-Static

- Static, indicated with underline.
- Associated with the class (and not with instances).
- Static data is stored only once in the class.
  All instances share the one copy.
- Example
  - Each book (instance) may have a different colour.
  - There is only one total number of books.



Book

– pages : BookPage []
– softCover : boolean
– author : string
– title : string
– numberOfBooks : integer

+ Book (in author : string, in title : string)
+ toPage (in pageNum : integer) : BookPage
+ skimPages (in start : integer, in end : integer) : integer
– makeNewPage (in contents : string) : BookPage

Note on Previous Slide

Notice how the `pages` attribute is an array of class type `BookPage`.

We really should not be drawing this as an attributes. We should be drawing this as an association – later lecture.

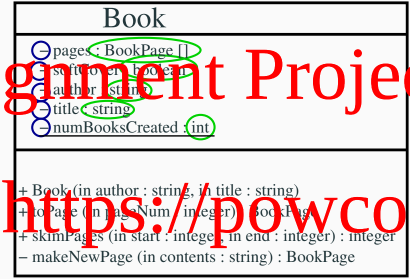
Attributes should mainly be of primitive (or built-in) types.

We use associations to show relationships between classes.

Going from UML to Java

- Most of the work is done in the design phase.
- Matter of translating syntax to desired language.
- UML is independent of programming language.
- In our case, we translate to Java.
- This is far easier than the design. Get the design right first!

Attributes to Instance Variables



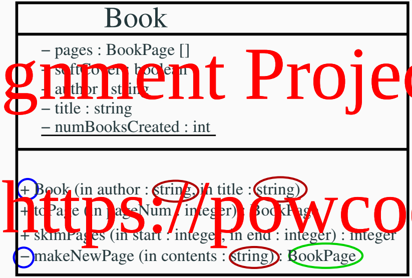
Book

– pages : BookPage []
– softCover : boolean
– author : string
– title : string
– numBooksCreated : int

+ Book (in author : string, in title : string)
+ toPage (in pageNum : integer) : BookPage
+ skimPages (in start : integer, in end : integer) : integer
– makeNewPage (in contents : string) : BookPage

```
public class Book
{
  private BookPage[] pages;
  private boolean softCover;
  private String author;
  private String title;
  private static int numBooksCreated;
  ....
}
```

Behaviours to Methods



Book

– pages : BookPage []
– softCover : boolean
– author : string
– title : string
– numBooksCreated : int

+ Book (in author : string, in title : string)
+ toPage (in pageNum : integer) : BookPage
+ skimPages (in start : integer, in end : integer) : integer
– makeNewPage (in contents : string) : BookPage

```
public Book (String author, String title)
{
  <implementation for constructor>
}

private BookPage makeNewPage (String contents)
{
  <implementation>
}
```

UML Class Diagram Exercise

- A student has a name, student number, and a program of study.
- It should have methods to set and get each of these attributes.
- It has methods such as going to class, sleeping, doing assignments.
- When a student is constructed, it should automatically have the the next unique student number, but its programme of study should be passed as parameter.