Faculty of Science, Engineering and Technology

Object Oriented Programming

Topic 3: Object Collaboration

Lecture Demo - The Spell Book

Return to your School of Magic program. This week you will add a **Spell Book** to this program.

1. Use the following description to **draw** your own **UML class diagram** of the Spell Book and its relationship with your Spell class.

   - A Spell Book contains zero of more spells.
   - spells can be added and removed by the Wizard at any time.
   - The wizard can then access the spells by fetching the spell at a given index in the book.

Hint: The ability to access a Spell at a given index can be implemented as an **indexer** in C#. In the UML show this as a property named this[int i], i.e.:

```
this[int i] :: Spell <<readonly property>>
```

Tip: Draw this on a whiteboard, take a photo, we will love it. This is a great way to think through your designs quickly, and record the results. Dont spend ages with a drawing program, we want the content not a pretty picture!

2. Create a Spell Book Tests file and add unit tests for adding, removing, and fetching Spells from a Spell Book.
3. Implement the new Spell Book class, and add the methods, properties, and fields you indicated in your UML. The following code demonstrates how to implement an indexer in C#.

```
public class SpellBook
{
    private List<Spell> _spells;
    public Spell this[int i]
    {
         get
         {
              return _spells [i];
         }
    }
}
SpellBook myBook; …
Spell s = myBook[0];
```

4. Run your tests and make sure they pass.
5. Add XML documentation to your Spell Book class.

Note: An **indexer** is a special kind of property that allows the caller to access your object using an index like an array. So by adding an indexer to your Spell Book callers can ask for **myBook[0]**. This will call the getter, passing in 0 as the value i.

You can also have setters, so **myBook[0] = new Spell(…);** can also work but does require a **set** inside the indexer.