From Design Model to Code

ISEP / LETI / ESOFT

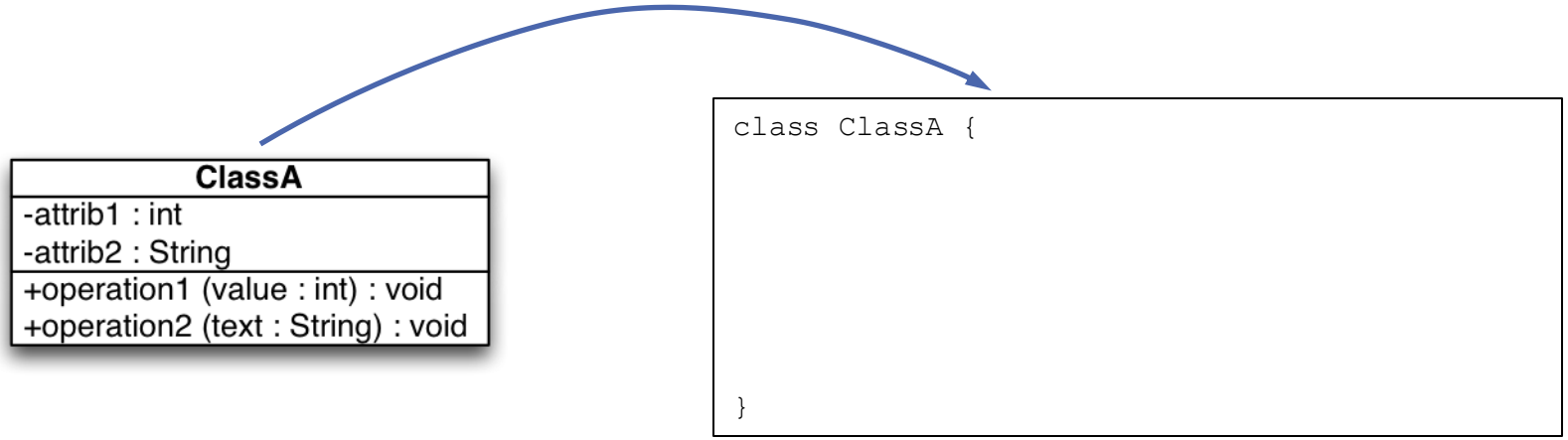
Topics

- Class Mapping
- Association Mapping
- Class Diagram Mapping Exercise
- Sequence Diagram Mapping
- Sequence Diagram Mapping Exercise

Class Mapping

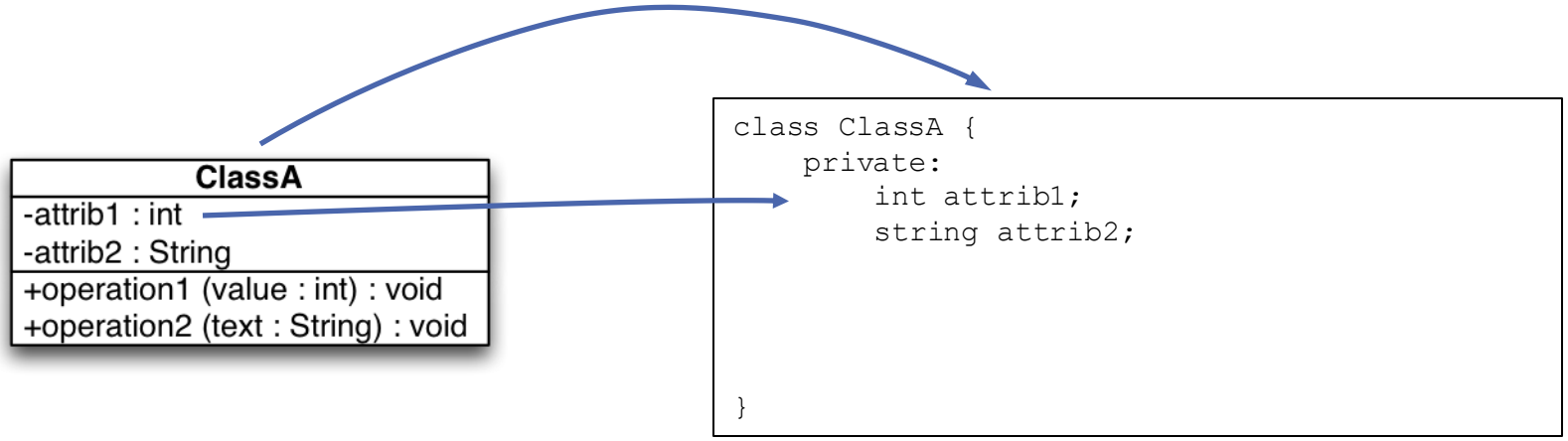
Class Mapping (1/4)

- A UML class represents a **class** in the code



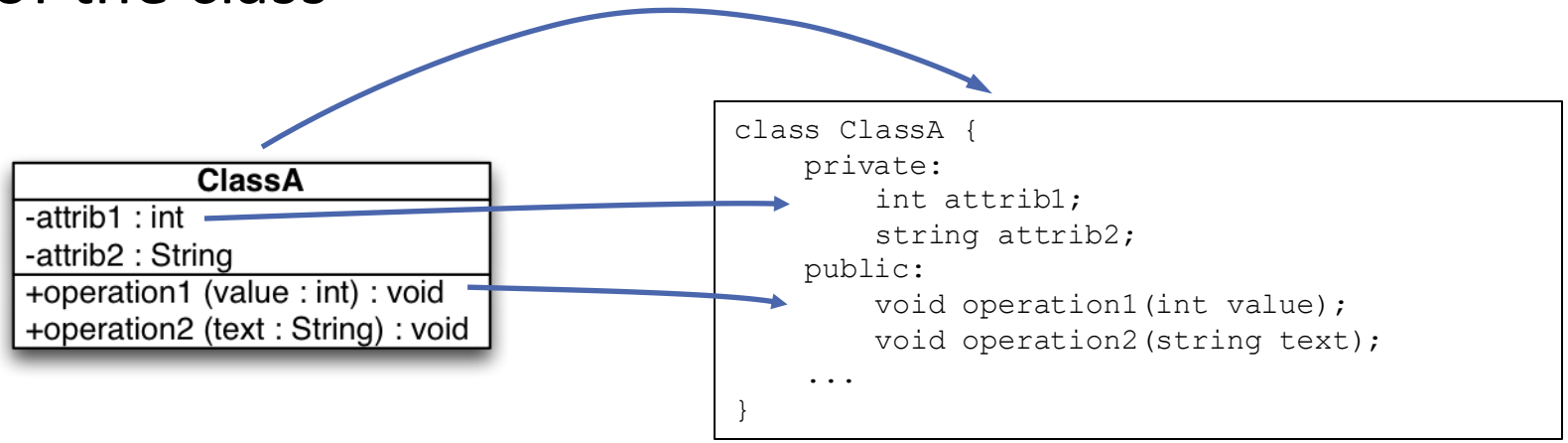
Class Mapping (2/4)

- A UML class represents a **class** in the code
- An attribute represents a **member variable**



Class Mapping (3/4)

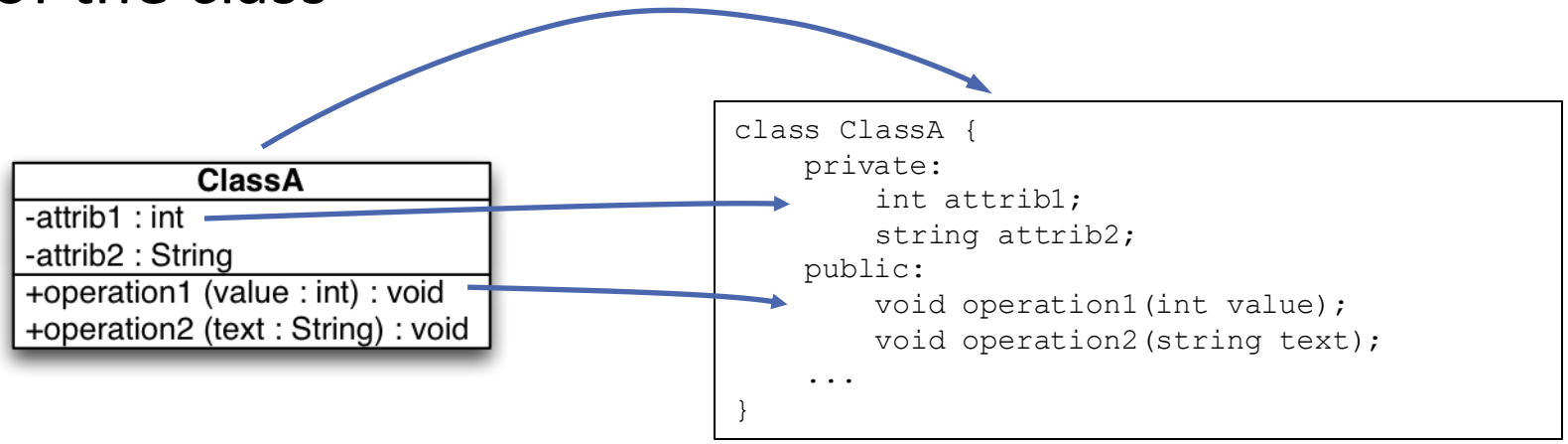
- A UML class represents a **class** in the code
- An attribute represents a **member variable**
- An operation represents a **function** of the class



Class Mapping (4/4)

- A UML class represents a **class** in the code
- An attribute represents a **member variable**
- An operation represents a **function** of the class

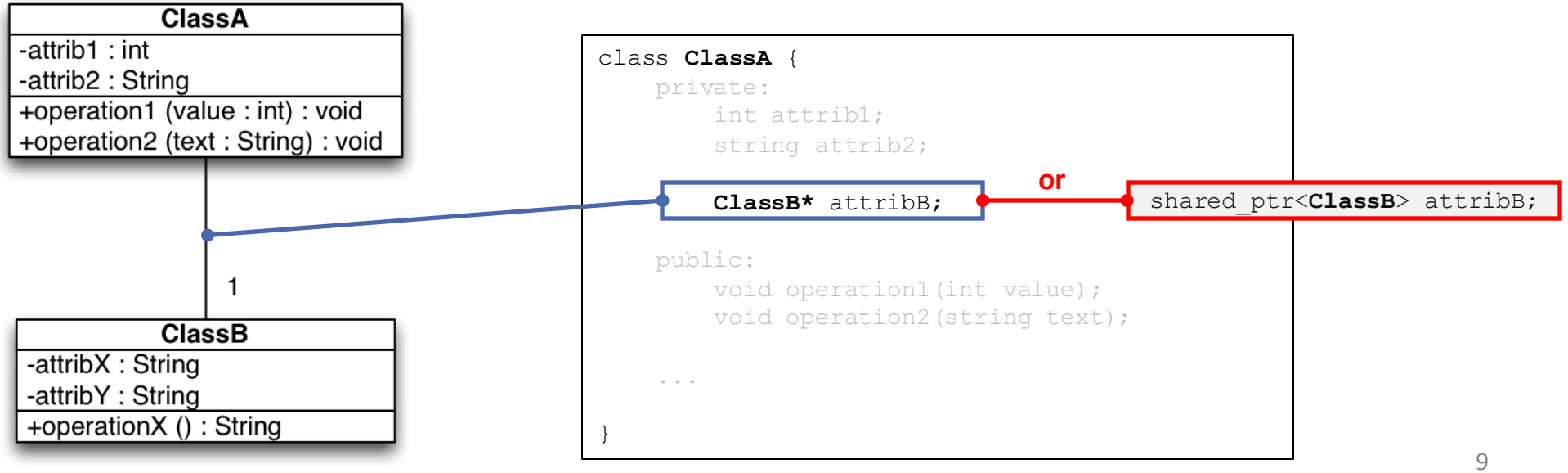
- Regarding visibility, if not depicted in the CD, then:
  - Member variables are private
  - Functions are public



Association Mapping

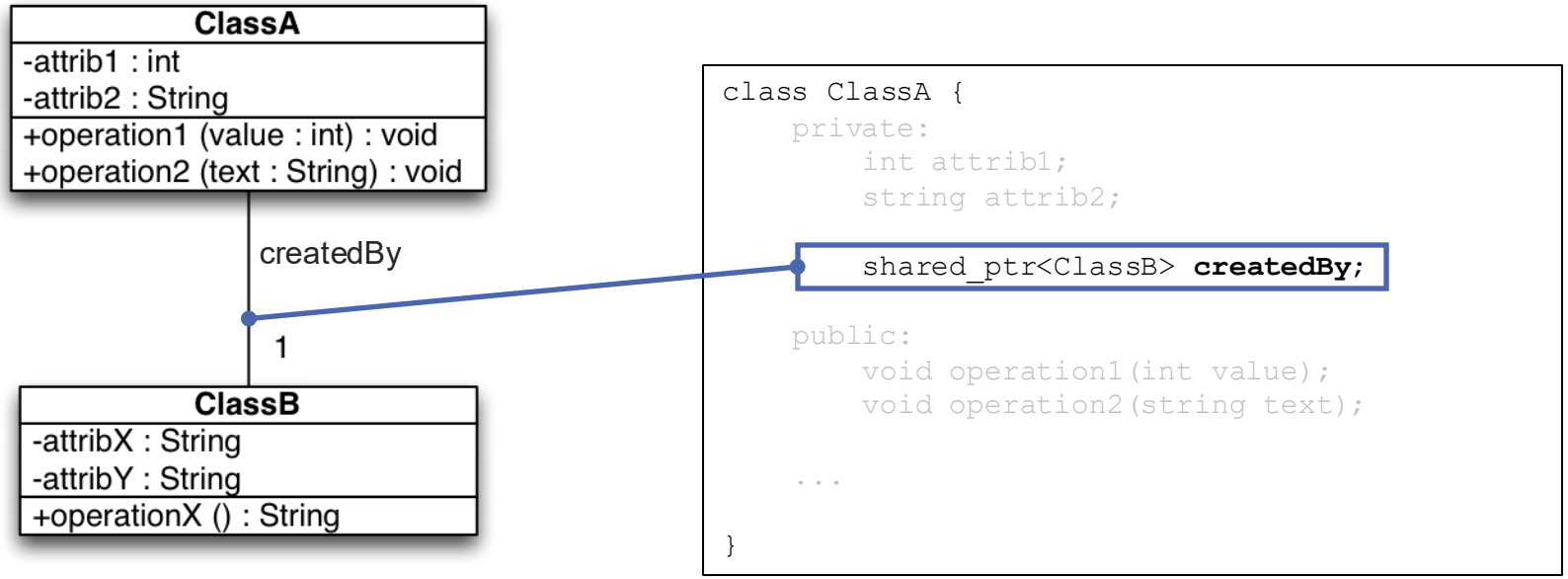
Association Mapping – Unnamed N to 1 (*:1)

- An association is represented as a **pointer member** or a **smart pointer member**
  - I.e., it points to an object, not a primitive data type
- (smart) pointer members are inferred from associations
  - They do not need to be explicit in the CD (they can be unnamed)



Association Mapping – Named N to 1 (*:1)

- Using the association name to name the (smart) pointer member

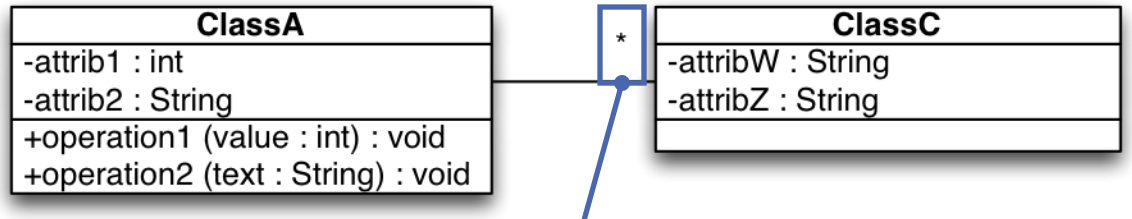


Association Mapping – 1 to N (1:*)

- Mapping an association whose multiplicity is one-to-many requires the use of Container objects

- In C++, several standard Container objects are available
  - E.g.: list, vector, set, map
  - Check [1] for a full list and comparison

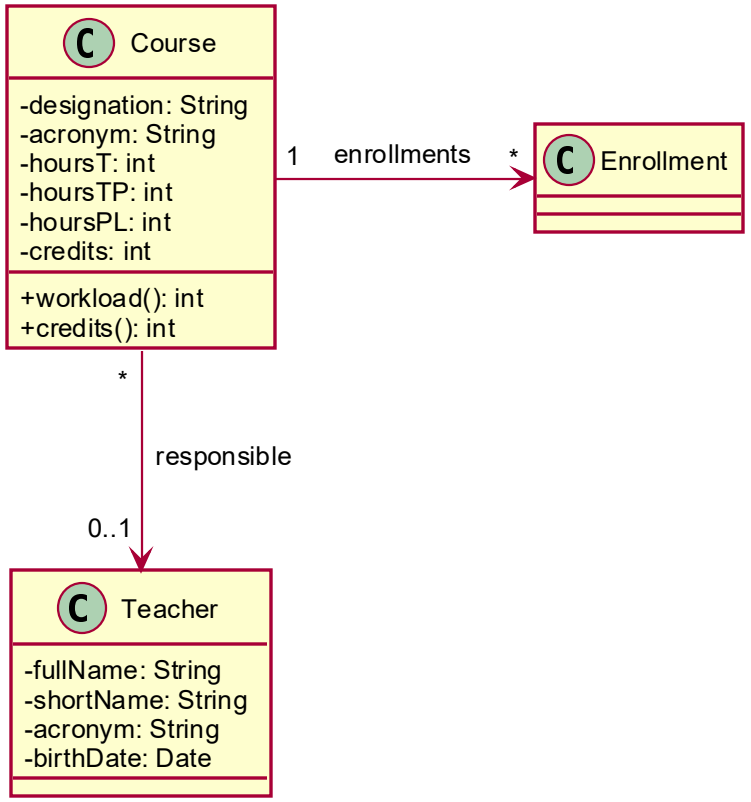
[1] https://www.cplusplus.com/reference/stl/




```
class ClassA {
    private:
        int attrib1;
        string attrib2;

        list<shared_ptr<ClassC>> listOfCs;

    public:
        void operation1(int value);
        void operation2(string text);

    ...

}
```

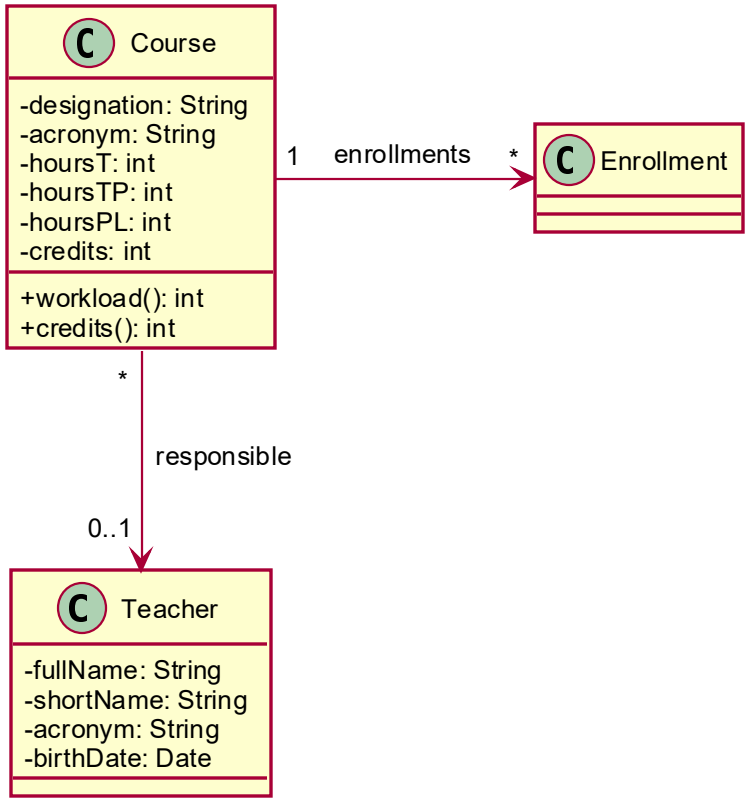
Class Diagram Mapping Exercise

Class Diagram Mapping Exercise (1/2)




```
class Course {




}
```

Class Diagram Mapping Exercise (2/2)

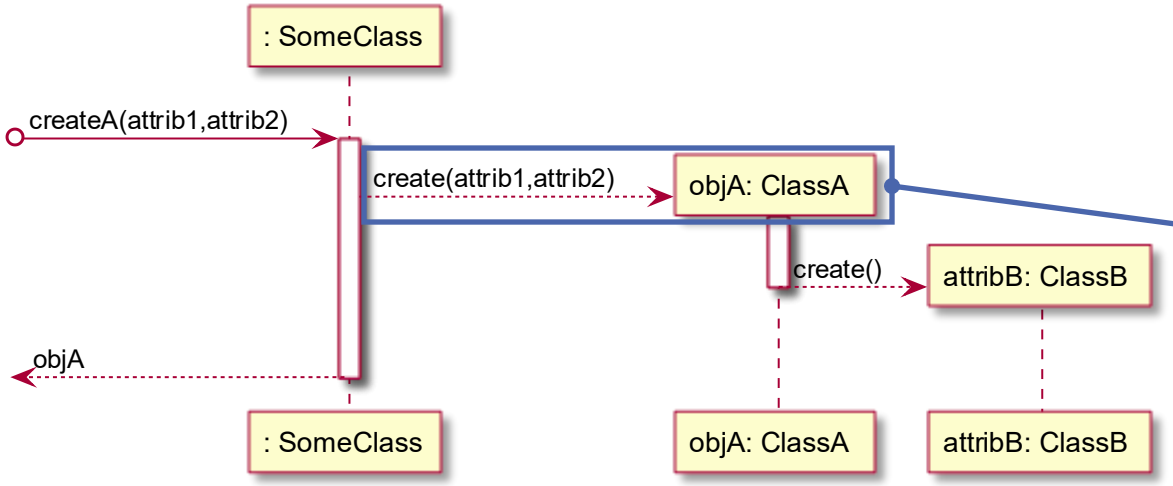



```
class Course {
   private:
      string designation;
      string acronym;
      int hoursT;
      int hoursTP;
      int hoursPL;
      int credits;
      shared_ptr<Teacher> responsible;
      list<shared_ptr<Enrollment>> enrollments;

   public:
      Course(...);
      int workload();
      int credits();

   ...
}
```

Sequence Diagram Mapping

Code mapping from a Sequence Diagram (1/2)

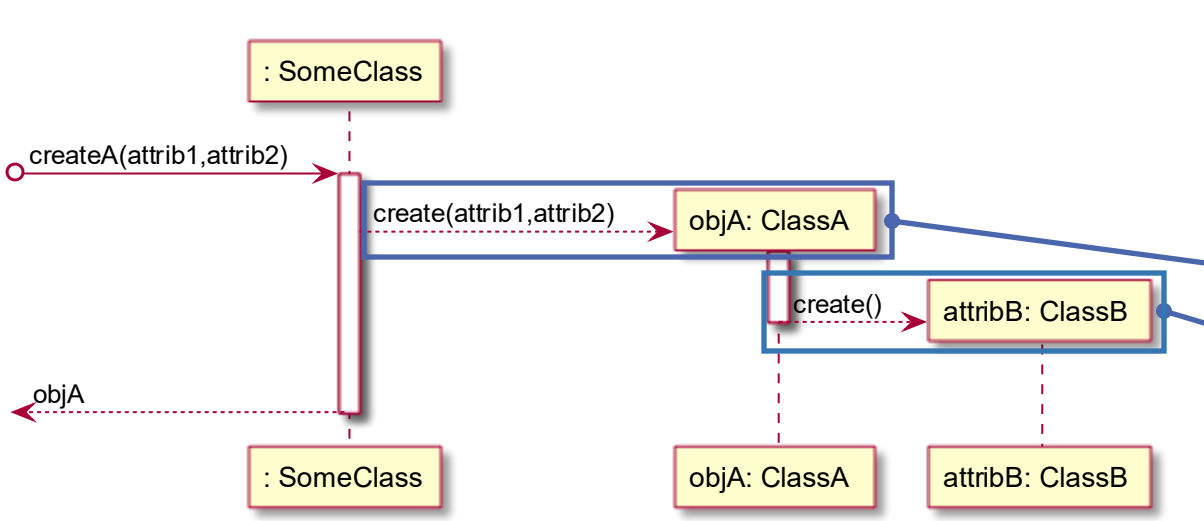
- Creating an object invokes its constructor

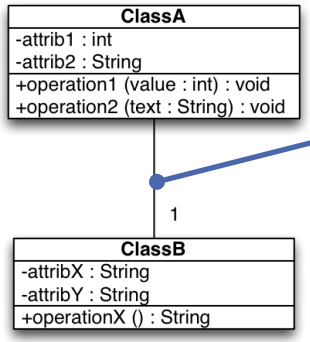



```
class ClassA {
  private:
    int attrib1;
    tring attrib2;
    shared_ptr<ClassB> attribB;

  public:

    ClassA(int attrib1, string attrib2) {
      this.attrib1 = attrib1;
      this.attrib2 = attrib2;

      attribB = shared_ptr<ClassB>(new ClassB());

  }

  ...
}
```

Code mapping from a Sequence Diagram (2/2)

- Creating an object invokes its constructor
- In C++, the UML **create** message maps to object instantiation using smart pointers, either with **shared_ptr** combined with **new**, or preferably with **make_shared**.





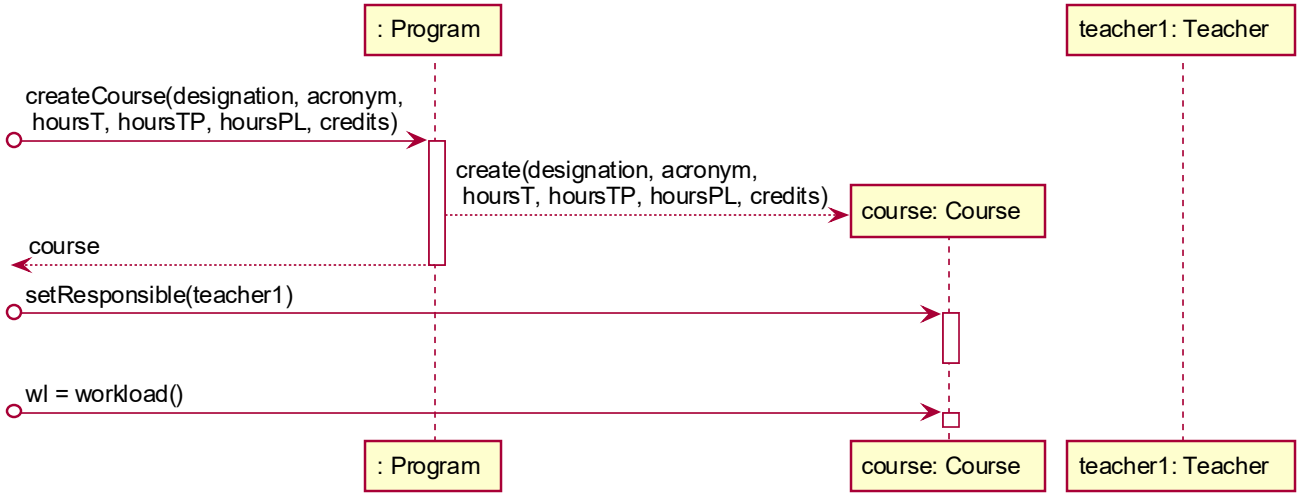

```
class ClassA {
  private:
    int attrib1;
    tring attrib2;
    shared_ptr<ClassB> attribB;

  public:

    ClassA(int attrib1, string attrib2) {
      this.attrib1 = attrib1;
      this.attrib2 = attrib2;

    attribB = shared_ptr<ClassB>(new ClassB());

  }

  ...
}
```

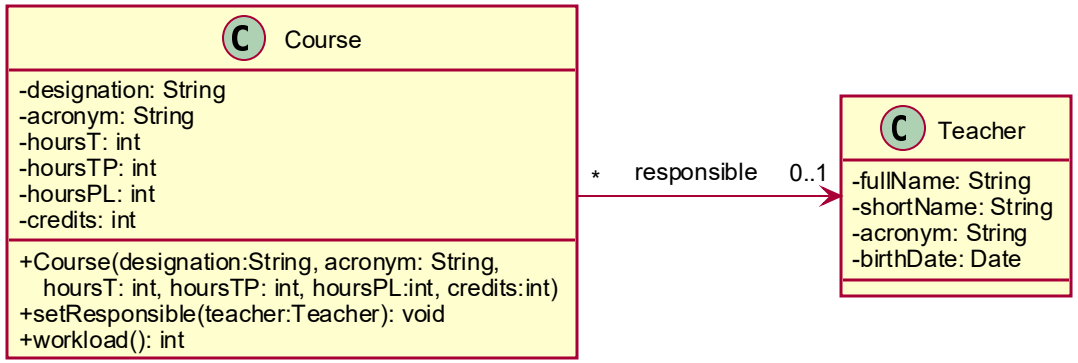
or

```
attribB = make_shared<ClassB>();
```

Sequence Diagram Mapping Exercise

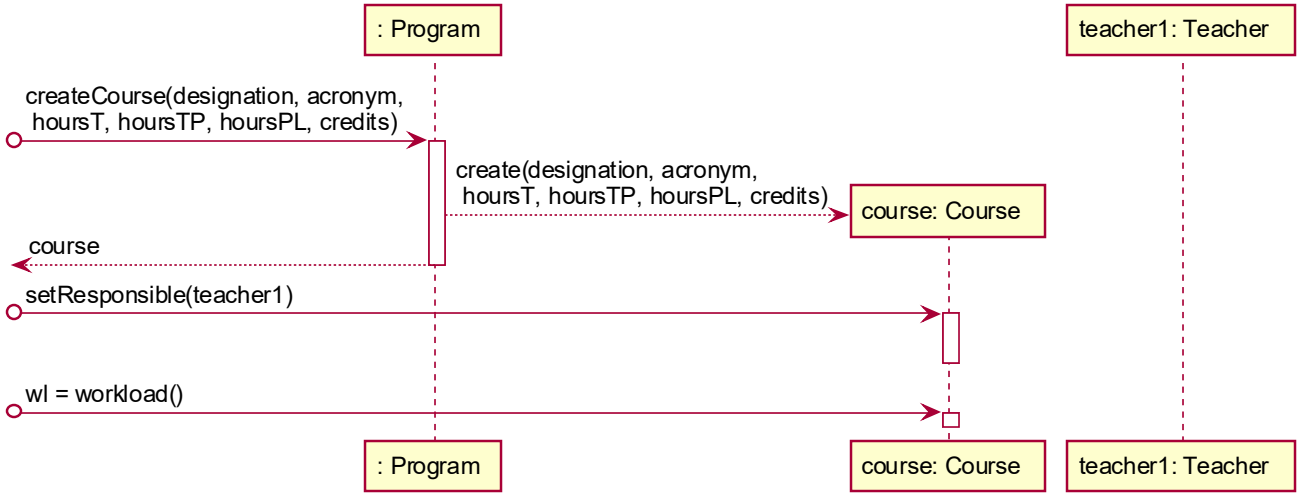
Sequence Diagram Mapping Exercise (1/2)

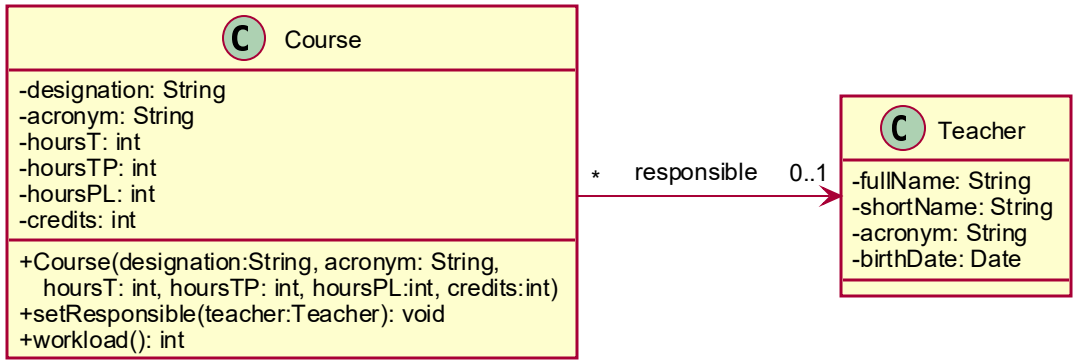





```
class Course {

}
```

Sequence Diagram Mapping Exercise (2/2)






```
class Course {
   private:
      string designation;
      string acronym;
      int hoursT;
      int hoursTP;
      int hoursPL;
      int credits;
      shared_ptr<Teacher> responsible;

   public:
      Course(string designation, string acronym,
             int hoursT, int hoursTP, int hoursPL,
             int credits) {
         this->designation = designation;
         this->acronym = acronym;
         ...
      }

      void setResponsible(shared_ptr<Teacher> teacher) {
         this->responsible = teacher;
      }

      int workload() {
         return this->hoursT + this->hoursTP
                + this->hoursPL;
      }

   ...
}
```

Summary

- Notice how the development of the intended software product is guided by:
  - Executing the SDP (main) activities
  - The realization of each user scenario (i.e. functional requirement)
- Outputs of one activity are used as inputs by the next activity
- Each activity is a step forward to successfully meet the functional requirements

- **Code must be coherent with all design artifacts**

Bibliography

- Larman, C. (2004). Applying UML and Patterns (3rd ed.). Prentice Hall. ISBN: 978-0-131-48906-6
- Malik, D.S. (2018) C++ Programming: From Problem Analysis to Program Design
- Pikus, F.G. (2019). Hands-On Design Patterns with C++. Packt