From Design Model to Code

UPskill

Digital Skills & Jobs

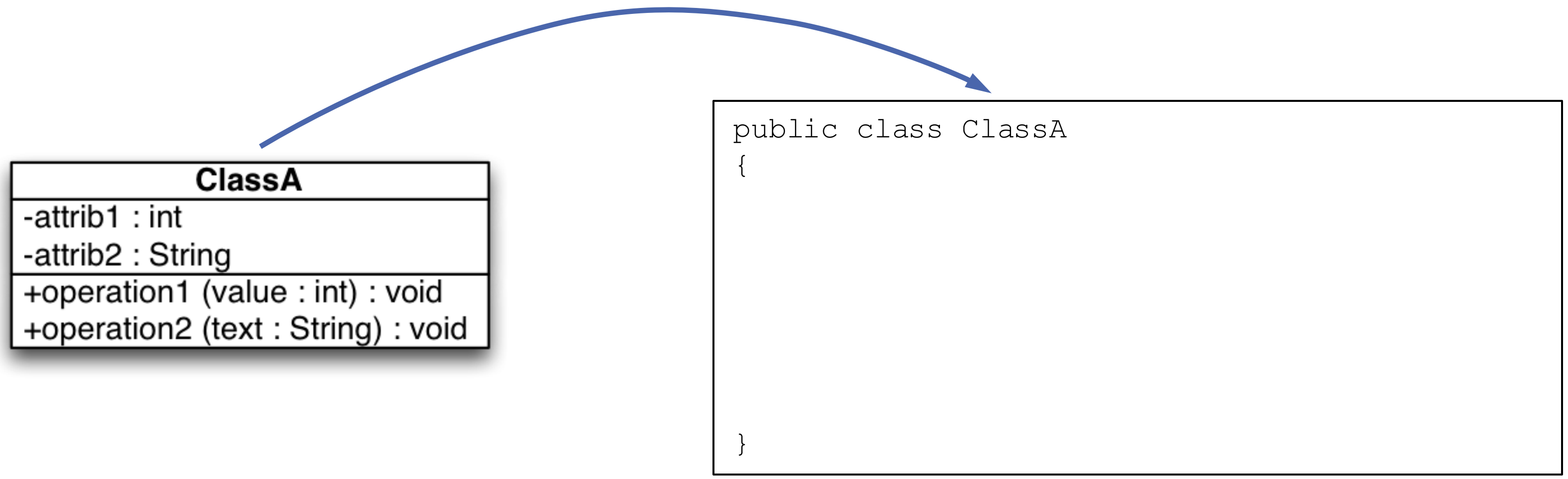
Topics

- 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



ClassA

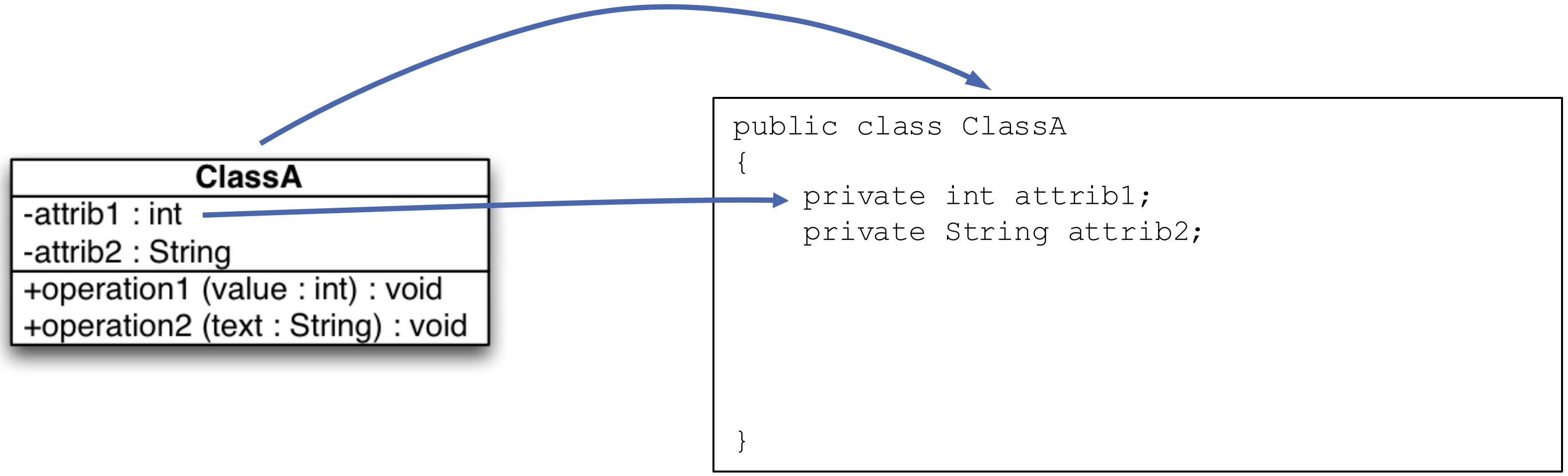
-attrib1 : int
-attrib2 : String

+operation1 (value : int) : void
+operation2 (text : String) : void

```
public class ClassA
{






}
```

Class Mapping (2/4)

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

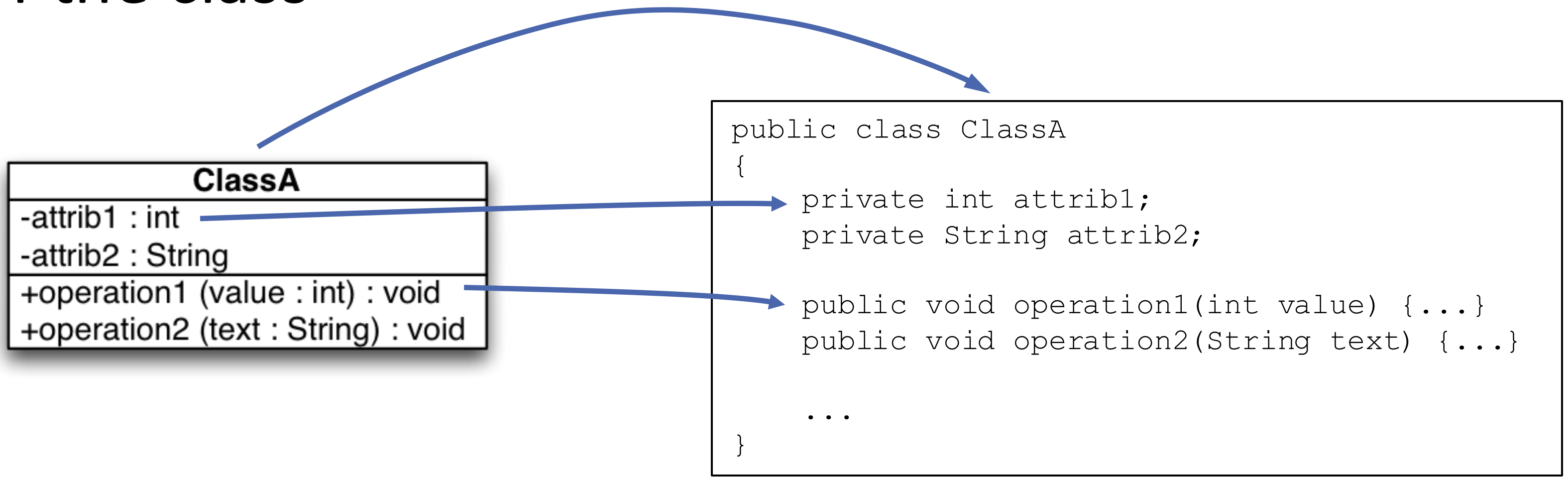


ClassA
-attrib1 : int
-attrib2 : String
+operation1 (value : int) : void
+operation2 (text : String) : void

```
public class ClassA
{
    private int attrib1;
    private String attrib2;




}
```

Class Mapping (3/4)

- A UML class represents a **class** in the code
- An attribute represents an **instance variable**
- An operation represents a **method** of the class

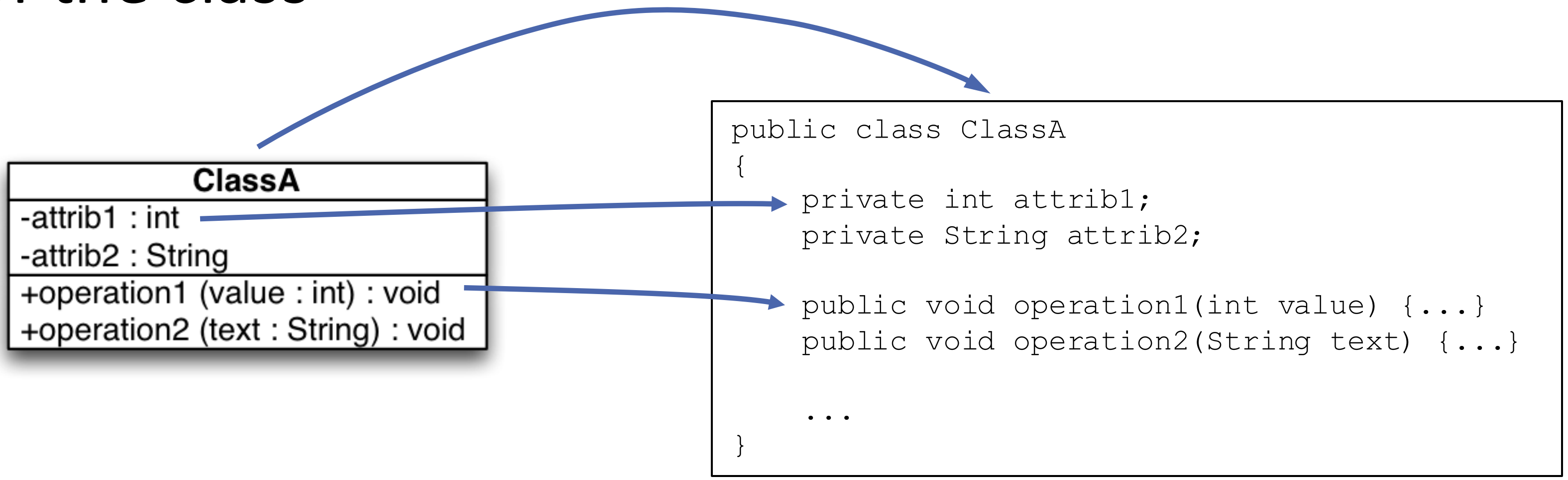

ClassA
-attrib1 : int
-attrib2 : String
+operation1 (value : int) : void
+operation2 (text : String) : void

```
public class ClassA
{
    private int attrib1;
    private String attrib2;

    public void operation1(int value) {...}
    public void operation2(String text) {...}

    ...
}
```

Class Mapping (4/4)

- A UML class represents a **class** in the code
- An attribute represents an **instance variable**
- An operation represents a **method** of the class

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



ClassA

-attrib1 : int

-attrib2 : String

+operation1 (value : int) : void

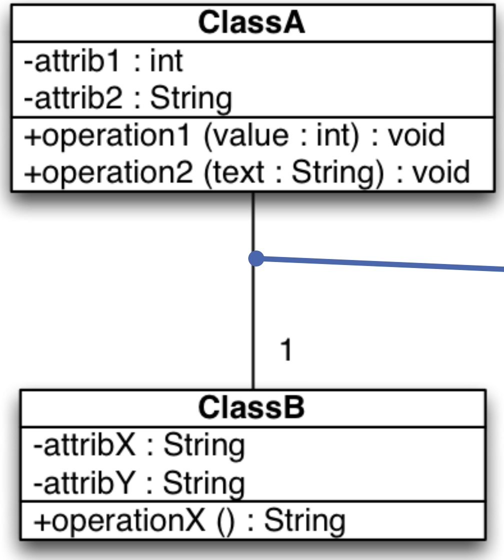
+operation2 (text : String) : void

```
public class ClassA
{
    private int attrib1;
    private String attrib2;

    public void operation1(int value) {...}
    public void operation2(String text) {...}

    ...
}
```

Association Mapping

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

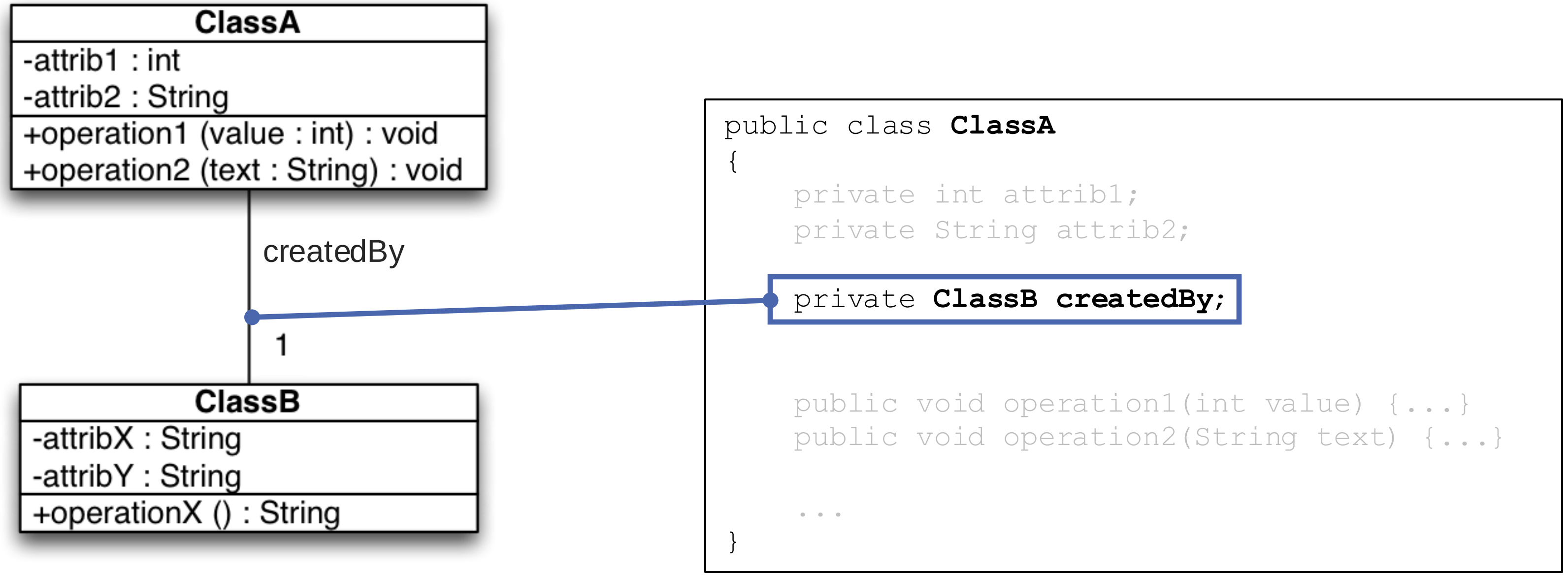
- An association is represented as a **reference attribute**
  - I.e., it references an object and not a primitive data type
- Reference attributes are inferred from associations
  - They do not need to be explicit in the CD (they can be unnamed)



```
public class ClassA
{
    private int attrib1;
    private String attrib2;

    private ClassB attribB;


    public void operation1(int value) {...}
    public void operation2(String text) {...}

    ...
}
```

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

- Using the association name to name the reference attribute

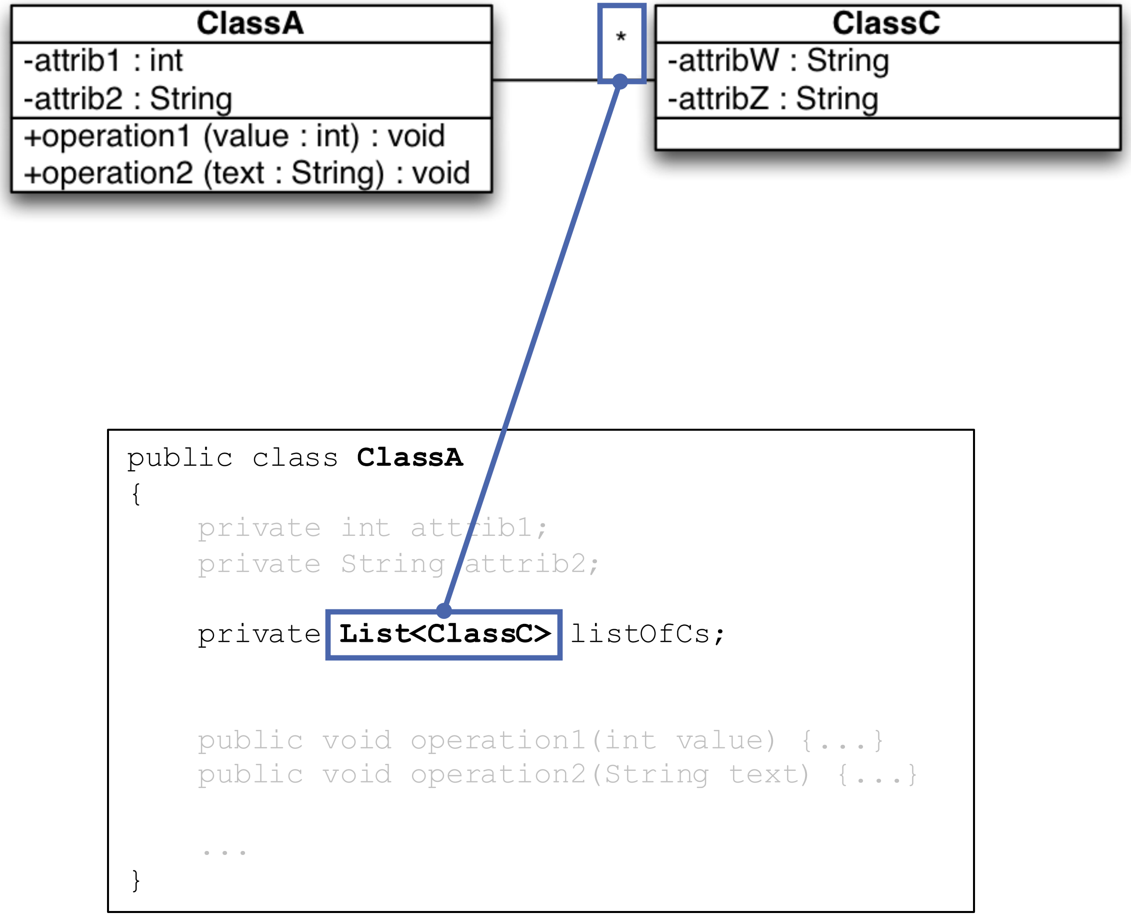


```
public class ClassA
{
    private int attrib1;
    private String attrib2;

    private ClassB createdBy;


    public void operation1(int value) {...}
    public void operation2(String text) {...}

    ...
}
```

Association Mapping – 1 to N (1:*)

- Mapping an association whose multiplicity is one-to-many requires the use of **Collection** objects (e.g. List, Set, Map)

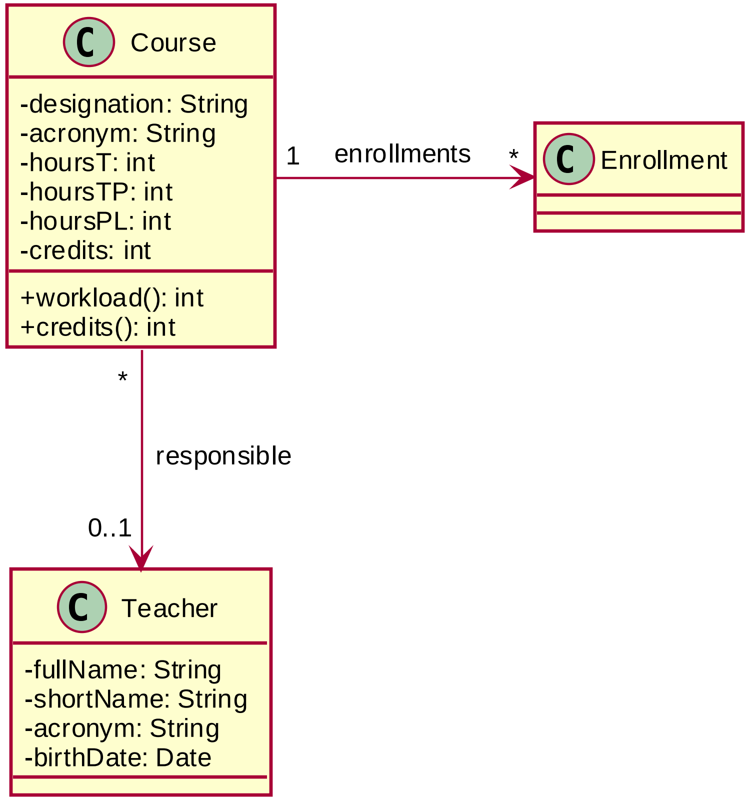
- In Java, the **ArrayList**, HashSet and HashMap classes implement the **List**, Set and Map interfaces, respectively



```
public class ClassA
{
    private int attrib1;
    private String attrib2;

    private List<ClassC> listOfCs;


    public void operation1(int value) {...}
    public void operation2(String text) {...}

    ...
}
```

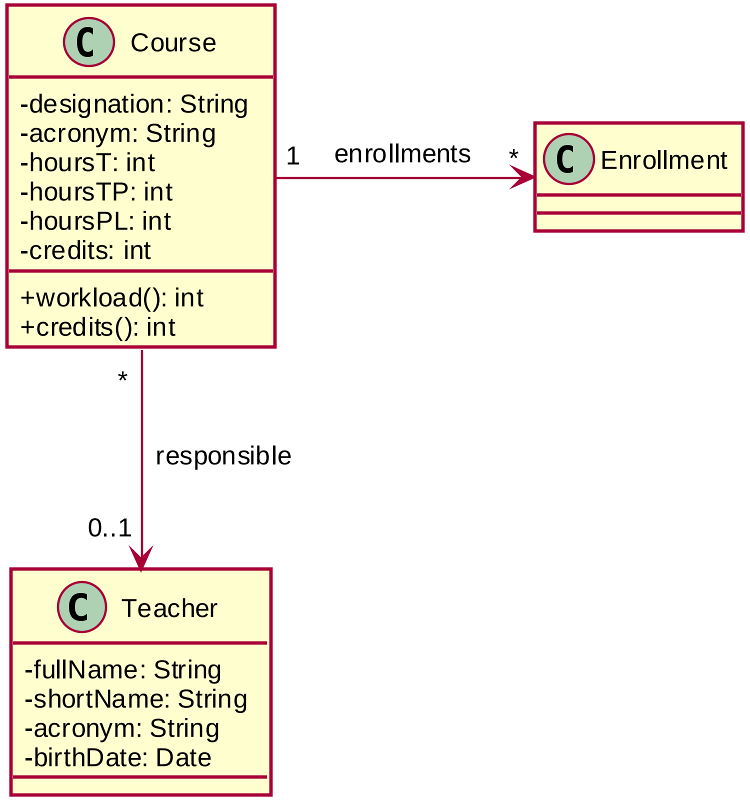
Class Diagram Mapping Exercise

Class Diagram Mapping Exercise (1/2)



```
public class Course
{





















}
```

Class Diagram Mapping Exercise (2/2)

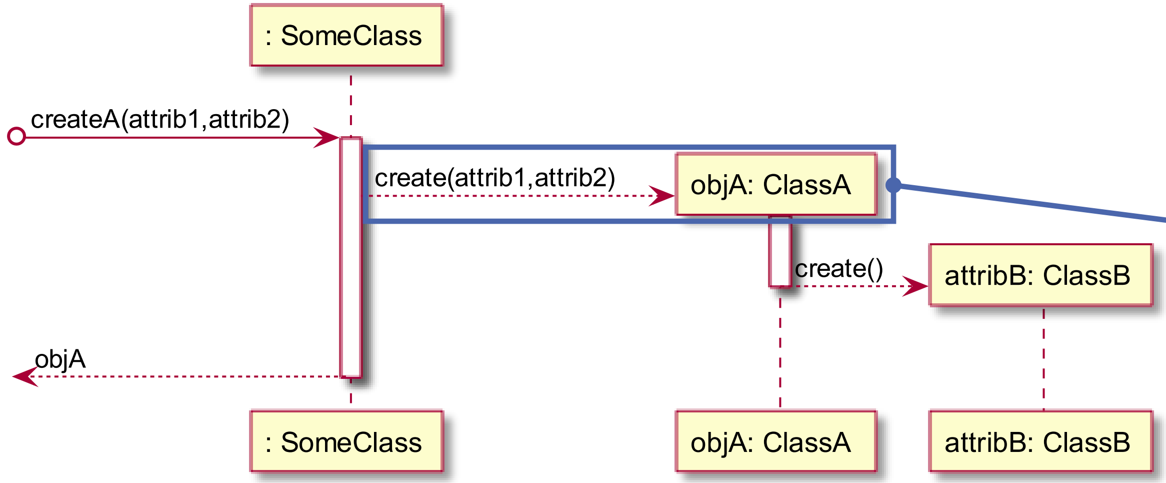


```
public class Course
{
   private String designation;
   private String acronym;
   private int hoursT;
   private int hoursTP;
   private int hoursPL;
   private int credits;
   private Teacher responsible;
   private List<Enrollment> enrollments;

   ...

   public int workload() {...}
   public int credits() {...}

   ...
}
```

Sequence Diagram Mapping

Code mapping from a Sequence Diagram (1/2)

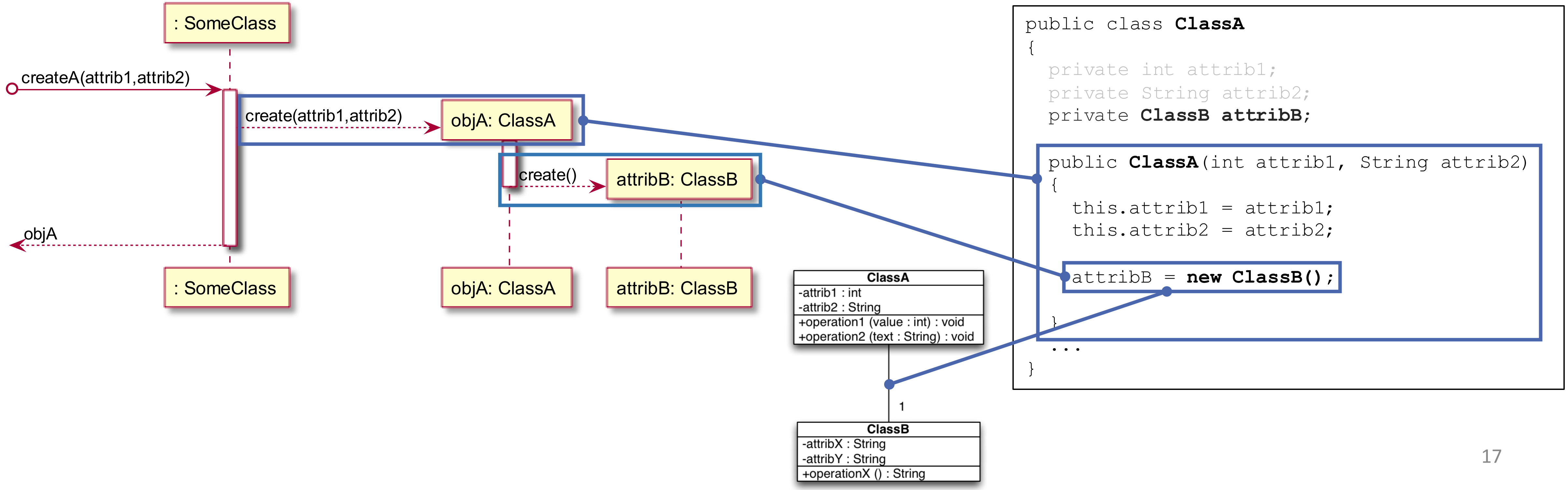
- Creating a class instance invokes its constructor



```
public class ClassA
{
  private int attrib1;
  private String attrib2;
  private ClassB attribB;

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

    attribB = new ClassB();

  }
  ...
}
```

Code mapping from a Sequence Diagram (2/2)

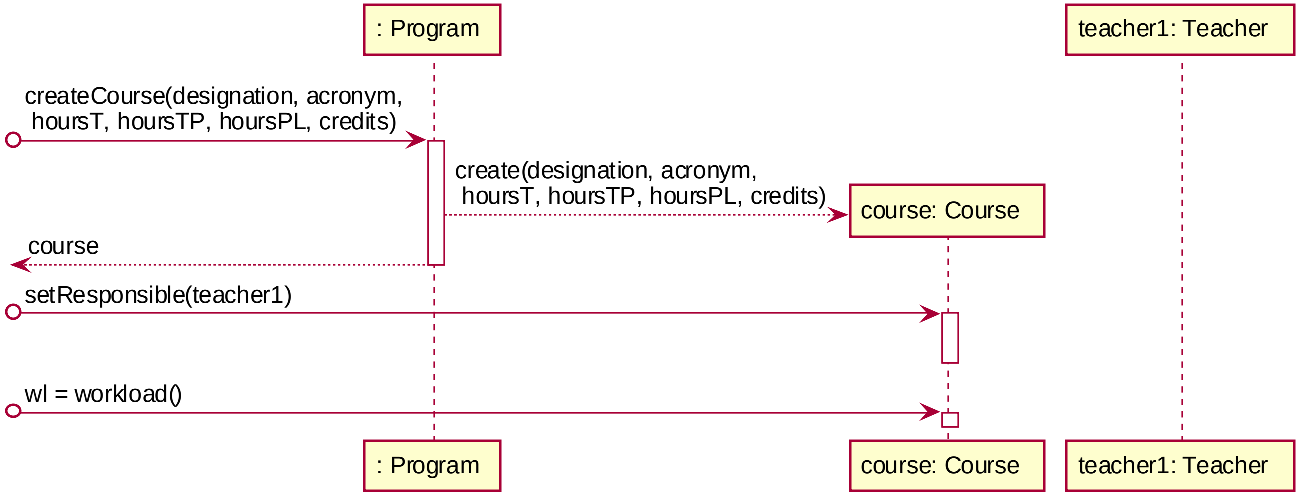
- Creating a class instance invokes its constructor
- The UML **create** message is mapped to the **new** operator (in Java) followed by a constructor of the class to instantiate



```
public class ClassA
{
  private int attrib1;
  private String attrib2;
  private ClassB attribB;

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

    attribB = new ClassB();

  }
  ...
}
```

Sequence Diagram Mapping Exercise

Sequence Diagram Mapping Exercise (1/2)



: Program

teacher1: Teacher

createCourse(designation, acronym, hoursT, hoursTP, hoursPL, credits)

create(designation, acronym, hoursT, hoursTP, hoursPL, credits)

course: Course

course

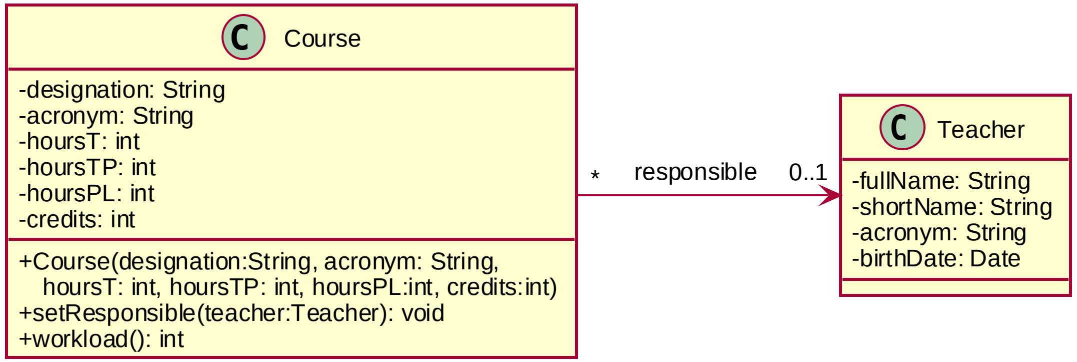
setResponsible(teacher1)

wl = workload()

: Program

course: Course

teacher1: Teacher



Course

-designation: String
-acronym: String
-hoursT: int
-hoursTP: int
-hoursPL: int
-credits: int

+Course(designation:String, acronym: String, hoursT: int, hoursTP: int, hoursPL:int, credits:int)
+setResponsible(teacher:Teacher): void
+workload(): int

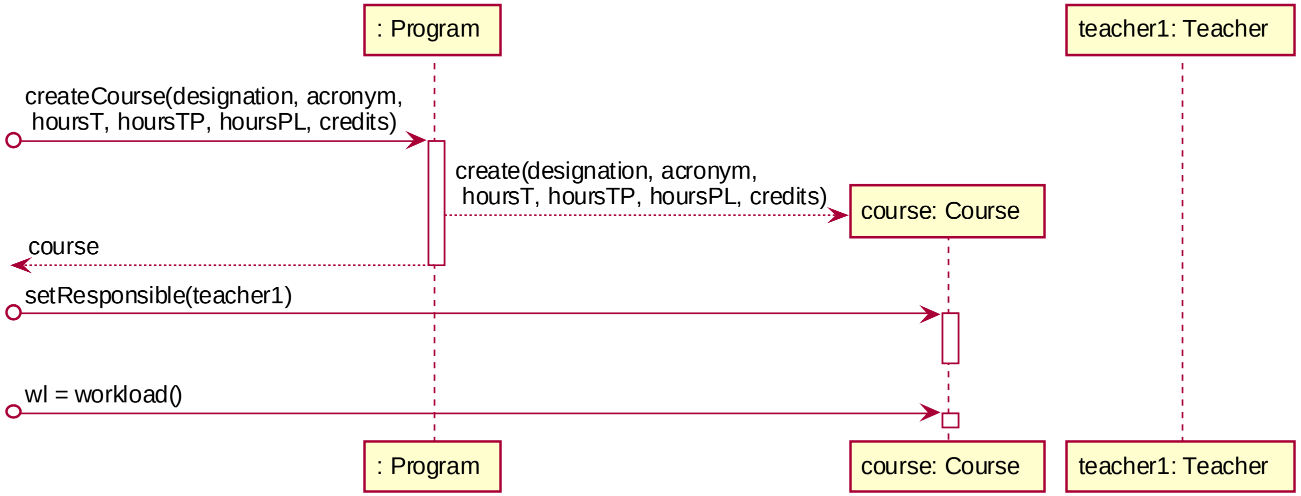
* responsible 0..1

Teacher

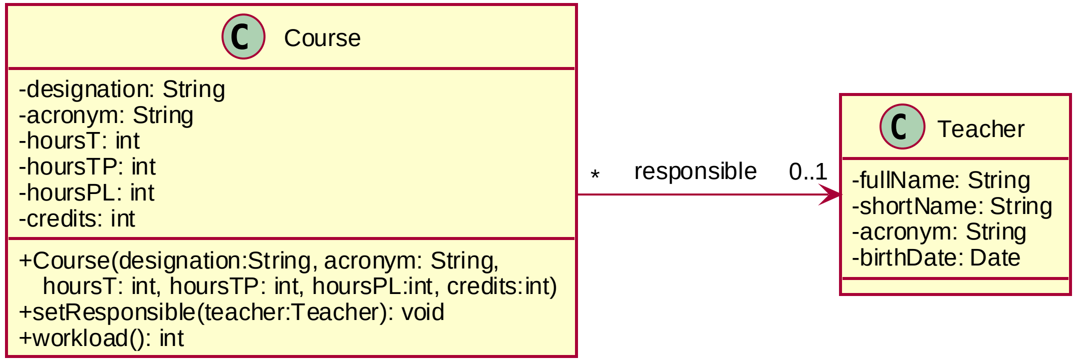
-fullName: String
-shortName: String
-acronym: String
-birthDate: Date

```
public class Course
{











}
```

Sequence Diagram Mapping Exercise (2/2)



: Program
teacher1: Teacher
createCourse(designation, acronym,
hoursT, hoursTP, hoursPL, credits)
create(designation, acronym,
hoursT, hoursTP, hoursPL, credits)
course: Course
course
setResponsible(teacher1)
wl = workload()
: Program
course: Course
teacher1: Teacher



```
public class Course
{
   private String designation;
   private String acronym;
   private int hoursT;
   private int hoursTP;
   private int hoursPL;
   private int credits;
   private Teacher responsible;

   public Course(String designation, String acronym,
                 int hoursT, int hoursTP, int hoursPL,
                 int credits) {
      this.designation = designation;
      this.acronym = acronym;
      ...
   }

   public void setResponsible(Teacher teacher) {
      this.responsible = teacher;
   }

   public int workload() {
      return hoursT + hoursTP + 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**

References & Bibliography

- Larman, Craig; Applying UML and Patterns; Prentice Hall (3rd ed.); ISBN 978-0131489066
- Fowler, Martin. UML distilled: a brief guide to the standard object modeling language. Addison-Wesley Professional, 2004.
- Rational Unified Process: Best Practices for Software Development Teams; Rational Software White Paper; TP026B, Ver 11/01.