Object Oriented Design

Other Diagrams

Dr. Seán Russell

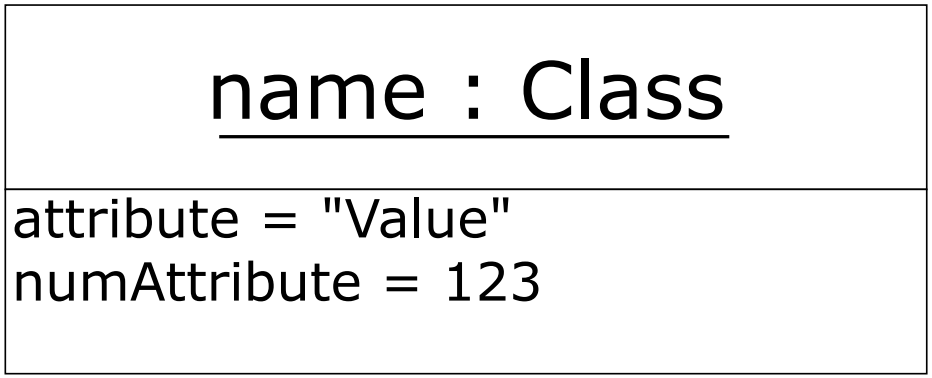
School of Computer Science,
University College Dublin

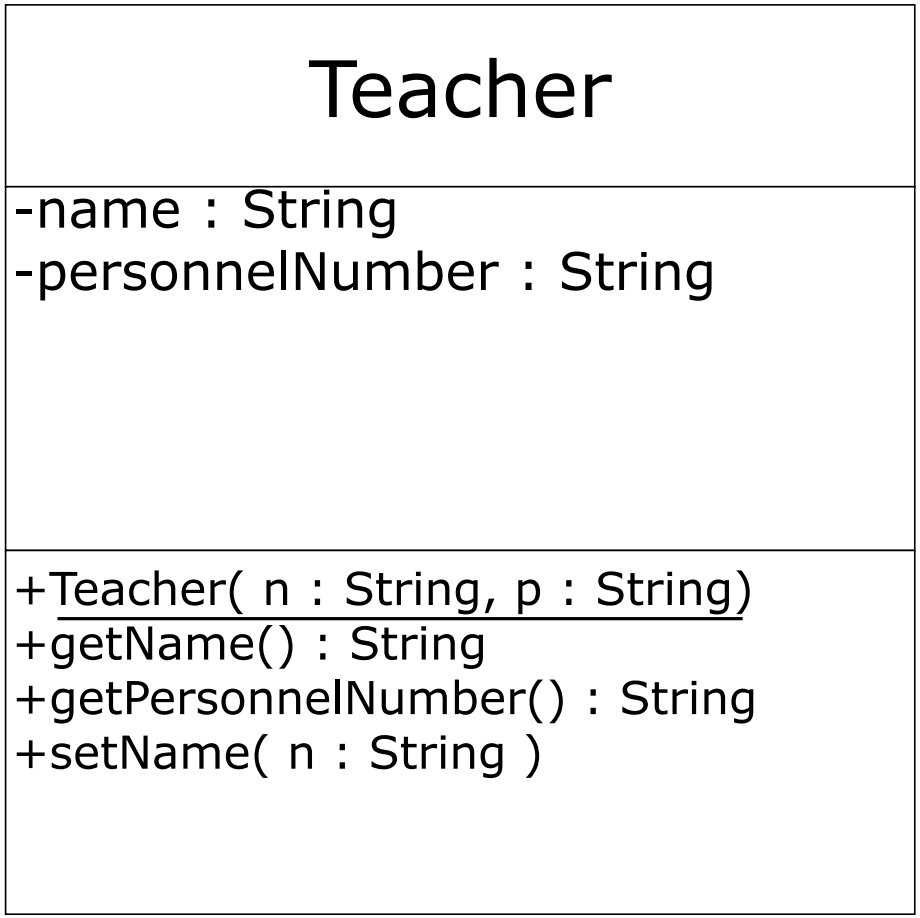
Lecture 04

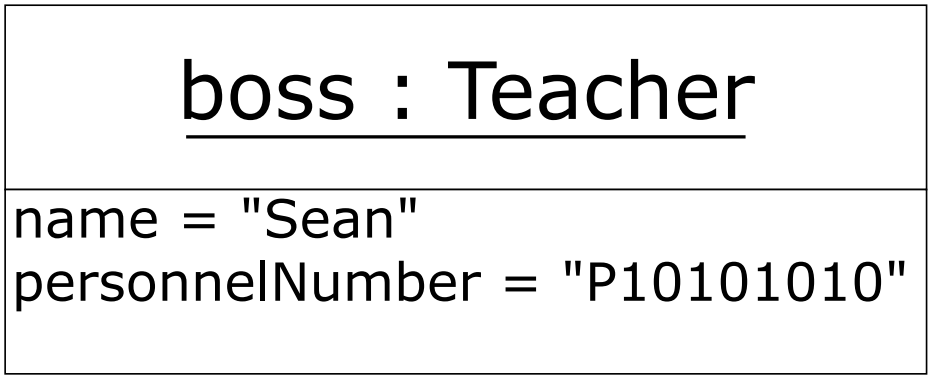
Table of Contents

Section Contents

- Object diagrams in UML are made up of objects, their data and the connections between them
- For every object we can represent three things:
  1. The class the object belongs to
  2. The name of the object
  3. The values of attributes it contains
- The attributes allowed are defined in the class


name : Class
attribute = "Value"
numAttribute = 123


Teacher
-name : String
-personnelNumber : String
+Teacher(n : String, p : String)
+getName() : String
+getPersonnelNumber() : String
+setName(n : String)

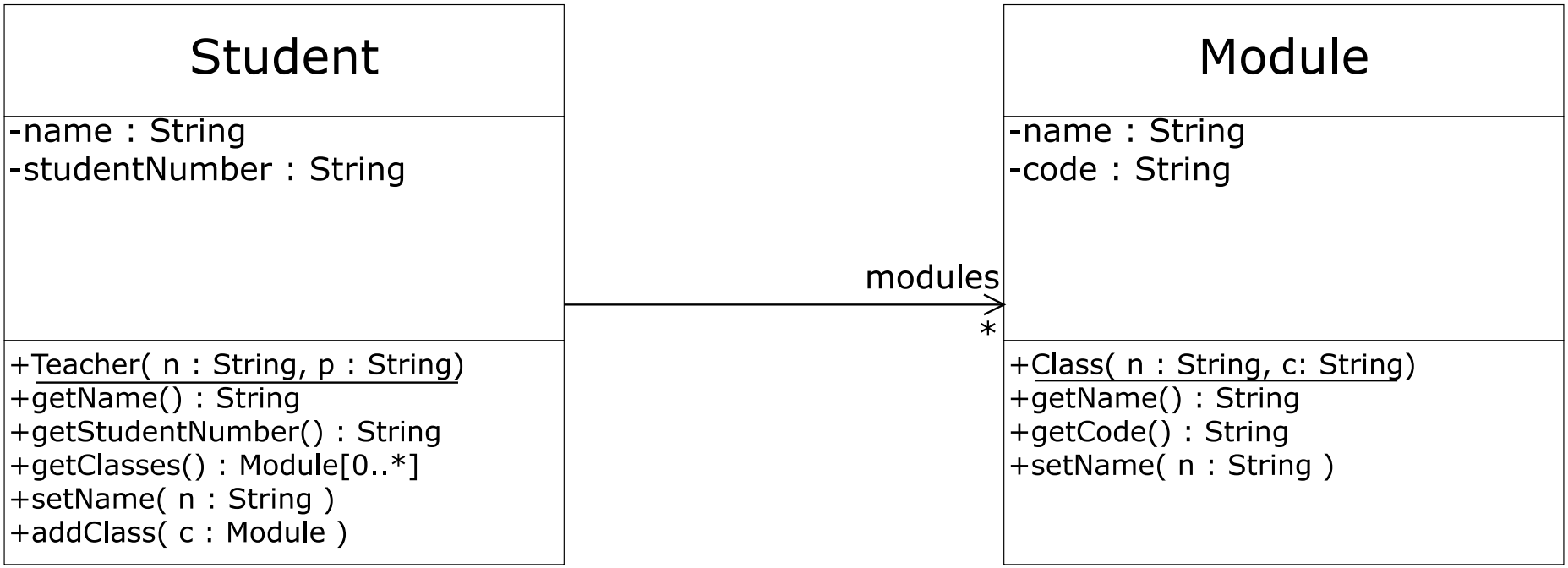

boss : Teacher
name = "Sean"
personnelNumber = "P10101010"

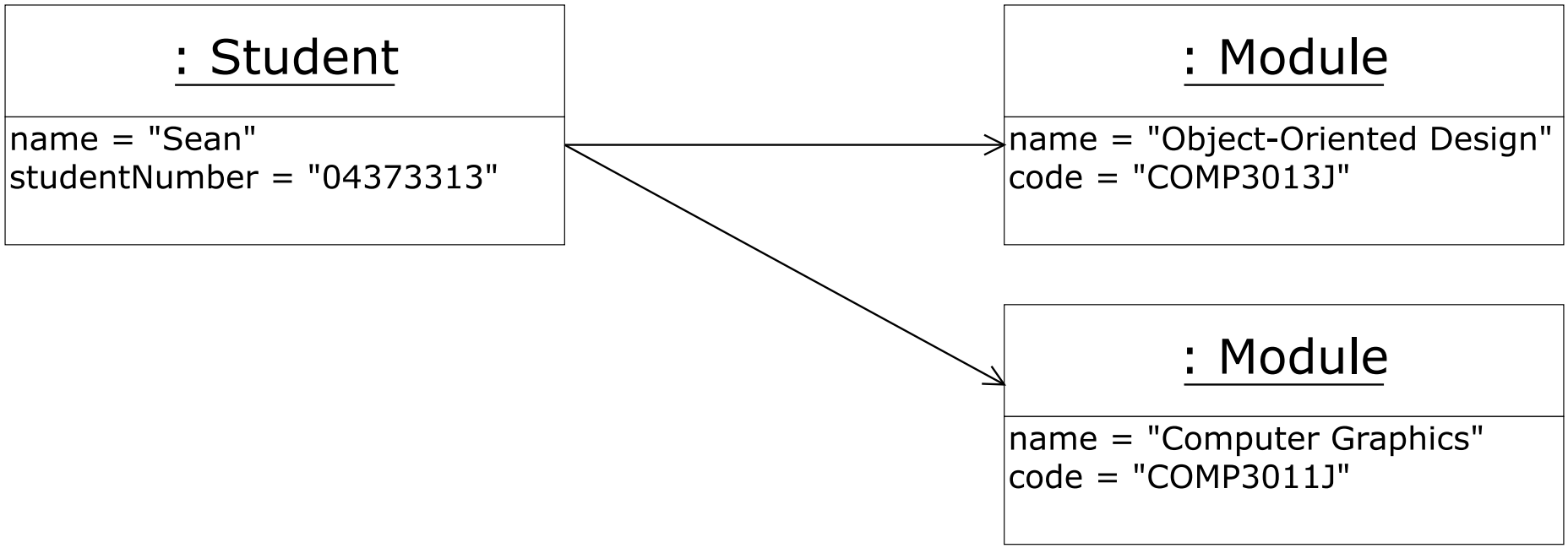
- Object name and class will always be underlined
- A colon will always be included even if a name is not
- Values of variables are included when the information is relevant or required
- Type information is not required, but may be included
- Details of operations is never included

- Object names are **optional** and often not used
- When names are used it is usually to explain how an object will be used
- Sometimes, variable names can be used but this is not common because variables can change

Section Contents

- A link is an instance of an association
- An association shows which classes can be connected and the rules applied to that connection
- A link shows which specific objects are connected
- A link may have the following annotations
  - Role names
  - Link name
  - Navigation arrows


Student
-name : String
-studentNumber : String
+Teacher(n : String, p : String)
+getName() : String
+getStudentNumber() : String
+getClasses() : Module[0..*]
+setName(n : String)
+addClass(c : Module)
modules
*
Module
-name : String
-code : String
+Class(n : String, c: String)
+getName() : String
+getCode() : String
+setName(n : String)

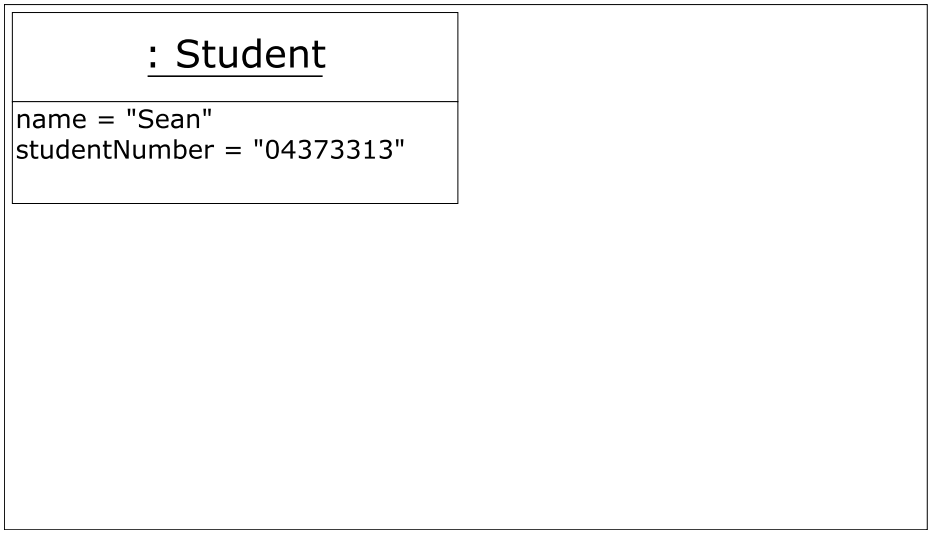

: Student
name = "Sean"
studentNumber = "04373313"
: Module
name = "Object-Oriented Design"
code = "COMP3013J"
: Module
name = "Computer Graphics"
code = "COMP3011J"

Section Contents

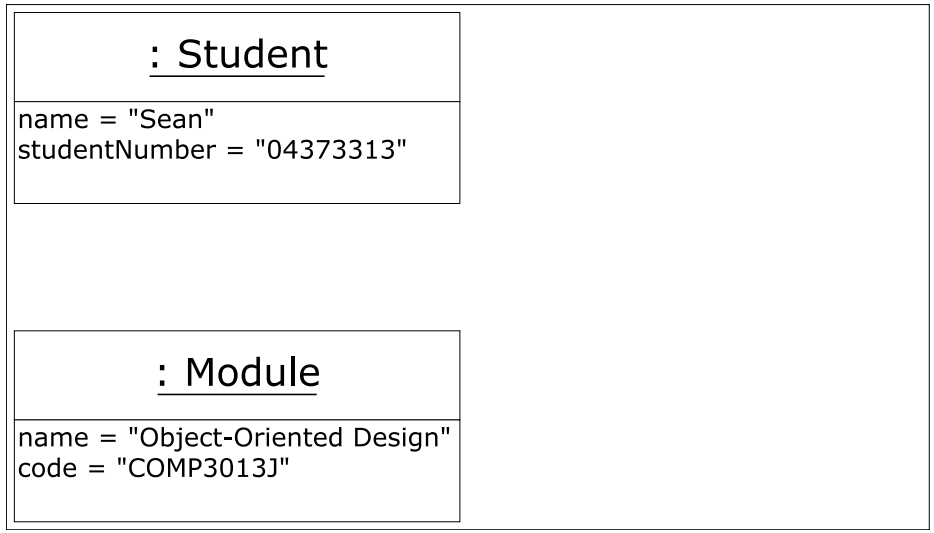
- Object diagrams are a static representation a group of objects at a specific point in time
- This includes both their internal data as well as the connections between them
- Object diagrams can be used to capture changes in the object graph over time
  - This is not the best way of showing changes in objects

```
1  public class Example {
2    public static void main(String[]
     ↪   args){
3      Student s = new Student("Sean"
       ↪  , "04373313");
4      Module m1 = new Module(
       ↪  "Object Oriented Design",
       ↪  "COMP3013J");
5      Module m2 = new Module(
       ↪  "Computer Graphics",
       ↪  "COMP3011J");
6      s.addClass(m1);
7      s.addClass(m2);
8    }
9  }
```

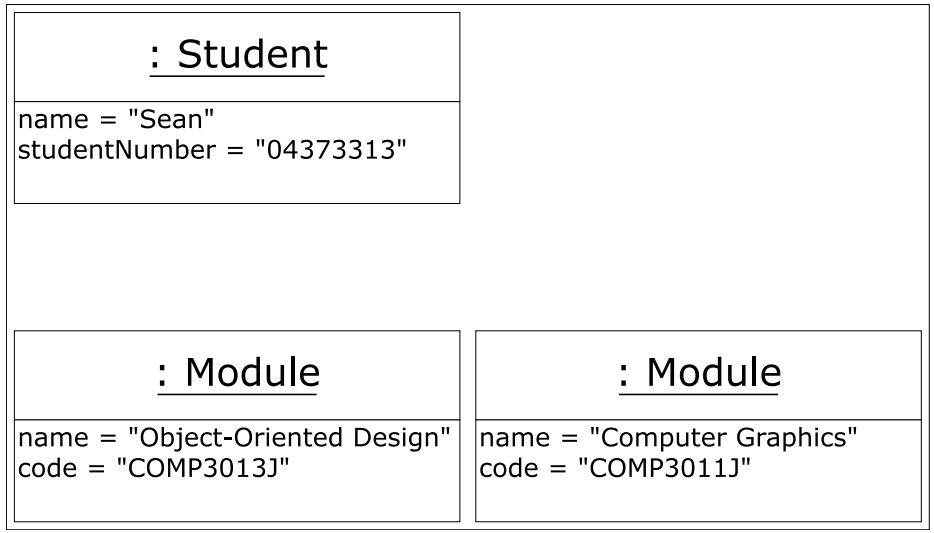
```
1  public class Example {
2    public static void main(String[]
     ↪   args){
3      Student s = new Student("Sean"
       ↪  , "04373313");
4      Module m1 = new Module(
       ↪  "Object Oriented Design",
       ↪  "COMP3013J");
5      Module m2 = new Module(
       ↪  "Computer Graphics",
       ↪  "COMP3011J");
6      s.addClass(m1);
7      s.addClass(m2);
8    }
9  }
```



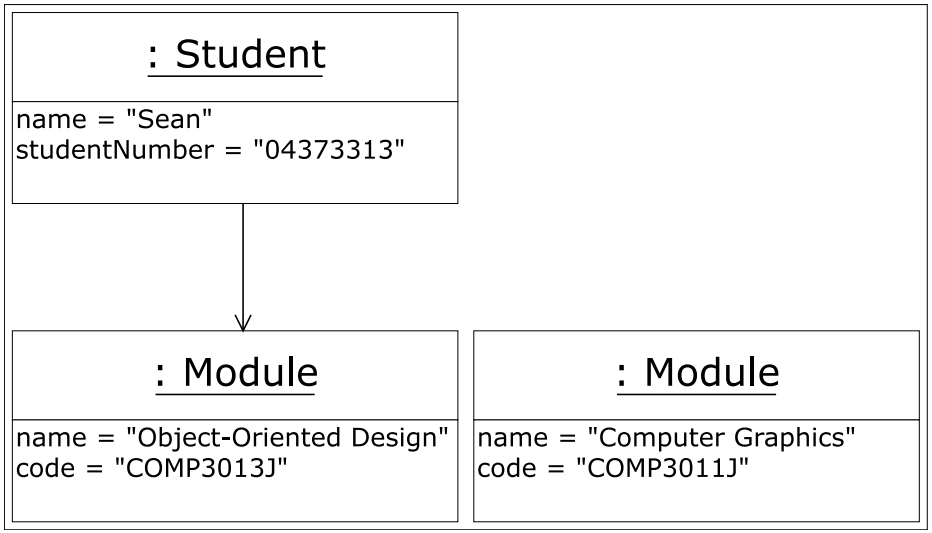
```
1  public class Example {
2    public static void main(String[]
     ↪   args){
3      Student s = new Student("Sean"
       ↪  , "04373313");
4      Module m1 = new Module(
       ↪  "Object Oriented Design",
       ↪  "COMP3013J");
5      Module m2 = new Module(
       ↪  "Computer Graphics",
       ↪  "COMP3011J");
6      s.addClass(m1);
7      s.addClass(m2);
8    }
9  }
```



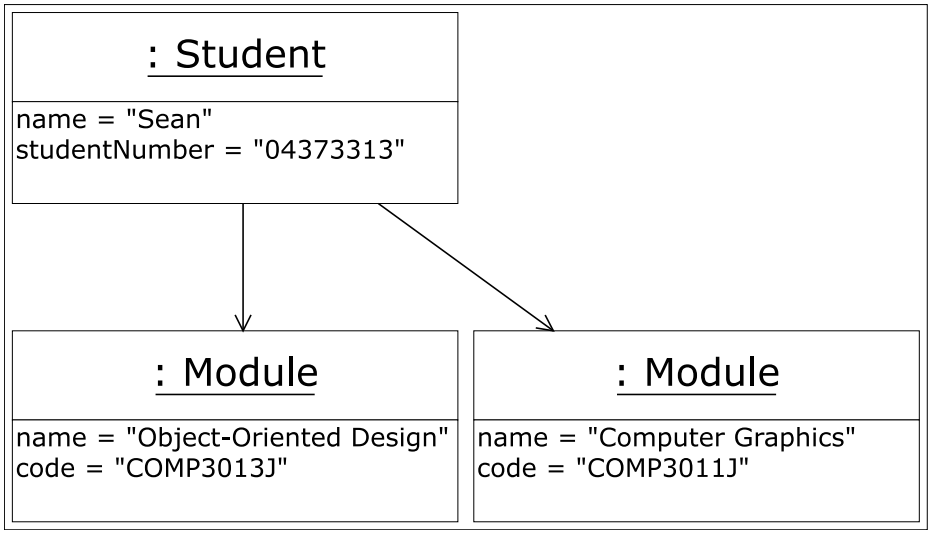
```
1  public class Example {
2    public static void main(String[]
     ↪   args){
3      Student s = new Student("Sean"
       ↪  , "04373313");
4      Module m1 = new Module(
       ↪  "Object Oriented Design",
       ↪  "COMP3013J");
5      Module m2 = new Module(
       ↪  "Computer Graphics",
       ↪  "COMP3011J");
6      s.addClass(m1);
7      s.addClass(m2);
8    }
9  }
```



```
1  public class Example {
2    public static void main(String[]
     ↪   args){
3      Student s = new Student("Sean"
       ↪  , "04373313");
4      Module m1 = new Module(
       ↪  "Object Oriented Design",
       ↪  "COMP3013J");
5      Module m2 = new Module(
       ↪  "Computer Graphics",
       ↪  "COMP3011J");
6      s.addClass(m1);
7      s.addClass(m2);
8    }
9  }
```

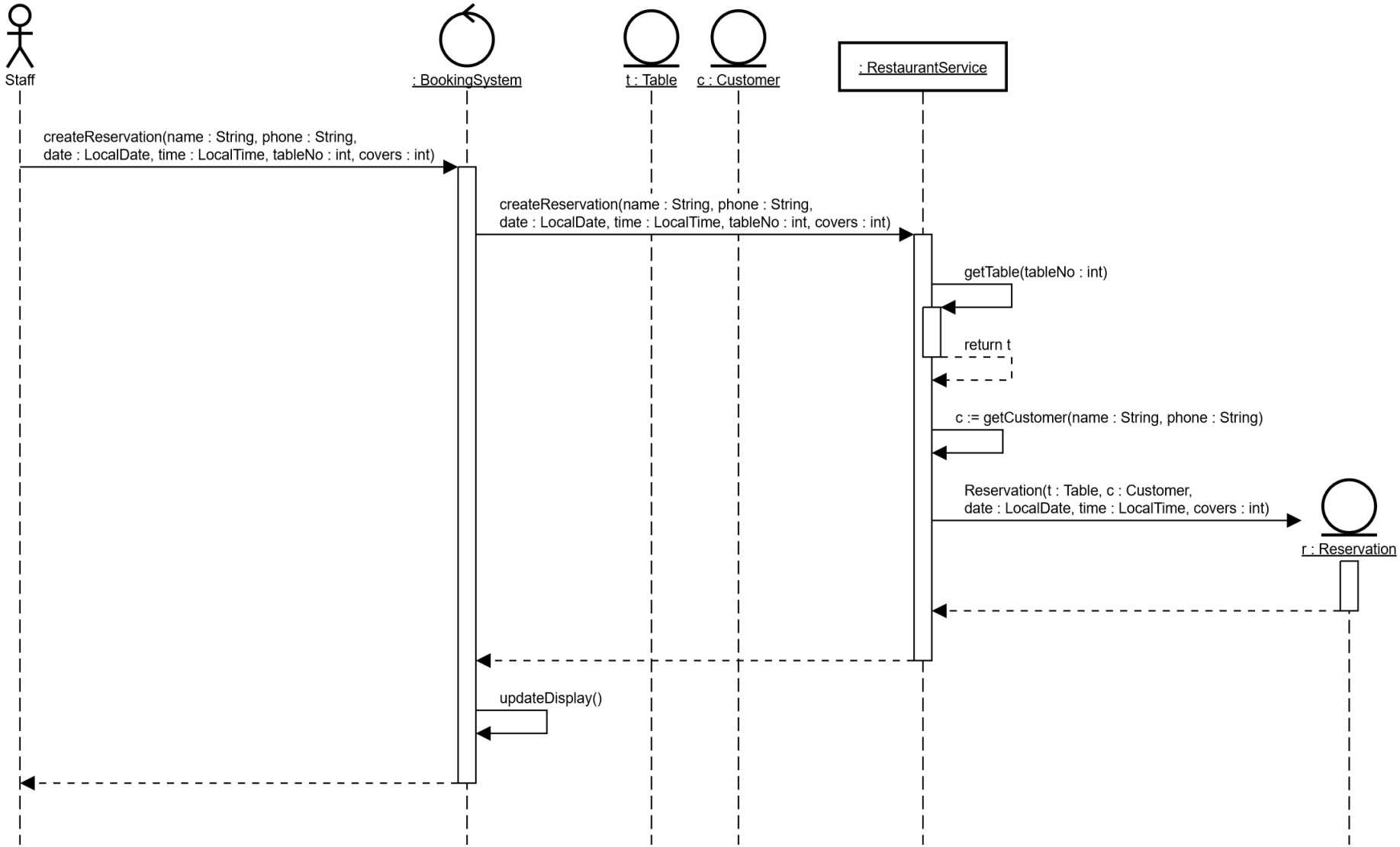


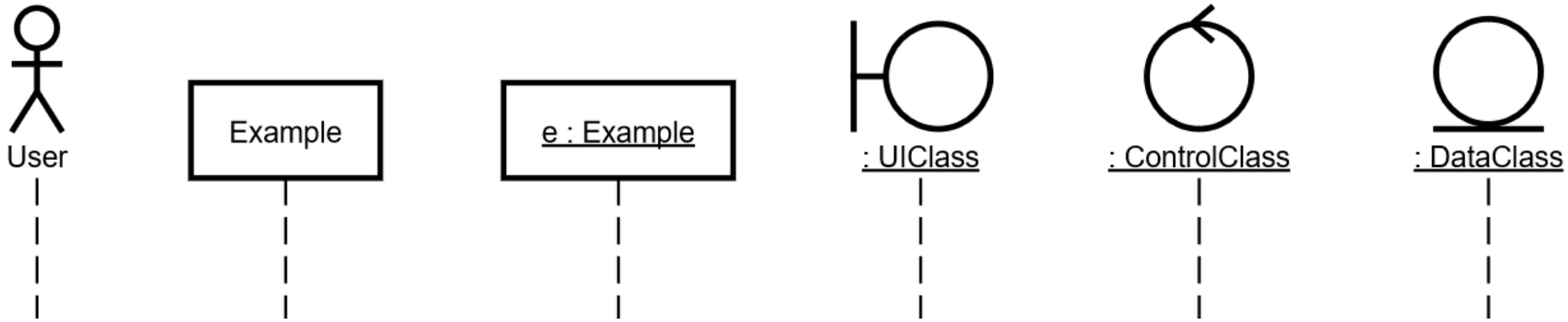
```
1  public class Example {
2    public static void main(String[]
     ↪   args){
3      Student s = new Student("Sean"
       ↪  , "04373313");
4      Module m1 = new Module(
       ↪  "Object Oriented Design",
       ↪  "COMP3013J");
5      Module m2 = new Module(
       ↪  "Computer Graphics",
       ↪  "COMP3011J");
6      s.addClass(m1);
7      s.addClass(m2);
8    }
9  }
```



Section Contents

- Sequence diagrams are a way of representing dynamic behaviour in our programs
- They focus on interactions between a number of objects
- These diagrams highlight the messages passed between objects as well as the **flow of control** in the program

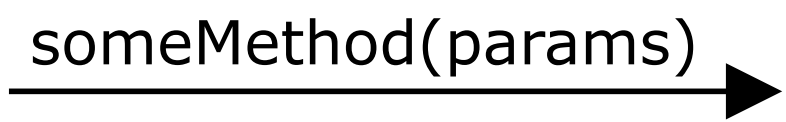

Staff
: BookingSystem
t : Table
c : Customer
: RestaurantService
createReservation(name : String, phone : String,
date : LocalDate, time : LocalTime, tableNo : int, covers : int)
createReservation(name : String, phone : String,
date : LocalDate, time : LocalTime, tableNo : int, covers : int)
getTable(tableNo : int)
return t
c := getCustomer(name : String, phone : String)
Reservation(t : Table, c : Customer,
date : LocalDate, time : LocalTime, covers : int)
r : Reservation
updateDisplay()



- Different icons can be used to show more information
- The dashed lines below the icon represent the lifeline of the object/class

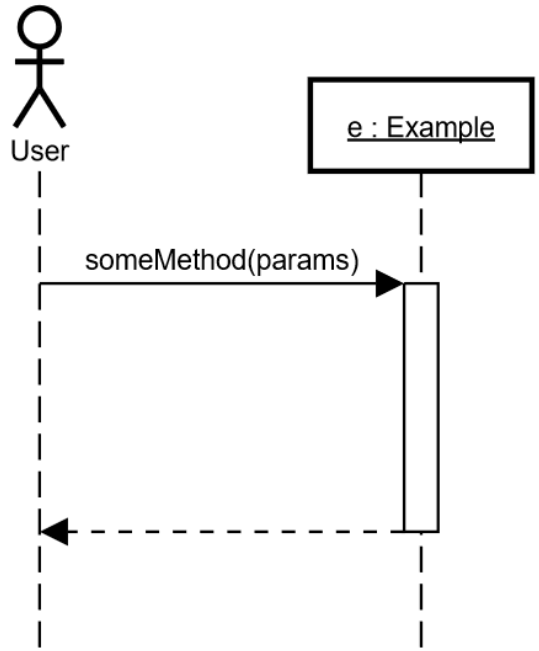
- The vertical axis of sequence diagrams represents time
- However, this is only in the sense of the order the events happen in
- If something is shown higher in the digram then it happened before, if it is lower then it happened after
- Icons shown at the top of a diagram already existed before this interaction started

- Sequence diagrams show interactions between objects
- Messages are shown as arrows between lifelines
- The arrow is labelled with the name of the method being called and often the names of parameters

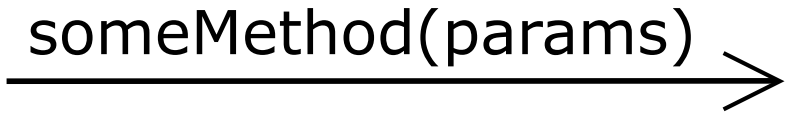


  - This is an example of a synchronous method call
  - A paired arrow showing the return of control is expected later

- When a method is called, the code begins executing
- This is called an activation
  - Shown as the long rectangle overlaid on the lifeline
- When the method is finished, a dashed line is shown connecting back to the object that called the method
  - Sometimes an explanation of what is returned is added

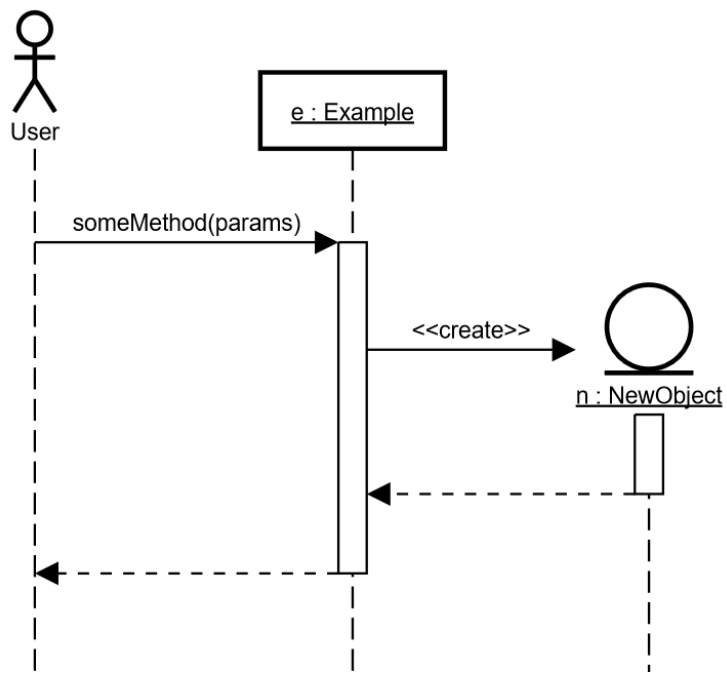


- Not all interactions are synchronous
- Sometimes the method of an object will be called and the calling object will not wait for a response

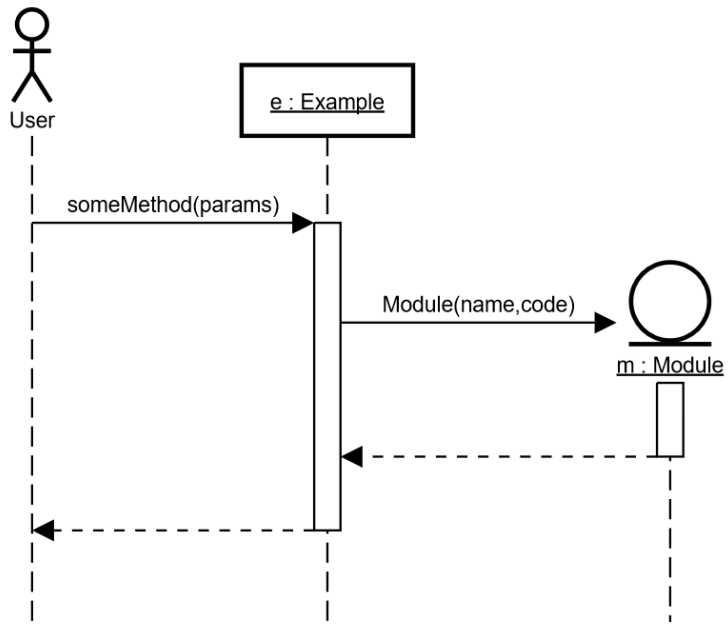


  - This is an example of an asynchronous method call
  - There is no need for a paired return later

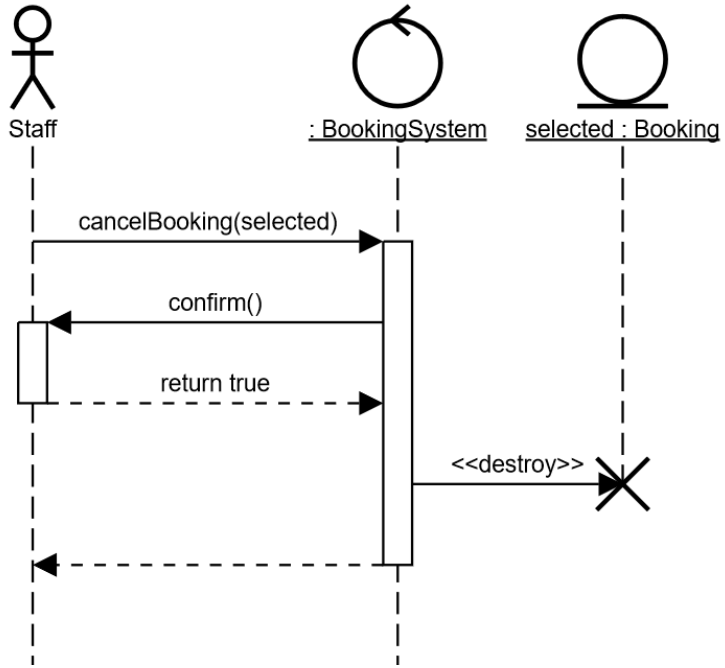
- When a new object is created, its icon is shown in that position
- This can be shown using a message with the stereotype `<<create>>`
- Or with a message showing the constructor



- When a new object is created, its icon is shown in that position
- This can be shown using a message with the stereotype `<<create>>`
- Or with a message showing the constructor

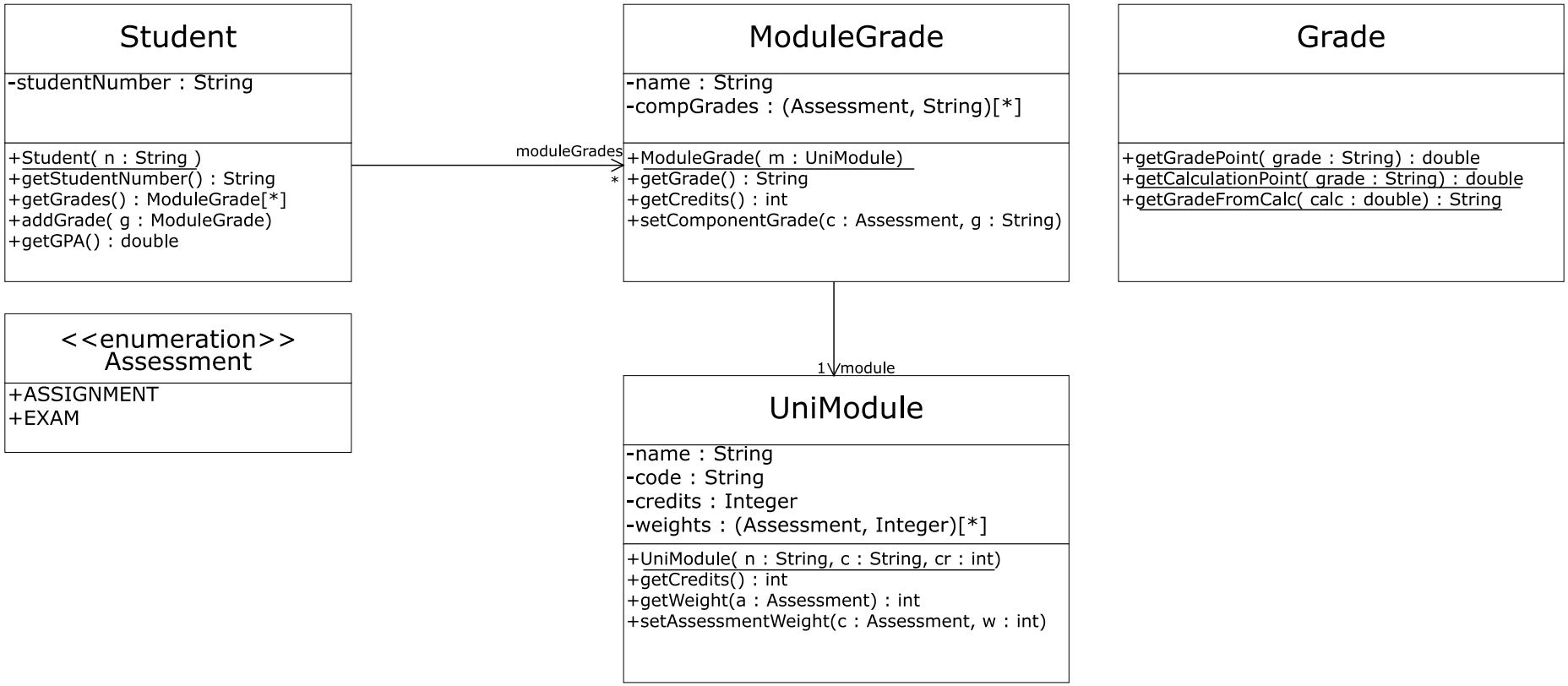


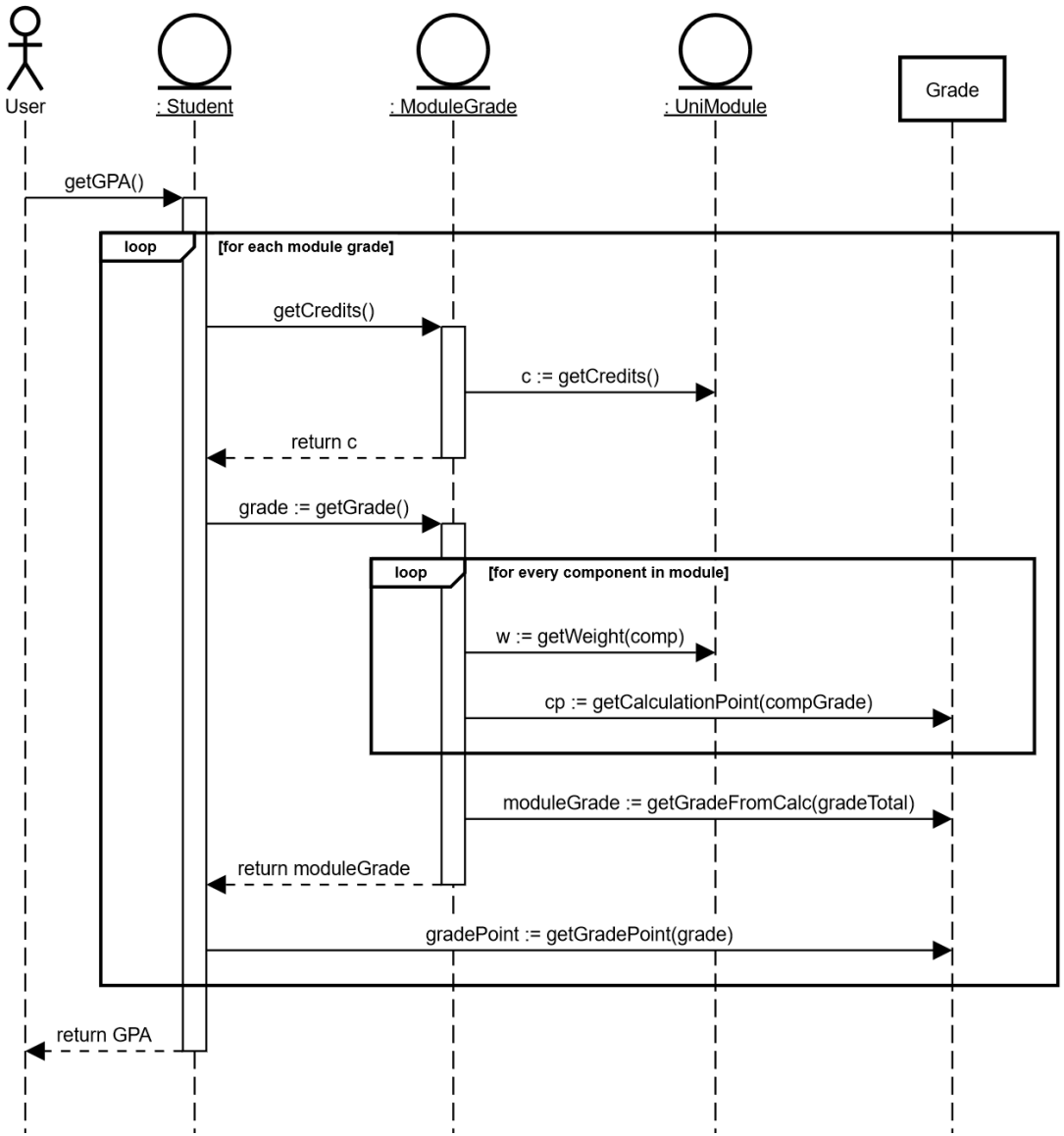
- When a new object is destroyed or deleted, its lifeline ends
- This can be shown using a message with the stereotype `<<destroy>>`
- A large X is typically placed at the point of destruction



Section Contents

- Sequence diagrams show the interactions between objects in a system
- From this information we can know some information about how a method could be implemented
- But any code not related to interactions will not be shown
- We will examine this idea by looking at an example


Student
-studentNumber : String
+Student(n : String)
+getStudentNumber() : String
+getGrades() : ModuleGrade[*]
+addGrade(g : ModuleGrade)
+getGPA() : double
moduleGrades
*
ModuleGrade
-name : String
-compGrades : (Assessment, String)[*]
+ModuleGrade(m : UniModule)
+getGrade() : String
+getCredits() : int
+setComponentGrade(c : Assessment, g : String)
Grade
+getGradePoint(grade : String) : double
+getCalculationPoint(grade : String) : double
+getGradeFromCalc(calc : double) : String
<<enumeration>>
Assessment
+ASSIGNMENT
+EXAM
1
module
UniModule
-name : String
-code : String
-credits : Integer
-weights : (Assessment, Integer)[*]
+UniModule(n : String, c : String, cr : int)
+getCredits() : int
+getWeight(a : Assessment) : int
+setAssessmentWeight(c : Assessment, w : int)


User
: Student
: ModuleGrade
: UniModule
Grade
getGPA()
loop
[for each module grade]
getCredits()
c := getCredits()
return c
grade := getGrade()
loop
[for every component in module]
w := getWeight(comp)
cp := getCalculationPoint(compGrade)
moduleGrade := getGradeFromCalc(gradeTotal)
return moduleGrade
gradePoint := getGradePoint(grade)
return GPA

```java
public double getGPA(){
    double total = 0.0;
    int totalWeight = 0;
    for(ModuleGrade mg : moduleGrades){
        int credits = mg.getCredits();
        totalWeight += credits;
        total += Grade.getGradePoint(mg.getGrade()) * credits;
    }
    return total/totalWeight;
}
```

- The diagram tells us the sequence of method calls for `getCredits`, `getGrade`, and `getGradePoint`
- But it does not tell us what the method does with the information

```java
public String getGrade(){
  int totalWeight = 0;
  double totalPoints = 0.0;
    for (Assessment a : Assessment.values()){
      if (compGrades.get(a) != null){
        int weight = module.getWeight(a);
        totalWeight += weight;
        totalPts += Grade.getCalculationPoint(compGrades.get(a)) * weight;
      }
    }
    if (totalWeight != 100){
      throw new RuntimeException("Total weight is not 100");
    }
    return Grade.getGradeFromCalc(totalPts/100);
}
```

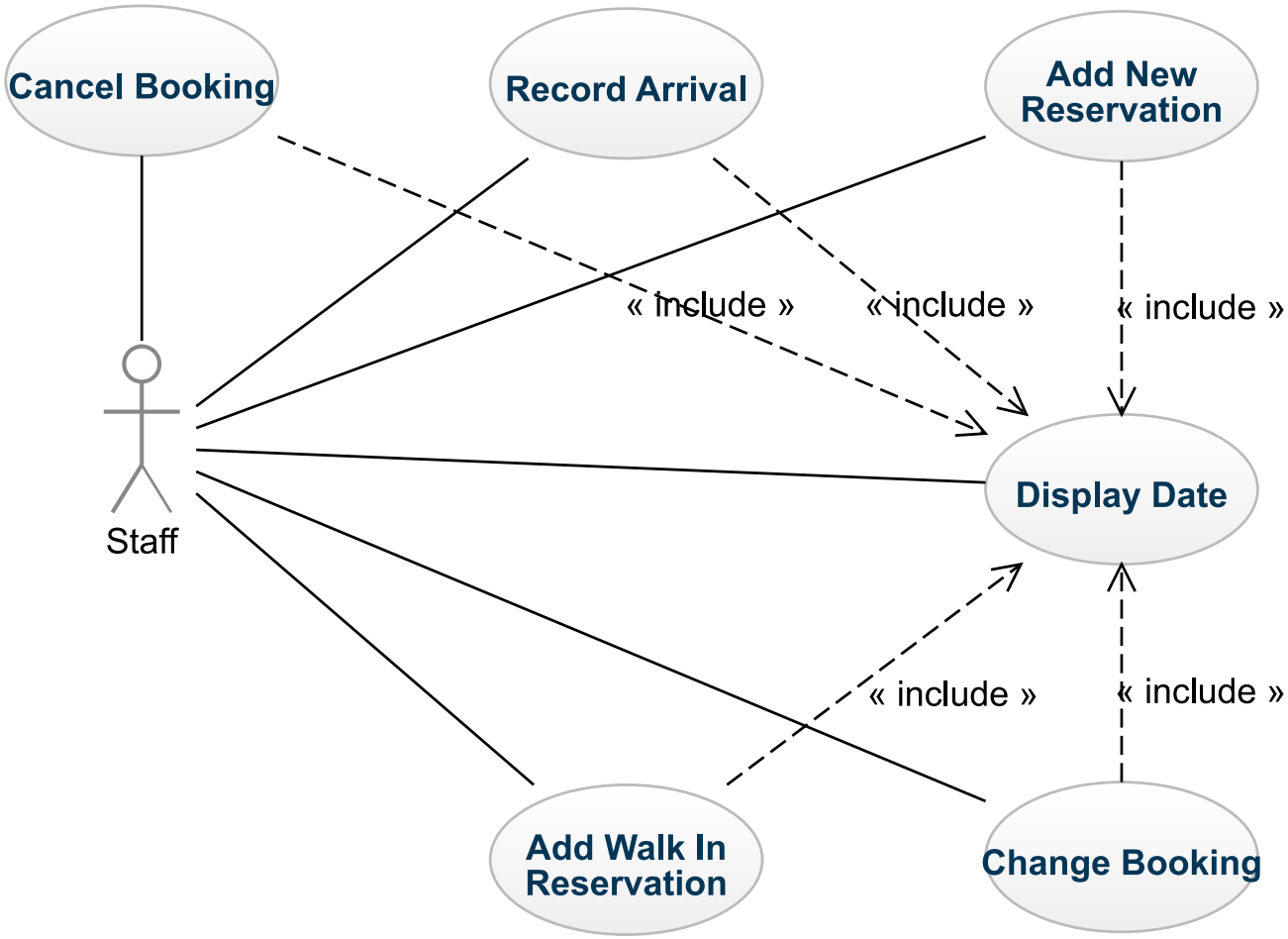
- The diagram tells us the sequence of method calls for `getWeight`, `getCalculationPoint`, and `getGradeFromCalc`
- But it does not tell us what the method does with the information

Section Contents

- Use case modelling is creating a structured view of a systems functionality
- We define the **actors** and **use cases**
  - An actor is a role that can be played in interaction
  - A use case describes a specific task that we can achieve

- Use cases are usually defined by consulting with users
- We will return to the process in the next lecture
- In this lecture we will look at the diagram itself

- Actors represent the different roles that users have in the system
- Use cases are shown as ovals containing the name of the use case
- Lines connect the actors to the use cases that they will perform
- Relationships between use cases are shown with dashed arrows
  - There are includes and extends relationships


Cancel Booking
Record Arrival
Add New Reservation
« include »
« include »
« include »
Staff
Display Date
« include »
« include »
Add Walk In Reservation
Change Booking