



**Software Engineering
Bootcamp**

Hyperiondev

Object-Oriented Design with Context & Sequence Diagrams

Welcome

Your Lecturer for this session



Armand Le Roux

Lecture – Housekeeping

- ❑ The use of disrespectful language is prohibited in the questions, this is a supportive, learning environment for all - please engage accordingly.
- ❑ No question is daft or silly - **ask them!**
- ❑ There are Q/A sessions midway and at the end of the session, should you wish to ask any follow-up questions.
- ❑ You can also submit questions here:
<http://hyperiondev.com/sbc4-se-questions>
- ❑ For all non-academic questions, please submit a query:
www.hyperiondev.com/support
- ❑ Report a safeguarding incident:
<http://hyperiondev.com/safeguardreporting>
- ❑ We would love your feedback on lectures:
<https://hyperiondev.wufoo.com/forms/zsgv4m40ui4i0g/>

Objectives

1. Context diagrams
 - a. What is a context diagram?
 - b. Benefits of creating context diagrams
2. Sequence diagrams
 - a. What is a sequence diagram?
 - b. Benefits of Sequence diagrams
 - c. Basic symbols & components
 - d. Common message symbols

Github Repository – Lecture Examples

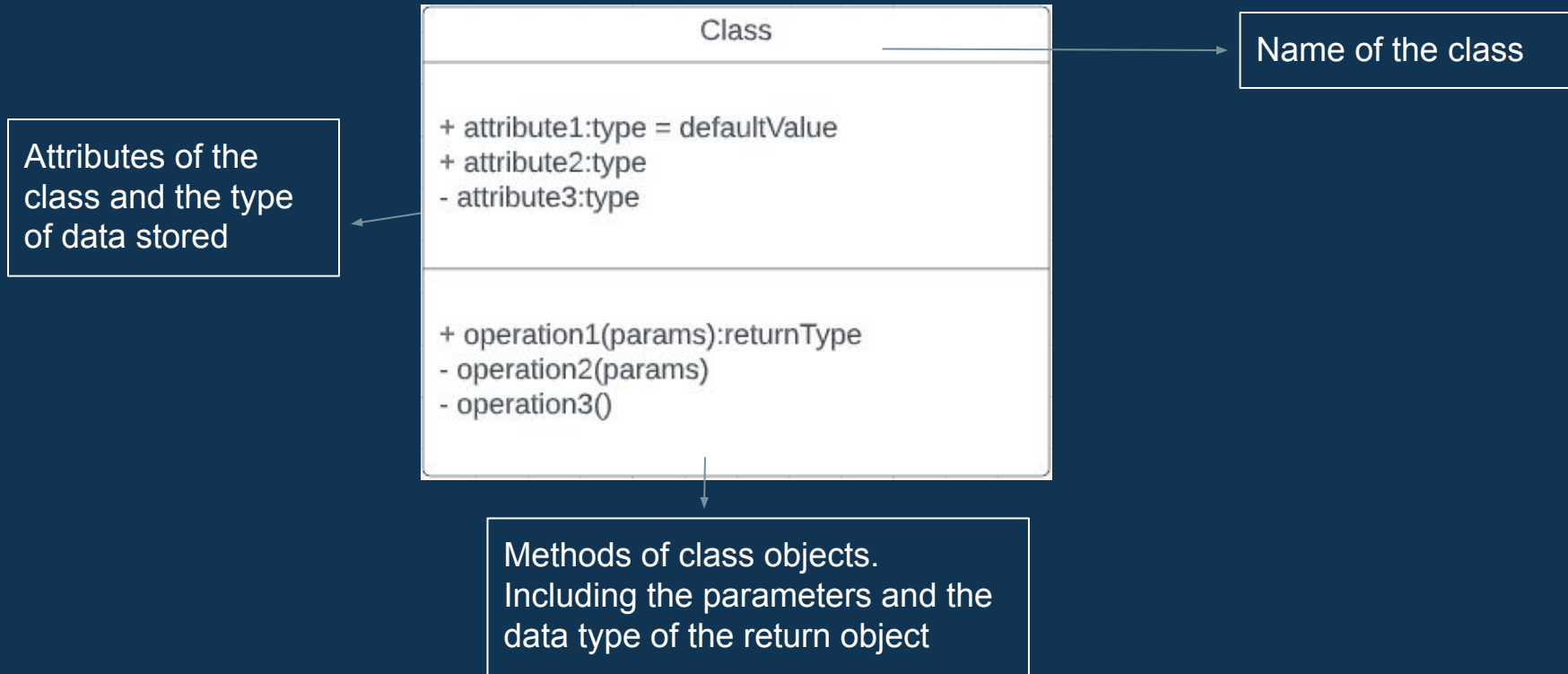
https://github.com/HyperionDevBootcamps/C4_SE_lecture_examples

Platform to create context & sequence diagrams

<https://www.lucidchart.com/pages/uml-class-diagram>

Recap on Class Diagrams

The Basics of Class Diagrams



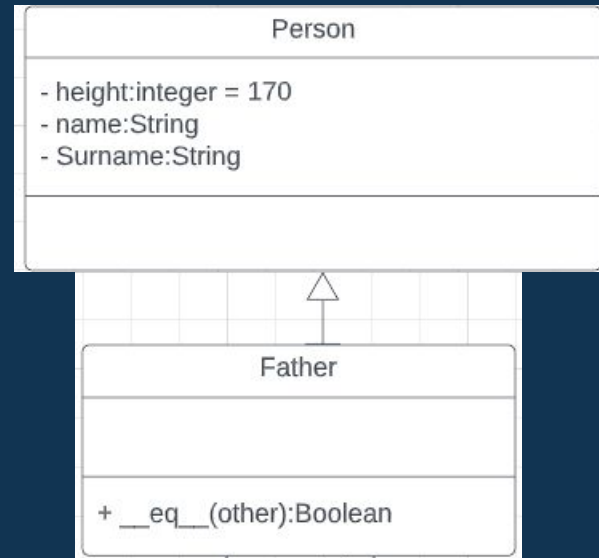
Drawing you own class diagram

- Let's have a look at one of our previous inheritance examples.

```
# Create a parent class called person
class Person:
    height = 170
    def __init__(self, name, surname):
        self.name = name
        self.surname = surname
```

```
# Create a child class called Father
class Father(Person):
    def __init__(self, name, surname = "Reeds"):
        super().__init__(name, surname)
        self.height = super().height - 10
```

```
def __eq__(self, other):
    if self.height == other.height:
        return True
    else:
        return False
```



- We also had two child classes of Father class

```
# Create a child class of Father called Son
class Son(Father):
```

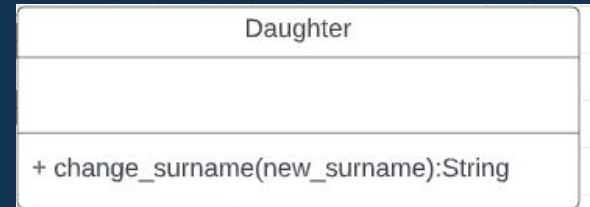
```
    def __init__(self, name):
        super().__init__(name)
        self.height = super().height + 10
```

```
    # Create a method to change the height of the son
    def set_height(self, new_height):
        self.height = new_height
```

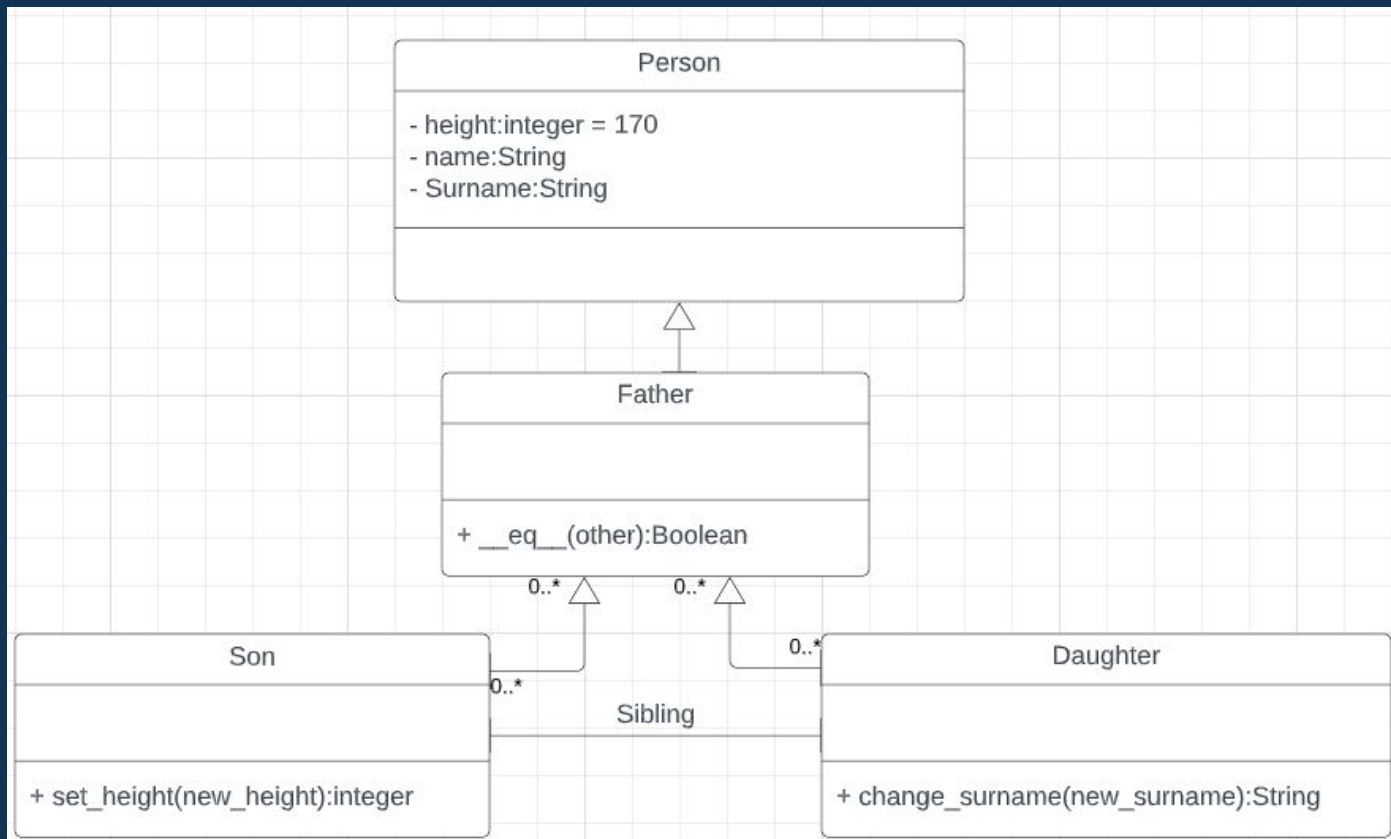
```
# Create another child class of Father called daughter
class Daughter(Father):
```

```
    def __init__(self, name):
        super().__init__(name)
```

```
    # Create methods to change the surname of the daughter
    def change_surname(self, new_surname):
        self.surname = new_surname
```



The relationship between classes



What is a context diagram?

- A context diagram is a high-level view of a system.
- It's a basic sketch meant to define an entity based on its scope, boundaries, and relation to external components like stakeholders.
- Otherwise known as a Level 0 data flow diagram, a context diagram provides a general overview of a process, focusing on its interaction with outside elements rather than its internal sub-processes.

Benefits of creating a context diagram

- A context diagram zeroes in on the external factors that must be considered when building a project or product's internal architecture.
- It is most helpful during the planning phase when proponents are just starting to interpret the landscape they are working with. At this point, a context diagram can ensure that the systems developed are relevant to the project's requirements and restrictions, and therefore able to reduce potential risks.
- It's an excellent tool for sharing critical information with team members and helping them develop a better appreciation of projects.

Context Diagram Example

A system context diagram represents the scope of a technical program or application like an online community.



What is a sequence diagram?

- A sequence diagram is a type of interaction diagram that describes how—and in what order—a group of objects works together.
- These diagrams are used by software developers and business professionals to understand requirements for a new system or to document an existing process.



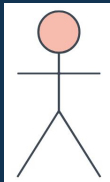
Benefits of Sequence Diagrams

Sequence diagrams can be useful references for businesses and other organizations. Try drawing a sequence diagram to:




- Model the logic of a sophisticated procedure, function, or operation.
- See how objects and components interact with each other to complete a process.
- Plan and understand the detailed functionality of an existing or future scenario.

Basic symbols & components


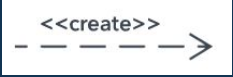

To understand what a sequence diagram is, you should be familiar with its symbols and components. Sequence diagrams are made up of the following icons and elements:

Name	Symbol	Description
Object symbol		Represents a class or object in UML. The object symbol demonstrates how an object will behave in the context of the system. Class attributes should not be listed in this shape.
Activation box		Represents the time needed for an object to complete a task. The longer the task will take, the longer the activation box becomes.
Actor Symbol		Shows entities that interact with or are external to the system.

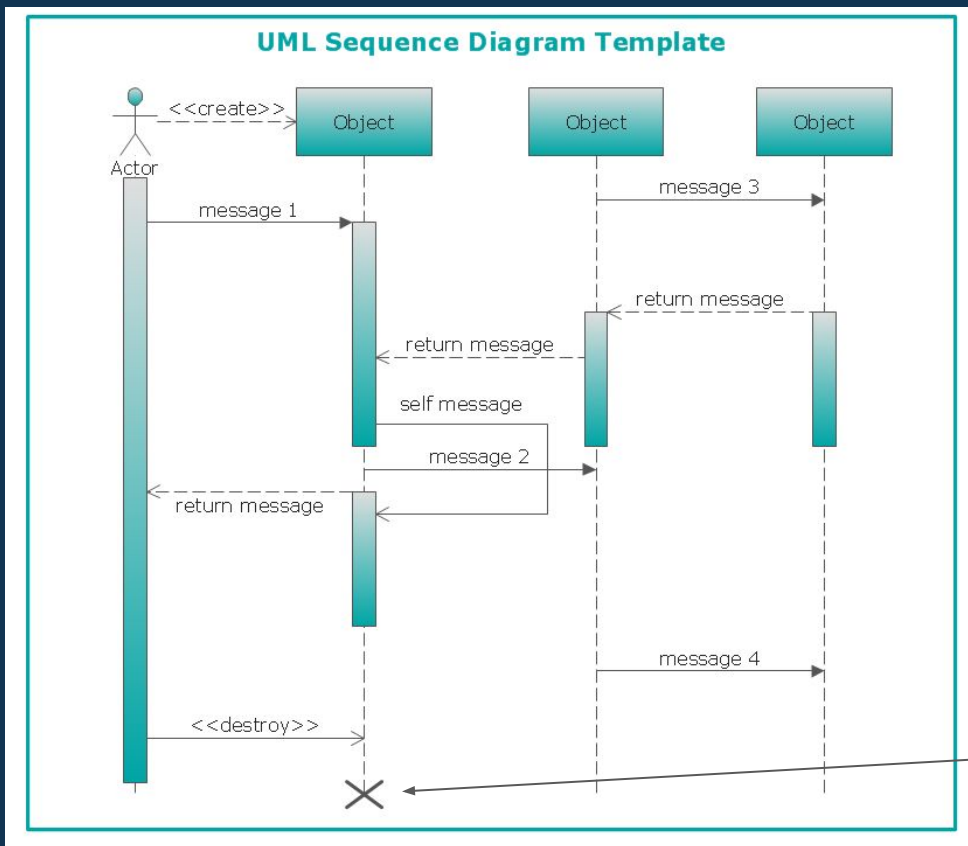
Common message symbols

Name	Symbol	Description
Synchronous message symbol		Represented by a solid line with a solid arrowhead. This symbol is used when a sender must wait for a response to a message before it continues. The diagram should show both the call and the reply.
Reply message symbol		Represented by a dashed line with a lined arrowhead, these messages are replies to calls.
Asynchronous message symbol		Represented by a solid line with a lined arrowhead. Asynchronous messages don't require a response before the sender continues. Only the call should be included in the diagram.

Common message symbols

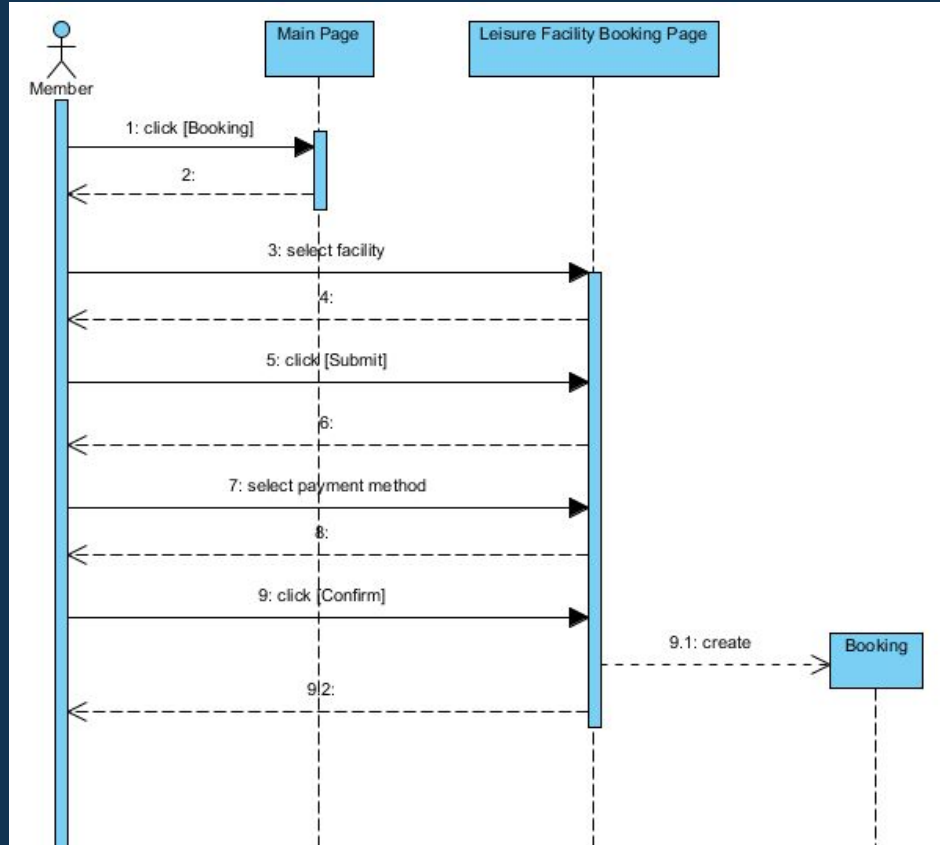
Name	Symbol	Description
Asynchronous return message symbol		Represented by a dashed line with a lined arrowhead.
Asynchronous create message symbol		Represented by a dashed line with a lined arrowhead. This message creates a new object.
Delete message symbol		Represented by a solid line with a solid arrowhead, followed by an X. This message destroys an object.

Sequence Diagram Example 1

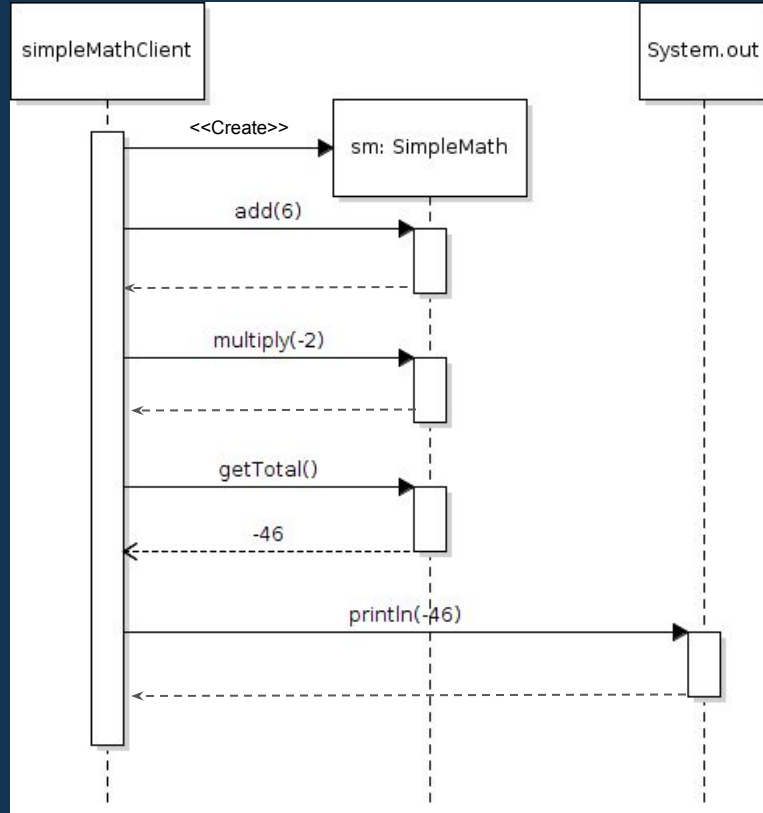


Delete message symbol:
This message destroys
an object.

Sequence Diagram Example 2



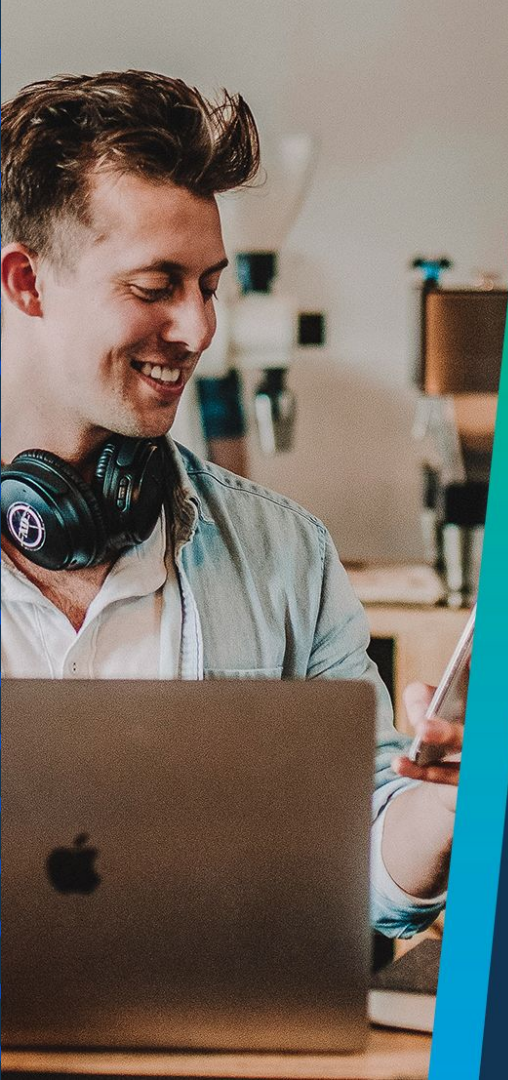
Sequence Diagram Example 3



Hyperiondev

Q & A Section

Please use this time to ask any questions relating to the topic explained, should you have any



Hyperiondev

Thank you for joining us

**Take regular breaks.
Stay hydrated.
Avoid prolonged screen time.
Remember to have fun :)**