Sequence Diagrams

Contents

[1 Introduction 1](#_Toc119417494)

[2 Examples 3](#_Toc119417495)

[3 Conditional Behavior 5](#_Toc119417496)

[4 Iteration 6](#_Toc119417497)

[5 Object Deletion 7](#_Toc119417498)

[6 Blank Template for Sequence Diagram (next page) 7](#_Toc119417499)

[Appendix 1 Resources 9](#_Toc119417500)

[Appendix 2 Real Examples 10](#_Toc119417501)

# Introduction

A *sequence diagram* is a UML diagram that shows the method calls that are necessary for a system to accomplish some task: (a) user story, (b) a method, (c) anything. For example, consider this code:

**public** **class** Driver {

**public** **static** **void** main(String[] args) {

A a = **new** A();

a.foo();

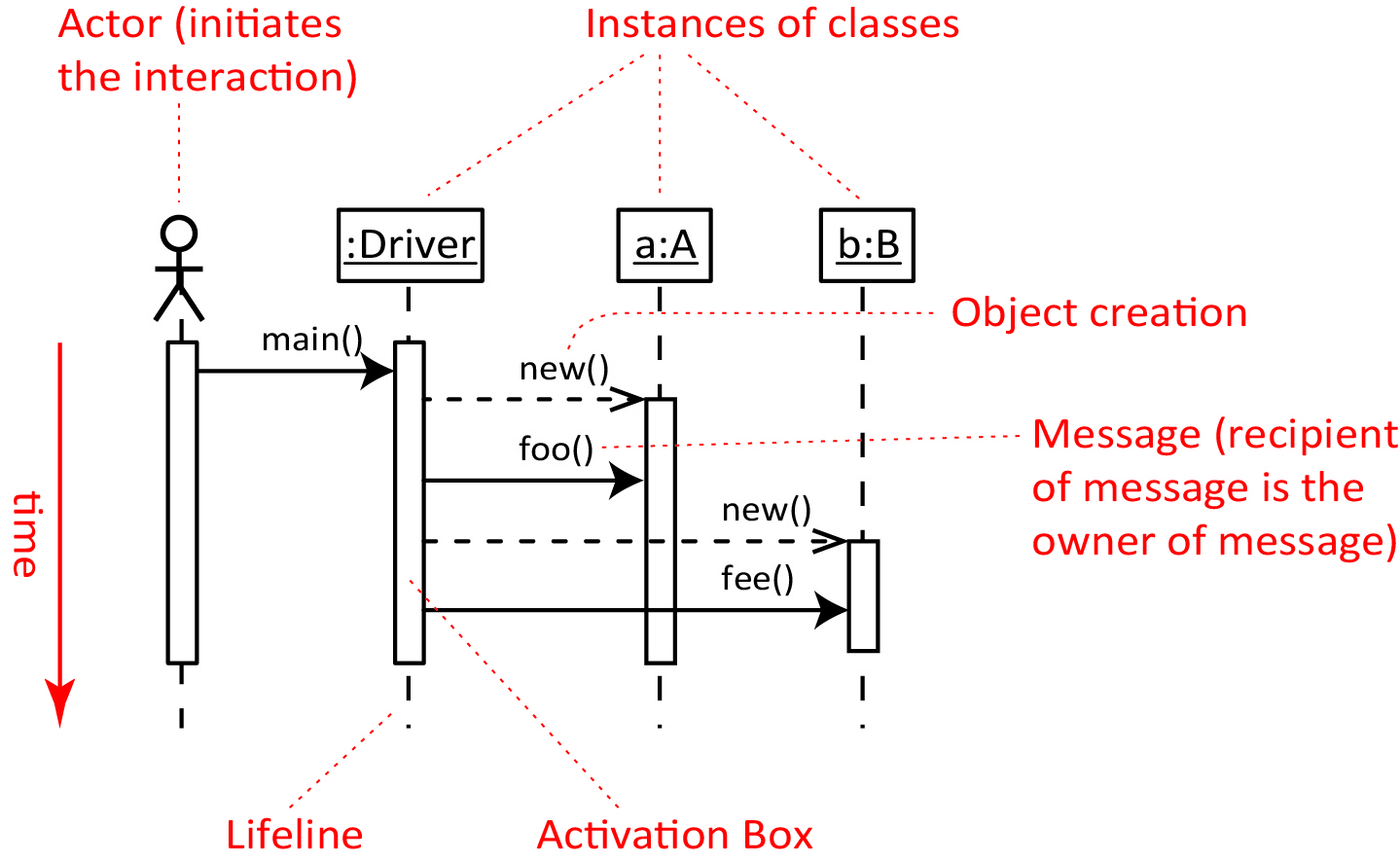
B b = **new** B();

b.fee();

}

}

A sequence diagram for the *main* method is:



Another way of describing a sequence diagram is that it shows the sequence of *messages exchanged* by the set of objects performing a certain task.

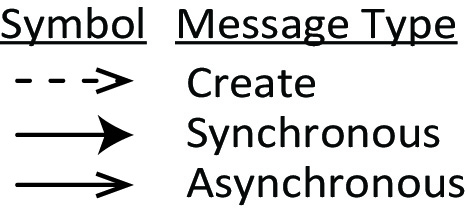
A sequence diagram is a type of *interaction* diagram in UML and is used to model the dynamic aspects of a system. There are two types of interactions diagrams: *Sequence Diagrams* and *Collaboration Diagrams*, which are virtually identical. We will only consider the sequence diagram.

Sequence diagrams are created, as needed, to:

* Document/communicate the dynamic aspects of the design of some piece of (complex) functionality
* At the design stage they are useful for refining a class diagram, *e.g.* are the methods in the correct class? Are the parameters correct, missing, navigability, *etc*.

Anatomy of a Sequence Diagram:

* Instances of classes – Shown as boxes with the class and object identifier underlined
* The objects are arranged horizontally across the diagram. The order doesn’t matter; however, it is useful to order them in the order they are called. Sometimes, it is useful to rearrange them so the diagram is minimally cluttered.
* An actor that initiates the interaction is often shown on the left. Use the stick-person symbol as in use case diagrams. The actor is not necessarily a person, it is frequently “some other object”.
* The vertical dimension represents time.
* A vertical (dashed) line, called a *lifeline*, is attached to each object or actor.
* The lifeline becomes a broad box, called an *activation box* during the *live activation* period. In the figure above, this is the time that the object is in memory. Stricter adherence to the UML standards gives a slightly different meaning to the live activation period: the time when the method is active. In other words, the box stops when the method ends.
* A message is represented as an arrow between activation boxes of the sender and receiver.
* A message is labelled and can have an argument list and a return value.
* The recipient of the message is the owner of the message. In the diagram above, the *foo* method is defined in the *A* class and the *fee* method is defined in the *B* class.
* Technically, the arrows below are the correct representation for the corresponding message type.



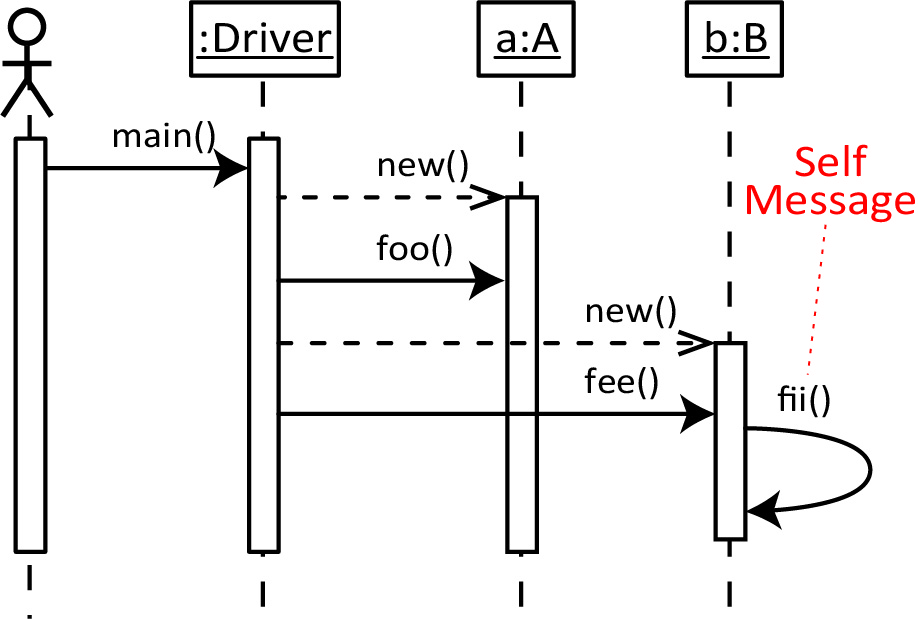
However, in much of this document, we will be much looser (due to my lack of understanding when I made these diagrams) and simply use the asynchronous arrow for both create and synchronous messages (we won’t use asynchronous messages)

# Examples

**Example 1** – For the example code above, if we had been given the code for the two classes *A* and *B*,

|  |  |  |
| --- | --- | --- |
| **public** **class** Driver {  **public** **static** **void** main(...) {  A a = **new** A();  a.foo();  B b = **new** B();  b.fee();  }  } | **public** **class** A {  **public** A() {}  **public** **void** foo() {}  } | **public** **class** B {  **public** B() {}  **public** **void** fee() {  fii();  }  **public** **void** fii() {}  } |

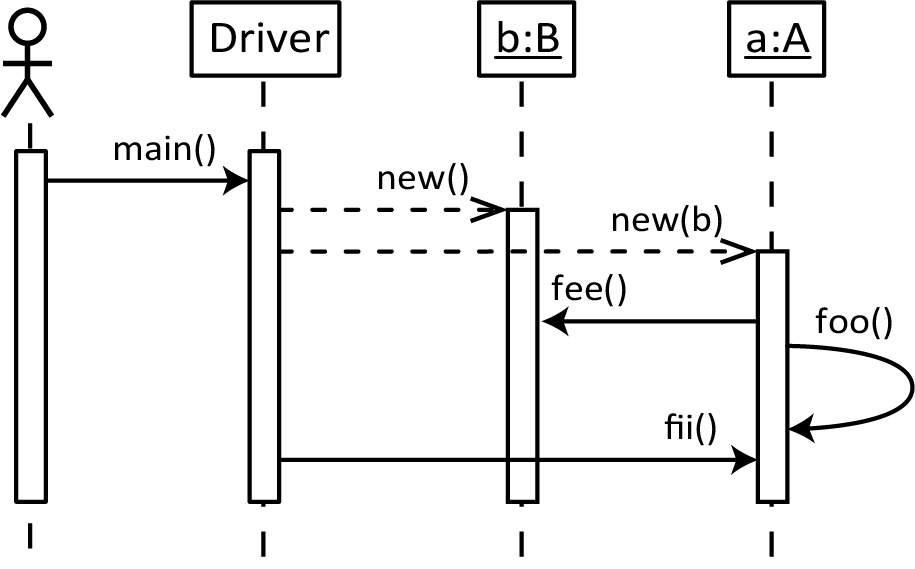
and we wanted to draw a *complete* sequence diagram of all interactions, the diagram would need to reflect any messages sent inside other messages. For example, the *fee* method in *B* calls the *fii* method. In this case, the *fii* message is called a *self-message*.



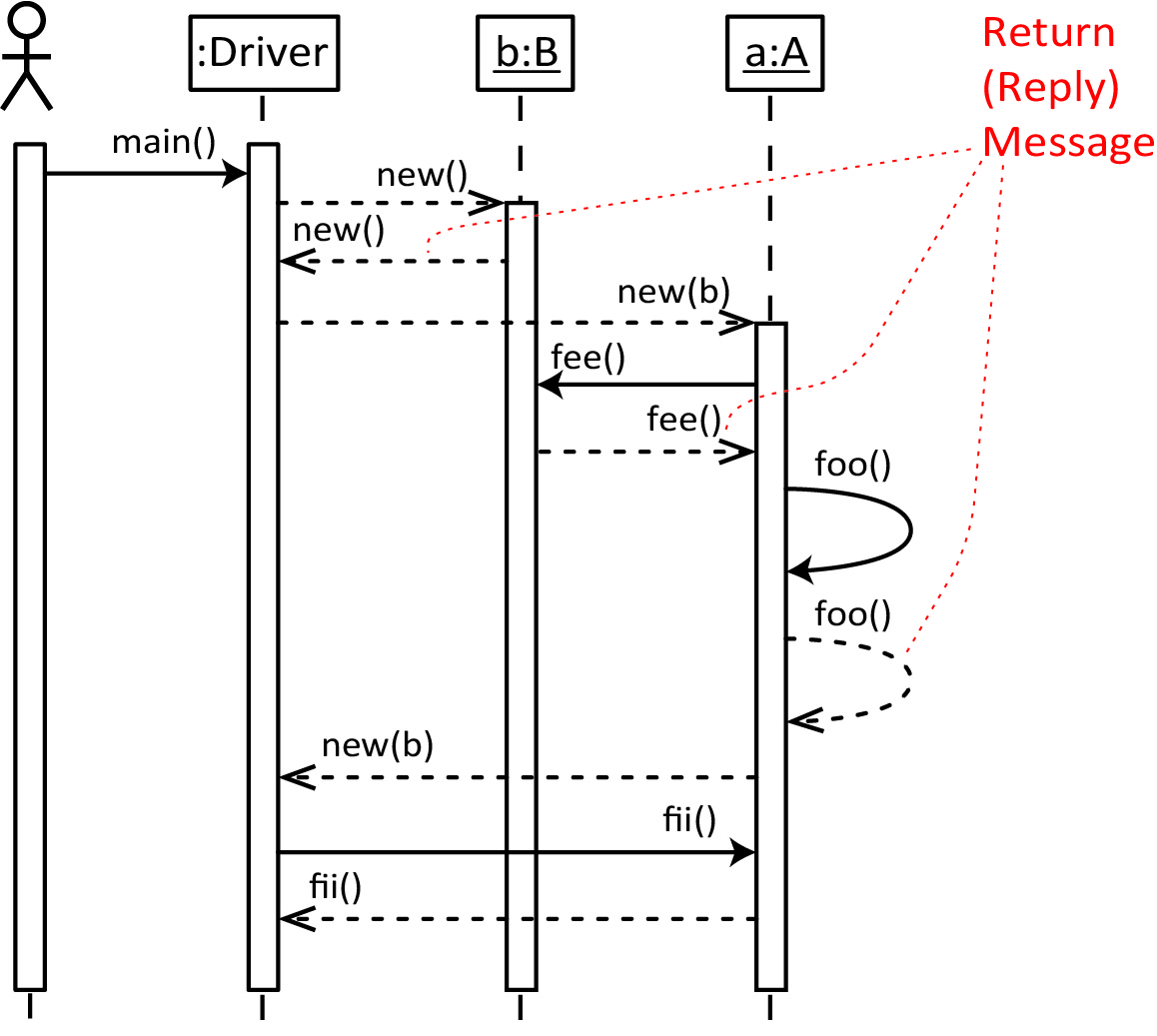
**Example 2** – Consider the code below.

|  |  |
| --- | --- |
| **public** **class** Driver {  **public** **static** **void** main(String[] args) {  B b = **new** B();  A a = **new** A(b);  a.fii();  }  } | **public** **class** A {  B b;  **public** A(B b) {  **this**.b=b;  b.fee();  foo();  }  **public** **void** foo() {}  **public** **void** fii() {}  }  **public** **class** B {  **public** B() {}  **public** **void** fee() {}  } |

A complete sequence diagram for the case when the system is run is shown below. Notice that *A’s* constructor calls back to the *b* object with the *fee* method, then calls its own *foo* method.



A sequence diagram can be ambiguous at times, without the indication of when a method finishes. Thus, we can add return (reply) arrows (dashed arrow with method name) when necessary. The example above with returns:



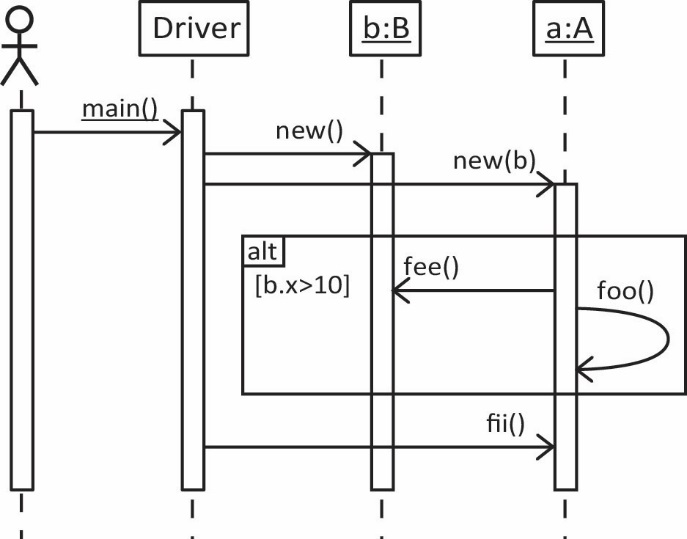
# Conditional Behavior

Conditional behavior in a sequence diagram is indicated by putting a box around the conditional code, and labeled with an *alt* tag, as shown in the example below. The condition (guard) for entering is shown in brackets. An [else] block can be added. Technically, in UML this is called a *combined fragment*.

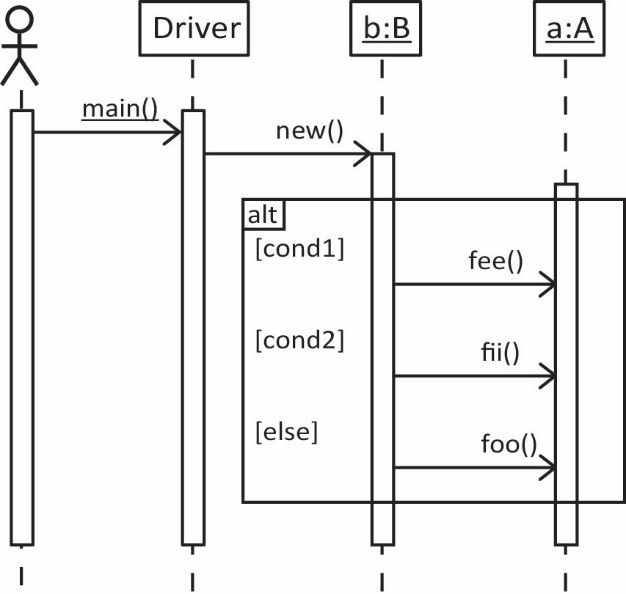
Example – Consider this code:

|  |  |  |
| --- | --- | --- |
| **public** **class** Driver {  **public** **static** **void**  main(String[] args) {  B b = **new** B(14);  A a = **new** A(b);  a.fii();  }  } | **public** **class** A {  B b;  **public** A(B b) {  **this**.b=b;  **if**(b.x>10) {  b.fee();  foo();  }  }  **public** **void** foo() {}  **public** **void** fii() {}  } | **class** B {  **int** x;  **public** B(**int** x) {  **this**.x=x;  }  **public** **void** fee() {}  } |

A complete sequence diagram for the case when the system is run is shown below.



A multi-part conditional (unrelated to the example above) is shown below:



# Iteration

Iteration in a sequence diagram is indicated by placing a box around code to be repeated, labeled with *loop*, and specifying how the iteration occurs as shown in the example below.

Consider this code:

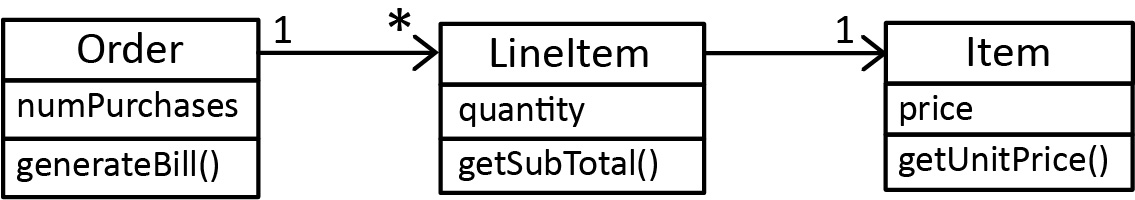
|  |  |
| --- | --- |
| **class** Product {  **double** price;  **public** **double** getPrice() {  **return** price;  }  **public** Product(**double** price) {  **this**.price = price;  }  } | **public** **class** Order {  List<Product> prods = **new** ArrayList<>();  **public** Order() {}  **public** **void** addProd(Product p) {  prods.add(p);  }  **public** **double** getTotal() {  **double** sum = 0.0;  **for**(Product p : prods) {  sum += p.getPrice();  }  **return** sum;  }  } |

A sequence diagram for the case when the *getTotal* is called is shown below using slightly different depictions for the collection of *Products*.

|  |  |  |
| --- | --- | --- |
|  |  |  |

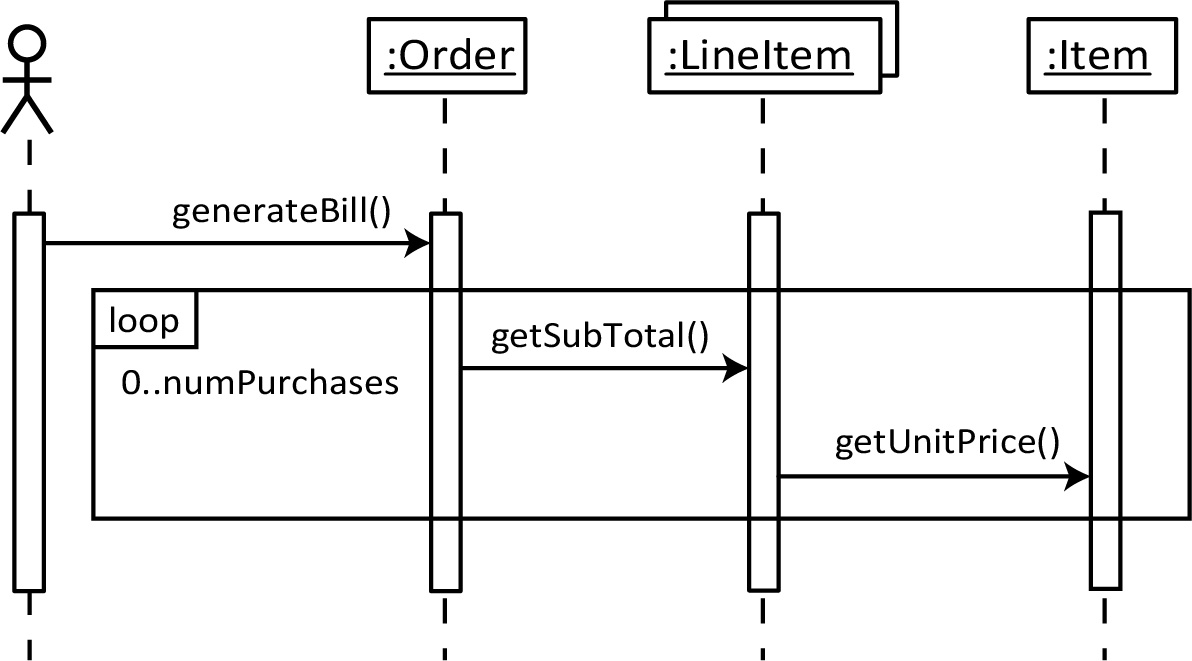
As another example, consider the use case and class diagram shown below to draw the corresponding sequence diagram.

**Class diagram:** A *Bill* contains a number of *LineItems* where each *LineItem* corresponds to one *Item* (Product). A *LineItem* specifies how many of the Item we want (and possibly other things).



**Use case:** ObtainBill – Obtain the total bill.

**Sequence diagram:**

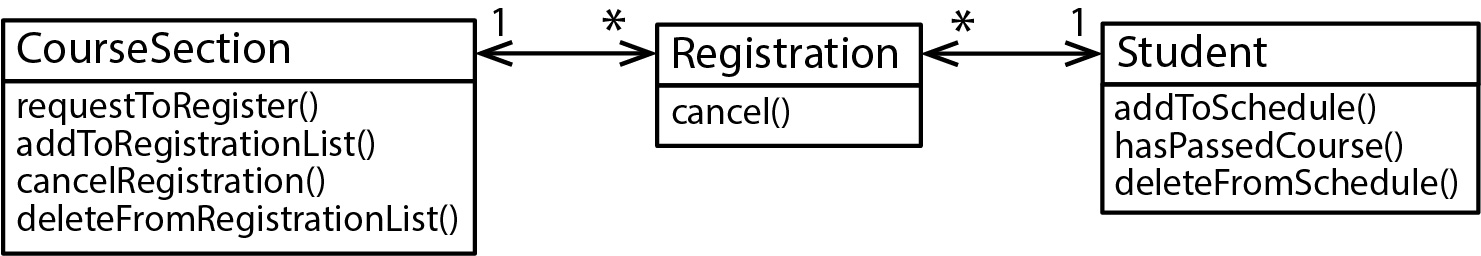


# Object Deletion

Object deletion in a sequence diagram is indicated by placing an “X” where the deletion occurs.

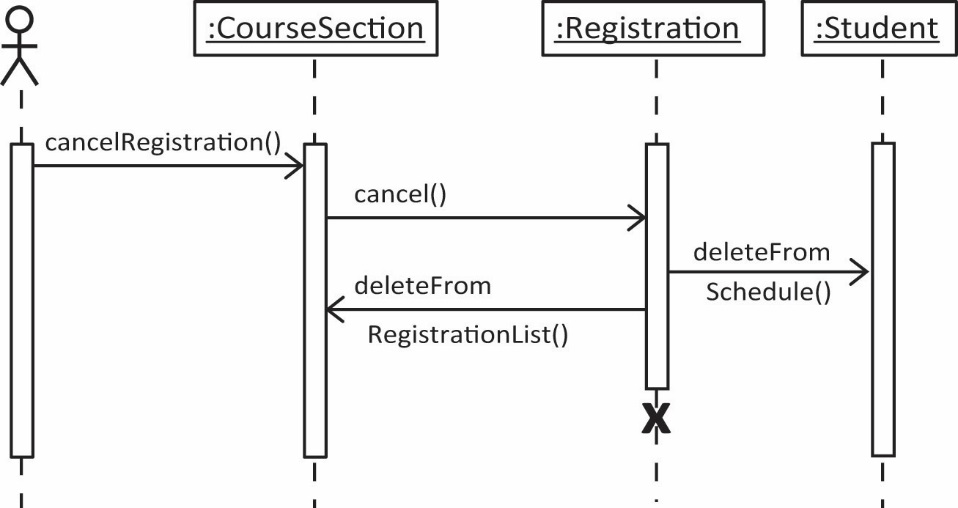
Example – Consider the use case and class diagram shown below to draw the corresponding sequence diagram.

**Class diagram:** This is the same situation as Example 1, except that we have added 4 methods: *cancelRegistration* and *deleteFromRegistrationList* in the *CourseSection* class, *deleteFromSchedule* in the *Student* class, and *cancel* in the Registration class.



**Use case:** CancelRegistration – Cancel the registration for a course.

**Sequence diagram:**



# Blank Template for Sequence Diagram (next page)



Appendix

1. Resources

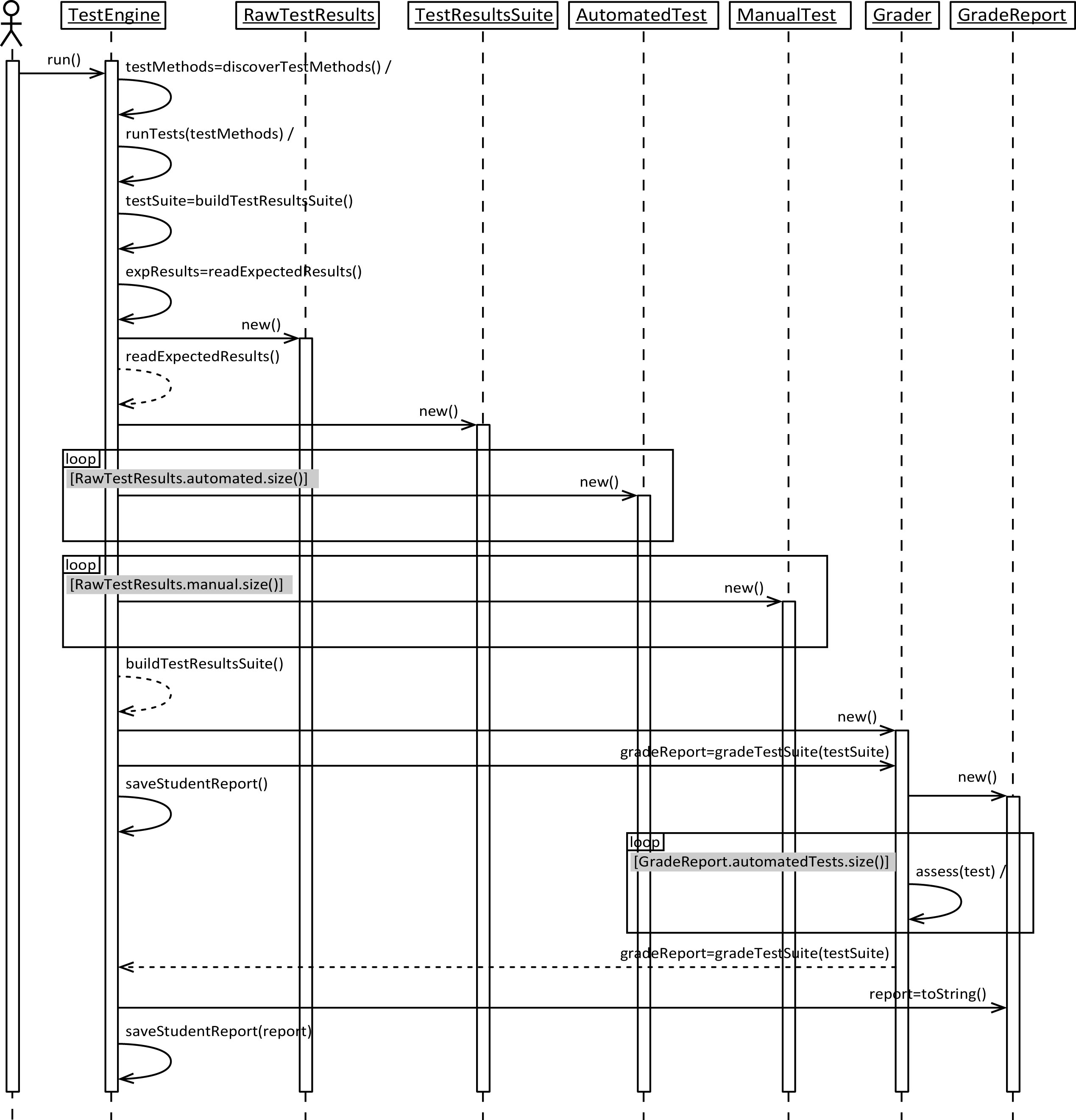
<https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-sequence-diagram/>

<https://www.uml-diagrams.org/sequence-diagrams.html>

<http://www.uml-diagrams.org/sequence-diagrams-reference.html>

1. Real Examples

This is from a system I wrote, *jGrader,* to automatically grade java programs:



The *gradeTestSuite* method above has a bit more detail than shown

