**Sequence Diagrams**

**Sequence Diagrams**

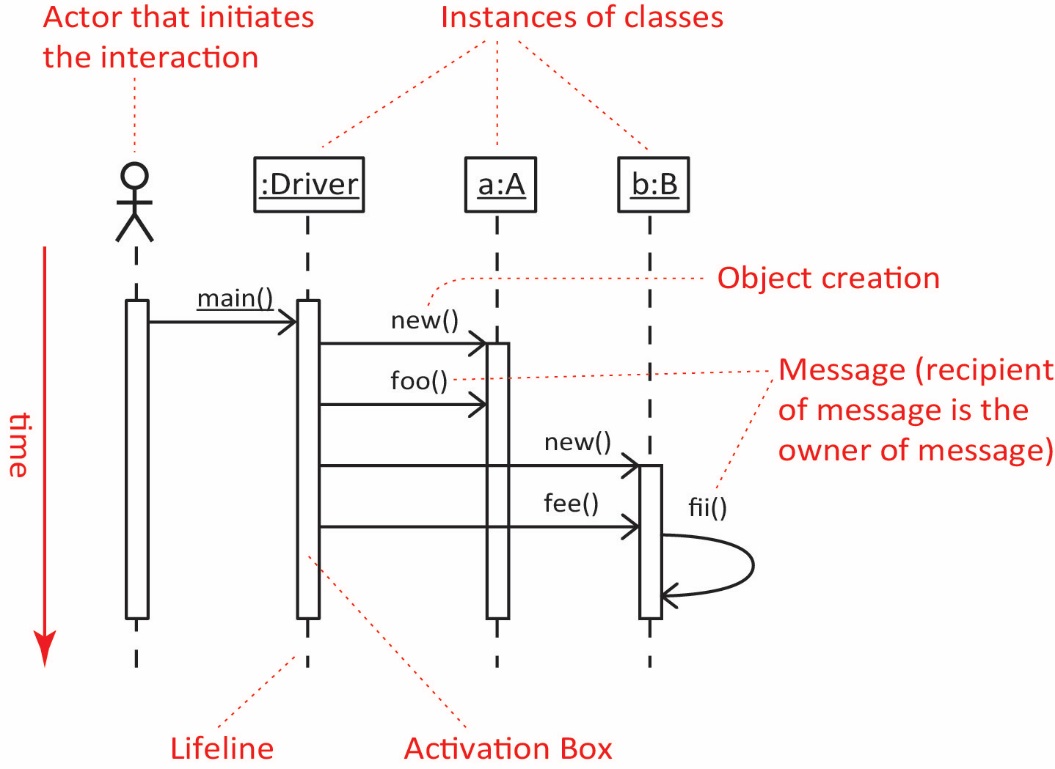
1. Sequence Diagrams –

* A sequence diagram shows the sequence of messages exchanged by the set of objects performing a certain task: the steps of a use case, a method, or the steps of some other piece of functionality. They are used to model the dynamic aspects of a software system.
* 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*.
* There are two types: *Sequence Diagrams* and *Collaboration Diagrams*, which are virtually identical. We will only consider the sequence diagram.
* Some references: <http://www.uml-diagrams.org/sequence-diagrams-reference.html>

1. 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();  }  } | **public** **class** A {  **public** A() {}  **public** **void** foo() {}  }  **public** **class** B {  **public** B() {}  **public** **void** fee() {  fii();  }  **public** **void** fii() {}  } |

Draw a sequence diagram for the case when the system is run:

s

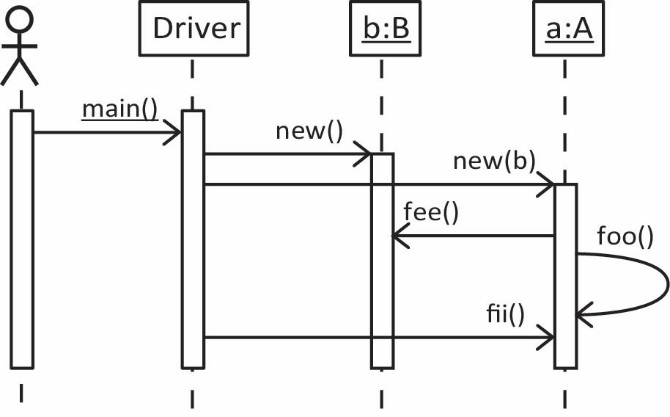
1. Anatomy of a Sequence Diagram
2. Instances of classes – Shown as boxes with the class and object identifier underlined
3. The objects are arranged horizontally across the diagram. The order doesn’t matter however you probably want to arrange them so that the diagram is not to cluttered.
4. An actor 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”.
5. The vertical dimension represents time.
6. A vertical (dashed) line, called a *lifeline*, is attached to each object or actor.
7. The lifeline becomes a broad box, called an *activation box* during the *live activation* period.
8. 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.

1. Example – Consider this code:

|  |  |
| --- | --- |
| **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() {}  } |

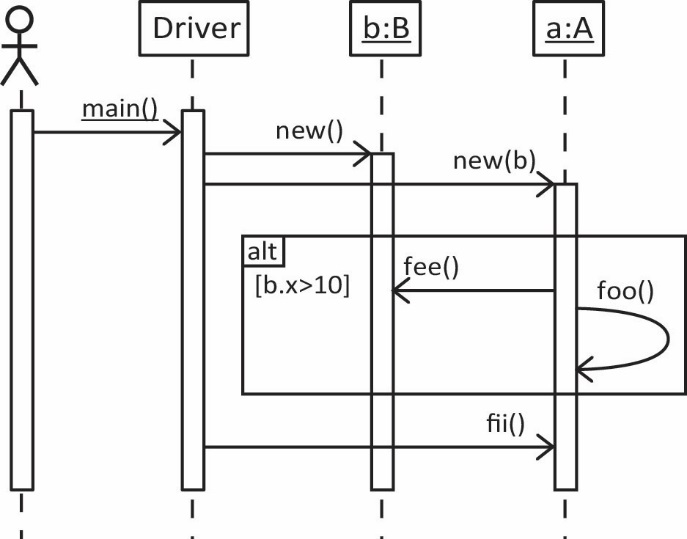
Draw a sequence diagram for the case when the system is run:



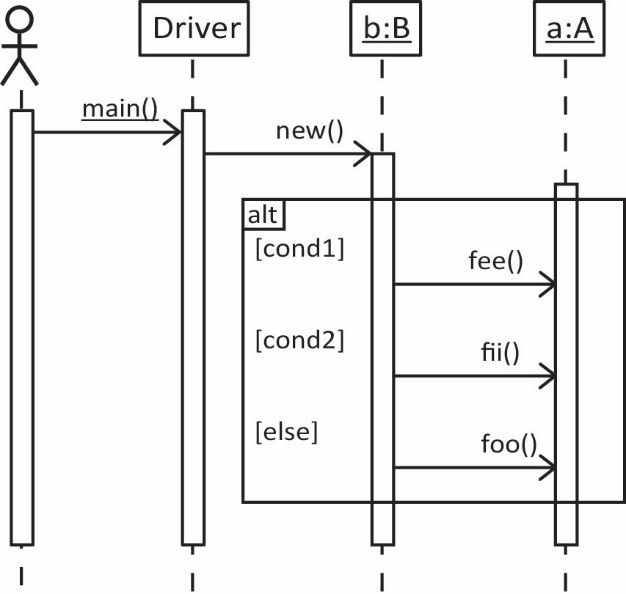
1. Conditional behavior in a sequence diagram is indicated by putting a box around the conditional code as shown in the example below. The condition for entering is shown in brackets. An [else] block can be added.
2. Example – Consider this code:

|  |  |
| --- | --- |
| **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() {}  } |

Draw a sequence diagram for the case when the system is run:



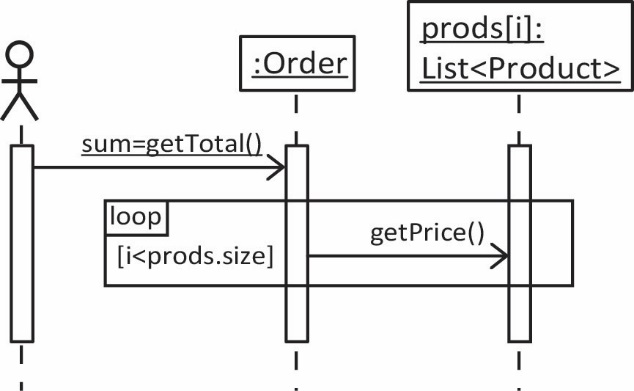
1. Example – Multi-part conditional



1. Iteration in a sequence diagram is indicated by placing a box around code to be repeated and specifying how the iteration occurs.
2. Example – Consider this code:

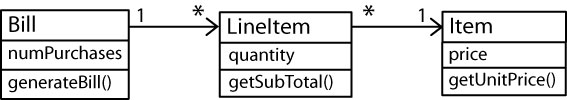
|  |  |
| --- | --- |
| **class** Product {  **double** price;  **public** **double** getPrice() {  **return** price;  }  **public** Product(**double** price) {  **this**.price = price;  }  } | **public** **class** Order {  List<Product> prods = **new** ArrayList<>();  **public** Order() {}  **public**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

Draw a sequence diagram for the case when the *getTotal* is called:



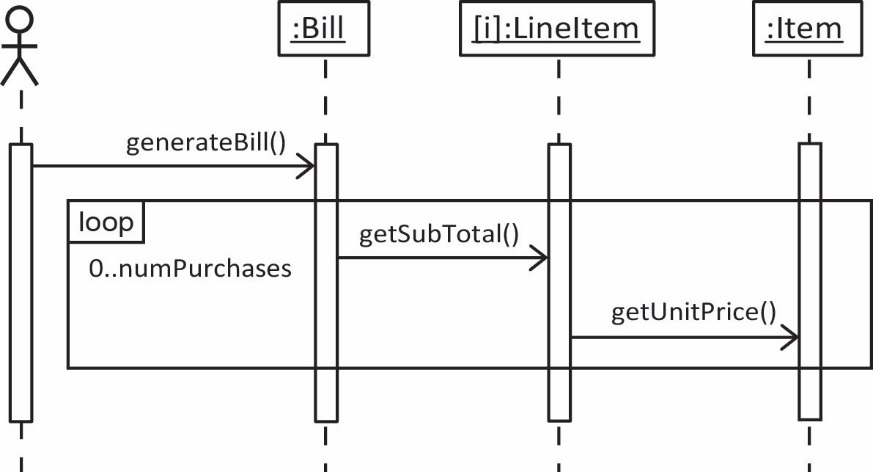
1. Example – Consider the use case and class diagram shown below to draw the corresponding sequence diagram.

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



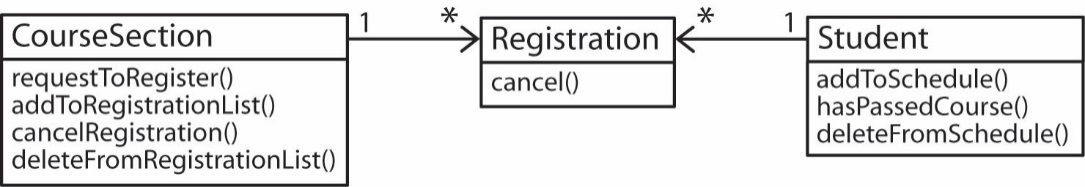
**Use case:** ObtainBill – Obtain the total bill.

**Sequence diagram:**



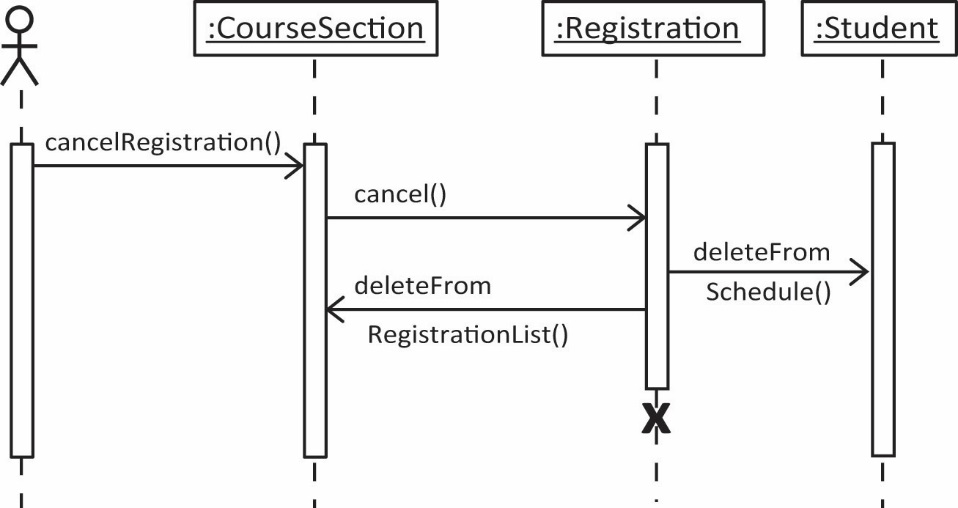
1. Object deletion in a sequence diagram is indicated by placing an “X” where the deletion occurs.
2. Example – Consider the use case and class diagram shown below to draw the corresponding sequence diagram.

**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:**



1. A sequence diagram can be ambiguous at times, without the indication of when a method finishes. Thus, we should add return arrows (dashed arrow with method name) when necessary. An example is shown below.
2. Example 7 – Consider the following code to draw the corresponding sequence diagram for the situation where we are starting the system.

**public class A {  
 public static void main(String[] args) {  
 B b = new B();  
 b.launch();  
 }  
}  
  
class B {  
 public B(){}  
  
 public void launch() {  
 System.out.println("B being launched");  
 }  
}**

|  |  |
| --- | --- |
| **Sequence Diagram for Starting System** | **Sequence Diagram with returns** |
| E:\Data-Classes\CS 4321 - Software Engineering 1\Topics\Ch08-UML 2\Fall12\pics\dd1.jpg | E:\Data-Classes\CS 4321 - Software Engineering 1\Topics\Ch08-UML 2\Fall12\pics\dd2.jpg |

**Blank Template for Sequence Diagram (next page)**



**Homework**

1. Example – Consider the following code to draw the corresponding (a) class diagram, (b) sequence diagram for the situation where we are starting the system. (Answer provided in Appendix below)

|  |  |
| --- | --- |
| public class A2 {  public static void main(String[] args) {  B b = new B();  b.launch();  } }  public class C {  public C() {  setup();  }    public void setup() {  System.out.println("C setup");  } } | class B {  C c;    public B() {  c = new C();   }   public void launch() {  System.out.println("B being launched");  init();  }    private void init() {  System.out.println("B being init'd");  } } |

1. Consider the classes shown below. (a) Draw a class diagram, (b) Draw a complete sequence diagram for starting the system.

|  |  |
| --- | --- |
| **public** **class** A {  C c;  D d;  **double** dim;  **public** **static** **void** vv(String[] args) {  A a = **new** A();  }  **public** A() {  d = **new** D();  F f = **new** F(d.getParam());  c = **new** C(**this**, f);  }  **public** **void** setDim(**double** x) {  dim = x+1;  }  **public** **double** getData(**int** y) {  F f = **new** F(y\*d.getParam());  **return** f.getCoord();  }  } | **public** **class** D {  **double** soln;  **public** D() {  setSolution();  }  **public** **void** setSolution() {  soln = 1.24523;  }  **public** **double** getParam() {  **return** soln/2;  }  } |
| **public** **class** C {  A a;  **double** x;  **public** C(A a, F f) {  **this**.a = a;  x = f.getCoord();  a.setDim(x);  }  **public** **void** updateModel( **int** y ) {  x = a.getData(y);  }  } | **public** **class** F {  **double** param;  **public** F(**double** p) {  **this**.param=p;  }  **public** **double** getCoord() {  **return** param;  }  } |

**Appendix – Answers for Problems**

|  |  |
| --- | --- |
| **Class Diagram:** | **Sequence Diagram for Starting the System** |
| **E:\Data-Classes\CS 4321 - Software Engineering 1\Topics\Ch08-UML 2\Fall12\pics\dd6.jpg** | **E:\Data-Classes\CS 4321 - Software Engineering 1\Topics\Ch08-UML 2\Fall12\pics\dd5.jpg** |

