UML
Unit 4

Dr. Magdi AMER

UML history

- The ***Unified Modeling Language*** was the outcome of the work of the pioneers of Object Oriented Design, Booch, Rumbaugh and Jacobson. A company Called Rational, which was lately bought by IBM, had the genius idea of employing these scientists, which were working previously independently, each developing his own tools and methodology.
- They work together, taking the best ideas from their previous work and coming up with a methodology for designing object oriented software. The result was the UML.
- The UML is composed of a set of diagrams; each diagram is used for explaining a part of the system. During this course, we are going to learn about the each type of diagram and learn how to use them.

Use Cases

- The first step in any software project is the analysis. Analysts are responsible for meeting with the client, determining the current processes and workflow that the client uses in his business. The client may be using a manual system where each request is submitted using a paper form and a file is being transferred from one department to another. The current system may also be a computerized one, which was partially developed or that became obsolete with time. In both cases the analyst should document the current system. The analyst should also document the current problems that the system is facing and suggest improvements.
- Use cases are written in plain English. They describe the result of one user action. The use case only describes events that are visible to the user. It does not describe, for example, the encoding of a file goes through before being saved.
- Let's take an example of a typical use case for a point of sale system. First think about the normal sequence of event. This will be **Use Case 1**:
  - 1. Cashier swipes product over scanner, scanner reads UPC code.
  - 2. Price and description of item, as well as current subtotal appear on the display facing the customer. The price and description also appear on the cashier's screen.
  - 3. Price and description are printed on receipt.
  - 4. System emits an audible "acknowledgement" tone to tell the cashier that the UPC code was correctly read.

Use Cases: Exceptional Path

- Now think about what may go wrong. First think, the reader may not read correctly. This will be **Use Case 2**:
  - 1- If the scanner fails to capture the UPC code, the system should emit the “re-swipe” tone telling the cashier to try again.
  - 2- The cashier tries to re-swipe again. If it reads correctly, then we continue with the Use Case 1, Step 2.
  - 3- If after three tries the scanner still does not capture the UPC code, the cashier enters it manually.

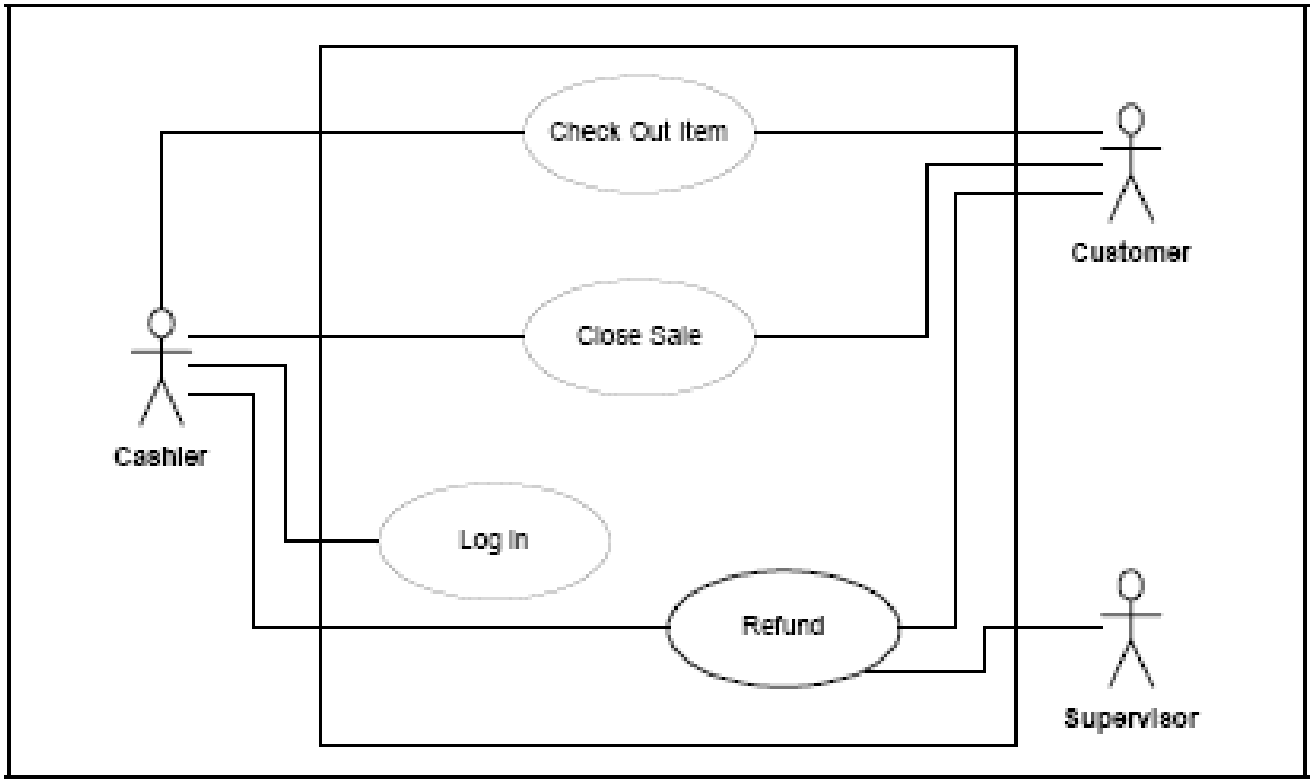
- The second un-normal sequence of events is the item does not have a UPC. This will be **Use Case 3**:
  - If the item does not have a UPC code, the cashier get the reference catalog and get the price of the item and enters it manually. We continue with the Use Case 1, Step 2.
  - If the item is not present in the catalog, then put it away the item and refuse to sell. The item should be reported to the administrator to enter it on the system or remove it from the inventory.

Use Cases: Exceptional Path

- Your job as analyst is to speak with the high level managers and get information about the main processes. Notice that generally managers will only know about the normal processes, and often without accurate information. The detail of the processes and the unusual use cases are only known to the actual employee that performs a specific task within the business process.

- Notice also that uncommon use cases will often lead to the discovery of uncommon processes. For example, in *Use Case 3* mentioned above, we discovered that there are two business processes that will be needed by the store administrator; one to add an item to the inventory, the second is to remove an item from the shelves.

Use Case Diagram



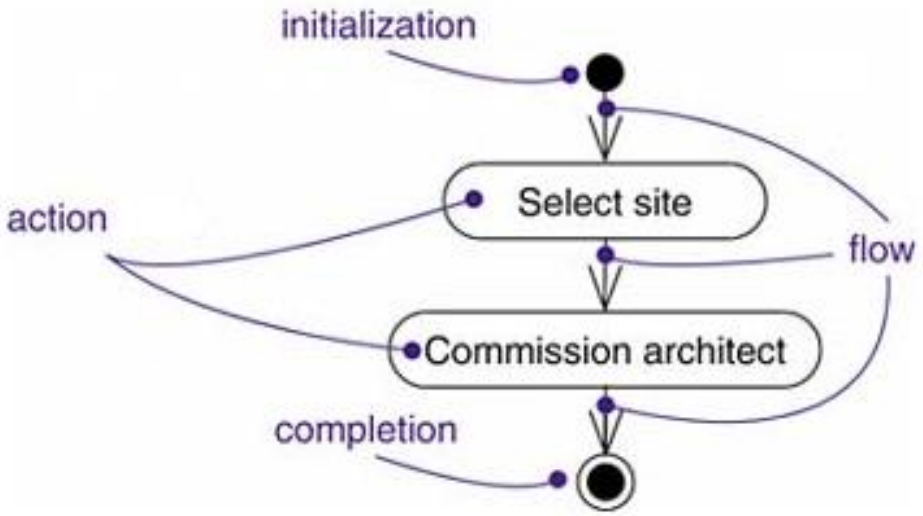
- Use Case Diagrams are used to draw the system boundaries. It shows the entity that may perform an action on the system. These entities may be either a person or another software system that interact with the system that we are building. Entities outside the system that perform actions on the system are called ***Actors***.

Activity Diagram

- Activity Diagrams are used as a visual way of explaining a use case. Remember that an image can be better than a thousand words. In fact, it is much easier to show a picture and explain the steps of the use case than to try to read the use case to other members of your team.
- **The Starting and Ending Points**
  - Any activity diagram must have at least one starting point and one ending point. A Starting point is denoted with , the Ending point is denoted by .
- **Actions**
  - Actions are specific steps in the diagram. It is denoted by a rounded rectangle containing the description of the activity, such as

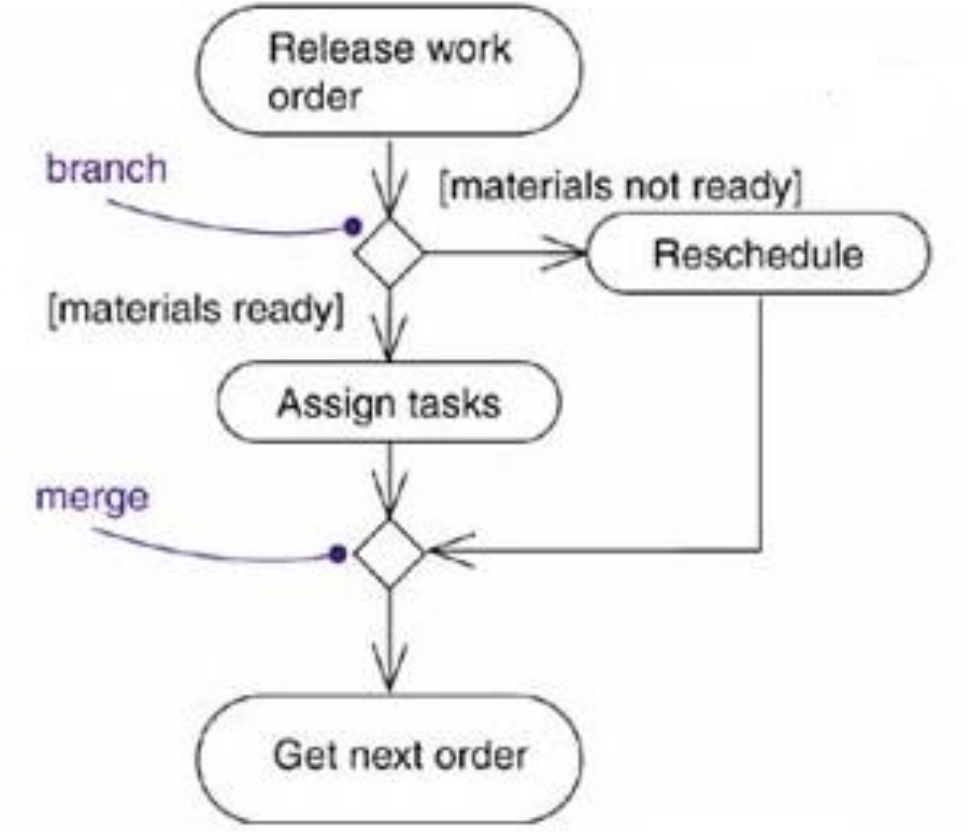
- **Action flow**

  It is shown by using arrows, from top to button, indicating the sequence of events and actions.



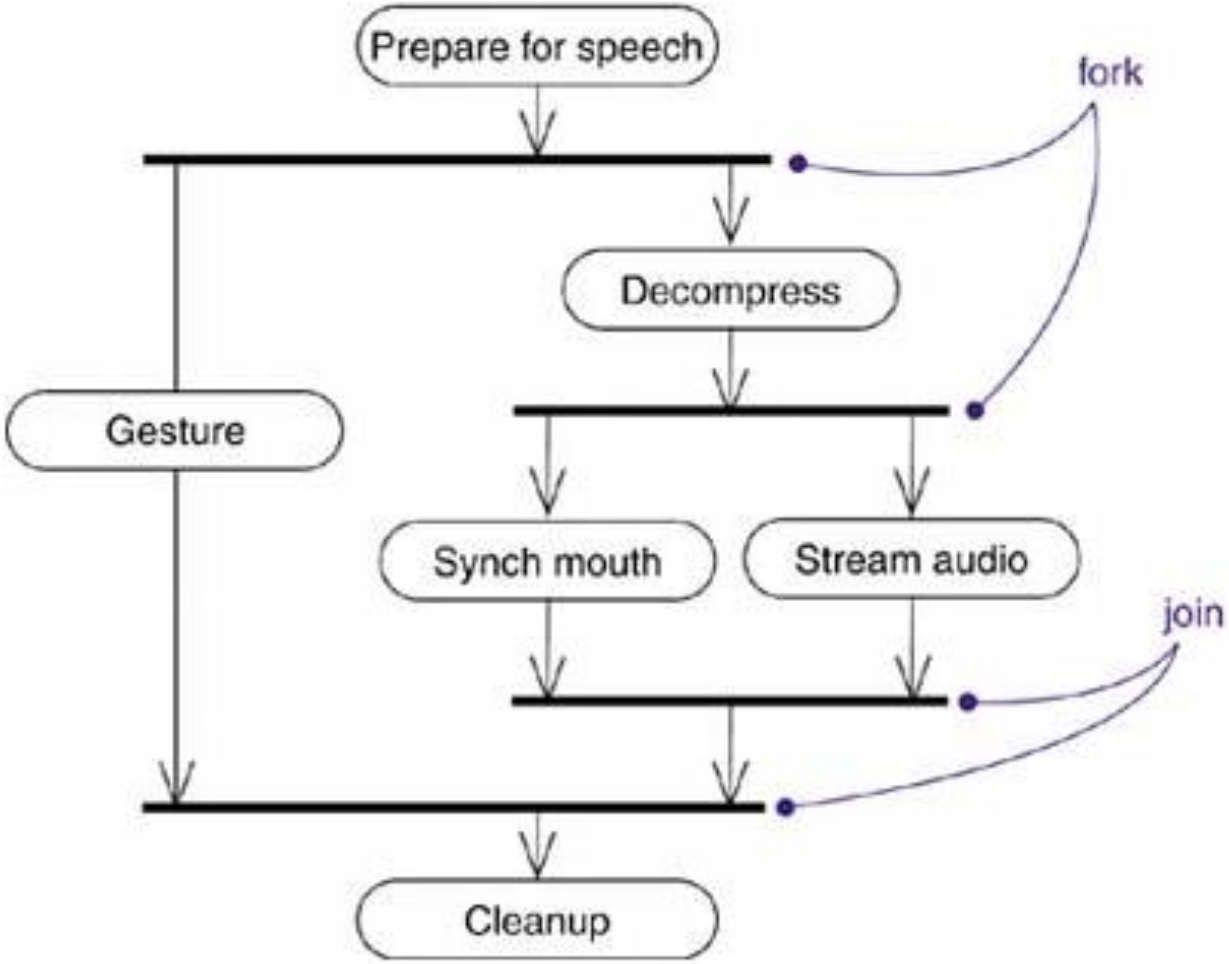
Activity Diagram

- **Branching and Merging**
  - Conditions are shown by a diamond shape. On each outgoing flow, a Boolean expression is placed that indicates that this path should be followed if the expression is true. Merge may occur if 2 branches would follow the same sequence of events after the steps that were related to the condition. It is shown by a diamond shape.



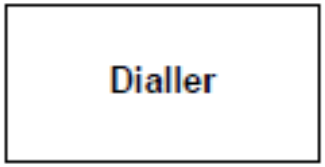
Activity Diagram

- **Forking and Joining**
  - Forking occurs when activities are done in parallel. Joining occurs when we have to wait for all activities before proceeding.

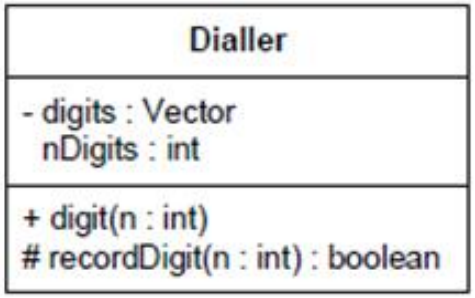


Class Diagram

- **Class**



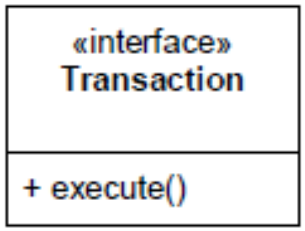

```
public class Dialler
{
}
```



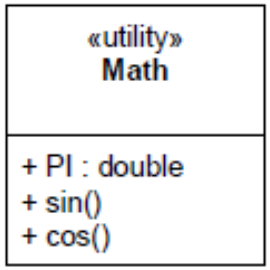

```
public class Dialler
{
  private Vector digits;
  int nDigits;
  public void digit(int n);
  protected boolean recordDigit(int n);
}
```

Class Diagram

- **Class Interface**




```
interface Transaction
{
  public void execute();
}
```



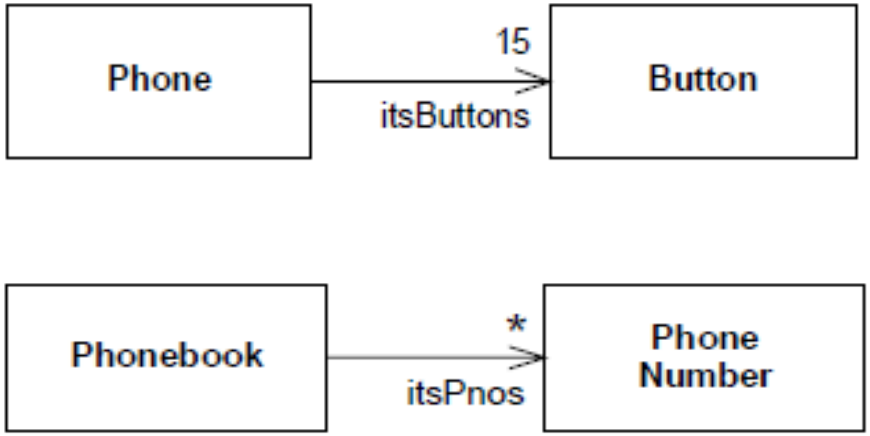

```
public class Math
{
  public static final double PI =
                              3.1415926535897932;

  public static double sin(double theta){...}
  public static double cos(double theta){...}
}
```

All the methods and variables of a «utility» class are static.

Class Diagram

- **Association**

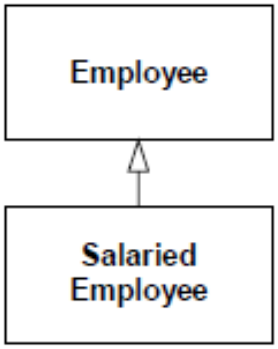



```
public class Phone
{
  private Button itsButtons[15];
}
```

```
public class Phonebook
{
  private Vector itsPnos;
}
```

Class Diagram

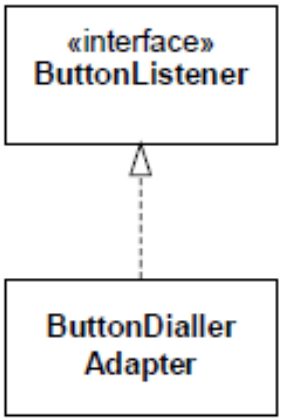
- **Inheritance**




```
public class Employee
{
   ...
}

public class SalariedEmployee extends Employee
{
   ...
}
```

- **Implementing an interface**

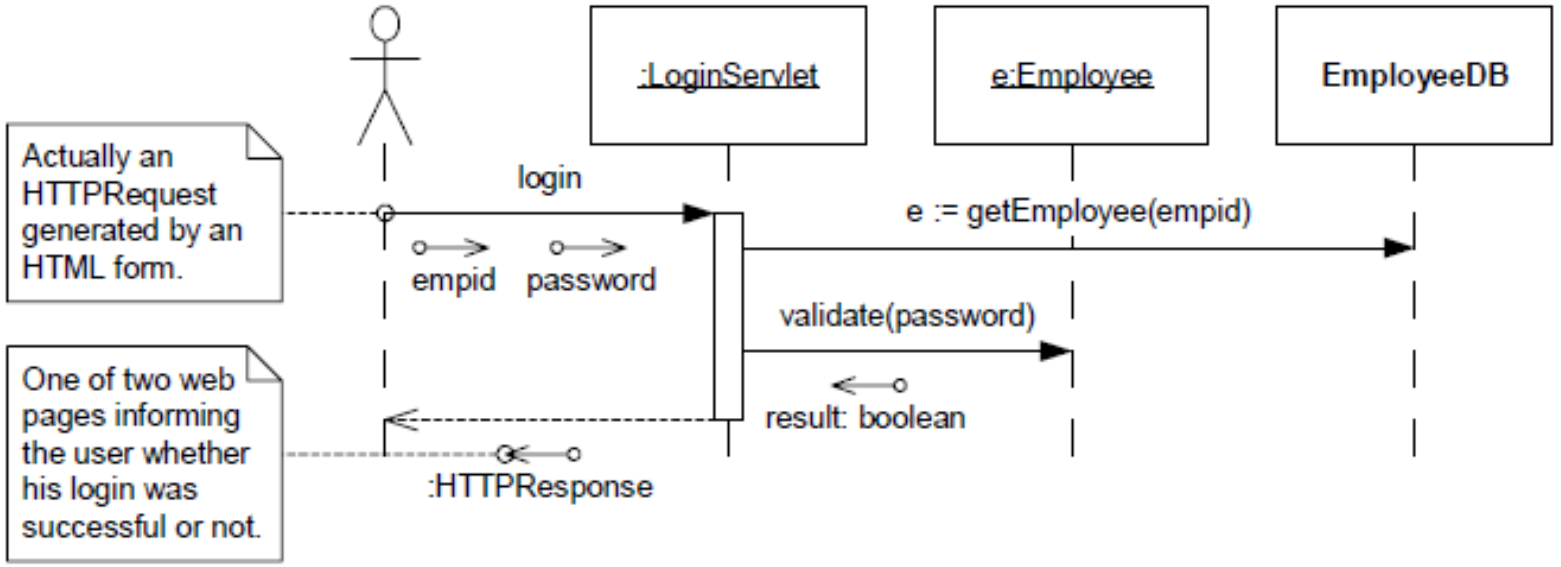



```
interface ButtonListener
{
   ...
}

public class ButtonDiallerAdapter
   implements ButtonListener
{
   ...
}
```

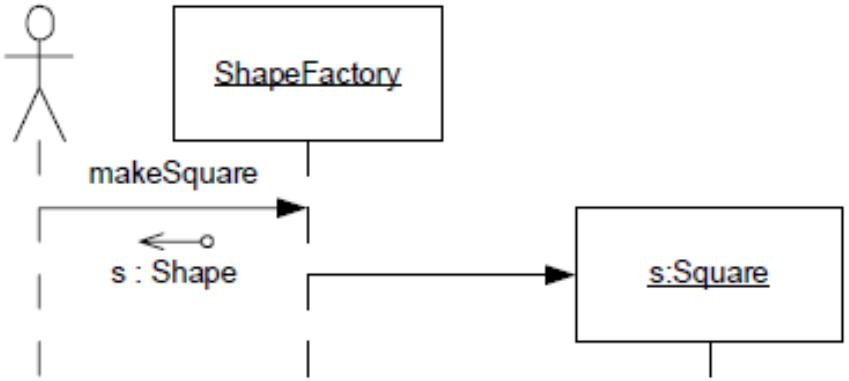
Sequence Diagram

•Sequence diagrams should be used to describe how a group of objects collaborate to perform an action.
•It is a tools that is occasionally used to describe complex interactions, rather than a required documentation.



Sequence Diagram

Creating an object

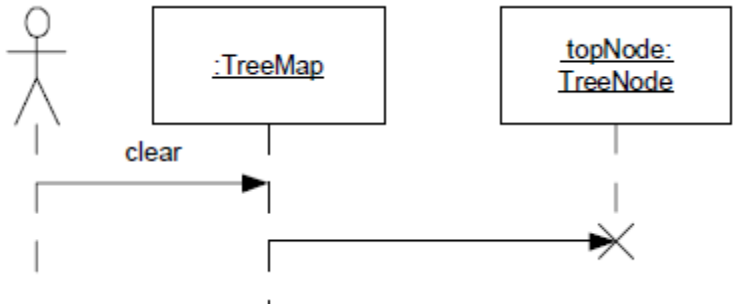



```
public class ShapeFactory
{
  public Shape makeSquare()
  {
    return new Square();
  }
}
```

Sequence Diagram

Destroying an object

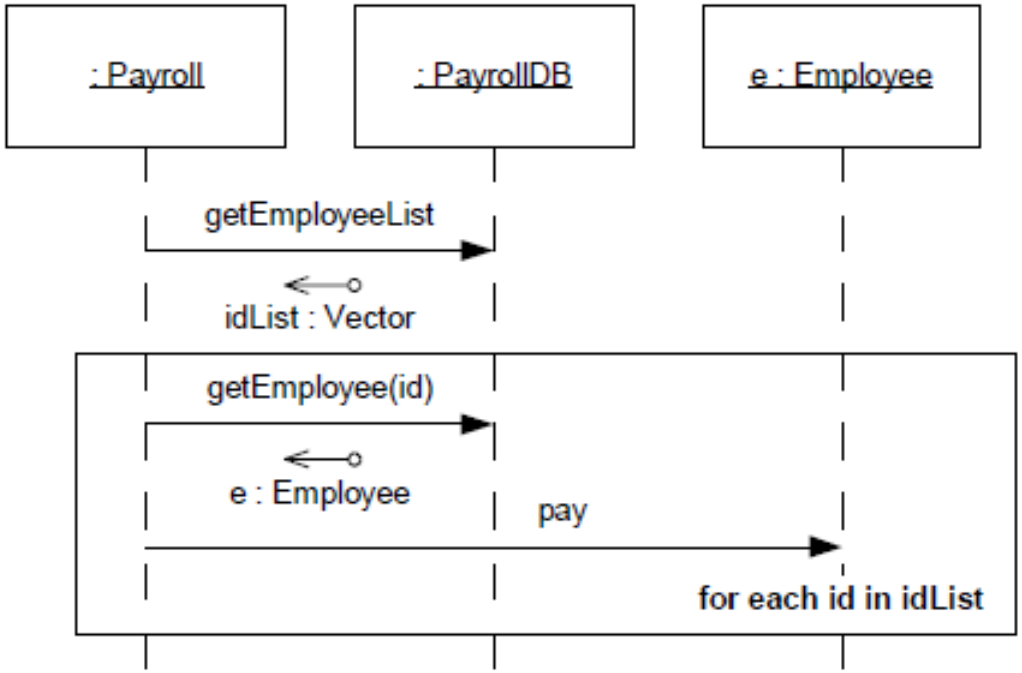
(Releasing an object to be garbage collected).




```
public class TreeMap
{
        private TreeNode topNode;
        public void clear()
        {
           topNode = null;
        }
}
```

Sequence Diagram

Loops



Sequence Diagram: Example