Association - Direction

- **Unidirectional** – One object can send messages to the other, but this other object cannot send messages back and has no knowledge about the object referring to it.
- **Bidirectional** – Both objects know about each other and messages can be passed in either direction.

Unidirectional Association - Example

- Account has exactly one Guarantor.
- Account has to access (invoke methods) on the Guarantor.
- The guarantor has no knowledge (cannot invoke methods) of the account.

Association – Example

- Unidirectional association from Account to DebitCard.
- Account has exactly zero or one DebitCard

Association – Implementation

```java
public class Account {
    private DebitCard theCard;
    public String getCardNumber() {
        return theCard.getNumber();
    }
    public void setCard(DebitCard card) {
        theCard = card;
    }
    public void removeCard() {
        theCard = null;
    }
}
```

Account sends a message to DebitCard

Association – Implementation

```java
public class DebitCard {
    private String theCardNumber;
    public String getNumber() {
        return theCardNumber;
    }
    // ...
}
```

Unidirectional association from Account to Guarantor

Account has exactly one Guarantor
Association – Implementation

**Unidirectional association from Manager to Account**

Manager can supervise a number of accounts but it is never necessary to know who is managing the accounts.

```
Manager -------- Account
```

**Bidirectional association from Account to Debit Card. Account can send messages to Debit Card and Debit Card can send messages to Account**

Account has exactly zero or one Debit Card

```
Account -------- DebitCard
```

**Implementation 1 provides the data members necessary to store bidirectional links. However creating the objects require multiple step and does not guarantee referential integrity.**

```
Account acc1 = new Account()
DebitCard card1 = new DebitCard(acc1)
acc1.setCard(card1)

Account acc2 = new Account()
DebitCard card2 = new DebitCard(acc2)
acc1.setCard(card2)
```
Association – Implementation

Implementation 2

```java
public class Account {
    private DebitCard theCard;
    public DebitCard getCard() { ... }
    public void setCard(DebitCard card) { ... }
    public void removeCard() { ... }
    public void addCard() {
        theCard = new DebitCard(this);
    }
}
```

```java
public class DebitCard {
    private Account theAccount;
    DebitCard(Account a) { theAccount = a; }
    public Account getAccount() { ... }
}
```

Implementation 2 ensures referential integrity.

Account acc1 = new Account();
acc1.addCard();

Class Relationships

- Association
- Generalization
- Realization
- Dependency

Generalization (Inheritance)

- Child class is a special case of the parent class

Generalization (Inheritance) e.g.

```java
public class Circle {
}
```

```java
public class GraphicCircle extends Circle {
}
```
Abstract Class

```
Shape
```

```
Circle
```

```
Rectangle
```

Abstract Methods (Operations)

```
Shape
draw()
```

```
Circle
draw()
```

```
Rectangle
draw()
```

Notes

- If the class is defined as Abstract no objects of the class can be instantiated.
- If at least one method in a class is defined as abstract then the class is abstract.
- In UML an abstract class and operations are denoted in italics.

Abstract class and method implementation

```
public abstract class Shape {
    public abstract draw(); // declare without implementation
    .......
}
```

```
public class Circle {
    public draw(){
        ....
    }
    ....
}
```

Class Relationships

- Association
- Generalization
- Realization
- Dependency

Realization - Interface

- Interface is a set of operation the class carries out
Realization - Implementation

```java
public interface TypeWriter {
    void keyStroked();
}
public class KeyBoard implements TypeWriter {
    public void keyStroked()
    {
        //...
    }
}
```

Class Relationships

- Association
- Generalization
- Realization
- Dependency

Dependency

- Change in specification of one class can change the other class. This can happen when one class is using another class.

```
Circle
Move(p:Point)
Point
```

Dependency cont

- Dependency relationship can be used to show relationships between classes and objects.

```
Circle
< instance Of circleA:Circle
< instance Of circleB:Circle
```

Class Diagrams

- The UML class diagram consists of several Classes, connected with Relationships.

Class Diagram - Example

- Draw a class diagram for a information modeling system for a school.
- The classes are School, Department, Student, Course, Instructor.
Class Diagram Example

- School
- Department
- Student
- Course
- Instructor

School has 1..* Department
Department offeredBy 1
Student attends * 1
Course teaches 1..*
Instructor assignedTo 1

Object Diagram

- Object Diagram shows the relationship between objects.
- Unlike classes objects have a state.

Object Diagram - Example

- c1: Company
- d1: Department
  - name= "Sales"
  - manager
  - employee
    - p1: Person
      - name= "John"
    - p2: Person
      - name= "David"
- d2: Department
  - name= "R&D"