UML Part VI

» Implementing UML relationships in code (C++)
Relationships Among Classes

- Dependency
- Association
- Composition & Aggregation
- Generalization
Association

- Semantic dependency between classes without direction
- Cardinality
  - one to one
  - one to many
  - many to many
Association example
Association example

class Product {
public:
private:
    Sale *lastSale;
};

class Sale {
public:
private:
    Product **productSold;
};

• Each instance of Product has a pointer to its last sale
• Each instance of Sale has a collection of pointers denoting the products sold
Composition example

mailbox \rightarrow messageQ
Composition

• A part of relationship (physical containment) when object is destroyed, so is attribute

```cpp
class mailbox {
public:
    mailbox();
    ~mailbox();
    message getMessage(const InputReader&);

private:
    messageQ lst;
};
```
Aggregation example
Aggregation

- A part of relationship - when container destroyed, attribute is not.

```cpp
class company {
public:
  company();
  ~company();
  int numberOfEmployees() const;

private:
  employee *lst;
};
```
Dependency

- Peer to peer link
- Directional client/server relationship
- Refinement of association
Dependency

- A using relationship

```cpp
class mailbox {
public:
    mailbox();
    ~mailbox();
    message getMessage(const InputReader&);

private:
    messageQ lst;
};
```
Generalization

• One class shares the structure/behavior of one (single inheritance) or more (multiple inheritance) classes
• Subclass typically augments or restricts the existing structure and behavior of the superclass
Single Inheritance

Telemetry Data

Telemetry Data

Telemetry Data

Telemetry Data

Telemetry Data

Telemetry Data
Single Inheritance

class TelemetryData {
public:
    TelemetryData();
    virtual ~TelemetryData();
    virtual void transmit();
    Time currentTime() const;
private:
    int id;
    Time timeStamp;
};

class ElectricalData : public TelemetryData {
public:
    ElectricalData(float v1, float v2,
                   float v1, float v2);
    virtual ElectricalData();
    virtual void transmit();
    float currentPower() const;
private:
    float fuelCell1Voltage, fuelCell2Voltage;
    float fuelCell1Amperes, fuelCell2Amperes;
};
Single Inheritance

```cpp
void TelemetryData::transmit() { 
    // Transmit the id
    // Transmit the timeStamp
}

void ElectricalData::transmit() { 
    // Transmit the voltages
    // Transmit the amperes
}

void transmitFreshData(TelemetryData& d, const Time& t){
    if (d.currentTime() >= t) d.transmit();
}

TelemetryData telemetry;
ElectricalData electrical(5.0, -5.0, 3.0, 7.0);

transmitFreshData(telemetry, Time(60));
transmitFreshData(electrical, Time(60));
```
Multiple Inheritance

- Bank Account
  - Saving Account
  - Checking Account
- Insurable Item
- Asset
  - Real Estate
  - Stock
- Interest Bearing
- Security
  - Bond
Multiple Inheritance

class Asset;
class InsurableItem;
class InterestBearing;

class BankAccount : public Asset,
                public InsurableItem,
                public InterestBearing {};

class RealEstate : public Asset,
                    public InsurableItem {};

class Security : public Asset,
                 public InterestBearing {};

class SavingsAccount : public BankAccount {};
class CheckingAccount : public BankAccount {};

class Stock : public Security {};
class Bond : public Security {};