IF2261 Software Engineering

OOSE - Construction

Program Studi Teknik Informatika
STEI ITB

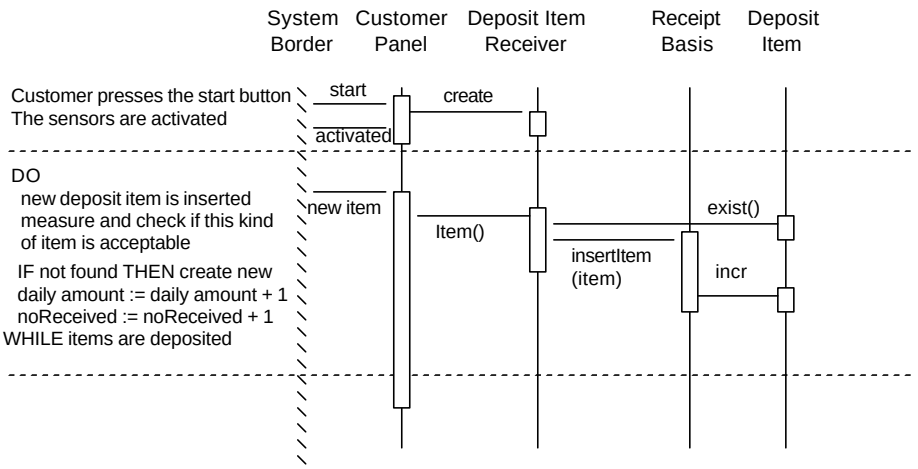
Block Design

- Through the design of the use cases, we now have a complete description of all external requirements of the block
  - From these requirements we can now design the block
- The actual implementation of the blocks in the source code can start when the block interfaces start to stabilize and are frozen
  - Normally, ancestor block should be implemented prior to descendants blocks

Block Design

- The Block Interfaces
  - From the interaction diagram, extracting all the operations defined for that block
    - when use case design involves several people, these operations are usually not collected in one document/diagram
    - can be done by using a CASE tool
  - By going through all interaction diagrams, we will get the complete interface to a block
  - By freezing the interface of the blocks, we can start block design activities in parallel

Example – Interaction Diagram for Use Case Returning Item

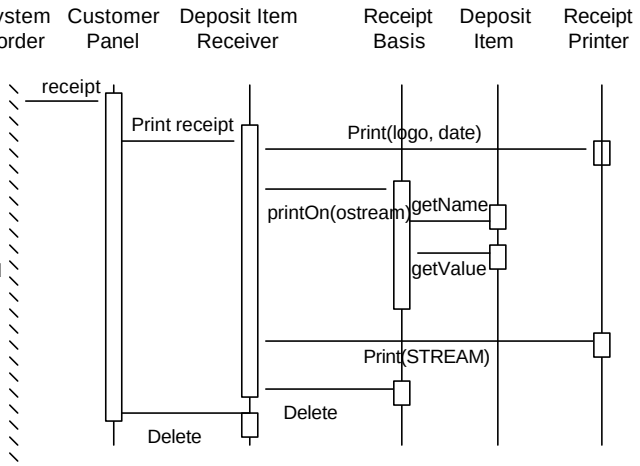


Example – Interaction Diagram for Use Case Returning Item

Customer presses the receipt button

Print logotype and date

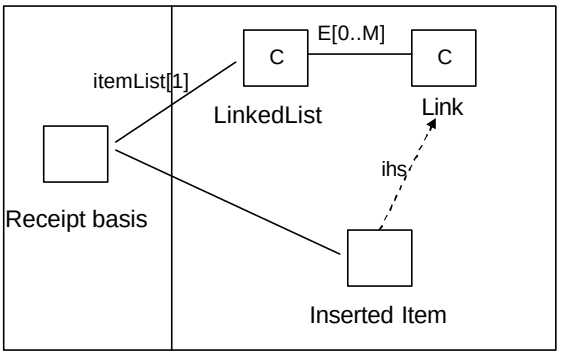
Print info on STREAM
FOR all basises DO
  find name on this type
  find deposit value on this type
  totalValue := depositValue*noReceived
  Print name, depositValue, totalValue
    on the STREAM
  sum := sum + totalValue
ENDFOR
Print sum on STREAM
Delete receipt
Delete deposit item receiver
Ready to net customer



Example - The Block Interface

- Interfaces for Deposit Item
  - Exists()
  - Incr
  - getName
  - getValue
- Interfaces for Receipt Basis
  - insertItem(item)
  - printOn(ostream)
  - delete

Example - The Block Interface – Receipt Basis



Example - The Block Interface – Receipt Basis

Class ReceiptBasis;
Operations:
  create;
  insertItem(item);
  printOn(ostream);
  delete;
Attributes:
  itemList: **listOf** ReturnedItem;
  sum: ECU;
endClass

Example - The Block Interface – Receipt Basis

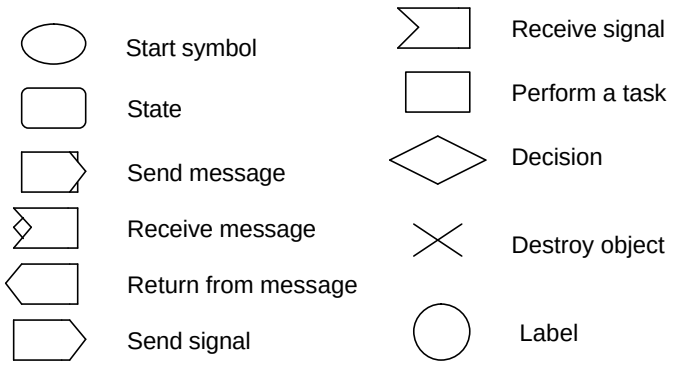
```
// File ReceiptBasis.h

#include “linkedlist.h”
#include “stream.h”

// private classes (encasulated to the block)
Class InsertedItem : public Link {
  private
    DepositItem  *returnedItem;
    int                totalNumber;
  public
    insertedItem(DepositItem *di);
    DepositItem *getItem();
    void  incr();
    int  getTotal();
    ~InsertedItem();
};
```

```
// public classes

Class ReceiptBasis {
//Attributes
  private:
    LinkedList *itemList;
    float  sum;

//Operations
public:
  ReceiptBasis();
  void insertItem(DepositItem*);
  void printOn(ostream&);
  ~ReceiptBasis();
}
```

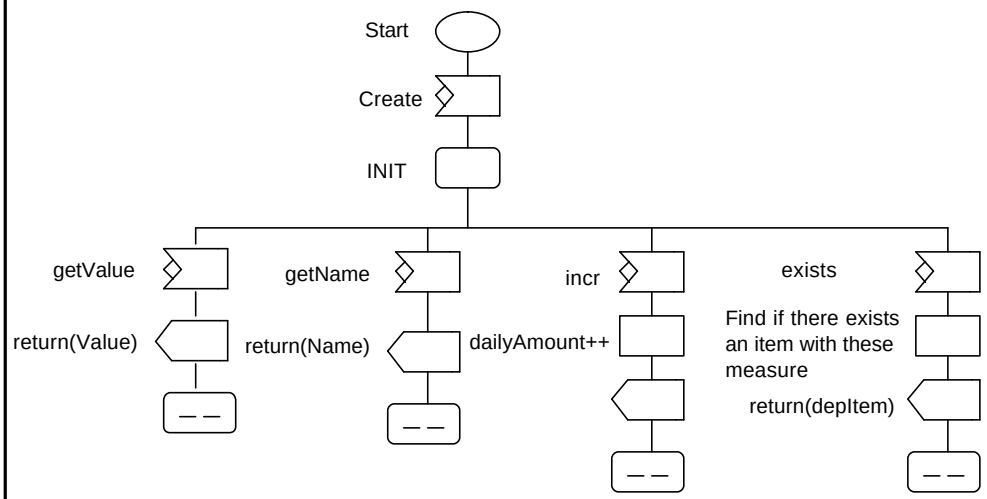

Object Behavior

- As an intermediate level for the internals of the object, use state transition graphs
- The purpose is
  - to provide a simplified description that increases understanding of the block without having to go down to source code level
  - to provide a description that is less dependent on the selected programming language
- The graph describes which stimuli can be received and what will happen when a stimulus is received

State Transition Graph - Notation



Example – STD – Deposit Item



The internal block structure

- will consist of object modules
  - classes in OO language
  - module units in procedural language

Defining the classes in the system

- Design reusable and high-quality classes
- Homogenized the classes
  - We don't want two classes offering similar functionalty, unless they are related through inheritance
- Consider to split one class into two classes
  - If half of the operations access half of the instance variables
- Use another heuristics of good class design
  - Such as, include judgement of the potential reusability value of the class (will the change make the class more reusable?)
  - A common estimation: 5-10 times longer to design a component class than an ordinary class

Implementation Model

- Implement STD in the programming language
  - There are tools for that translation
  - Normally, human beings is still needed to make final transition to source code
- Maintain traceability from the analysis model to the source code
- The specialization to the programming language describes how to translate the terms used in the design into the terms and properties in the language

Traceability for C++

Analisis	Design	C++
Analysis object	Block	1..N classes
Behavior in object	Operation	Member function
Attribute (class)	Attribute (class)	Static variable
Attribute (instance)	Attribute (instance)	Instance variable
Acquaintance assc.	Acquaintance assc.	Instance variable
Communication assc.	Communication assc.	Reference to a function
Interaction between object	Stimulus	Call to a function
Use case	Designed Use case	Sequence of calls
subsystem	subsystem	File

Traceability for Ada

Analisis	Design	C++
Analysis object	Block	Package
Behavior in object	Operation	Procedure or task
Attribute (class)	Attribute (class)	Variable (global)
Attribute (instance)	Attribute (instance)	Variable (private)
Acquaintance assc.	Acquaintance assc.	Variable reference
Communication assc.	Communication assc.	Existence of procedure call
Interaction between object	Stimulus	Procedure call or entry call
Use case	Designed Use case	Sequence of calls
subsystem	subsystem	Package

Implementation of probe

- The probe must be implemented in the block where the sequence should be inserted
  - Thus we must add one variable to hold a reference to the inserted functionality

Example – implementation of probe

```
class CustomerPanel {
public:
  . . .
  void stuck();
  void reset();
  . . .
private:
  . . .
  alarmist myAlarm;
  . . .
}

Void CustomerPanel::stuck()
{
  myAlarm.alarm;
}
  . . .
```


Working with the construction

- Specialization
  - Adapting the design to he relevant implementation environment
- Existing product
  - A common requirement is that the system must use existing product
  - The cases:
    - The exisiting product is the same system but in an older version
    - We build a new system that only makes use of existing product because it already exists
  - An assesment must be made:
    - Whether you want to use an existing product, and thus are forced to adapt your design to it
    - Whether you will not use the existing product but decide to develop this functionality yourself
  - A common problem:
    - You want to develop a system that has not been designed according to an OO method
      - Involves great adaptations, difficult, and often requires much re-engineering
  - Consider the testing cost

Working with the construction

- Abstractions
  - Requirement to handle the system complexity
    - Issue for abstractions on different levels
  - Use subsystem
    - Service package: the lowet level of the subsystem
    - Subsystem is indivisible
      - Delivered in their entirety or not at all
- Development is incremental
  - Many iterations must be done
  - Start construction early (focus: to identify implementation environment)

Working with the construction

- Further issues which must be considered in real project:
  - Documentation rules
  - Process issues, such as interactions between different parts
  - Other activities, such as configuration management and review