OO Terms
- Class- A structure that encapsulates the data and the procedural abstractions required to describe the content and behavior of some real world entity
  - Through encapsulation the class is decoupled from the rest of the system
- Data Abstractions – attributes
- Procedural Abstractions – operations, methods or services
  - Algorithms that process the data
- Objects interact through messages
- Encapsulation
  - Information hiding
  - Component reuse
  - Interfaces are simplified
    - Less parameters passed
    - Coupling is reduced
- Inheritance reuse accomplished directly
  - Subclass inherits all data and operations with no further work
  - Changes are made to base class they are immediately propagated through the system
  - At each level new attributes and operations can be added
- Refactoring
  - Change the internal structure of a program without changing its functionality
  - Used to combat software entropy
    - Law of increasing complexity, law of decreasing quality, etc
  - Separate refactoring work from adding functionality
  - Refactor when old code “gets in the way” or can’t understand code
  - Thoroughly test refactored code

Developing a class
- From scratch
- Inherit from an existing class
  - Possibly adding features
- Class hierarchy can be restructured
- Override existing operations
  - Polymorphism

Advantages of OO Developed Systems
• Lead to reuse
  • Leads to faster developed programs with higher quality
    • More used, reused the better its tested and understood
• Easier to maintain
  • Structure is inherently decoupled
    • Leads to fewer side effects when changed
• Easier to adapt and scale
  • Assembled of reusable parts (subsystems)

**OO Lifecycle Model**

**Object Oriented Life Cycle**

![Object Oriented Life Cycle Diagram]

**UML**

• 5 views of the system
• User Model View
  o Users perspective
  o Use–Case diagram and scenarios
• Structural model view
• Classes, objects and relationships
  • Behavioral Model View
    o Dynamic or behavioral aspects
    o Interaction or collaboration between structural elements
  • Implementation View
    o How they are to be built
  • Environmental View
    o User Interface

Process
  • Use Cases
  • Determine classes, responsibilities, and collaborators – CRC Cards
  • Define structures and hierarchies
    o Generalization/specialization
    o Composite, aggregate
  • Define subjects and subsystems
    o A group of classes that collaborate among themselves to accomplish a set of cohesive tasks
    o Provide a contract of the responsibilities that can be expected of them
  • Define object – relationship model
    o Class diagram – cardinality
    o To derive
      • Using CRC cards determine the network of collaborators
      • Give name and direction to the network
      • Determine cardinality
  • Define behavior model
    o Model the dynamic behavior
    o Go through the Use Cases documenting events where information has been exchanged
    o Build State transition diagram

**OOA**    ->    **OOD**
CRC          responsibilities
Use Cases    subsystem
Object –relationship    message design
Object-behavioral    subsystem
System Design Activities

- Partition the analysis model into subsystems which share something in common
  - Classes which only collaborate with each other
  - All accomplish same function
  - All are on same piece of hardware
  - Manage a set of resources
- Concurrency and Subsystem Allocation
  - Concurrent only if classes are active at the same time
  - Either apply them to separate processors or a single processor using concurrency
- User Interface
  - Use cases
  - Subsystems to determine menus
- Data Management Component
  - Design attributes and management operations
  - Determine file/database system
- Inter-subsystem Collaboration

OOA – Static Modeling

- Determine classes, attributes and methods (operations/responsibilities)
- Form basis of OO paradigm
  - Use Cases
  - Class network
- Three types of classes
  - Entity class: models information that is long lived (persistent when the system is idle)
  - Boundary Class: models the interaction between the software product and its actors (UI – both input and output)
  - Control Class: models complex computations and algorithms
### Abbots Heuristic’s for mapping parts of speech to model components

<table>
<thead>
<tr>
<th>Part of Speech</th>
<th>Model Component</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proper Noun</td>
<td>Instance</td>
<td>Jane</td>
</tr>
<tr>
<td>Common Noun</td>
<td>Class</td>
<td>Passenger</td>
</tr>
<tr>
<td>Doing Verb</td>
<td>Operation</td>
<td>Creates, Submits, Selects</td>
</tr>
<tr>
<td>Being Verb</td>
<td>Inheritance</td>
<td>Is a kind of, is one of either</td>
</tr>
<tr>
<td>Having Verb</td>
<td>Aggregation</td>
<td>Has, Consists, includes</td>
</tr>
<tr>
<td>Modal Verb</td>
<td>Constraints</td>
<td>Must be</td>
</tr>
<tr>
<td>Adjective</td>
<td>Attribute</td>
<td>Address</td>
</tr>
</tbody>
</table>

**Extracting Classes**
- Functional modeling – Present scenarios for all of the use cases
- Entity Class modeling – determine the entity classes and their attributes
- Dynamic modeling – determine the operations performed by each entity class
- Done iteratively and incrementally

**Identifying Entity Classes**
- Grammatical parse on use-case text or processing narratives
- Nouns or noun clauses
  - Look for
  - External entities
    - Produce or consume information
  - Things
    - Reports or displays, etc.
  - Occurrences or events
  - Roles
  - Places – establish context
  - Structures
- Coad/Yourdon selection criteria
  - Satisfy almost all of these
  - Retained info
  - Needed service
  - Multiple attributes
  - Common attributes
  - Common operations
- Essential requirements

- Rumbaugh – reasons to reject
  - Vague
  - Attribute
  - Redundant
  - Irrelevant
  - Implementation

- Specify Attributes -- adjectives
- Define Operations
  - Manipulate
  - Perform calculation
  - Inquire about state
  - Monitor for a controlling event
  - Verbs in the grammatical parse
- Messages – Communication between classes

**Car Rental System – Example Zipcars**

Software is to be developed to automate the reservation and invoicing of a car rental company. The major requirements are:

(i). Vehicles are taken from one location and returned to the same location.
(ii). Different models of cars are grouped into a small number of price classes.
(iii). Different rental plans are available, with a special weekend rate to attract non-business customers.
(iv). The price charged is established in advance.
(v). Free options are: automatic or manual transmission, two or four doors, smoker or non-smoker cars.
(vi). Non-fitted extras are: roof rack, trailer, snow chains, child seats. These extras are charged to the customer.
(vii). The system must handle block booking of cars and keep track of car availability.
Identify Entity Classes and Objects -- Noun Parse

• client /customer
• rental plan
• price classes
• roof rack
• Trailer
• Snow chains
• Child seats
• vehicle/car
• models
• auto. transmission
• Location
• rate

Final Entity Classes after Refinement

• client -- car renter -- individual or corporate
• contract -- rental terms and conditions
• rental -- rental information filled out when taking out and returning a vehicle
• vehicle -- automobile in the fleet
• extras -- non-permanent options
• rate -- pricing conditions

Identify Attributes

Client:
- account number
- name
- address
- phone
- type of account
- drivers license #
- Balance

Rental:
- Contract id
- account number
- applicable rate
- license plate
- starting and ending mileage
- list of authorized drivers
- tag(s) of extras
- insurance

Contract:
- Contract id
- account number
- payment type
- credit card account
- options chosen
- departure and return dates
- amount

Vehicle:
- model of car
- license plate
- location
- availability
- mileage
- available features
- rate category

Extras
- tag #
- type
- availability
- rate category

Rate:
- rate category
- rate amount
Associations and Dependencies

• Relationships between classes

  – Multiplicity – the number of possible instances of the class associated with a single instance of the other class.

<table>
<thead>
<tr>
<th>Multiplication</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0..1</td>
<td>Zero or 1 instance (n..m)</td>
</tr>
<tr>
<td>0..* or *</td>
<td>Zero or many</td>
</tr>
<tr>
<td>1</td>
<td>Exactly 1 instance</td>
</tr>
<tr>
<td>1..*</td>
<td>At least one instance</td>
</tr>
</tbody>
</table>

• Association – binary relationship between two entity classes. There is an association if an instance in one class must know about the other to do its work.

  – Generalization – denotes inheritance with one class being the super class of the other, denoted by a triangle pointing to the super class
- Aggregation – one class belongs to a collection, denoted by a diamond pointing to the class containing the whole

- Composition – a stronger association in which the whole is made up of the parts, the parts cannot exist without the whole, denoted by a filled diamond at the whole
Class Diagram for the Car Rental example
Class Diagram for BankApp
Class Diagram for a media collection library
CRC Cards
- Class-responsibility-collaborator
- Index cards
- Top class name
- Left responsibilities
- Right collaborators
  - Can the class fulfill the operation on its own or does it need help from another class

Review CRC model
- Representatives of customer and developer
- Each participant given a subset of cards such that a single representative does not have any collaborators
- Read use case
  - When come to a class see if it can fulfill the task
  - Call on collaborators as needed
- Modify the model as needed to ensure all use cases can be carried out

CRC Model -- Client
- Responsibilities
  - add a client
  - return account #
  - delete client
  - change/return client info
    - address, phone, name, account type, drivers license #
  - update/return balance

CRC Model -- Contract
- Responsibilities
  - invoice client

  - Return contract id
  - Return contract info

  Collaborators
  - client
  - vehicle
  - rental
  - Extras
**CRC Model -- Vehicle**

- **Responsibilities**
  - check mileage
  - check availability
  - add vehicle
  - remove vehicle
  - rent
  - return
  - change/return license plate, location, rate category

**CRC Model -- Rental**

- **Responsibilities**
  - create rental
  - select vehicle
  - select extras
  - edit rental
  - rent
  - return
  - return rental info

**CRC Model -- Extras**

- **Responsibilities**
  - check availability
  - return tag #
  - return/change
    - rate category, type
  - rent
  - return
  - add extra
  - remove extra

**CRC Model -- Rate**

- **Responsibilities**
  - add rate category
  - change rate amount
  - return rate amount
**Analysis Packages**
- Categorize the classes into units of corresponding nature
- Package diagram
- Object Diagram
- Component Diagram
- Deployment Diagram

**Static Aspects**
- Define classes, their relationships and their behaviors
- Define class
  - Attributes
  - Methods (behaviors/operations/responsibilities)
- Define class hierarchy

**Dynamic Modeling**
- Use case diagram
- Interaction Diagrams
  - Sequence diagram
  - Collaboration Diagram
- Statecharts
- Activity Diagrams
  - Swimlane Diagrams

**Interaction Diagrams**
- Develop use-case(s)/scenarios and use-case diagram
- Describe how objects collaborate
  - Classes and actors from the use case and class diagrams participate to model the interactions to carry out a use case
- Two types– sequence and collaboration
  - Sequence – detail how operations are carried out
    - What messages are sent and when
    - Timing is important
  - Collaboration – detail how operations are carried out
    - Focus on objects roles and do not include timing
Rental Car Agency Use Case

- Reserve a Car
- rent A car
- Return A Car
- «uses» Manage Customers
- creditCardCompany
- Manage Fleet
- RentalCarAgency
Heuristics for drawing Sequence Diagrams

- First column should correspond to the actor who initiates the use case
- The second column should be a boundary objects that the actor uses to initiate the use case
- The third column should be the control object that manages the rest of the use case
- Control objects are created by boundary objects initiating use cases
- Entity objects are accessed by control and boundary objects
- Entity objects never access boundary or control objects, this makes it easier to share entity objects across use cases

Sequence Diagram for Renting a Car
Sequence Diagram for Ordering a Product over Internet
Sequence Diagram for
Open a Savings Account
UML Activity diagram
- Also supplements use-cases
- Provides graphical representation of the flow of the interaction
- Symbols
  - Rounded rectangles – system function
  - Directed arcs – flow through the system
  - Diamond – branching decision
  - Horizontal lines – parallel activities
- Adds more information by including constraints explicitly
enter password and user id

Valid user id/password

terminate session
choose drop classes
choose add classes
return to previous menu

Invalid user Id/password

Prompt for reentry

Tries remain

enter course number

view course data
display course does not exist

drop course

UML Activity Diagram
Register for a Class
UML Swimlane Diagram
- Adds actor responsibility to activity diagram when multiple actors are involved in a use case
- Partition the diagram into areas for each actor involved and place their functionality in their partition
UML Swimlane Diagram
Register for a Class