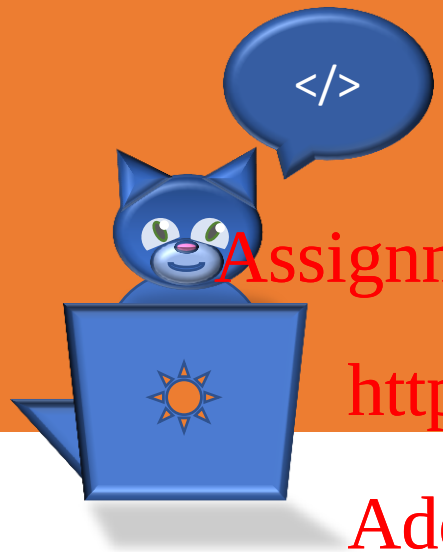
Please note that the slides published AFTER the lectures and workshops are the official slides and are the ones that should be used for revision.



COMP2013
Software Maintenance

Lecture 03

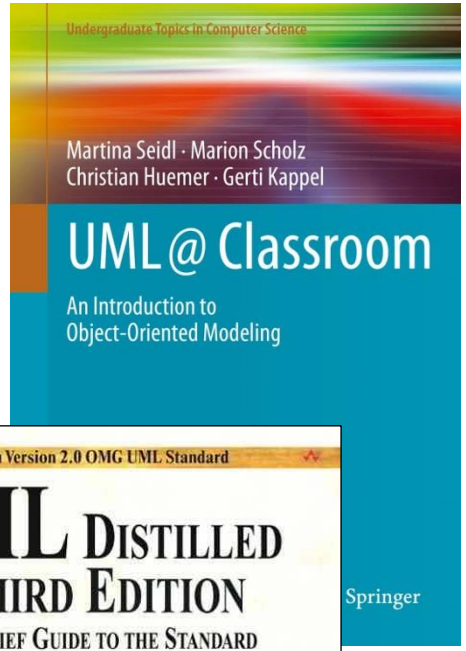
OO Analysis/Design with UML

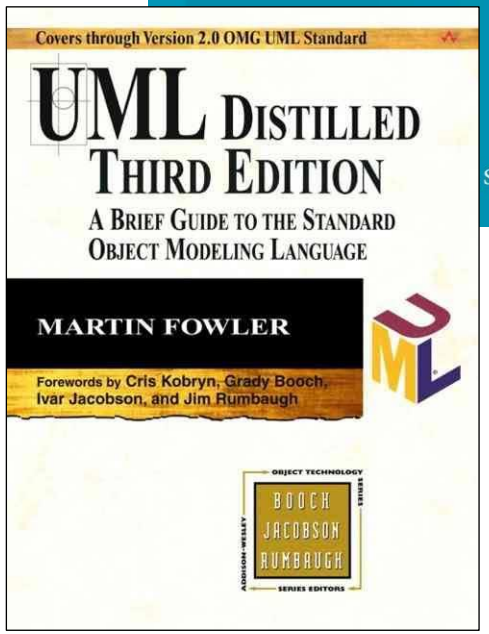
Peer-Olaf Siebers

University of Nottingham
UK | CHINA | MALAYSIA

Topics

- Lecture
  - OO Analysis and Design (OOA/D) with UML
  - Lab reflections
- Lab:
  - Group activity: Designing a software with the help of UML
- Workshop:
  - Being agile in SWM
  - Communication in teams





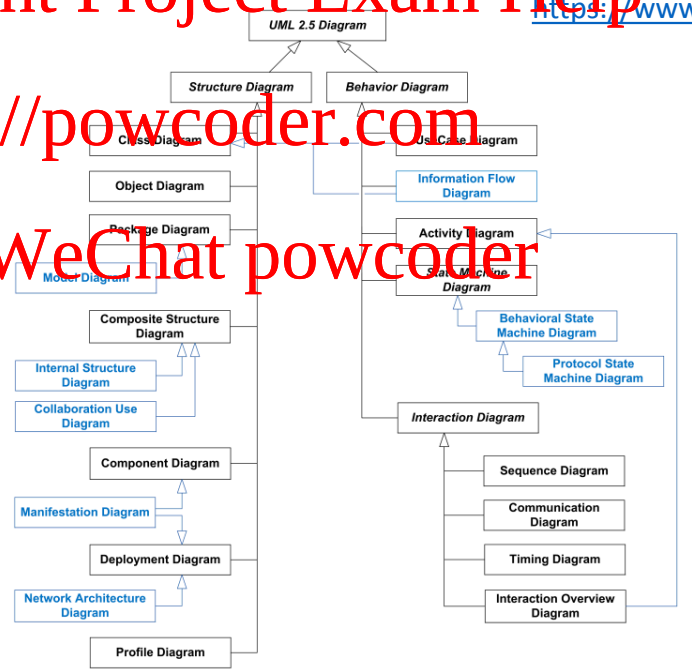
unified modelling language

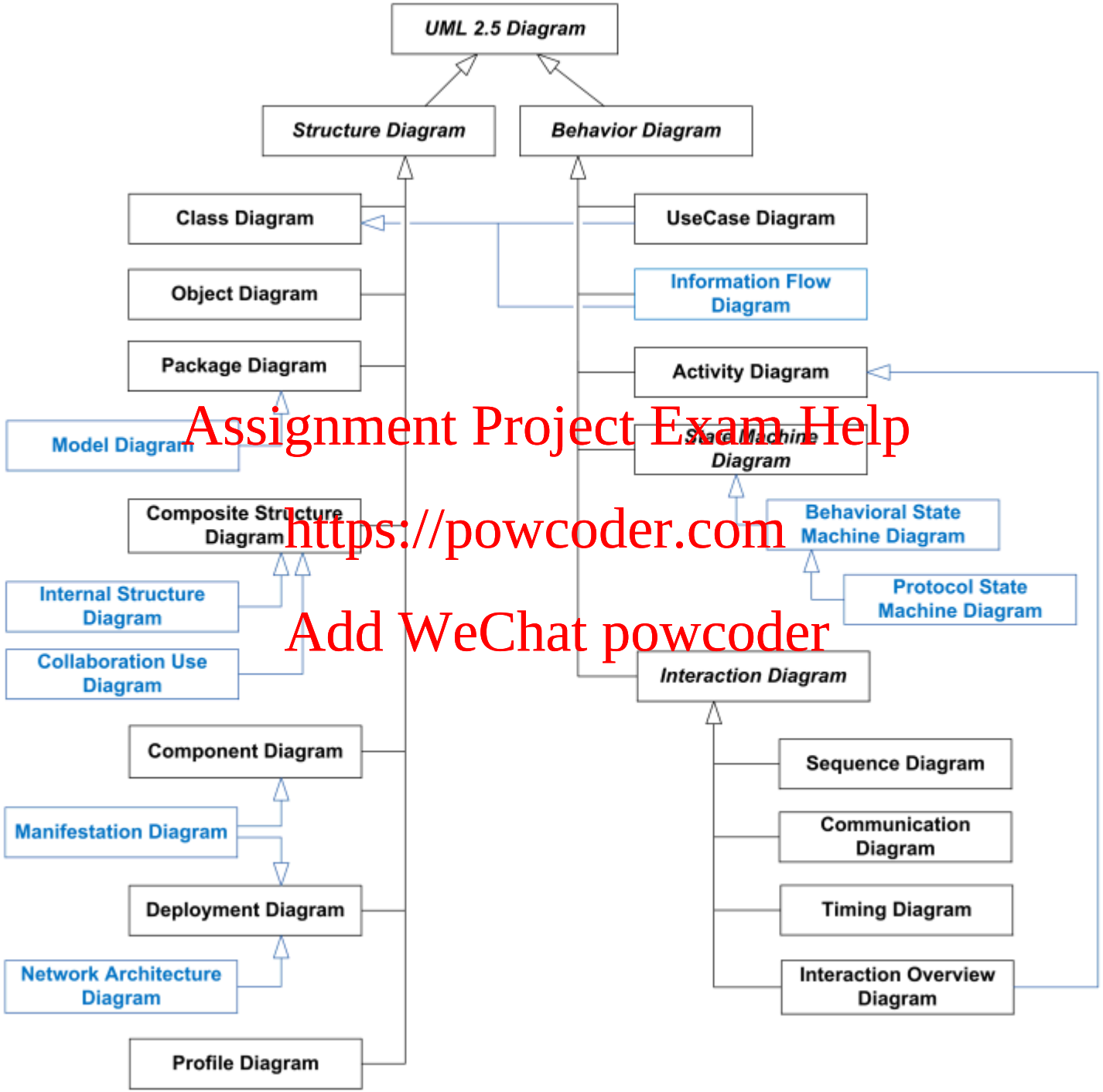
UML (Unified Modelling Language)

- UML: "A specification defining a graphical language for visualizing, specifying, constructing, and documenting the artifacts of distributed object systems."

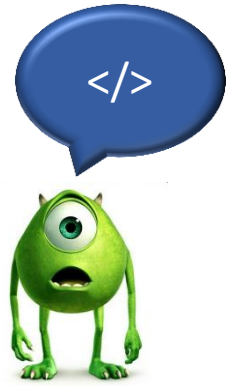
https://www.omg.org/spec/UML/About-UML/

- Latest Version: 2.5.1 (Dec 2017)

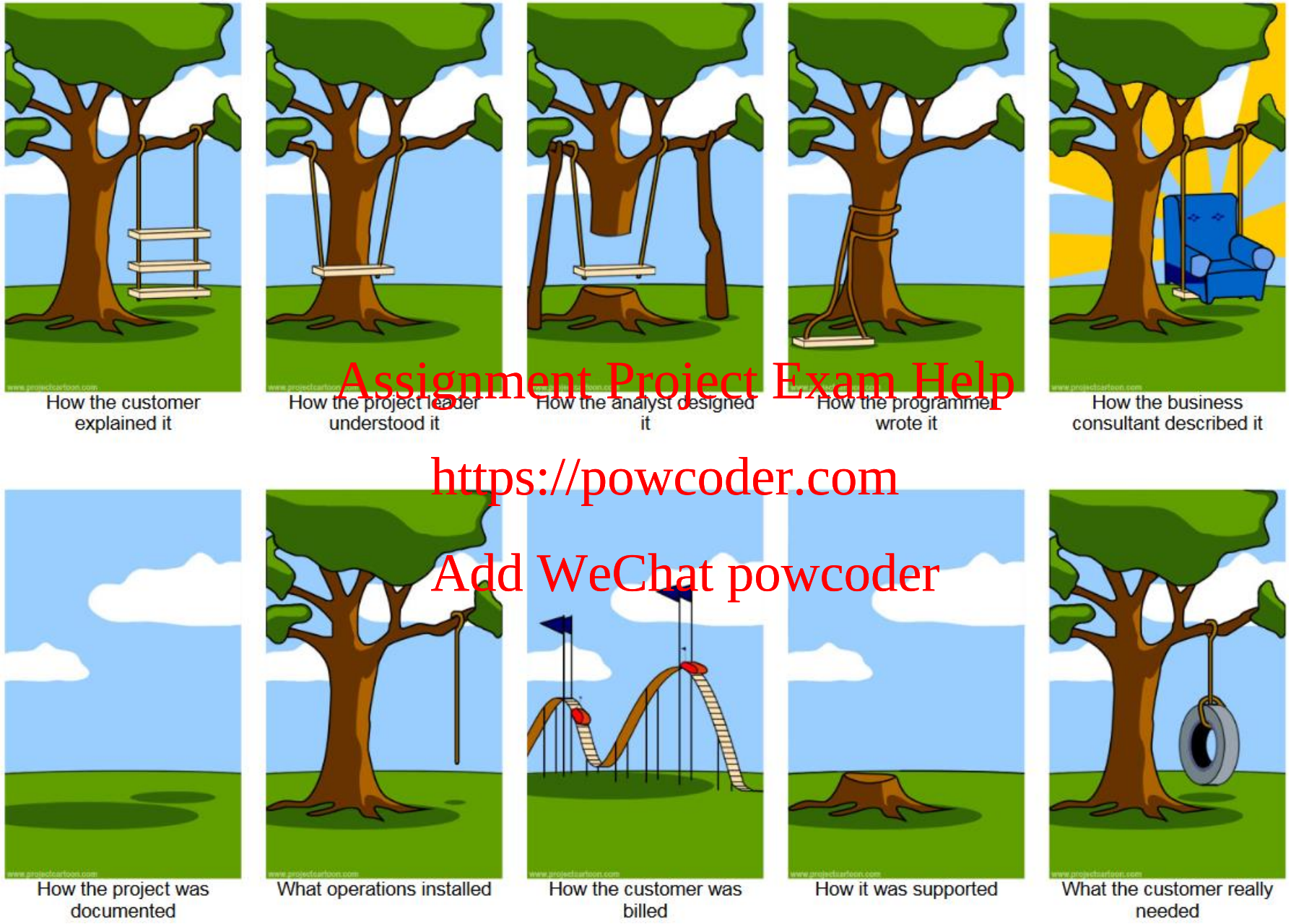




Why UML?

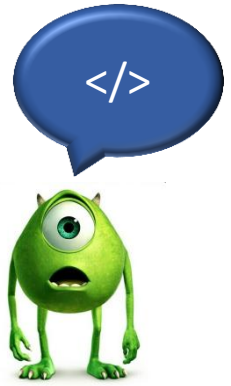


- Advantages of using UML:



Source: http://www.projectcartoon.com/

Why UML?



- Advantages of using UML:
  - Enhances communication and ensures the right communication
  - Captures the logical software architecture independent of the implementation language
  - Helps to manage the complexity
  - Enables reuse of design

case studies

Example: Library Booking System

Example: Library Booking System

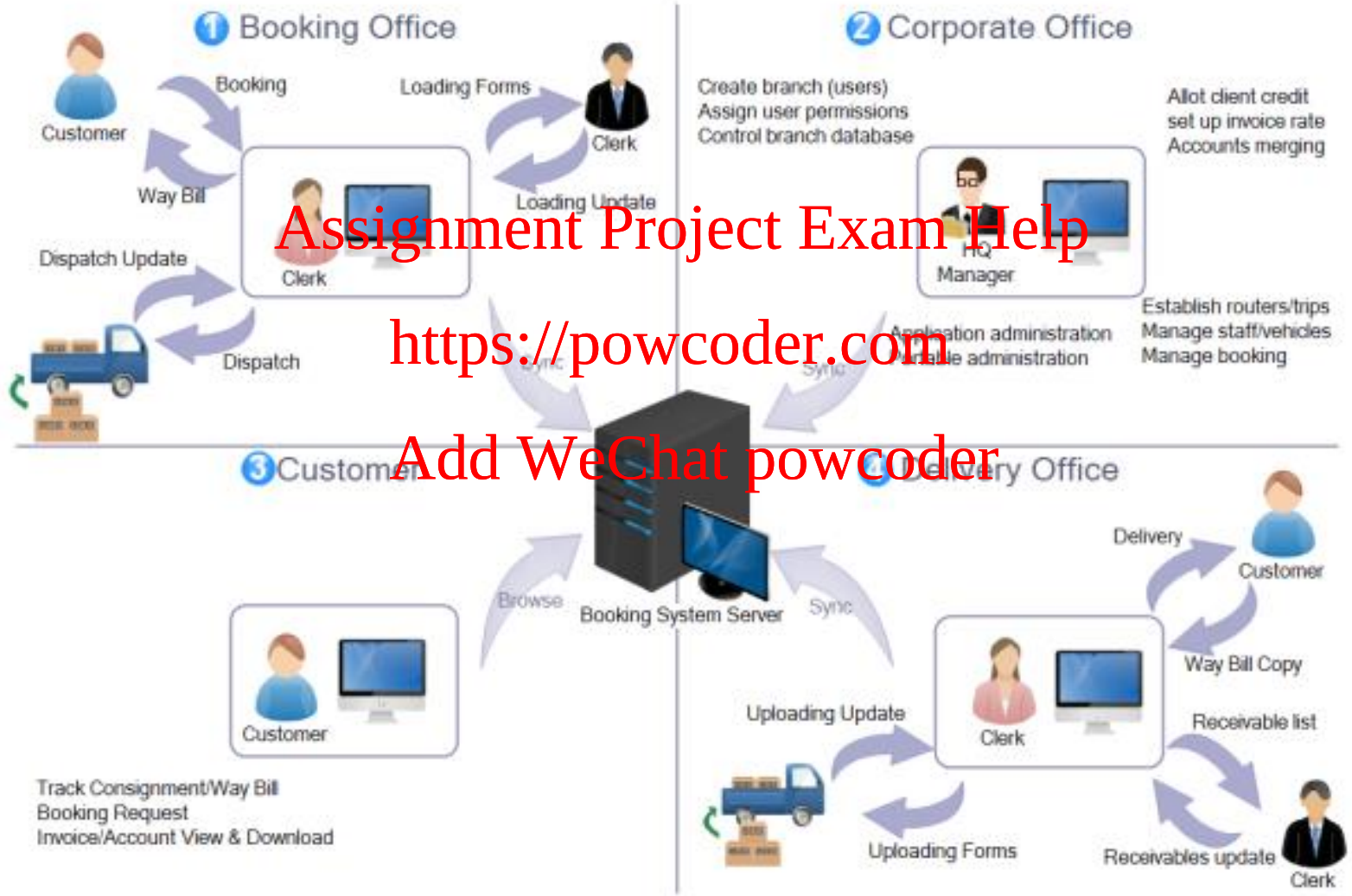
- Library Rules
  - The library contains books and journals; it may have several copies of a given book; some are for short term loan only, the others can be borrowed by any library member for three weeks
  - Normal members can borrow up to 6 books at the same time, staff members up to 12
  - Only staff members can borrow journals

- System Requirement
  - The system must keep track of when books and journals are borrowed and returned, enforcing the rules described above

Example: Library Booking System

- Our Job: Development of a Library Booking System
  - After discussing priorities with the university we decided that the first iteration of the system should consider the following user stories:
    - As a university member I want to be able to borrow a copy of a book
    - As a university member I want to be able to return a copy of a book
    - As a university staff member I want to be able to borrow a journal
    - As a university staff member I want to be able to return a journal

Activity: Fleet Logistics Management

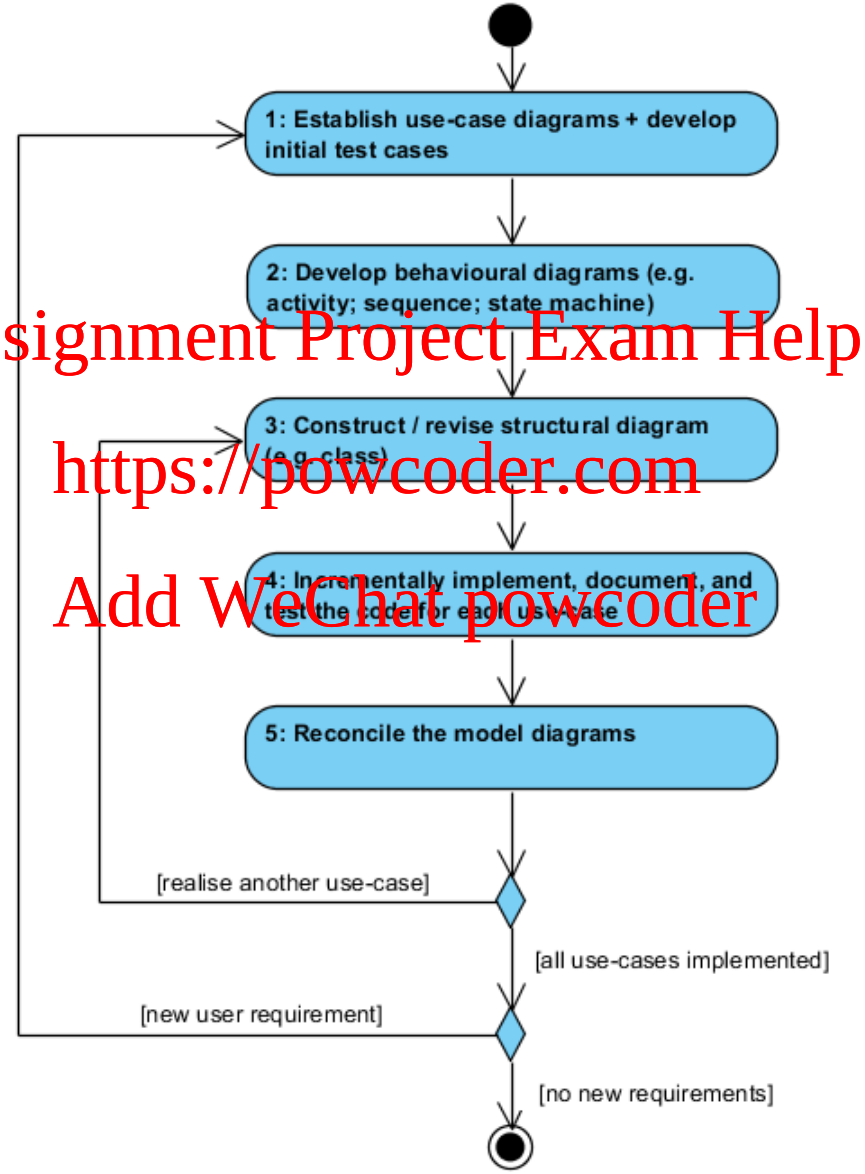


Activity: Fleet Logistics Management

- User stories
  - As a client I want to be able to check availability of lorries
  - As a client I want to be able to track cargo
  - As a manager I want to be able to see the finances
  - As an admin I want to be able to search for information
  - As an admin I want to be able to organise routes
  - As an admin I want to be able to track lorries and cargo

ooa/d process

"Use Case Driven" OOA/D Process



[after Barclay and Savage 2004]

object oriented analysis

Use Case Diagrams

- Use case diagrams
  - Behaviour diagrams used to describe a set of actions (use cases) that some system or systems (subject) should or can perform in collaboration with one or more external users of the system (actors)
  - They do not make any attempt to represent the order or number of times that the systems actions and sub-actions should be executed

- Use case diagram components:
  - Actors
  - Use cases
  - System boundary
  - Relationships

Use Case Diagrams

- Actors
  - Entities that interface with the system
  - Can be people or other systems
  - Think of actors by considering the roles they play

- Use cases
  - Based on user stories (derived from discussions with stakeholders)
  - Represent what the actor wants your system to do for them
  - In the use case diagram only the use case name is represented; in a use case specification each use case is formally described
    - Formal description must be a complete flow of activity (from the actors point of view) that provides observable and valuable result to the actor(s)

Use Case Diagrams

- Relationship between use case and actor:
  - Associations indicate which actors initiate which use cases
- Relationship between two use cases:
  - Specifying common functionality and simplifying use case flows
  - Using <<include>> or <<extends>>

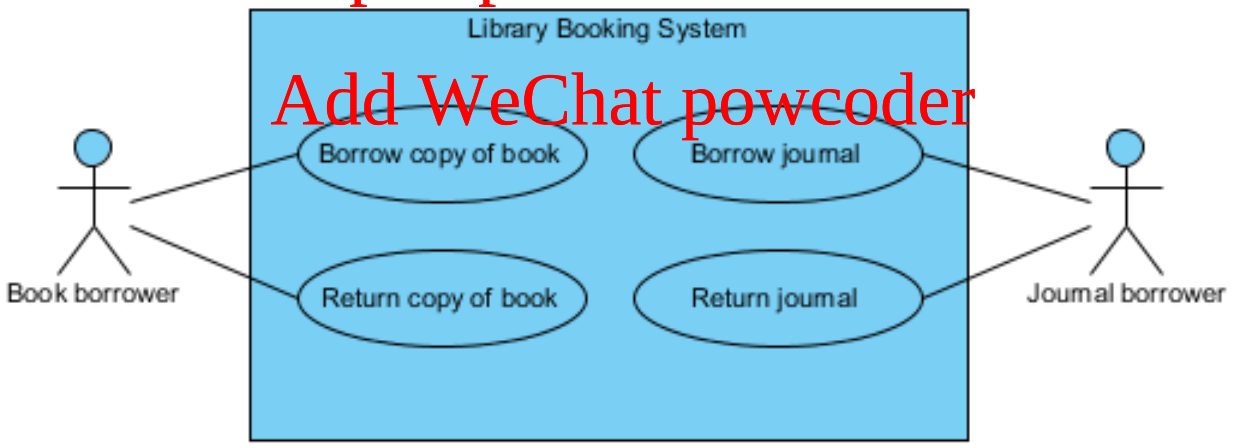
Use Case Diagrams

- <<include>>
  - Multiple use cases share a piece of same functionality which is placed in a separate use case rather than documented in every usecase that needs it
  - One use case is a functionality that another use case requires
  - The dependent use case ultimately re-uses the depended-on use case

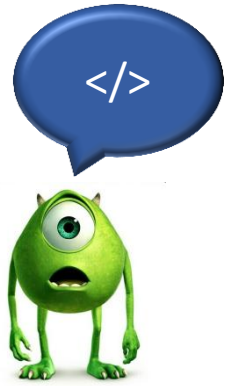
- <<extends>>
  - Use when activities might be performed as part of another activity but are not mandatory for a use case to run successfully
  - We are adding more capability

Example: Library Booking System

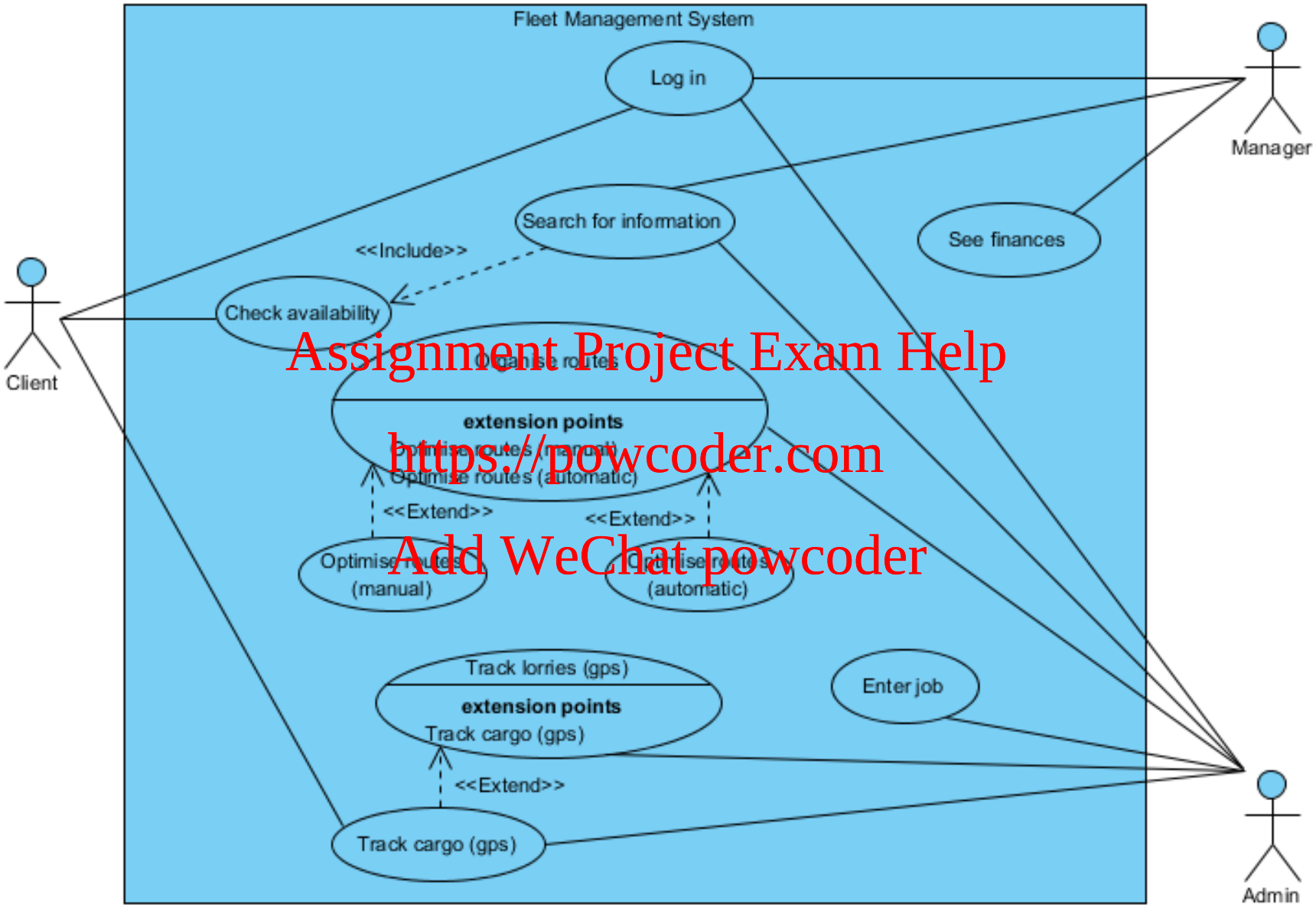
- Reminder
  - The library contains books and journals; it may have several copies of a given book; only staff members can borrow journals



Activity: Fleet Logistics Management



- Use Case Diagram?
  - Reminder
    - Client wants to check availability of lorries and track cargo. Manager wants to see the finances. Admin wants to search for information, organise routes and track lorries and cargo



Use Case Specification

- Use case specification elements
  - Use case name
  - Use case purpose
  - Pre-condition(s)
  - Base Path (optimistic flow)
  - Alternative Paths (pragmatic flows)
  - Post-condition(s)

Use Case Specification

- Base and alternative path:
  - Base Path (optimistic flow)
    - "happy day" scenario
  - Alternative Paths (pragmatic flows)
    - Every other possible way the system can be (ab)used
    - Includes perfectly normal alternative use, but also errors and failures

Example: Library Booking System

- Use Case: Borrow copy of books
  - Purpose:
    - The Book Borrower borrows a book from the library using the Library Booking System
  - Pre-condition(s):
    - The book must exist
    - The book must be available

Example: Library Booking System

- Use Case: Borrow copy of books
  - Base Path (optimistic flow)
    1. LBS requests membership card
    2. BB provides membership card
    3. BB is logged in by LBS
    4. LBS checks permissions / debts
    5. LBS asks for presenting a book
    6. BB presents a book
    7. LBS scans RFID tag inside book
    8. LBS updates records accordingly
    9. LBS disables anti-theft device
    10. BB is logged out by LBS
    11. LBS confirms that process has been completed successfully

BB = Book Borrower
LBS = Library Booking System

Example: Library Booking System

- Use Case: Borrow copy of books
  - Alternative Paths (pragmatic flows):
    - BB's card has expired: Step 3a: LBS must provide a message that card has expired; LBS must exit the use case
    - LBS cannot read membership card: Step 3a: LBS must provide a message that card could not be read correctly; LBS must go back to Step 1
    - ...
  - Post-condition(s):
    - The member has successfully borrowed the book
    - The system is up to date

Activity: Fleet Logistics Management

- Use Case: Search for Information
  - Use case:
    - Search for Information
  - Purpose:
    - Admin can search the DB for any kind of information related to lorries and jobs
  - Pre-condition(s):
    - Admin has to be logged in

Activity: Fleet Logistics Management

- Use Case: Search for Information
  - Base Path (optimistic flow)
    1. Admin opens search window
    2. Admin defines query using query editor
    3. Admin sends query to DB
    4. DB deals with query: finding results
    5. DB deals with query: organising them by relevance
    6. DB sends results back
    7. DB requests confirmation that result is sufficient
    8. Admin confirms that result is sufficient
    9. DB closes search window

Activity: Fleet Logistics Management

- Use Case: Search for Information

  - Alternative Paths (pragmatic flows)
    - Admin has not received the required information: Step 7a Admin denies that result is sufficient; Admin must go back to Step 2
    - DB is not accessible: Step 3a: DB returns warning message that DB is not accessible; use case needs to be left

  - Post-condition(s):
    - The admin has retrieved the required information

object oriented design

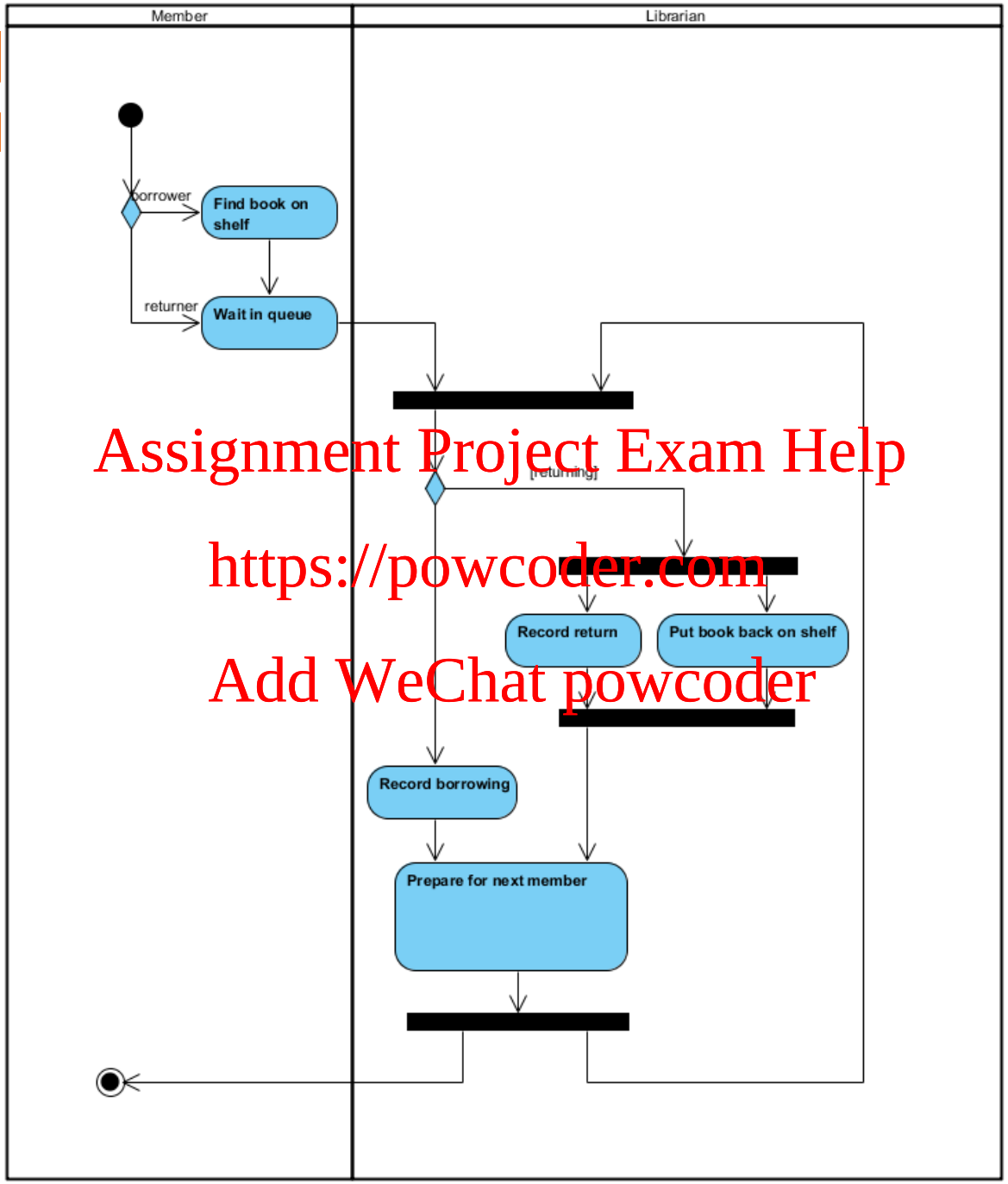
Activity Diagrams

- Activity diagram
  - Graphical representations of workflows of stepwise activities and actions related to an individual use case or across many use cases
  - Supports parallel behaviour (unlike flowcharts)

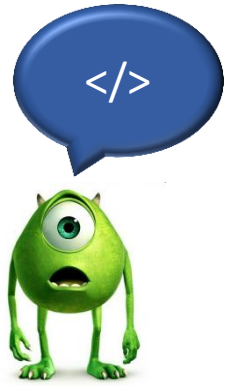
Activity Diagrams

- Activity diagram components
  - Activities: A state that is left once the activity is finished
  - Activity edge: Transition that fires when previous activity completes
  - Synchronisation bar: Describes the co-ordination of activities
  - Decision diamond: Used to show decisions
  - Start and stop markers: Used to define entry and exit points
  - Swim lane: A way to group activities performed by the same actor on an activity diagram or to group activities in a single thread

Example: Libra

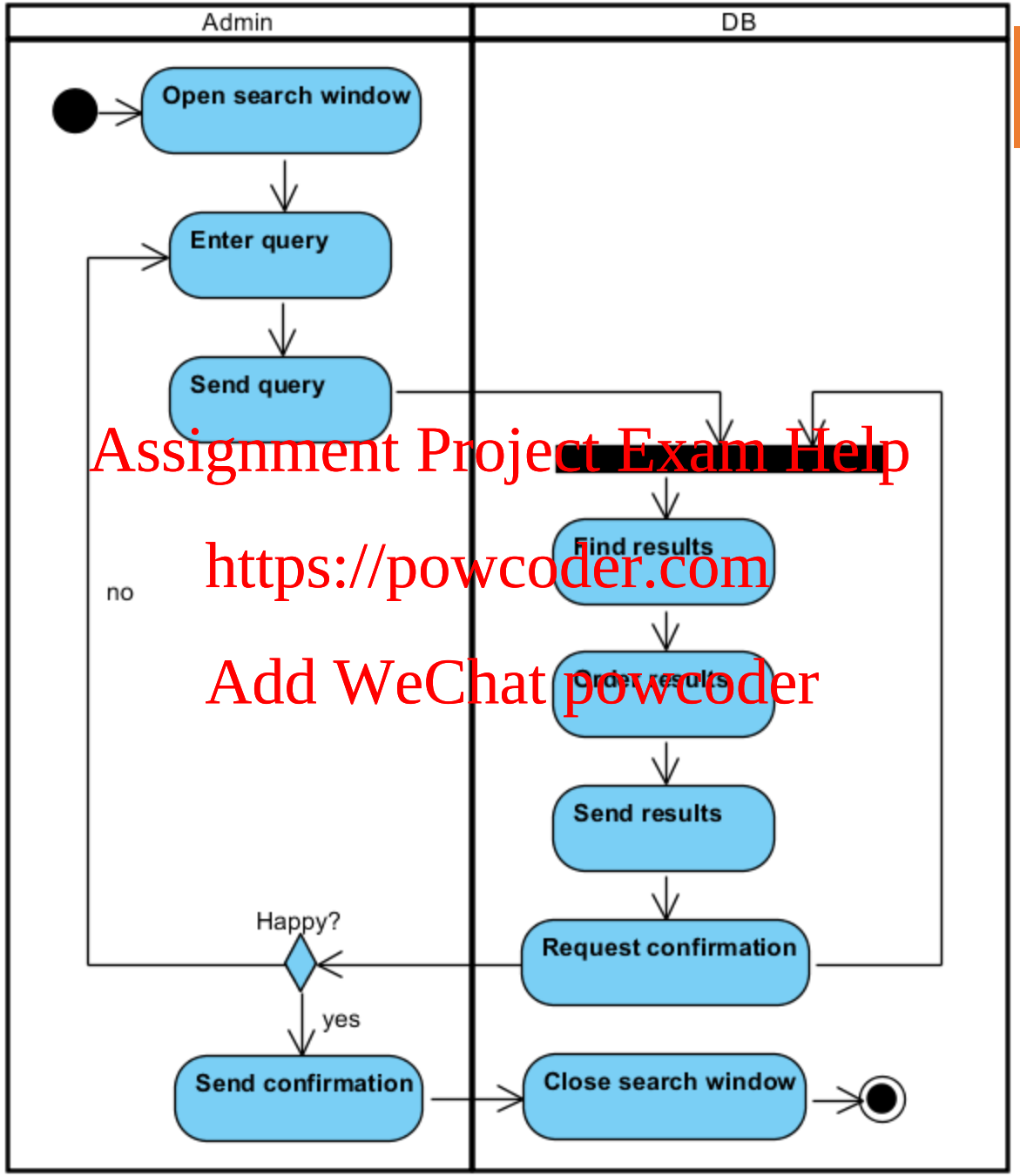


Activity: Fleet Logistics Management



- Activity Diagram for use case "Search for Information"?
  - Reminder: Base Path (optimistic flow) for this use case
    1. Admin opens search window
    2. Admin defines query using query editor
    3. Admin sends query to DB
    4. DB deals with query: finding results
    5. DB deals with query: organising them by relevance
    6. DB sends results back
    7. DB requests confirmation that result is sufficient
    8. Admin confirms that result is sufficient
    9. DB closes search window

Activity: Fleet



Sequence Diagrams

- Sequence diagrams are a temporal representation of objects and their interactions; they shows the objects and actors taking part in a collaboration at the top of dashed lines

- Sequence diagrams components
  - Participants are objects or actors that act in the sequence diagram
  - Lines represent time as seen by the object (lifeline)
  - Arrows from lifeline of sender to lifeline of receiver are messages (denoting events or the invocation of operations)
  - A narrow rectangle covering an object's life line shows a live activation of the object

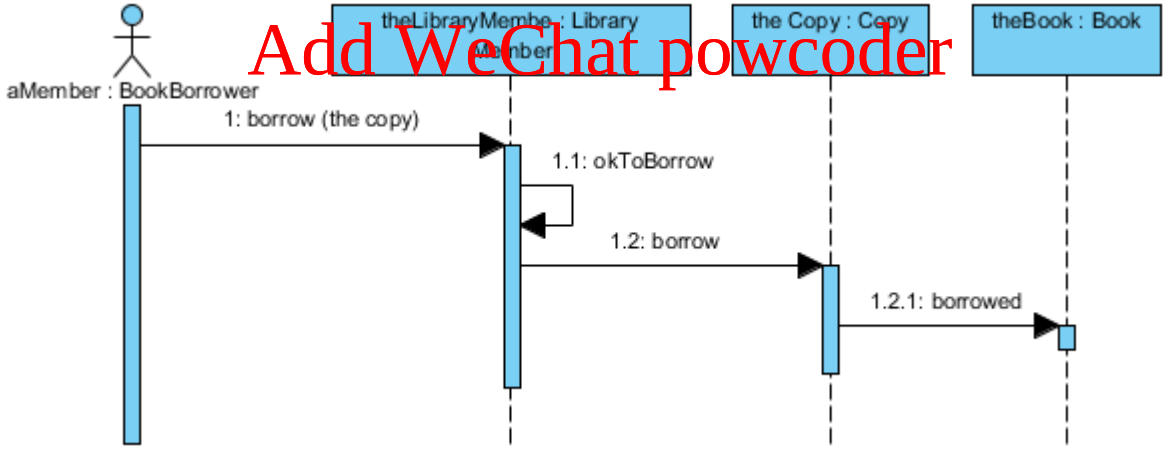
Sequence Diagrams

- The axes in a sequence diagram
  - Horizontal: which object/participant is acting
  - Vertical: time (down -> forward in time)
- Creation: arrow with 'new' written above it
  - Notice that an object created after the start of the scenario appears lower than the others
- Deletion: an X at bottom of object's lifeline
  - In some OOP languages this is handled automatically

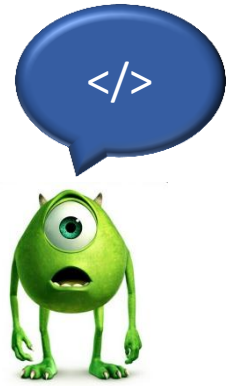
Example: Library Booking System

- Reminder
  - The library contains books and journals; it may have several copies of a given book; only staff members can borrow journals

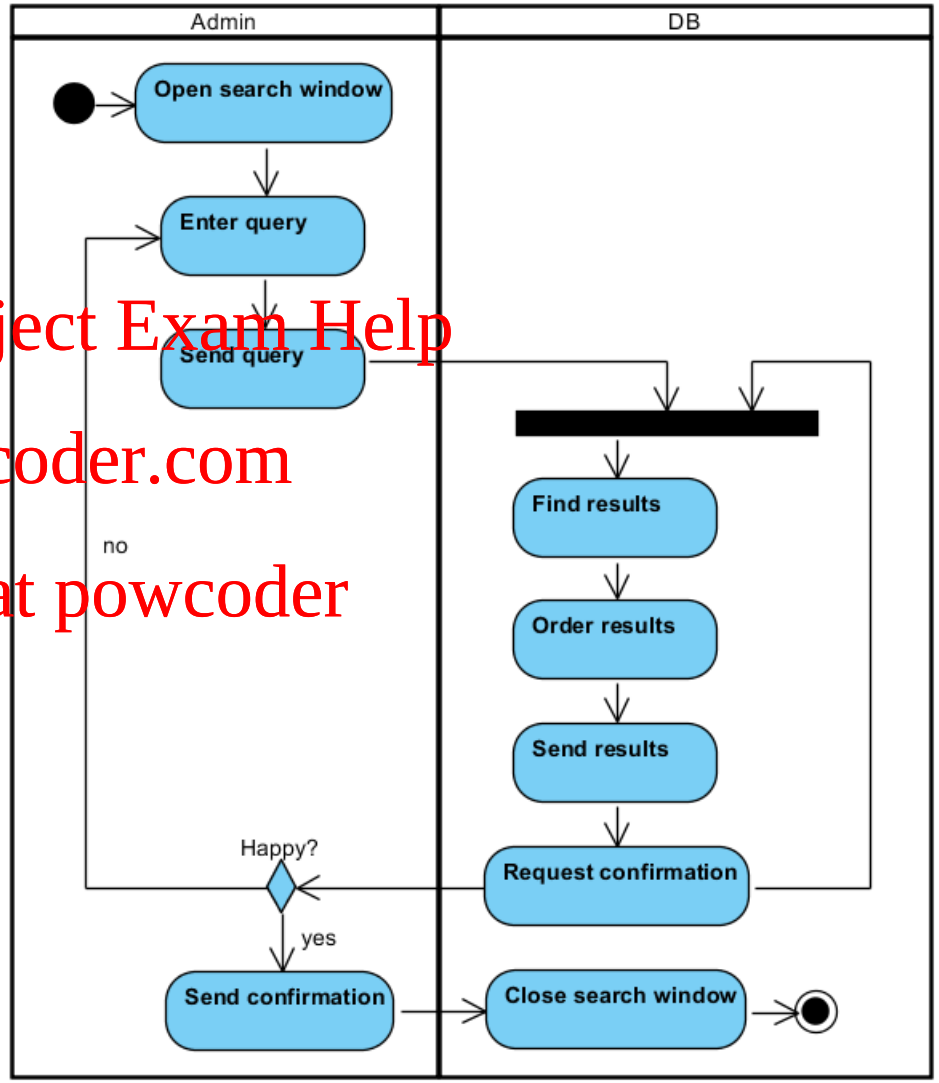
This is the record that keeps track of the books



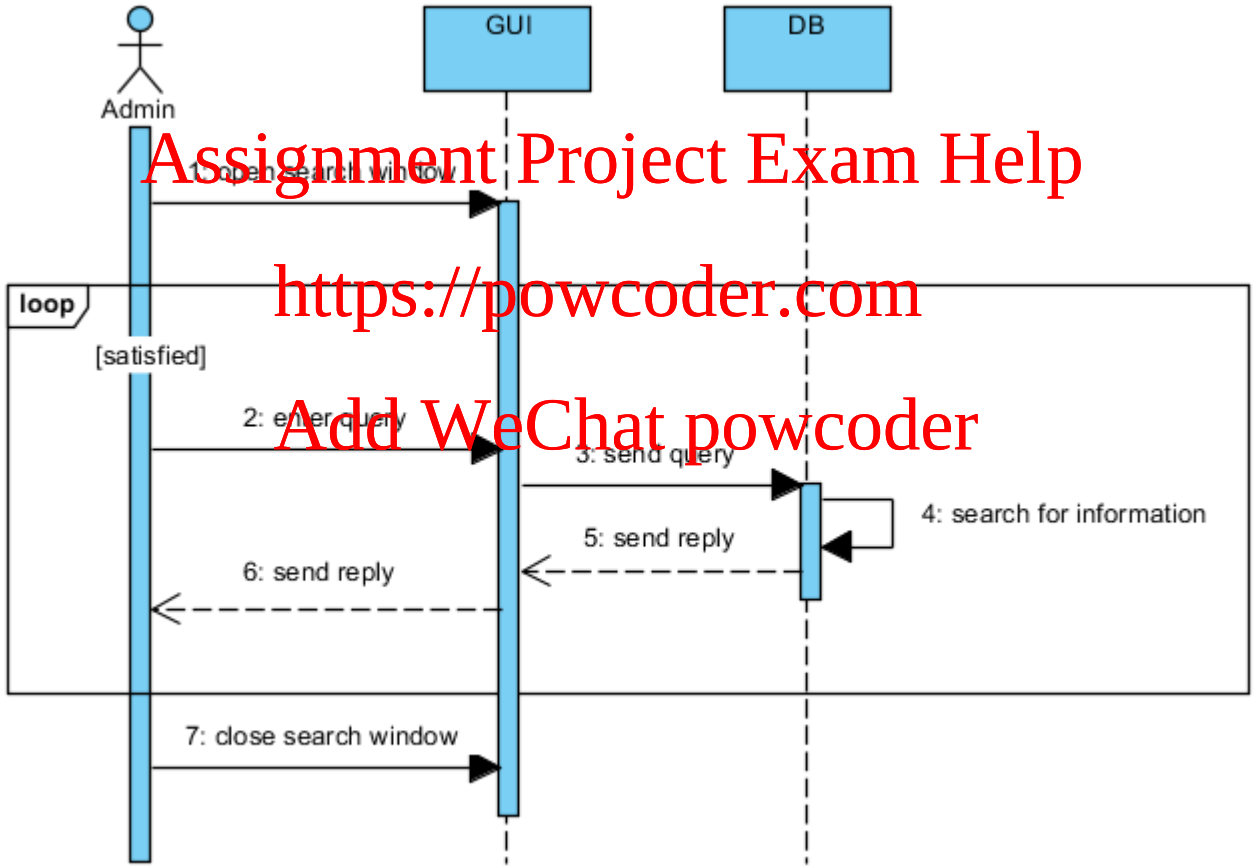
Activity: Fleet Logistics Management



- Sequence Diagram for use case "Search for Information"?
  - Reminder: Activity Diagram



Activity: Fleet Logistics Management



Class Diagrams

- Class diagrams shows the existence of classes and their structures and relationships in the logical view of a system
- Class diagram components:
  - Classes (their structure and behaviour)
  - Class relationships
    - Dependency
    - Association
    - Aggregation
    - Composition
    - Realisation
    - Generalisation
  - Multiplicity and navigation indicators

Class Diagrams

- What makes a class model good?
  - If you can build a system quickly and cheaply to the satisfaction of the client
  - If you can build a system that is easy to maintain and easy to extend
- Identifying classes
  - A class describes a set of objects with an equivalent role
  - Identify candidate classes by picking all nouns and noun phrases out of a requirement specification of a system
  - Discard candidates which appear to be inappropriate (redundant, vague, an event or operation, meta-language, outside the scope of the system, an attribute)

Class Diagrams

- What kind of things are classes?
  - Tangible (real world things)
  - Roles
  - Events
  - Interactions

- First two are much more common sources for classes – the other two might help to find and name associations between them

Class Diagrams

- Associations between classes
  - Correspond to verbs
  - Real world association that can be described by a short sentence (reader borrows a book)
  - Classes are associated if some object of class A has to know about some object of class B or vice versa

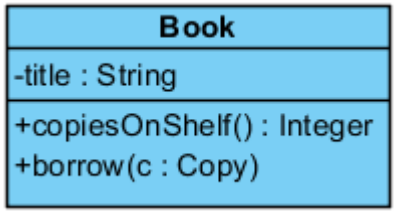
- Multiplicity
  - Number of links between each instance of the source class and instances of the target class
    - 1 = exactly 1; * = unlimited number (zero or more); 0..* = zero or more; 1..* = one or more; 0..1 = zero or 1; 3..7 = specified range (from 3 to 7)

Class Diagrams

- Class representation
  - In UML classes are depicted as rectangles with three compartments
    - Class name
    - Attributes: Describe the data contained in an object of the class
    - Operations: Define the ways in which objects interact

  - Additional symbols

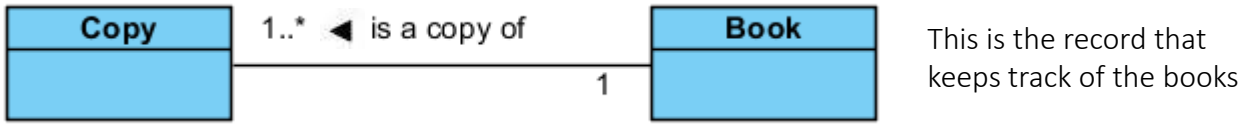
+	public
#	protected
-	private
/	derived
$	static



This is the record that keeps track of the books

Class Diagrams

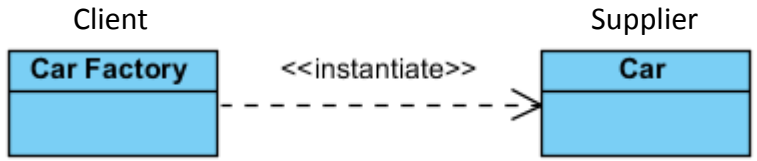
- Relationship: Association
  - These are the most general types of relationships
  - It shows bi-directional connection between two classes
  - It is a weak coupling as associated classes remain somewhat independent of each other



Every Copy is associated only with one Book; every Book is associated with one or more Copies

Class Diagrams

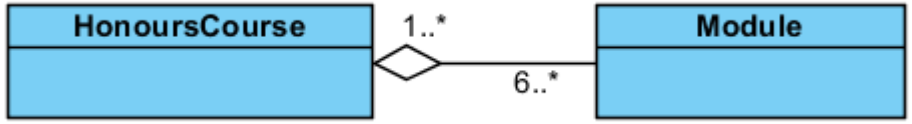
- Relationship: Dependency
  - A directed relationship which shows that an element or a set of elements require(s), need(s) or depend(s) on other elements for implementation
  - It is a supplier-client relationship, where supplier provides something to the client, and thus the client is in some sense incomplete without or semantically or structurally dependent on the supplier element(s)
  - Modification of the supplier may impact the client elements



CarFactory class depends on the Car class

Class Diagrams

- Relationship: Aggregation ("is part of" relationship)
  - This is a special type of association
  - It is used when one object logically or physically contains another; the container is called "aggregate"
  - The components of aggregate can be shared with others



Each Honours Course consists of 6 or more Modules; each Module could be part of one or more Honours Courses

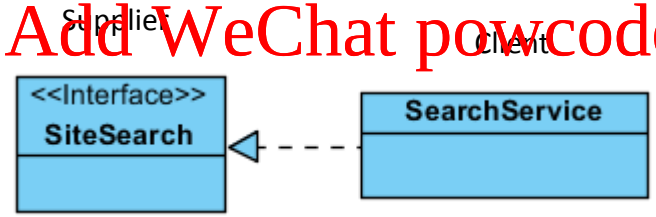
Class Diagrams

- Relationship: Composition
  - This is a strong form of aggregation (physical containment)
  - The multiplicity at the composition end is always 1 – as the parts have no meaning outside the whole
  - If the whole is copied or deleted its parts are copied or deleted together with it; the owner is explicitly responsible for creation and deletion of the parts

A board has 64 squares, and each square belongs to exactly one board

Class Diagrams

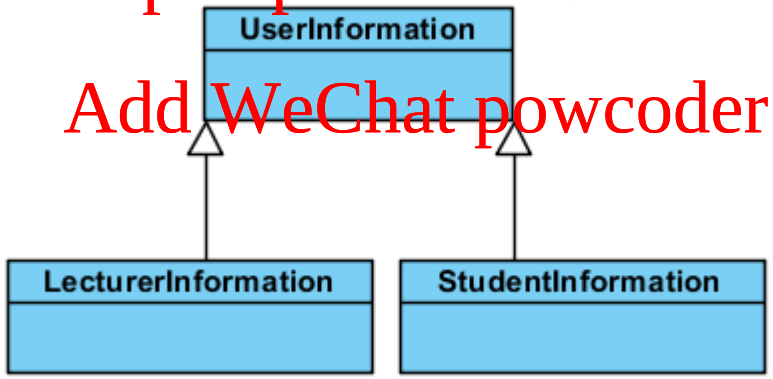
- Relationship: Realisation
  - A "Realisation" is a specialised abstraction relationship between two sets of model elements, one representing a specification (the supplier) and the other representing an implementation of the latter (the client)



Interface SiteSearch is realised (implemented) by SearchService

Class Diagrams

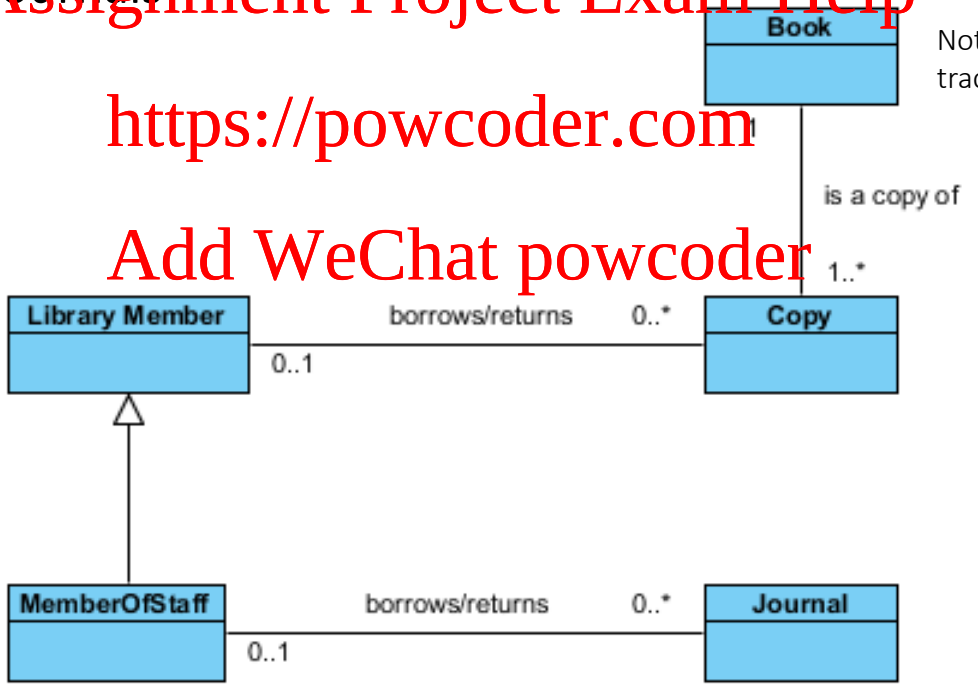
- Relationship: Generalisation ("is a" relationship) > Inheritance
  - A directed relationship between a more general classifier (superclass) and a more specific classifier (subclass)



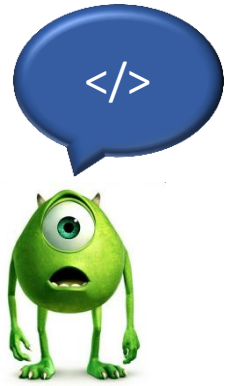
LecturerInformation and StudentInformation are generalised by UserInformation

Example: Library Booking System

- Reminder
  - The library contains books and journals; it may have several copies of a given book; only staff members can borrow journals

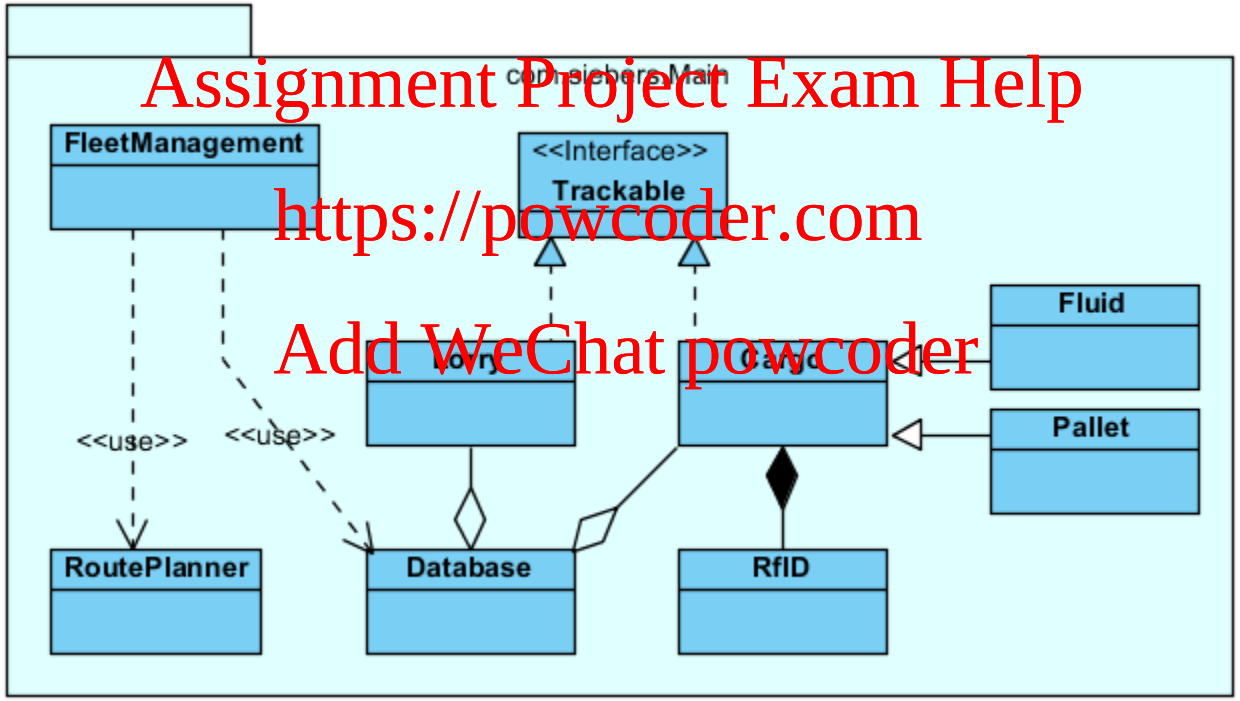


Activity: Fleet Logistics Management



- Class Diagram?
  - Reminder
    - Client wants to check availability of lorries and track cargo. Manager wants to see the finances. Admin wants to search for information, organise routes and track lorries and cargo

Activity: Fleet Logistics Management



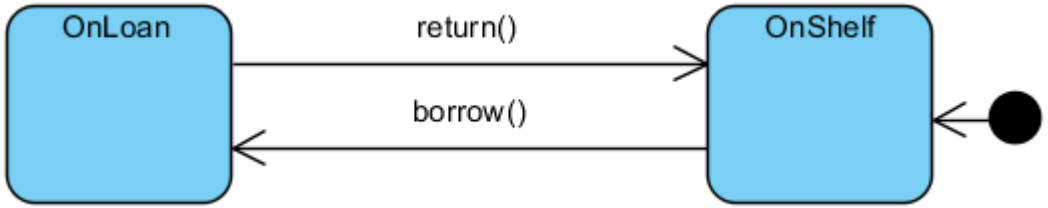
State Machine Diagrams

- State machine diagrams
  - In order to implement a class we need to understand what the dependencies are between the state of an object and its reaction to messages or other events
  - State machine diagrams show the states of a single object, the events or the messages that cause a transition from one state to another and the action that result from a state change.
  - You do not have to create a state machine diagram for every class!

State Machine Diagrams

- State: A condition during the life of an object when it satisfies some condition, performs some action, or waits for an event

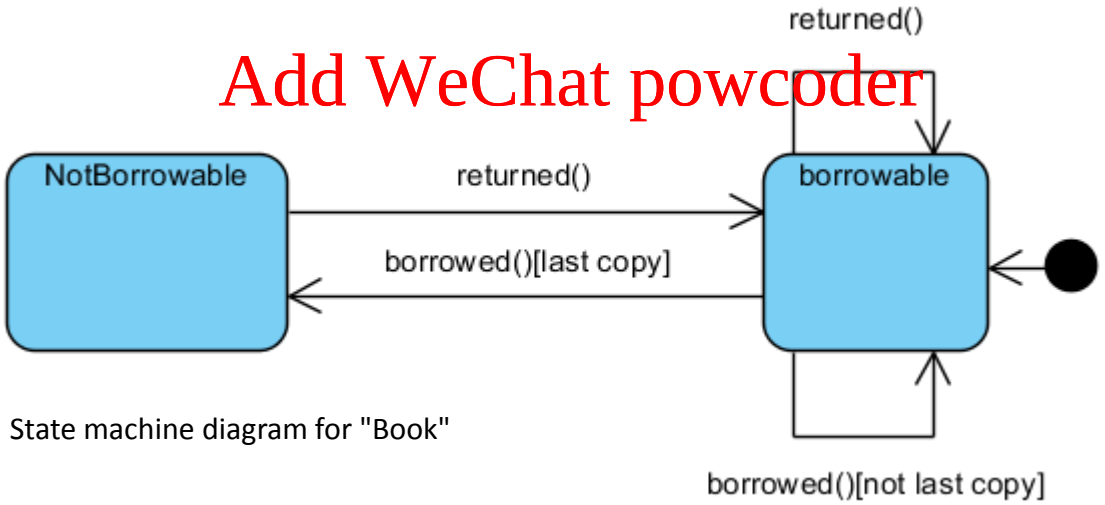
- There are two special states
  - Start state: Each state diagram must have one and only one start state
  - Stop State: An object can have multiple stop states



State machine diagram for "Copy"

State Machine Diagrams

- Guard
  - Sometimes a change of state of the object depends on the exact values of an object's attributes
  - Guard conditions affect the behaviour of a state machine by enabling actions or transitions only when they evaluate to TRUE and disabling them when they evaluate to FALSE.

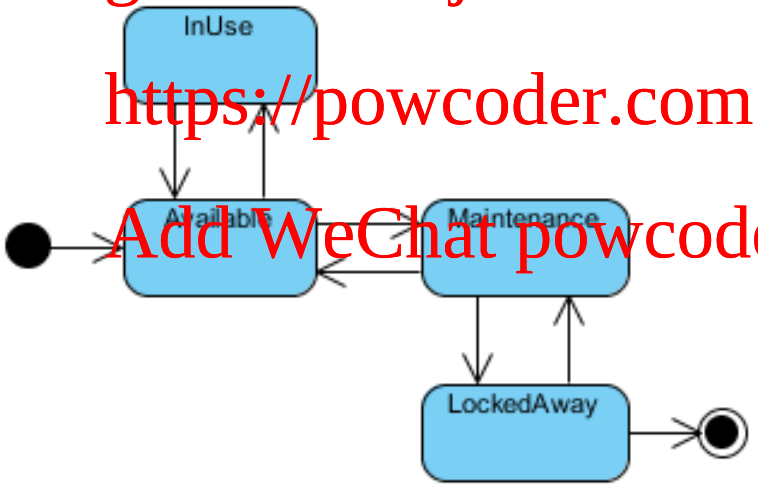


State machine diagram for "Book"

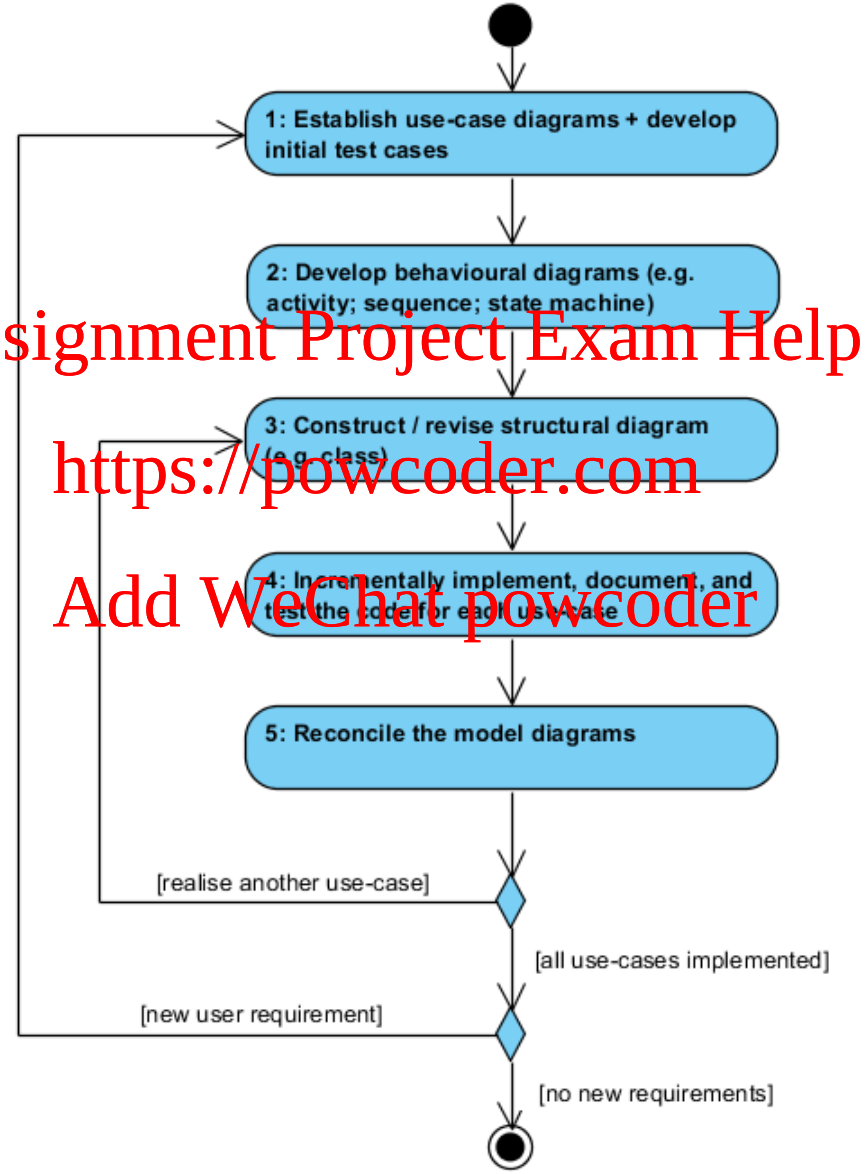
Activity: Fleet Logistics Management

- State Machine Diagram for "Lorry Class"?

Activity: Fleet Logistics Management



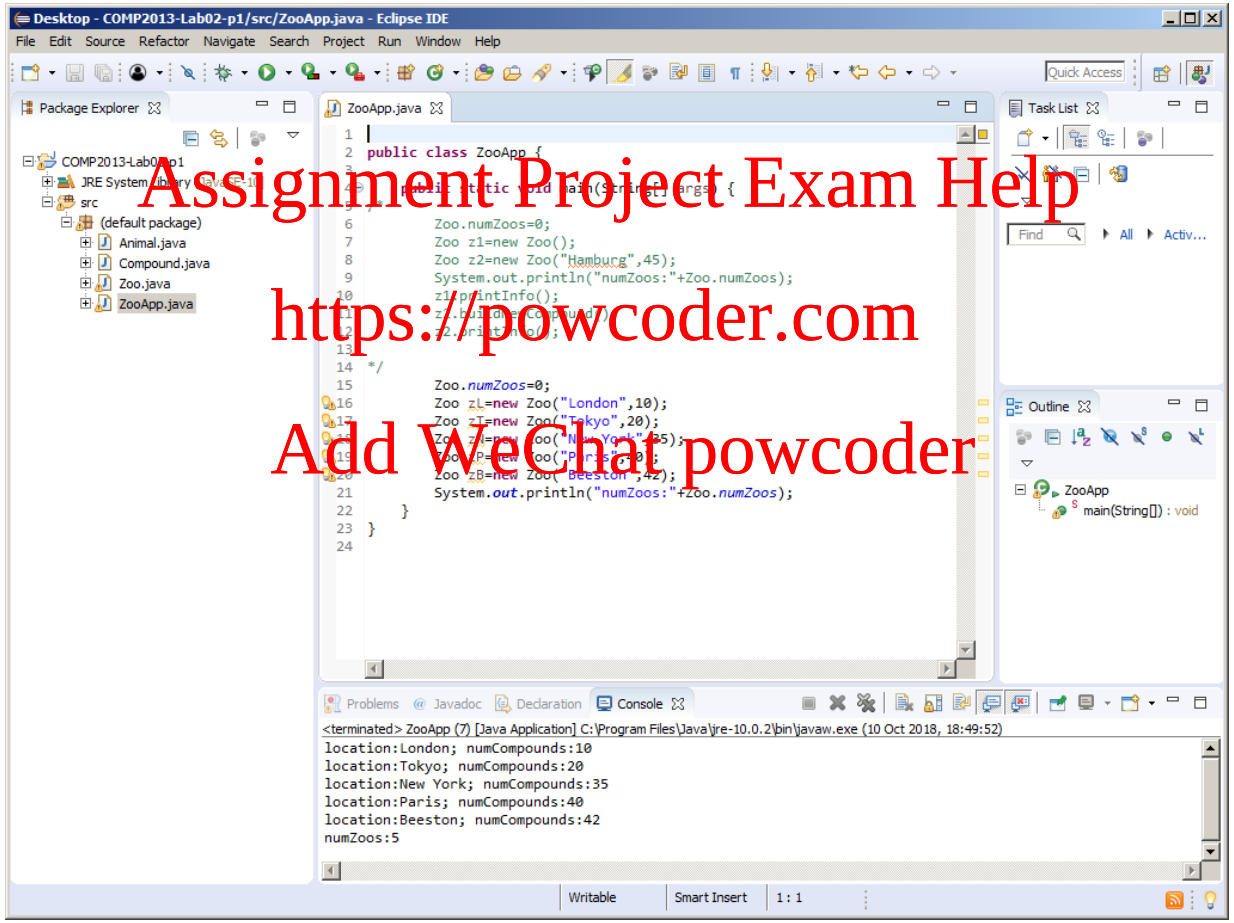
"Use Case Driven" OOA/D Process



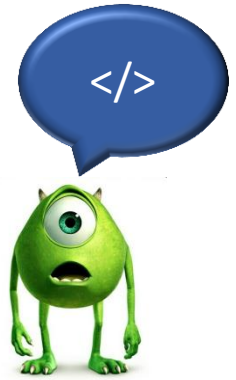
[after Barclay and Savage 2004]

lab reflections

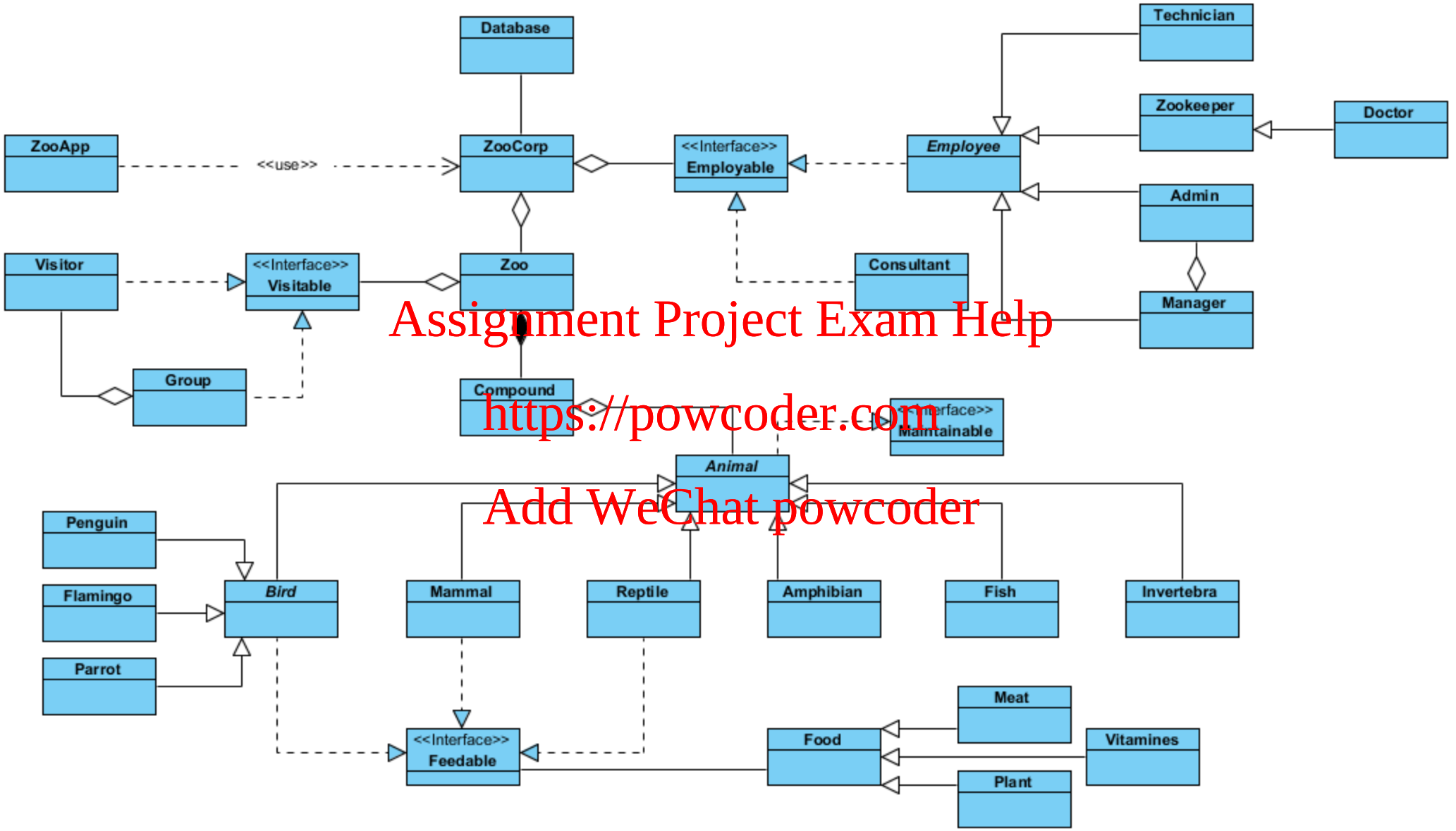
Lab 02: Part 1



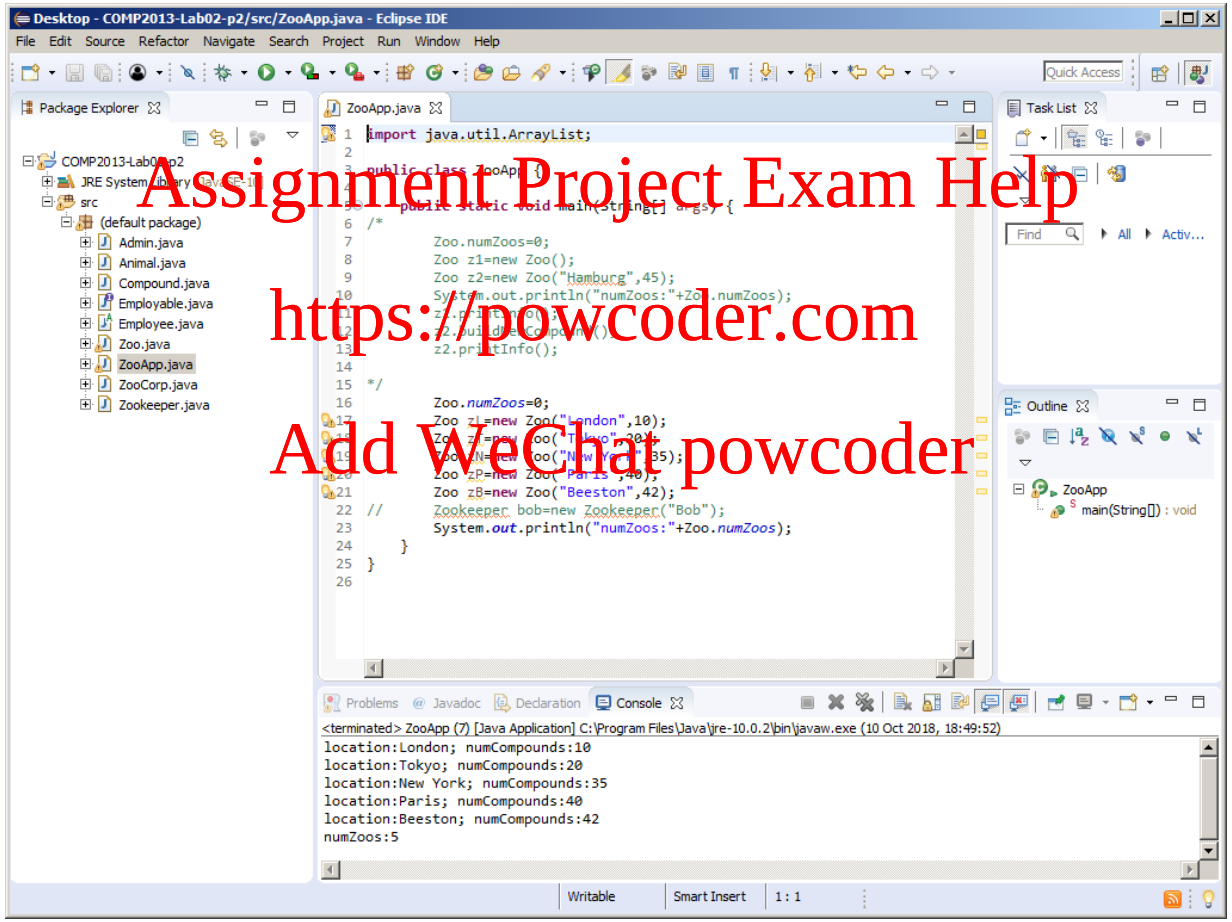
Lab 02: Part 2



- Adding "Employable" branch
  - ZooCorp + Employable: Aggregation
  - Employable: Interface
    - Does not define any field variables
  - Employee: Abstract
    - Why do we want employee to be abstract?
  - Zookeeper/Admin: Concrete
    - Make sure you choose the correct superclass
    - Need to implement all unimplemented methods from the interface



Lab 02: Part 2



And finally ...

References

- Barclay and Savage (2004) Object-Oriented Design with UML and Java