Object Oriented Analysis and Design with Java

UE19CS353

Dr. Phalachandra and Prof.J.Ruby Dinakar
Department of Computer Science and Engineering

Acknowledgements: Significant portions of the information in the slide sets presented through the course in the class, are extracted from the prescribed text books, information from the Internet. Since these are only intended for presentation for teaching within PESU, there was no explicit permission solicited. We would like to sincerely thank and acknowledge that the credit/rights remain with the original authors/creators only

UE19CS353: Object Oriented Analysis and Design with Java

Class Modelling: UML Class Diagrams and CRC

Dr. Phalachandra and Prof.J.Ruby Dinakar

Department of Computer Science and Engineering

Class Modelling

- A class model captures the static structure of a system by characterizing the objects in the system, the relationships between the objects and the attributes and operations for each class of objects.
- **Class diagram** is a graphical notation used to construct and visualize object oriented systems.
- Class diagram can be mapped directly with object oriented languages.
- Class diagrams capture the static structure of Object-Oriented systems as how they are structured rather than how they behave.
- It supports architectural design.
- They identify what classes are there, how they interrelate and how they interact.
- A UML class diagram is made up of:

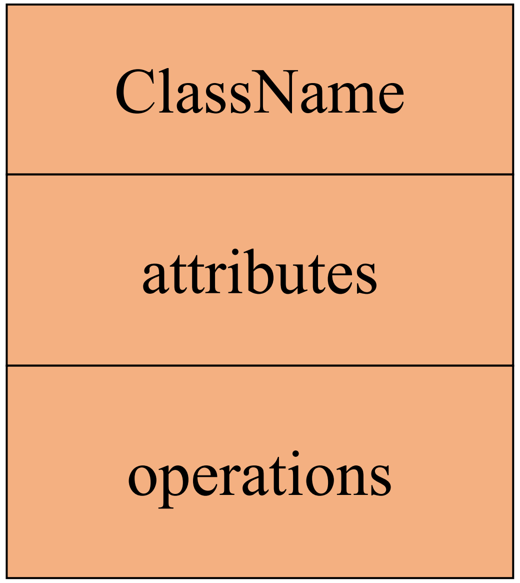
 A set of classes and

 A set of relationships between classes

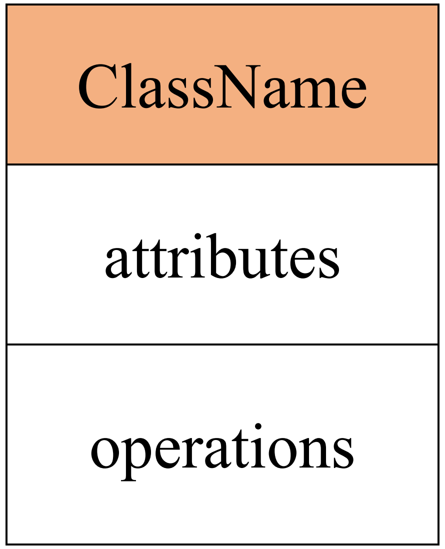
Class

A class is a description of a set of objects that share the same attributes, operations, relationships and semantics.

Graphically, a class is rendered as a rectangle, usually including its name, attributes, and operations in separate, designated compartments.

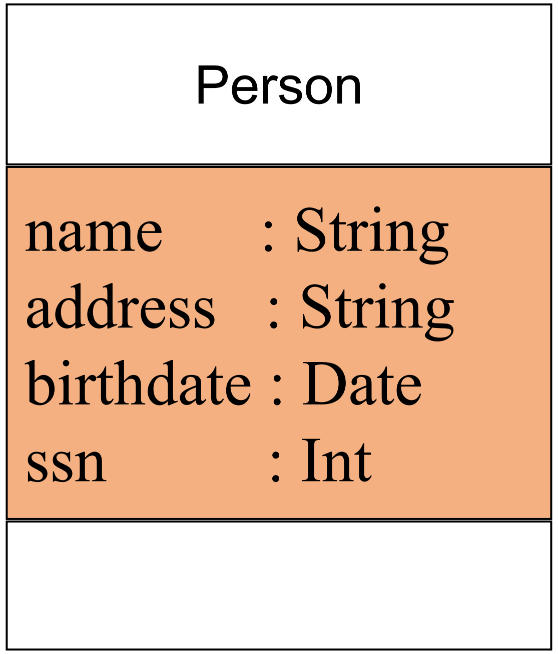


Class



The name of the class is the only required tag in the graphical representation of a class. It always appears in the top-most compartment.

Attributes



Each class can have attributes.

An *attribute* is a named property of a class that describes the object being modeled.

Each attribute has a type.

In the class diagram, attributes appear in the second compartment just below the name-compartment.

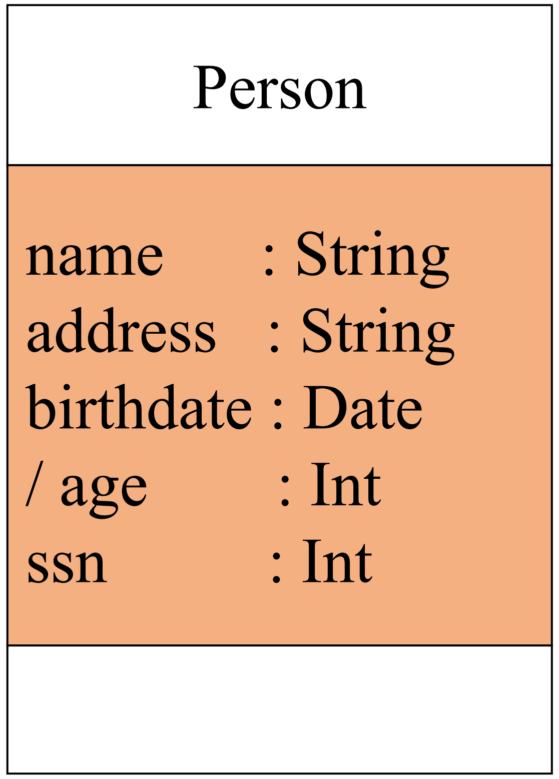
Derived Attribute

Attributes are usually listed in the form:

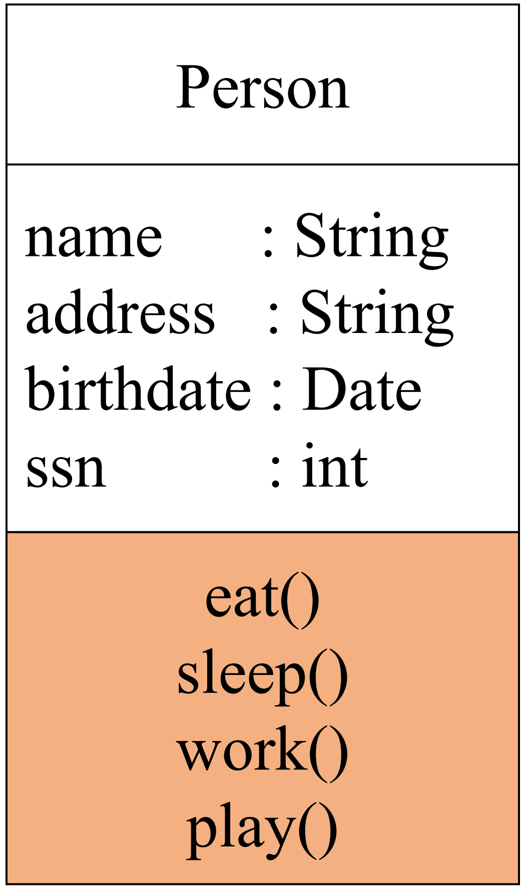
attributeName : Type

A *derived* attribute is one that can be computed from other attributes, but doesn't actually exist. For example,
a Person's age can be computed from his birth date.
A derived attribute is
designated by a preceding '/' as in:

/ age : Date

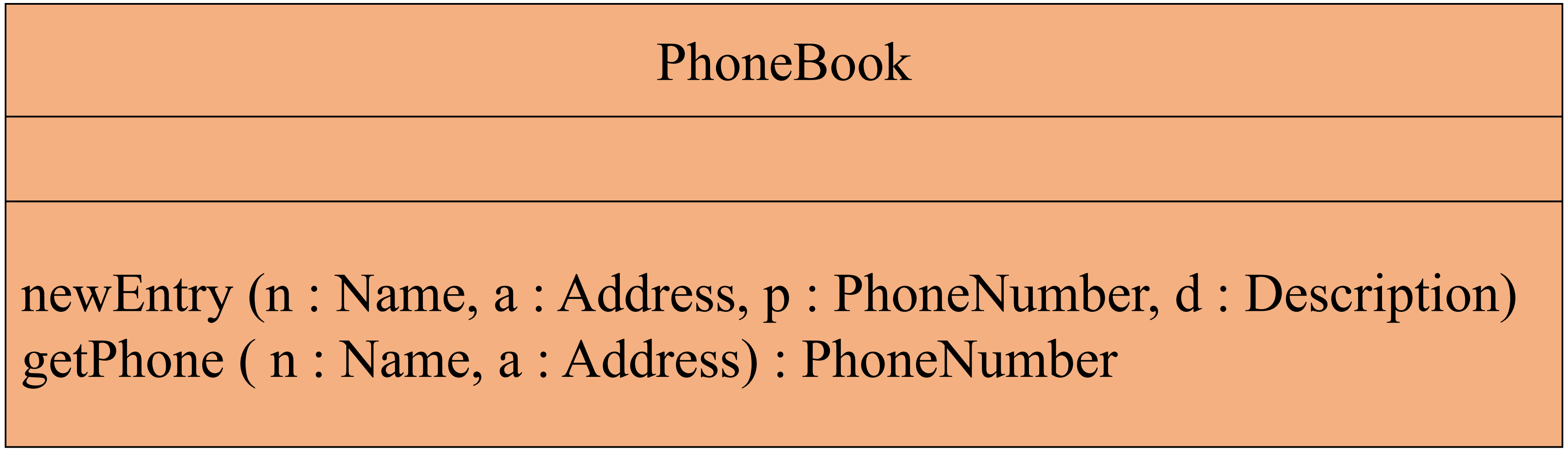


Class Operations



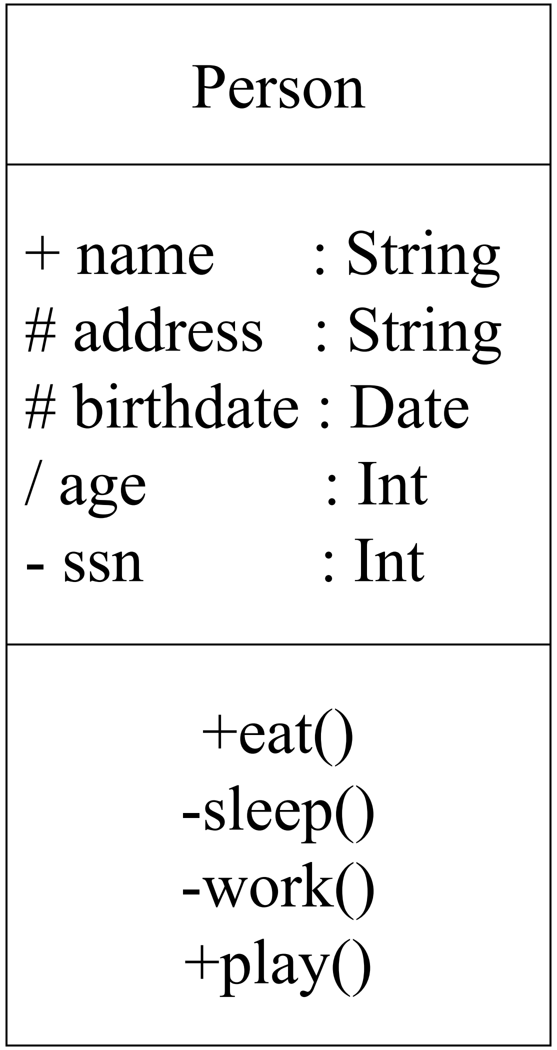
Operations describe the class behavior and appear in the third compartment.

Class operations


PhoneBook
newEntry (n : Name, a : Address, p : PhoneNumber, d : Description)
getPhone (n : Name, a : Address) : PhoneNumber

You can specify an operation by stating its signature: listing the name, type, and default value of all parameters, and, in the case of functions, a return type.

Visibility



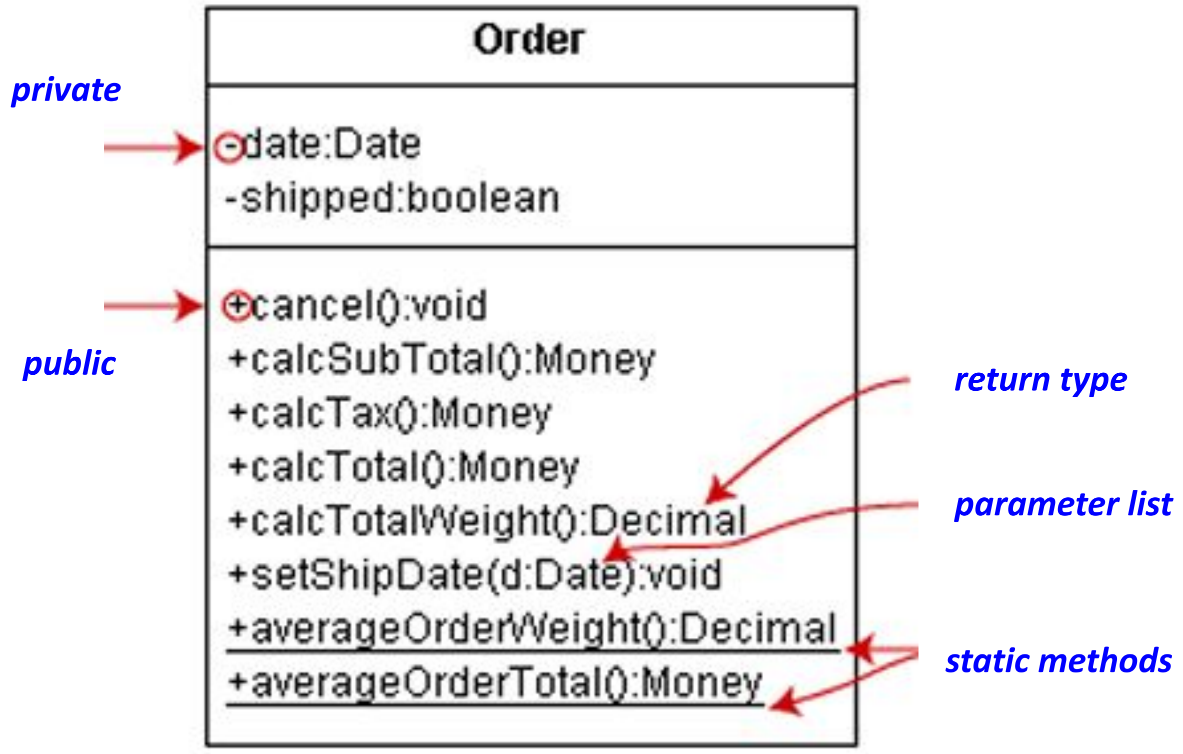
attributes and operations can be declared with different visibility modes:

+ public: any class can use the feature (attribute or operation);

# protected: any descendant of the class can use the feature;

- private: only the class itself can use the feature.

Complete Class



Note: Static members are underlined

Finding Classes

Finding classes in use case, or in text descriptions:
Look for nouns and noun phrases in the description of a use case or a problem statement;
These are only included in the model if they explain the nature or structure of information in the application.

Don't create classes for concepts which:
Are beyond the scope of the system;
Refer to the system as a whole;
Duplicate other classes;
Are too vague or too specific (few instances)

Finding classes in other sources:
Reviewing background information;
Users and other stakeholders;
Analysis patterns;
CRC (Class Responsibility Collaboration) cards.

Approaches for identifying classes

1. Noun phrase approach
2. Common class patterns approach
3. Classes, responsibilities and collaborators (CRC) approach
4. Use case driven approach (discussed in previous session)

1. Noun phrase approach

- Proposed by Rebecca, brian and lauren.
- The requirement document need to be read and noun phrase should be identified.
- Nouns to be considered as classes and verbs to be methods of the classes.
- All plurals changed to singular.
- Nouns are listed and divided into three categories:
  relevant, fuzzy and irrelevant classes
- Irrelevant classes need to be identified .
- Candidate classes can be identified from other two categories.

- Identifying tentative classes:
  - Look for noun and noun phrases in the use cases.
  - Some classes are implicit or taken from general knowledge.
  - Carefully choose and define class names.
- Finding classes is an incremental and iterative process.

selecting classes from the relevant and fuzzy categories

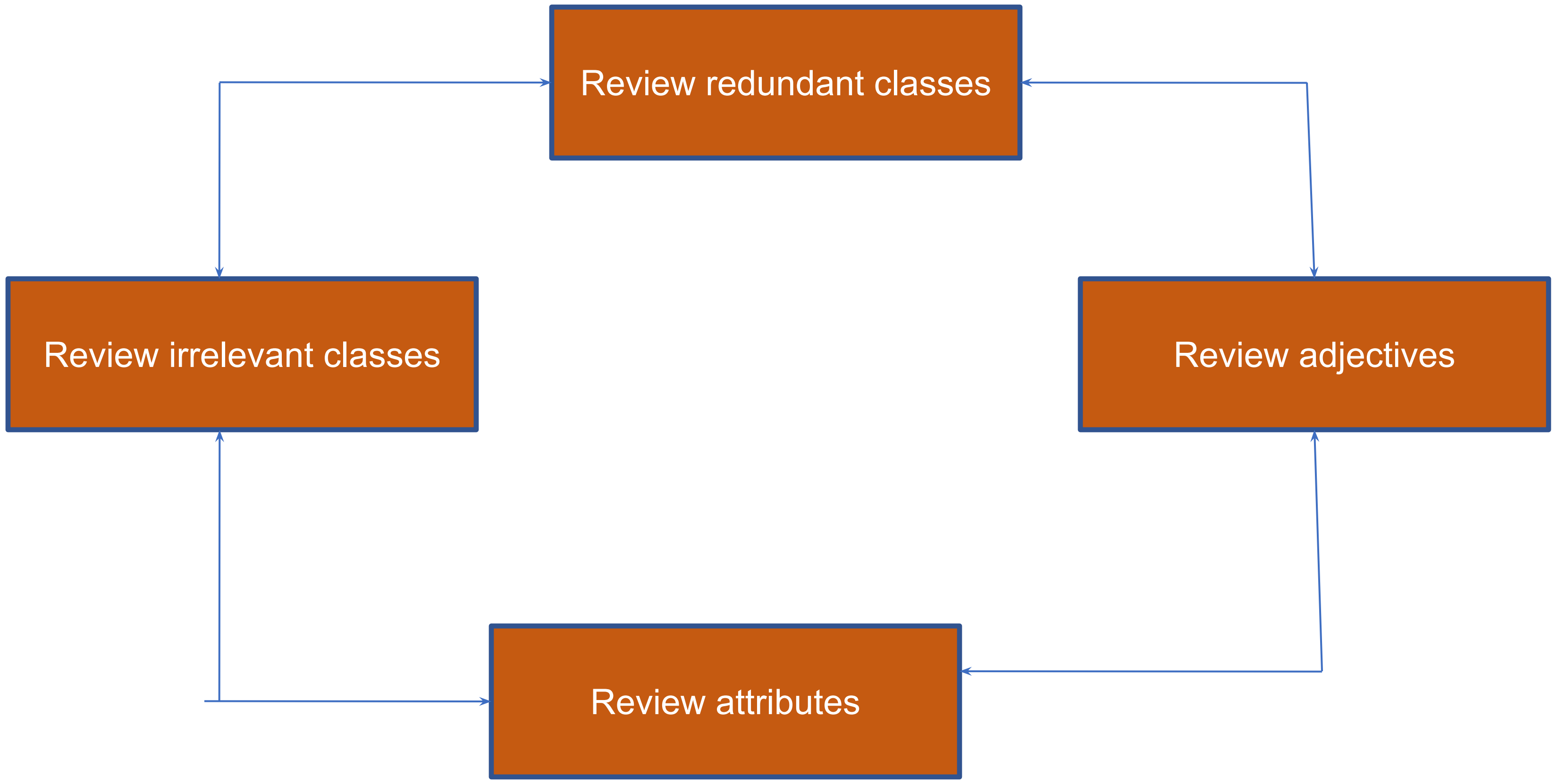
- Guidelines to select candidate classes from the relevant and fuzzy categories:
  - Redundant classes: select one class if more than one class describes the same information. This is part of building a common vocabulary for the whole system. Choose the word that is used by the user.
  - Attribute classes: Tentative objects that are used only as values should be defined or restarted as attributes and not as a class. For example client status can be the attribute not the class.

Object Oriented Analysis and Design with Java

  - Irrelevant classes: class should be defined clearly with its purpose. If the purpose of statement is not possible for any class, class should be eliminated.

- The process of identifying classes and refining is the iterative process.
- This is not the sequential process. You can move back and forth.

Object Oriented Analysis and Design with Java



2. Common class patterns approach

- Researchers like shlaer and mellor proposed some patterns for identifying the candidate class and objects
- Concept class:
  - Emphasis principles that are not tangible but used to organize or keep track of business activities or communications.
  - Ex: performance

- Event class:
  - These are points in time that must be recorded.
  - Things happen, usually to something else at given date and time or as a step in an ordered sequence.
  - Associated with things remembered are attributes such as who,what, when, where, how or why.
  - Ex: Landing,interrupt

Object Oriented Analysis and Design with Java

- Organization class:
  - it’s the collection of people, resources,facilities or groups to which the users belong.
  - Ex: An accounting department might be considered a potential class.

- People class:
  - It specifies different roles users play in interacting with the application.

  - *Two types:*
    - the users of the system (operator or clerk) or who interacts with the system.
    - Who do not use the system but whom information is kept by the system.

- Place class:
  - Places are physical locations that the system must keep information.
  - Ex: buildings, stores and offices
- Tangible things and device class:
  - This includes physical objects or groups of objects that are tangible.
  - The devices with which the application interacts.
  - Ex: cars,pressure sensors

CRC (Class-Responsibility-Collaborators)

CRC cards are tool used for brainstorming in OO design

CRC cards are created from Index cards and each member of the brain storming session with write up one CRC card for each relevant class/object of their design

Objects need to interact with other objects (Collaborators) in order to fulfill their responsibilities

Since the cards are small, prevents to get into details and give too many responsibilities to a class

They can be easily placed on the table and rearranged to describe and evolve the design

What is a CRC Card?

CRC stands for Class, Responsibility and Collaboration.

- Class

  An object-oriented class name

  Include information about super- and sub-classes

- Responsibility

  What information this class stores

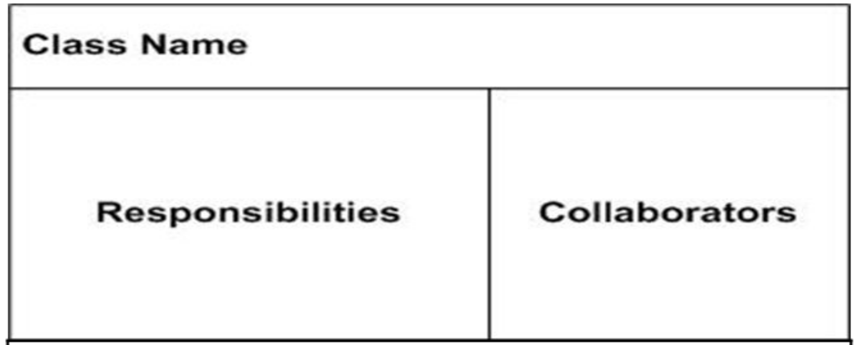
  What this class does

  The behaviour for which an object is accountable

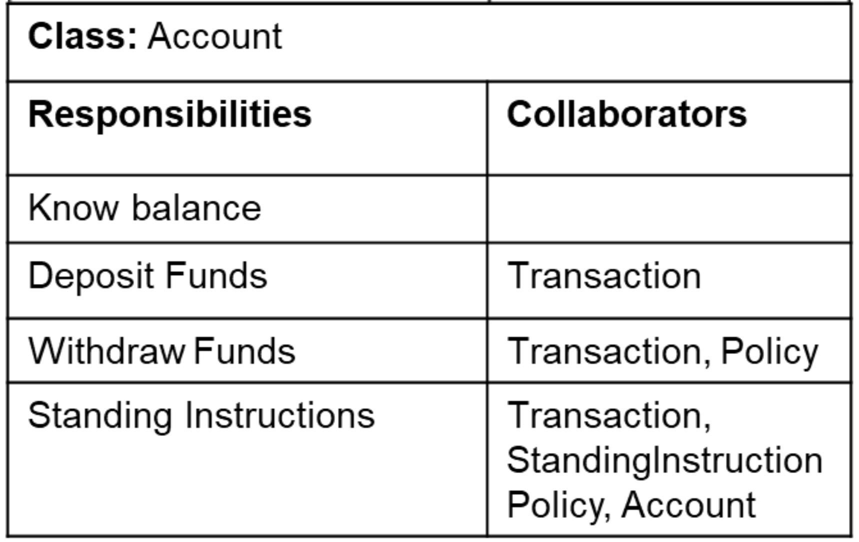
- Collaboration

  Relationship to other classes

  Which other classes this class uses

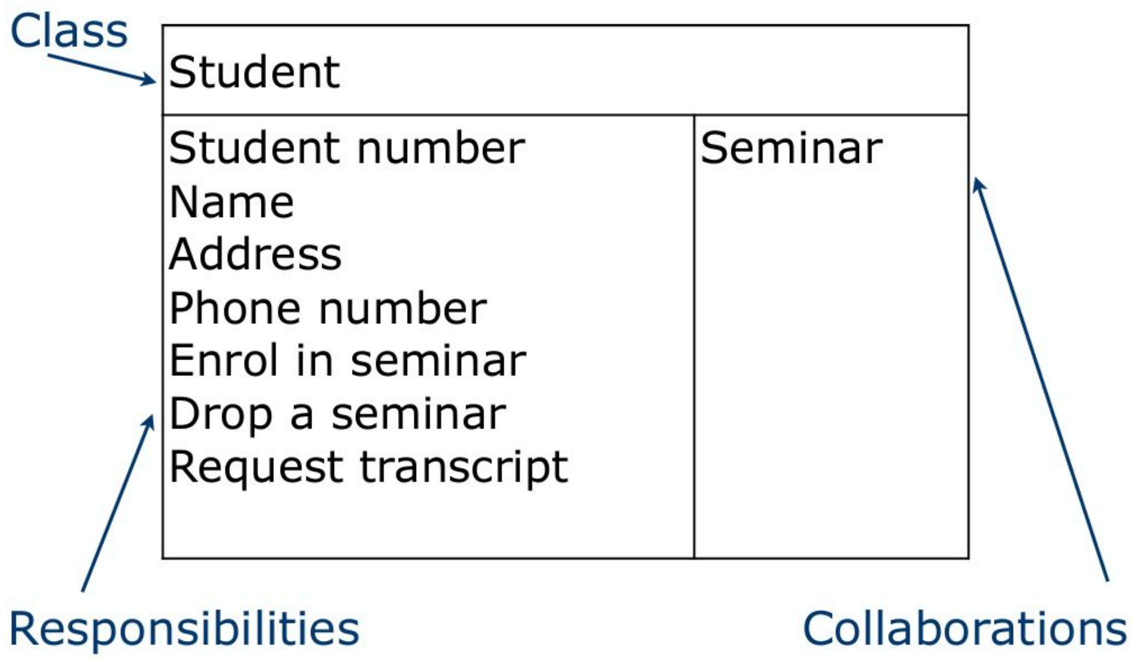


Class Name	
Responsibilities	Collaborators



Class: Account	
Responsibilities	**Collaborators**
Know balance	
Deposit Funds	Transaction
Withdraw Funds	Transaction, Policy
Standing Instructions	Transaction, StandingInstruction Policy, Account

CRC Card – Example



Students only have information about themselves (their names and so forth), and not about seminars. What the student needs to do is collaborate/interact with the card labeled *Seminar* to sign up for a seminar. Therefore, *Seminar* is included in the list of collaborators of *Student*.

A responsibility is anything that a class knows or does.
For example, students have names, addresses, and phone numbers. These are the things a student knows.
Students also enroll in seminars, drop seminars, and request transcripts. These are the things a student does.

How CRC Cards are used?

CRC cards are an aid to a group role-playing activity .
Index cards are used in preference to pieces of paper due to their robustness and to the limitations that their size (approx. 15cm x 8cm) imposes on the number of responsibilities and collaborations that can be effectively allocated to each class.

A class name is entered at the top of each card and responsibilities and collaborations are listed underneath as they become apparent.
Each collaboration is normally listed next to the corresponding responsibility.
From a UML perspective, use of CRC cards is in analyzing the object interaction that is triggered by a particular use case scenario.

The process of using CRC cards is usually structured as follows.
1.Conduct a session to identify which objects are involved in the use case.

2. Allocate each object to a team member who will play the role of that object.

3. Act out the use case.

4. Identify and record any missing or redundant objects.

How CRC Cards are used?

Before beginning a CRC session it is important that all team members are briefed on the organization of the session and a CRC session should be preceded by a separate exercise that identifies all the classes for that part of the application to be analyzed.

The team members to whom these classes are allocated can then prepare for the role playing exercise by considering in advance a first-cut allocation of responsibilities and identification of collaborations.

Here , it is important to ensure that the environment in which the sessions take place is free from interruptions and free for the flow of ideas among team members.

During a CRC card session, there must be an explicit strategy that helps to achieve an appropriate distribution of responsibilities among the classes.

THANK YOU

J.Ruby Dinakar

Department of Computer Science and Engineering

rubydinakar@pes.edu