
MAHARISHI UNIVERSITY of MANAGEMENT
Engaging the Managing Intelligence of Nature
Computer Science Department
CS401 Modern Programming Practices (MPP)
Professor Paul Corazza
1

Lecture 1: The OO Paradigm for Building Software Solutions

Unlocking the Blueprint of Creation

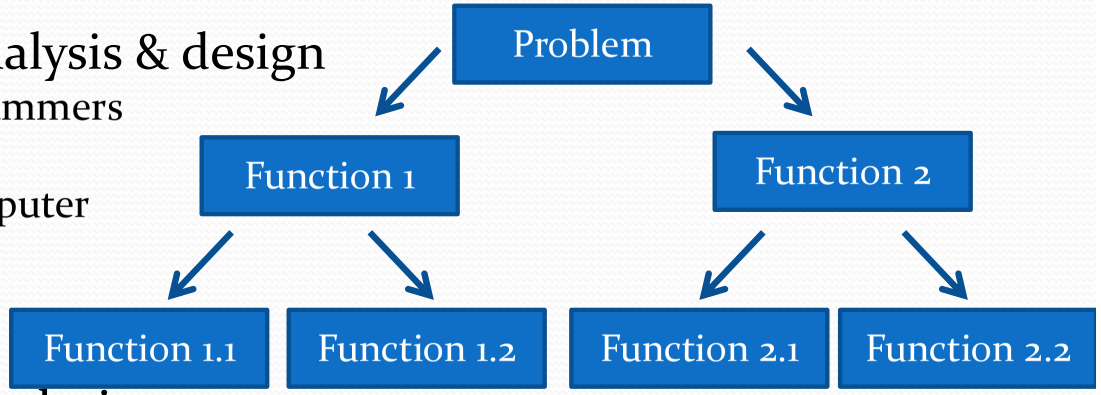
Wholeness Statement

In the OO paradigm of programming, execution of a program involves objects interacting with objects. *Analysis* is the process of understanding user requirements and discovering which objects are involved in the problem domain and their relationships, attributes, and behavior. *Design* turns these discovered objects into a web of software objects from which a fully functioning system is built. Each object has a type, which is embodied in a Java *class*. The intelligence underlying the functioning of any software object resides in its underlying class, which is the silent basis for the dynamic behavior of objects. Likewise, pure consciousness is the silent level of intelligence that underlies all expressions of intelligence in the form of thoughts and actions in life.

Origin of OO

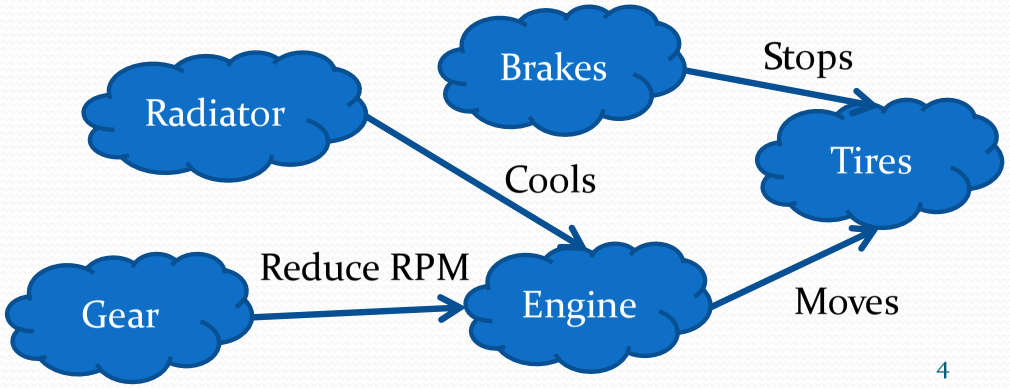
- Procedural analysis & design

In early days, programmers adapted real-world problems into "computer logic"



- OO analysis & design

Using OO, real-world objects are represented by software objects; real-world behavior by sending messages between objects



Object Oriented Principles

- Objects have *state*, *behavior*, and *identity*
- Encapsulation and Data Hiding
- Inheritance (Generalization)
- Polymorphism and Late Binding
- Delegation

The Goal

- We want to build a software system based on objects interacting with objects, following best practices of the OO paradigm
  Demo: `lesson1.lecture.objectdemo`
- Example: Recall the car example
- When we achieve this, there are obvious benefits:
  - Easy to maintain
  - Easy to extend and reuse
  - Easy to understand
- To achieve the goal, there are two important steps before writing code:
  - *Analysis:* Understand *what* is needed and model these requirements
  - *Design:* Determine *how* to put the elements discovered in analysis into a system of software objects that function together in a way that meets the user requirements

Steps to Achieve the Goal: The Software Development Lifecycle (SDLC)

To build a software system, these are the key steps in the process:

1. User Requirements (Analysis): Determine as precisely as possible what problem you are trying to solve and what requirements must be met in your software solution. The first step is some kind of *Problem Statement;* the next step is done by working out *Use Cases*
2. Create a Static Model (Analysis): Based on use cases and the problem statement, determine what the objects of the system are going to be. In this step you identify the *classes* and create a *class diagram.* Initially, you will identify classes and attributes (properties) for each class.
3. Add Relationships to the Static Model (Analysis). Determine from the use cases how classes in your model should be related. *Associations* and *dependencies* between classes help to identify how each class should behave and what services each should provide.

(continued)

4. Create a Dynamic Model (Analysis): For each important flow of a use case, see how a request from a user of the system should be handled by the classes you have identified. You accomplish this when you create *sequence diagrams*. Build sequence diagrams by studying use cases and by reviewing the associations and dependencies that have been included in your class diagrams. Sequence diagrams help to identify what responsibilities each of your classes will have; what services each of your classes should provide.
5. Enhance the Static and Dynamic Models (Design): Design is concerned with *how* to build the system. Previous steps were concerned with modeling the use cases but design is concerned with how to put all the ideas together to build a system. One aspect of design is the intelligent use of abstract classes and interfaces.
6. Transform UML into Code UML is like an architect's blueprint – it provides clear guidelines for the design, but turning the blueprint into a final product requires additional skills. Code is developed in conjunction with unit tests that verify correctness.

UML

UML (Unified Modeling Language) allows us to build a map of "objects-interacting-with-objects". It provides a language of diagrams for both analysis and design and supports each step of the SDLC. UML diagrams support:

1. Understanding user requirements (analysis) – in the form of a *use case diagram*
2. Representing the classes or key abstractions within the problem statement and use cases, in the form of a *class diagram*
3. Modeling the flow of the application, as determined by use cases, in the form of *sequence diagrams*.

Some Types of UML Diagrams

- Use Case Diagram – shows in a single diagram all the use cases for the system
- Class Diagram – shows attributes and operations in each of the (primary) classes of the system as well as relationships between them. Used in the Static Model during analysis, and later in design
- Sequence Diagram – shows the flow of communication between the running objects of the system, driven by the use cases of the system (e.g.: In an ATM system, a use case "withdraw money"; a sequence diagram will show how the system processes the request to withdraw money). Used for the Dynamic Model.
- Object Diagram – shows how, at a particular moment in time, how all the instances of the classes communicate with each other. This is part of the Dynamic Model
- Many Others! [*See the sample diagrams in the materials accompanying this lecture*]

Objectives for Next Few Lessons

- Modeling a Problem with UML – use the Student Registration System as an example
  - Use Case Diagram – to specify the use cases
  - Class Diagram – to specify the objects embedded in the problem statement and use cases
  - Sequence Diagrams – to model the flow of the application and identify behaviors and responsibilities of classes
- Coding
  - Convert UML Diagrams to OO code
  - Learn best practices for code and fundamental programming concepts
- Development of Consciousness
  - Regular practice of TM
  - Connecting CS to SCI and back to CS

Main Point 1

Software is by nature complex, and the only way to manage this complexity is through *abstraction*.

Abstraction is at work when we discover the objects in the problem domain during analysis, and work with these to build a system during design. Abstraction is also at work in creating maps of our objects in the form of UML diagrams.

In a similar way, to manage the complexities of life itself the technique is to saturate awareness with its more abstract levels so that all the details of any situation are appreciated from the broadest perspective. The abstract levels of awareness are experienced in the processs of transcending.

Overview

- The Student Registration System (SRS) Problem Statement.
- SRS use cases and Use Case Diagram
- SRS static model – first steps in building a Class Diagram

Problem Description for the SRS

We have been asked to develop an automated Student Registration System (SRS) for the university. This system will enable students to register online for courses each semester, as well as track their progress toward completion of their degree.

When a student first enrolls at the university, he/she uses the SRS to create a plan of study that lists the courses he/she plans on taking to satisfy a particular degree program, and chooses a faculty advisor. The SRS will verify whether or not the proposed plan of study satisfies the requirements of the degree that the student is seeking.

Once a plan of study has been established, then, during the registration period preceding each semester, students are able to view the schedule of classes online and choose whichever classes they wish to attend, indicating the preferred section (day of the week and time of day) if the class is offered by more than one professor.

The SRS will verify whether or not the student has satisfied the necessary prerequisites for each requested course by referring to the student's online transcript of courses completed and grades received (the student may review his/her transcript online at any time).

Assuming that (a) the prerequisites for the requested course(s) are satisfied, (b) the course(s) meet(s) one of the student's plan of study requirements, and (c) there is room available in each of the class(es), the student is enrolled in the class(es).

If (a) and (b) are satisfied, but (c) is not, the student is placed on a first-come, first-served wait list. If a class/section that he/she was previously waitlisted for becomes available (either because some other student has dropped the class or because the seating capacity for the class has been increased), the student is automatically enrolled in the waitlisted class, and an email message to that effect is sent to the student. It is the student's responsibility to drop the class if it is no longer desired; otherwise, he/she will be billed for the course.

Students may drop a class up to the end of the first week of the semester in which the class is being taught.

Plan for Lesson 1

- ☑ The Student Registration System (SRS) Problem Statement.
- ☐ SRS use cases and Use Case Diagram
- ☐ SRS static model – first steps in building a Class Diagram

Use Case Model for the Student Registration System

What is a Use Case?

A Use Case is a sequence of steps performed by a user, interacting with the system, for the purpose of achieving some goal.

(continued)

A ***Use Case Description*** describes the different flows that might occur in a Use Case and clearly indicates the steps in each flow: user actions and system responses. The Main Flow is the expected sequence of steps, but there are often other flows (Alternate Flows) in which the goal fails to be reached or is achieved in a different way.

Example Think of an ATM machine as a software system. Use cases for this system include:

Check Balance Withdraw Money Deposit Money

CHECK_BALANCE Use Case Description: Main Flow

User Action	System Response
1. User types in PIN into main screen	1. System checks validity of PIN and presents options to user on another screen
2. User selects "Check Balance"	2. System looks up user account and displays balance on another screen

(continued)

Some Use Cases for the Student Registration System:
What kinds of things should a user of the system expect to be able to do?

A Student should be able to:
- Register for a course
- Drop a course
- View schedule of classes

A Professor should be able to
- Post grades
- View a class list
- Update course description

Exercise 1.1: Use Case Description for Register Use Case

Fill in the Use Case description table below. Use the Problem Statement to determine the different user actions and system responses for the main flow of the Register For Class use case.

REGISTER_FOR_CLASS Use Case Description: Main Flow

User Action	System Response
1.	1.
2.	2.

A Solution

REGISTER_FOR_CLASS Use Case Description: Main Flow

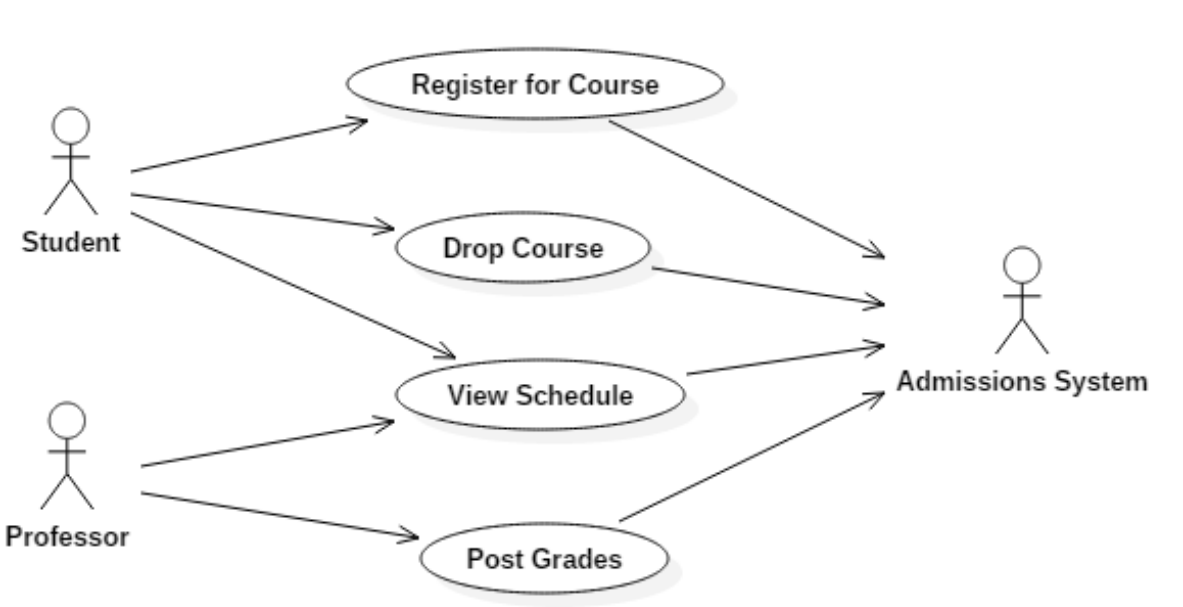
User Action	System Response
1. Student requests at main screen to view schedule of classes	1. System displays schedule of classes
2. Student selects a course and the preferred section of the course	2. System looks verifies student's eligibility for the course and availability of the section, and then enrolls the student according to the request.

Cataloging All Use Cases

- A *Requirements Specification* for a project lists all the use cases along with use case descriptions.
- A full account of use cases also requires a list of Actors and which uses cases each Actor interacts with.
- An Actor is any entity that either initiates action in the system (like a user) or that is acted upon by the system (like a database). Actors are *external to* the system.
- Name some Actors in the Student Registration System:

  Student, Professor, Registrar, Admissions System

A Partial Use Case Diagram for the Student Registration System



Main Point 2

Use cases are the unifying thread that runs through all stages of the development lifecycle. Therefore, proper formulation of the use cases is a central key for a successful project. Success in life also requires access to the thread that ties all diversity together. Bringing awareness to this unified level of life brings the ability to handle the diversity of circumstances, challenges, and personalities that one faces in life.

Upanishads
Know that by which all this is known

Plan for Lesson 1

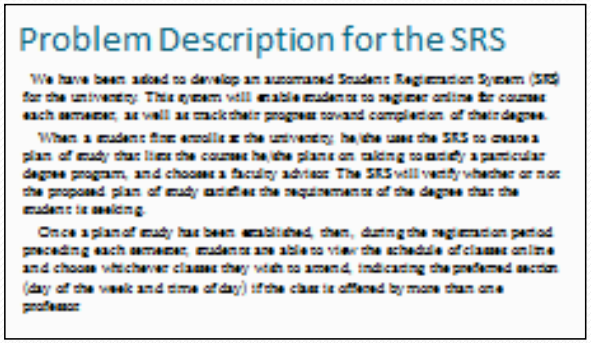
- ☑ The Student Registration System (SRS) Problem Statement.
- ☑ SRS use cases and Use Case Diagram
- ☐ SRS static model – first steps in building a Class Diagram

Exercise 1.2: Start Building the Static Model: Find *Noun Phrases*

In your small group create a list of all the *noun phrases* from the problem description.

Examples:

- student
- plan of study
- wait list



Problem Description for the SRS

We have been asked to develop an automated Student Registration System (SRS) for the university. This system will enable students to register online for courses each semester, as well as track their progress toward completion of their degree.

When a student first enrolls at the university, he/she uses the SRS to create a plan of study that lists the courses he/she plans on taking to satisfy a particular degree program, and chooses a faculty advisor. The SRS will verify whether or not the proposed plan of study satisfies the requirements of the degree that the student is seeking.

Once a plan of study has been established, then, during the registration period preceding each semester, students are able to view the schedule of classes online and choose whichever classes they wish to attend, indicating the preferred section (day of the week and time of day) if the class is offered by more than one professor.

Problem Description for SRS

We have been asked to develop an automated Student Registration System (SRS) for the university. This system will enable students to register online for courses each semester, as well as track their progress toward completion of their degree.

When a student first enrolls at the university, he/she uses the SRS to set forth a plan of study as to which courses he/she plans on taking to satisfy a particular degree program, and chooses a faculty advisor. The SRS will verify whether or not the proposed plan of study satisfies the requirements of the degree that the student is seeking.

Once a plan of study has been established, then, during the registration period preceding each semester, students are able to view the schedule of classes online, and choose whichever classes they wish to attend, indicating the preferred section (day of the week and time of day) if the class is offered by more than one professor.

The SRS will verify whether or not the student has satisfied the necessary prerequisites for each requested course by referring to the student's online transcript of courses completed and grades received (the student may review his/her transcript online at any time).

Assuming that (a) the prerequisites for the requested course(s) are satisfied, (b) the course(s) meet(s) one of the student's plan of study requirements, and (c) there is room available in each of the class(es), the student is enrolled in the class(es).

If (a) and (b) are satisfied, but (c) is not, the student is placed on a first-come, first-served wait list. If a class/section that he/she was previously waitlisted for becomes available (either because some other student has dropped the class or because the seating capacity for the class has been increased), the student is automatically enrolled in the waitlisted class, and an email message to that effect is sent to the student. It is the student's responsibility to drop the class if it is no longer desired; otherwise, he/she will be billed for the course.

Students may drop a class up to the end of the first week of the semester in which the class is being taught.

List of Noun Phrases (SRS)

system
students
courses
semester
progress
completion
degree
student
university
plan of study
courses
degree program
faculty advisor
plan of study
requirements of degree
student
plan of study
registration period

semester
students
schedule of classes
classes
preferred section
day of the week
time of day
class
professor
student
prerequisites
requested course
student
transcript
courses completed
grades received
student
transcript

student
waitlisted class
email message
student
responsibility
class
course
Students
class
end

NOTES:
1. Many duplicates
2. Prefer singular to plural ("student" instead of "students")

Sort and Eliminate Duplicates (SRS)

class
class/section that a student was previously wait-listed for
completion
course
courses completed
day of the week
degree
degree program
email message
end
faculty advisor
first-come, firstserved wait list
grades received
plan of study
plan of study requirements
preferred section
prerequisites

(continued)

professor
progress
registration period
requested course
requirements of degree
responsibility
room
schedule of classes
seating capacity
semester
student
system
time of day
transcript
university
waitlisted class

Streamline the List Further

- Eliminate terms that do not seem to be objects or that are essentially duplicates, such as: 'completion', 'end', 'progress', 'responsibility', and 'requirements of the degree'. (Note: 'requirements' will be wrapped into 'plan of study requirements'.)
- Eliminate reference to the system itself (SRS) and to "university" – our system will (probably) not need to maintain/modify information about the university itself.
- It may be hard to decide about some terms (like 'registration period' – we expect this term will be used in a different way later). Retain list of eliminated terms, so you can use them later if necessary.

Final List of Noun Phrases (SRS)

class
class/section that he/she was previously wait-listed for
course
courses completed
day of the week
degree
degree program
email message
faculty advisor
first-come, firstserved wait list
grades received
plan of study
plan of study requirements
preferred section
prerequisites
professor
requested course
room
schedule of classes
seating capacity
section
semester
student
system
time of day
transcript
waitlisted class

Terminology: These noun phrases are called Key Abstractions in software engineering

Main Point 3

The OO approach to building software solutions is to represent objects and behavior in the problem domain with software objects and behavior. One of the first steps in this process is to *locate* the objects implicit in the problem statement, and this is done by examining *nouns* and *noun phrases* ("key abstractions") in the problem statement. These words and phrases link the real world situation to the abstract realm of software objects. Likewise, linking individual awareness to its abstract foundation in fully expanded awareness is the basis for creating solutions to the real-world challenges of life.

Exercise 1.2, continued

The next step is to group together terms that are closely related, that belong together, that can be classified with a single concept.

- Example: *class, course, waitlisted class* belong together
- Note: Sometimes this step requires the assistance of a domain expert because sometimes there is a need to discriminate between subtle shades of meaning
- Exercise: In small groups, group together terms that belong together.

Group "Synonyms" (SRS)

class
course
waitlisted class
class/section that he/she was previously wait-listed for
preferred section
requested course
section
prerequisites

day of the week

degree
degree program

email message

faculty advisor
professor

first-come, firstserved wait list

plan of study
plan of study requirements

room
schedule of classes
seating capacity
semester
student
system
time of day

courses completed
grades received
transcript

Choose Class Name (SRS)

class
course
waitlisted class
class/section that he/she was previously wait-listed for
preferred section
requested course
section
prerequisites

* Words in bold indicate best choices

- Avoid choosing nouns that imply roles in a relationship between objects. For example, “prerequisite” is a role in an association between two courses.
  “Waitlisted class”?
  “Preferred section”?

(continued)

class
course
waitlisted class
class/section that he/she was previously wait-listed for
preferred section
requested course
section
prerequisites

day of the week

degree
degree program

email message

faculty advisor
professor

first-come, firstserved wait list

plan of study
plan of study requirements

room
schedule of classes
seating capacity
semester
student
system
time of day

courses completed
grades received
transcript

Note: Prefer shorter expressions to longer ones ("degree" instead of "degree program")

Note: The notion of "transcript" *includes* "courses completed" and "grades received" although they are not actually synonyms.

Which Nouns Should Become *Classes?*

- Are there any attributes for this class? [Sometimes, can think of attributes for a class but they are not relevant. Example in original noun list: "University"]
- Are there any services that would be expected of objects in this class? [Typically, a class will provide services, which in Java are represented by its public methods.]
- Can this item simply be included as an attribute of another class?
  [Example: Should "room" be a class on its own, or an attribute of "section"? Which others can we treat as just attributes?]

Examples of Noun Phrases That Are Attributes Rather Than Classes (SRS)

- Day of week
- Degree
- Seating capacity
- Semester
- Time of Day

Examples of Noun Phrases That Are Attributes Rather Than Classes (SRS)

- Day of week – attribute of Section
- Degree – attribute of Student
- Seating capacity – attribute of Section
- Semester – attribute of Section
- Time of Day – attribute of Section

Ignore Implementation Classes During Analysis

Two Main Types of Classes:

1. Domain Classes: abstractions that the end user will recognize and that represent real-world entities.
2. Implementation Classes: introduced solely to hold the application together (example: a dictionary to look up students based on ID number, or a special type of list to keep track of professors).

During Analysis, keep only Domain Classes; the others will be useful during design.

SRS Implementation Classes

- Email message (sending an email is a behavior we need but the message itself is an implementation class)
- Schedule of classes (could be a domain class – for now, think of it as a "computed value" – assembled from other information in the system and displayed to the user in a UI)

Final List of Classes Derived from Noun Phrases (SRS)

Course (rather than Class)
PlanOfStudy
Professor
Section
Student
Transcript

Data Dictionary of Classes (SRS)

- **Course**: a semester-long series of lectures, assignments, exams, etc. that all relate to a particular subject area, and which are typically associated with a particular number of credit hours; a unit of study toward a degree. For example, 'Software Engineering' is a required **course** for the Master of Science Degree in Computer Science.
- **Plan of Study**: a list of the **courses** that a student intends to take to fulfill the **course** requirements for a particular degree.

(continued)

- **Professor**: a member of the faculty who teaches **sections** and/or advises **students**.
- **Section**: the offering of a particular **course** during a particular semester on a particular day of the week and at a particular time of day (for example, **course** 'Software Engineering' is taught in the Spring 2012 semester on Mondays from 1:00 – 3:00 PM).
- **Student**: a person who is currently enrolled at the university and who is eligible to register for one or more **sections**.

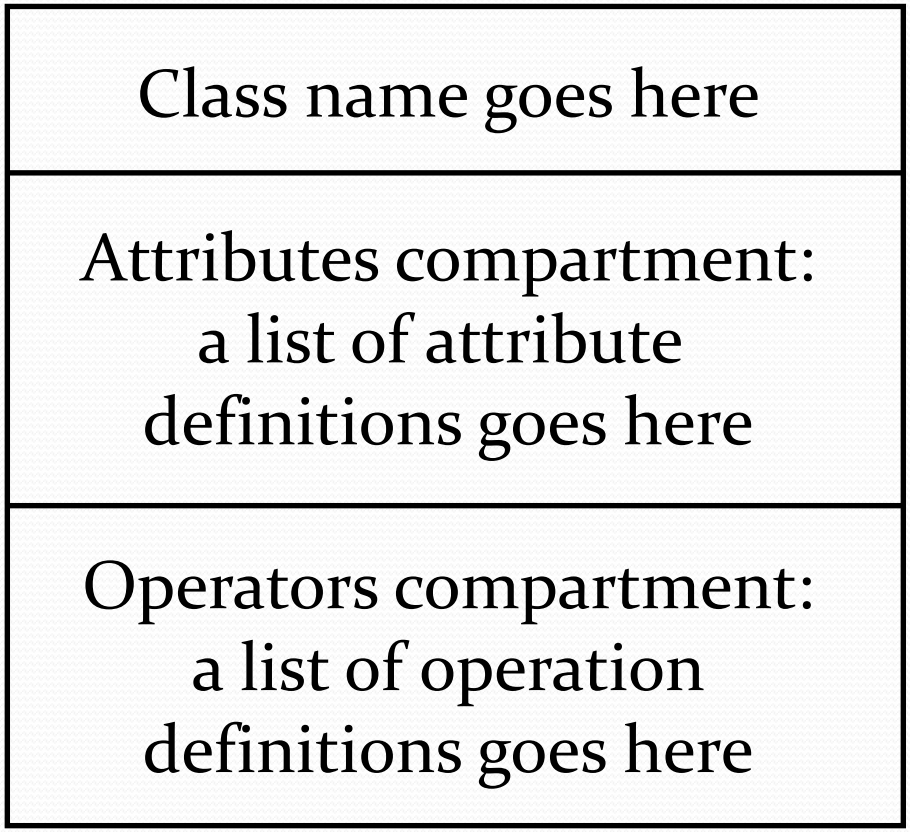
(continued)

- **Transcript**: a record of all of the **courses** taken to date by a particular **student** at this university, including which semester each **course** was taken in, the grade received, and the credits granted for the **course**, as well as reflecting an overall total number of credits earned and the **student's** grade point average (GPA).

The Class Model

- Gradually add detail to your UML classes: To begin, locate *attributes* and (next lesson) *relationships* between classes.
- *The Process: From Analysis to Design:*
  1. At first we try to understand the user requirements (*analysis*) and use UML to lay out the classes that are involved.
  2. Later, as we add more detail and understand the requirements more completely, we enhance our UML class diagrams with *design* elements and techniques for *building* the application.

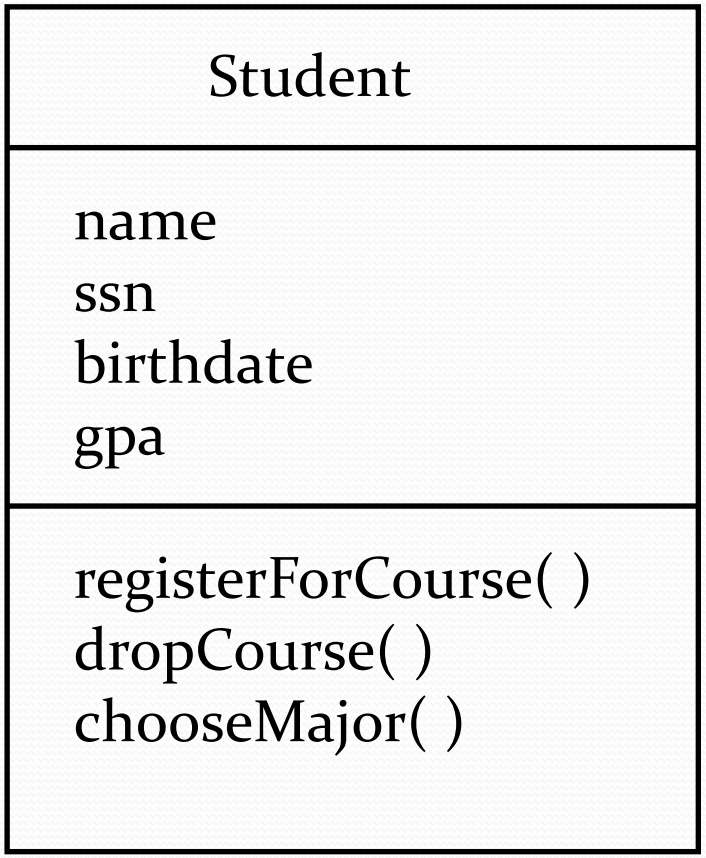
The Class Diagram



Exercise 1.3: Class Diagrams

- Create a UML class for Student in the SRS problem.
- Look back at the problem description and the definition of a Student.
  - What attributes naturally belong to Student?
  - What operations belong with Student? (We will discuss techniques for identifying operations in a later lesson.)

Solution: Student Class Diagram



Identifying Attributes

- Use requirements to find attributes of domain classes
- Use your prior knowledge of the domain to help find attributes (e.g. each student has an ID number)
- Talk to the domain expert (often you're not the expert)
- Examine old SRS system already in use to find attributes
- Note: Trying to understand domain classes in this way is part of the process of analysis.

Identifying Operations

- To identify operations, we need to know how our classes are supposed to *behave* and what their *responsibilities* are.
- One way to begin is to identify *relationships* between classes, represented in UML as *associations*.
- Associations can be further analyzed to help specify operations for each class.

Main Point 4

A class encapsulates *data,* stored as attributes, and *behavior*, represented as operations. These are the static and dynamic aspects of any class, and a UML diagram for a class provides compartments for each of these.

These two aspects of a class – data and behavior – are aspects of anything the we encounter in life. They give expression to the reality that life, at its basis, is a field of *existence* and *intelligence*.

Plan for Lesson 1

- ☑ The Student Registration System (SRS) Problem Statement.
- ☑ SRS use cases and Use Case Diagram
- ☑ SRS static model – first steps in building a Class Diagram

Connecting the Parts of Knowledge With the Wholeness of Knowledge

1. Class diagrams display the data and behaviors of a class
2. Class diagrams provide an (abstract) representation of a specific real world problem domain.

---

3. **Transcendental Consciousness** is the simplest state of awareness, where the mind goes beyond thoughts and concepts to the most abstract level of awareness – the "abstract content" of awareness
4. **Wholeness moving within itself**: in Unity Consciousness one experiences that all objects in the universe arise from pure consciousness and are ultimately nothing but consciousness.