Module 6 – Unified Modeling Language (UML)

UML is used in this chapter to develop object models.

6.2 Objects

DFDs treated data and processes separately – an object will include the data *and* the processes that affect the data.
Ex: customerObject contains name, address, account number, current balance. Customers can also place an order, pay a bill, change their address, etc.
Ex: How UML might describe a family with parents and children.

```
object PARENT:
  attribute name
  attribute age
  attribute sex
  attribute hair color

  method read_bedtime_story
  method drive_carpool_van
  method prepare_school_lunch

  if message FROM child == "good night" { read_bedtime_story() }
  if message FROM parent == "drive" { drive_carpool_van() }

object CHILD:
  attribute name
  attribute age
  attribute sex
  attribute hair_color
  attribute num_of_siblings

  method pick_up_toys
  method eat_dinner
  method get_ready_for_bed

  if message from PARENT == "dinner's ready" { eat_dinner() }
  if message from PARENT == "tuck away" { pick_up_toys() }
```

6.3 Attributes

Objects can have a specific attribute called **state**.
Ex:*

```
STATE past_student
STATE current_student
STATE future_student
```

6.4 Methods

Methods are tasks or functions that the object performs when it receives a message.
Ex:*

```
object CAR performs OPERATE_WIPERS method when sent a message from wiper con
object CAR performs APPLY_BRAKES method when sent a message from brake pedal
```

Ex 2:

```
method MORE_FIELDS {
```

```
method MORE_FRIES {
                  heat_oil();
                  fill_fry_basket();
                  lower_basket();
                  check_fries();
                  drain_basket();
                  deposit_into_warming_tray();
                  add_salt();
}
```

6.5 Messages

The same message to two different objects can yield two different results – this is called **polymorphism**.
Ex 1:

```
message to PARENT { read_bedtime_story(); }
message to DOG { go_to_sleep(); }
message to CHILD { get_ready_for_bed(); }
```

An object can be viewed as a **black box**, because a message to the object triggers changes within the object without specifying how the changes must be carried out. This is an example of **encapsulation**, i.e. all data and methods are self-contained.

encapsulation allows objects to be used as modular components throughout the system, and prevents its internal code from being altered.
Ex 2:

```
message FROM instructor TO student_record = "ENTER GRADE";
```

6.6 Classes

An object belongs to a group or category called a **class**.

```
object 4_runner in class suv;
object mustang in class car;
object f_150 in class truck;
```

All objects within a class share common attributes and methods. Objects can also be grouped into **subclasses**, which are more specific categories within a **class**.

```
object truck in class vehicles;
object car in class vehicles;
object mini_van in class vehicles;

class vehicle;
  attribute make;
  attribute model;
  attribute year;
  attribute weight;
```

A class can also belong to a more general category called a **superclass**.
Ex 1:*

```
class novel in superclass book;
class hardcover in class novel;
class paperback in class novel;
class digital in class novel;
```

Ex 2:

```
class person;
subclass employee;
sub-subclass instructor;
```

6.7 Relationships Among Objects and Classes

Relationships enable objects to communicate and interact as they perform business functions and transactions as required by the system. They describe what objects need to know about each other and how they should respond to each other.

The strongest relationship is called **inheritance**, which enables an object, called a **child**, to derive one or more of its attributes from its **parent** object.

In *Ex 2* from **6.6**, instructor inherits traits from its **employee** superclass, and **employee** inherits many traits from its **person** parent class.

After objects, classes, and relationships are defined – an **object relationship diagram** can be prepared to provide system overview. The model would show objects and how they interact to perform business functions and transactions.

6.8 The Unified Modeling Language (UML)

Structured analysis uses DFDs to model data and processes. Object Oriented analysis uses UML to document and model a system.

6.8.1 Use Case Modeling

A **use case** represents the steps in a specific business function or process. An external entity, called an **actor**, initiates a use case by requesting the system to perform a function or process.
Ex 1:*

```
actor patient { make_appointment();}
```

*The symbol for use case is oval with a label that describes the action or event. The actor is shown as a stick figure, with a label that identifies their role. The line from the actor to the use case is called an **association**, because it links a particular actor to a use case.*

1. Identify the actors and the functions or transactions they initiate.
2. For each use case, develop a **use case description** in the form of a table.
   (a) Document the name of the use case, the actor, description of use case, a step-by-step list of the tasks and actions required for completion, description of alternative courses of action, preconditions, postconditions, and assumptions.

Table 1: ADD NEW STUDENT Use Case

Attribute	**Description**
Name	Add New Student
Actor	Student/Manager
Description	Describes the process used to add a student to a fitness-class
Successful Completion	1. Manager checks *FITNESS-CLASS SCHEDULE* object for availability 2. Manager notifies student 3. Fitness-class is open and student pays fee 4. Manager registers student
Alternative	1. Manager checks *FITNESS-CLASS SCHEDULE* object for availability 2. *Fitness-class* is full 3. Manager notifies student
Precondition	Student requests *fitness-class*
Postcondition	Student is enrolled in *fitness-class* and fees have been paid
Assumptions	None

6.8.2 Use Case Diagrams

A **use case diagram** is a visual summary of several related use cases within a system or subsystem.

1. Identify the **system boundary** which shows what is includes in the system and what is not included in the system.

2. Place use cases on the diagram, add actors, and show relationships.