Chapter 1- Introduction to Object Oriented Programming with Java

By Biruk M.

Outline

- All that is to know on OOP
- Introduction
- Structural Programming
- Object Oriented Programming
- Basic Features
  - Class & Objects
  - Abstraction & Encapsulation
  - Inheritance and Polymorphism
- OOP-Design
- Exercise

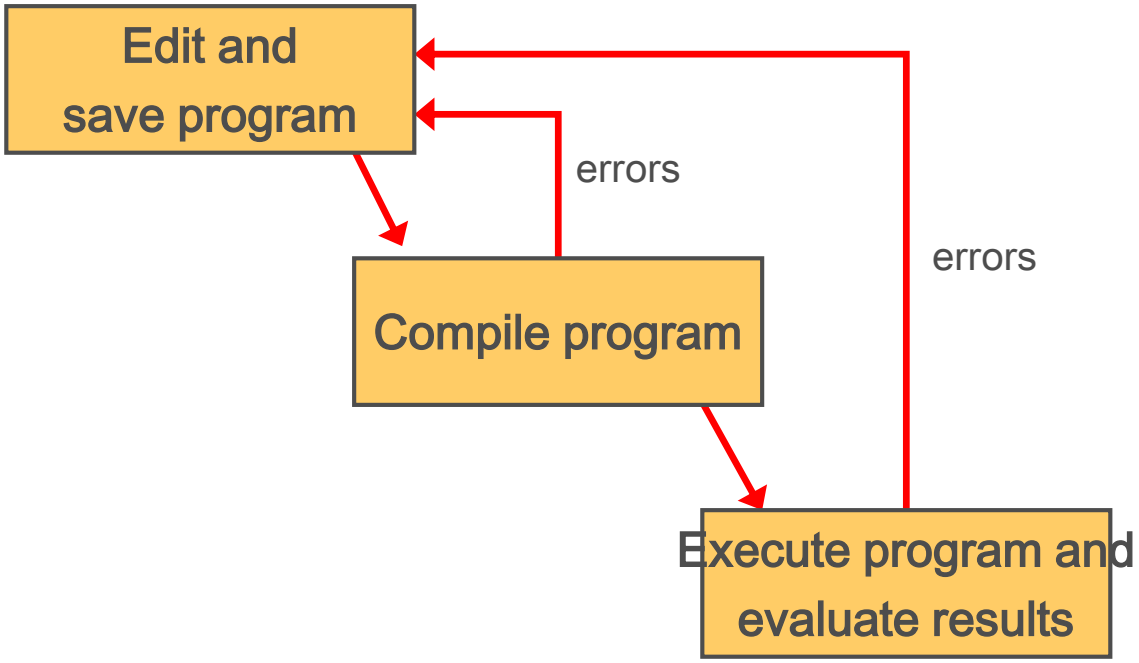
Program Development

The mechanics of developing a program include several activities

- writing the program in a specific programming language (such as Java)
- translating the program into a form that the computer can execute
- investigating and fixing various types of errors that can occur

Software tools can be used to help with all parts of this process

Basic Program Development



Programming Languages

- A *programming language* specifies the words and symbols that we can use to write a program
- A programming language employs a set of rules that dictate how the words and symbols can be put together to form valid *program statements*
- Each type of CPU executes only a particular *machine language*
- A program must be translated into machine language before it can be executed
- A *compiler* is a software tool which translates *source code* into a specific target language
    - Often, that target language is the machine language for a particular CPU type. The Java approach is somewhat different

Problem Solving

The purpose of writing a program is to solve a problem

Solving a problem consists of multiple activities:

- Understand the problem
- Design a solution
- Consider alternatives and refine the solution
- Implement the solution
- Test the solution

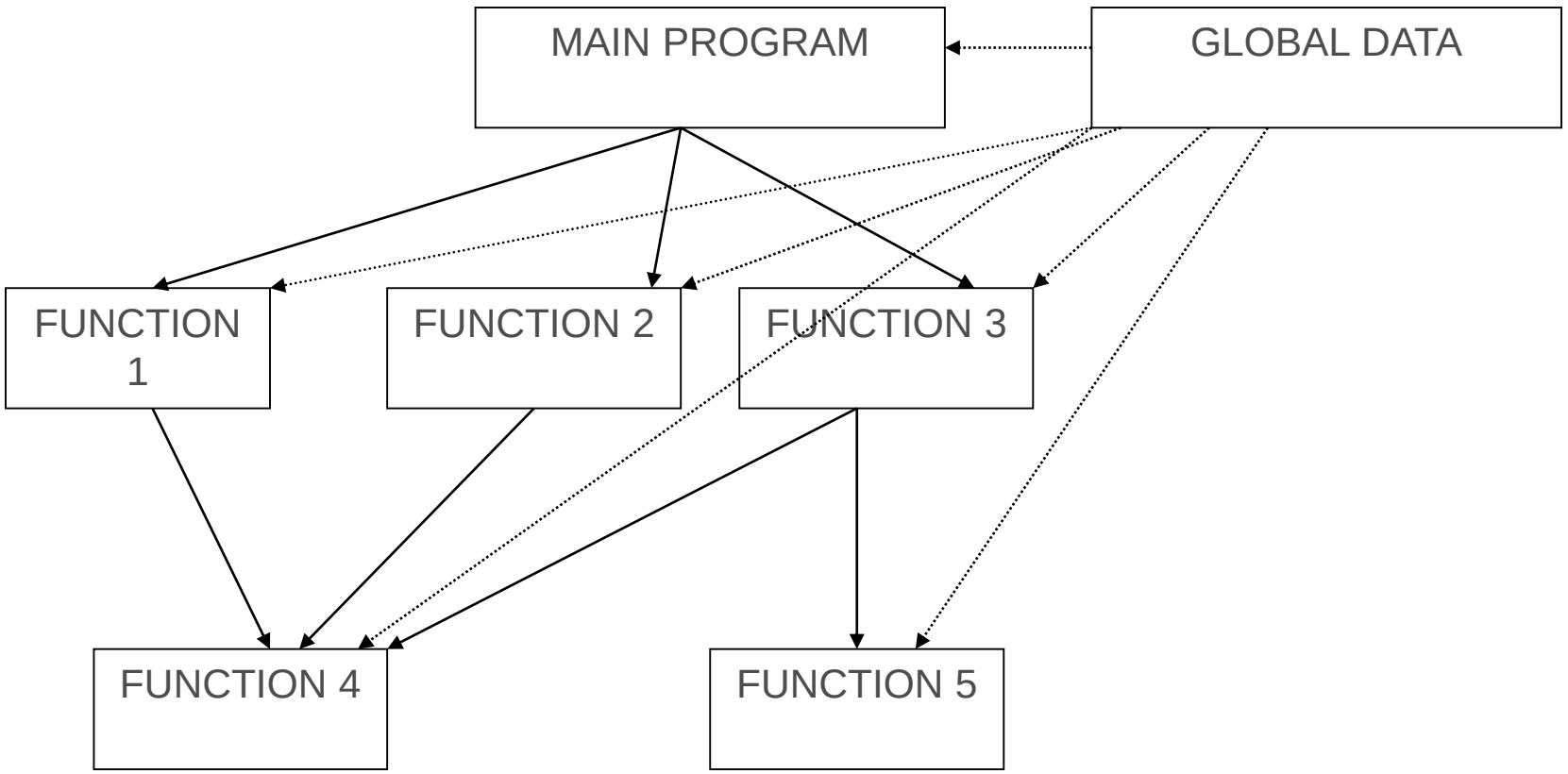
These activities are not purely linear – they overlap and interact

Problem Solving

- The key to designing a solution is breaking it down into manageable pieces
- When writing software, we design separate pieces that are responsible for certain parts of the solution
- An *object-oriented approach* lends itself to this kind of solution decomposition
- We will dissect our solutions into pieces called objects and classes

STRUCTURED vs. OO PROGRAMMING

STRUCTURED PROGRAMMING:



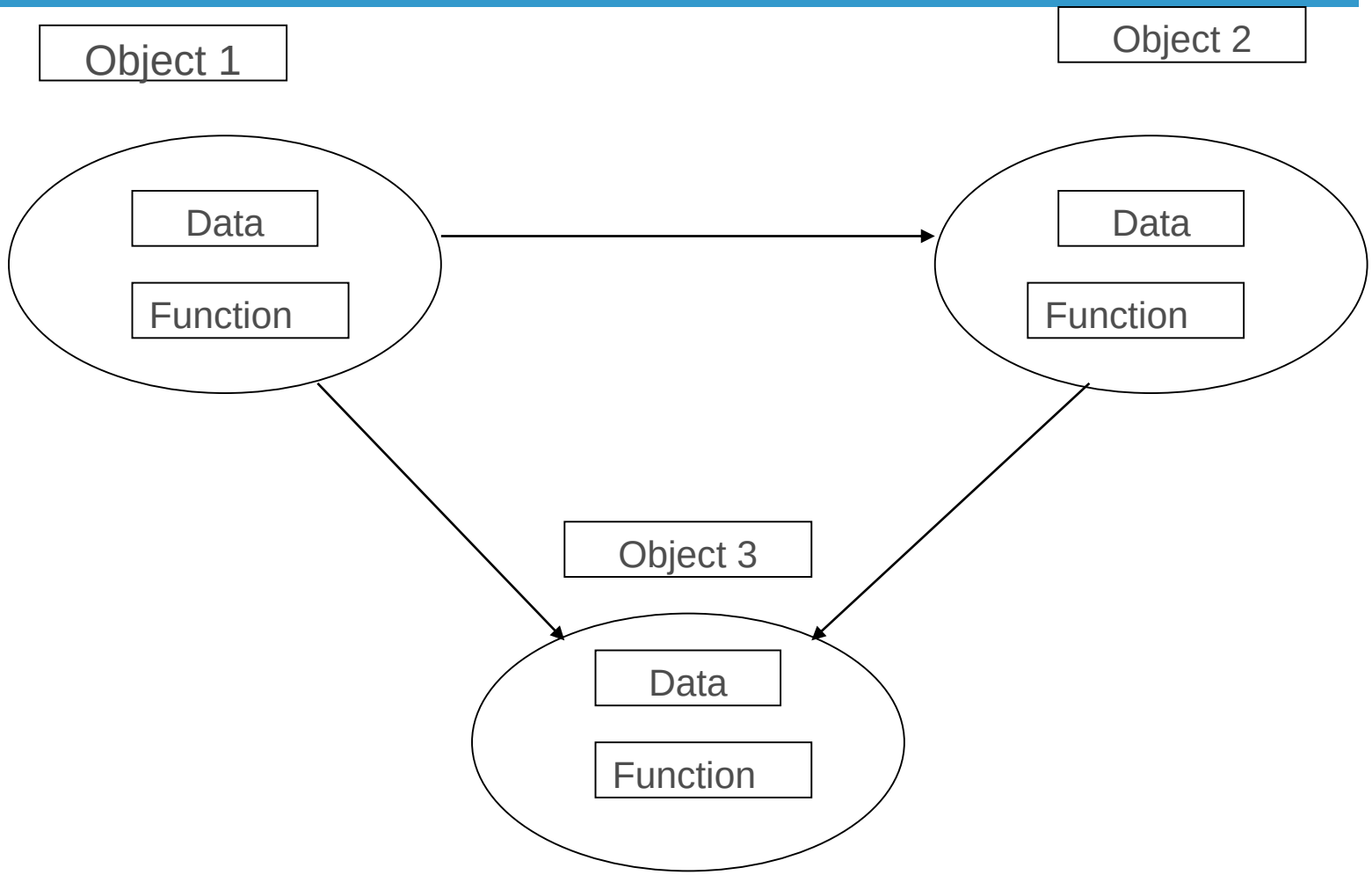
Structured Programming

Using function

Function & program is divided into modules

Every module has its own data and function which can be called by other modules.

Object Oriented Programming



OBJECT ORIENTED PROGRAMMING

- Objects have both data and methods
- Objects of the same class have the same data elements and methods
- Objects send and receive *messages* to invoke actions

Key idea in object-oriented:

The real world can be accurately described as a collection of objects that interact.

Object-Oriented Programming

- Java is an object-oriented programming language
- As the term implies, an object is a fundamental entity in a Java program
- Objects can be used effectively to represent real-world entities
- For instance, an object might represent a particular employee in a company
- Each employee object handles the processing and data management related to that employee

O-O is a different Paradigm

Central questions when programming.

- Imperative Paradigm:
  - What to do next ?
- Object-Oriented Programming
  - What does the object do ? (vs. how)

Central activity of programming:

- Imperative Paradigm:
  - Get the computer to do something.
- Object-Oriented Programming
  - Get the object to do something.

Why OOP?

- Save development time (and cost) by reusing code
  - once an object class is created it can be used in other applications
- Easier debugging
  - classes can be tested independently
  - reused objects have already been tested

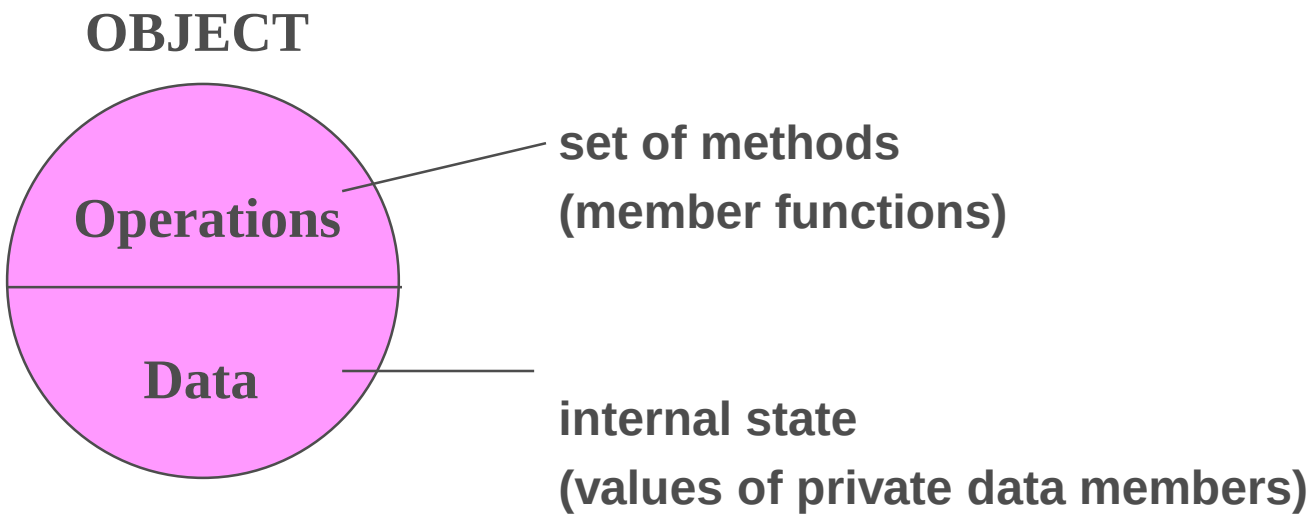
Object Oriented Programming Features

A methodology of programming

Four (Five ?) major design principles:

1. Class & Objects
2. Data Abstraction.
3. Encapsulation(Information Hiding).
4. Polymorphism (dynamic binding).
5. Inheritance. (particular case of polymorphism ?)

Classes and Objects



Class

The class is like a cookie cutter

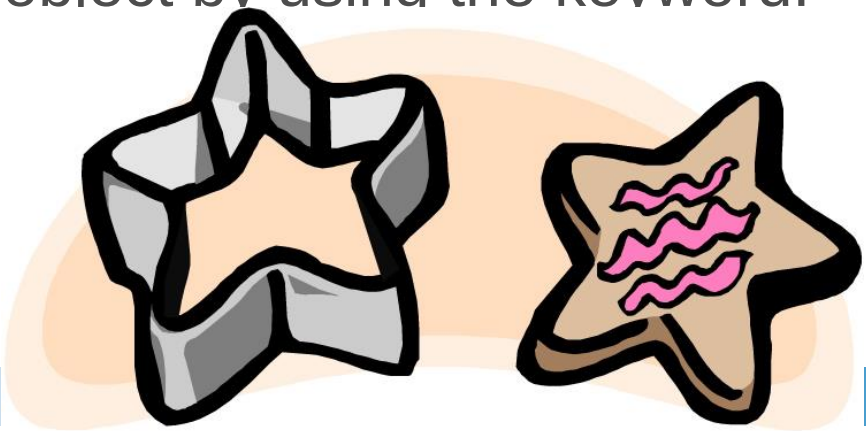
- It knows how much space each object needs (shape)
- Many objects can be created from the class

To create objects we ask the object that defines the class to create it

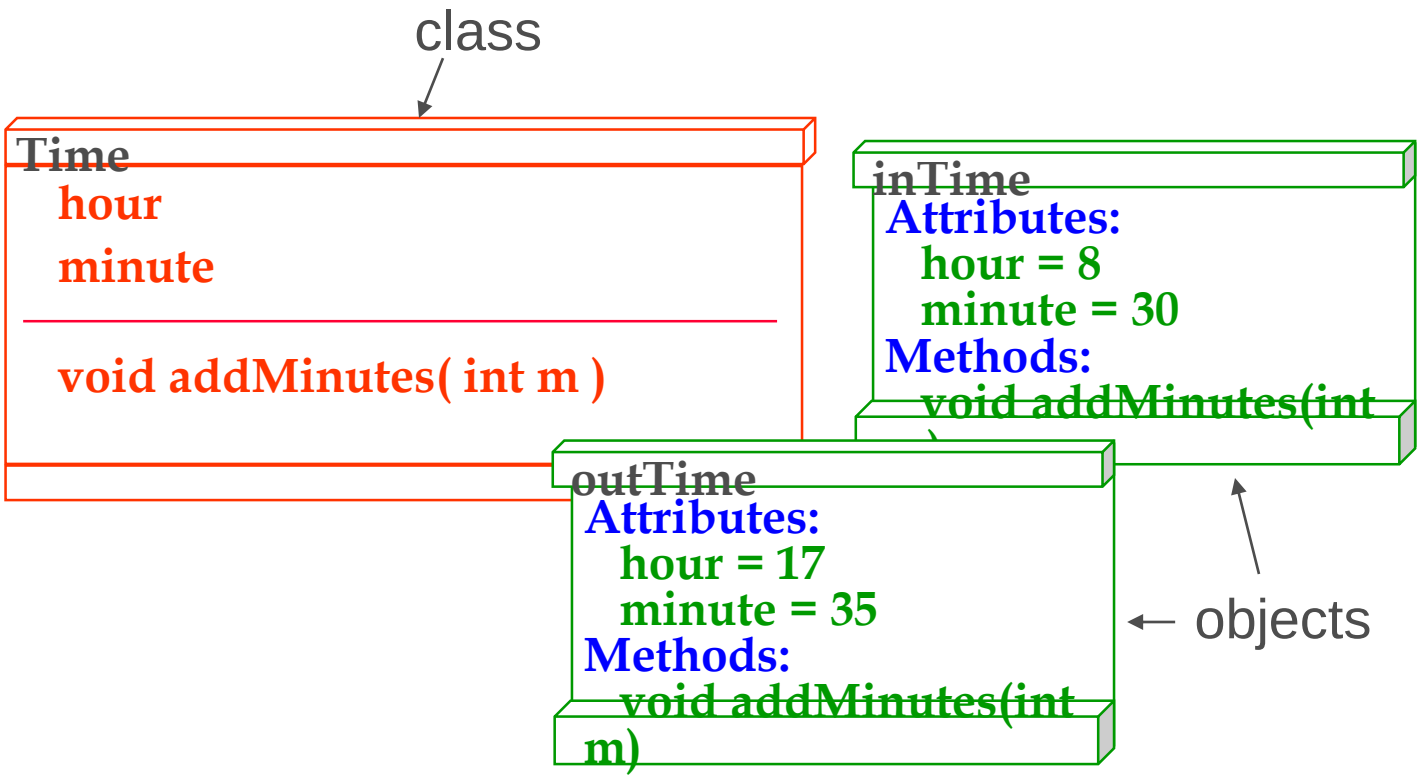
- Each object is created in memory with space for the fields it needs
- Each object keeps a reference to the class that created it

A class is like a factory that creates objects of that class

We ask a class to create an object by using the keyword:
new *ClassName*

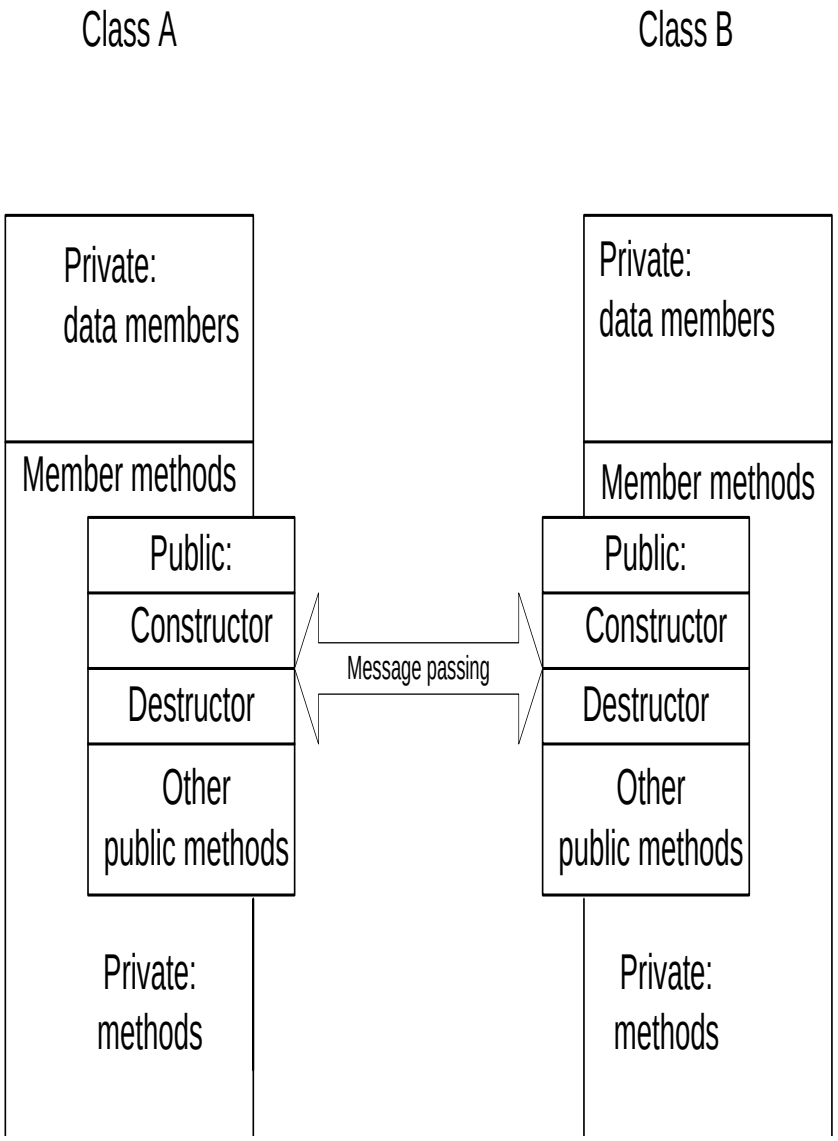


Objects


class
Time
hour
minute
void addMinutes(int m)
inTime
Attributes:
hour = 8
minute = 30
Methods:
void addMinutes(int
outTime
Attributes:
hour = 17
minute = 35
Methods:
void addMinutes(int
m)
objects

acting Objects

- cts don't "tell" each other what to do
  - y "ask" each other to do things
- cts can refuse to do what they are ked
  - object must protect it's data
    - ot let it get into an incorrect state
    - bank account object shouldn't let you withdraw more oney that you have in the account



Classes and Objects Declaration

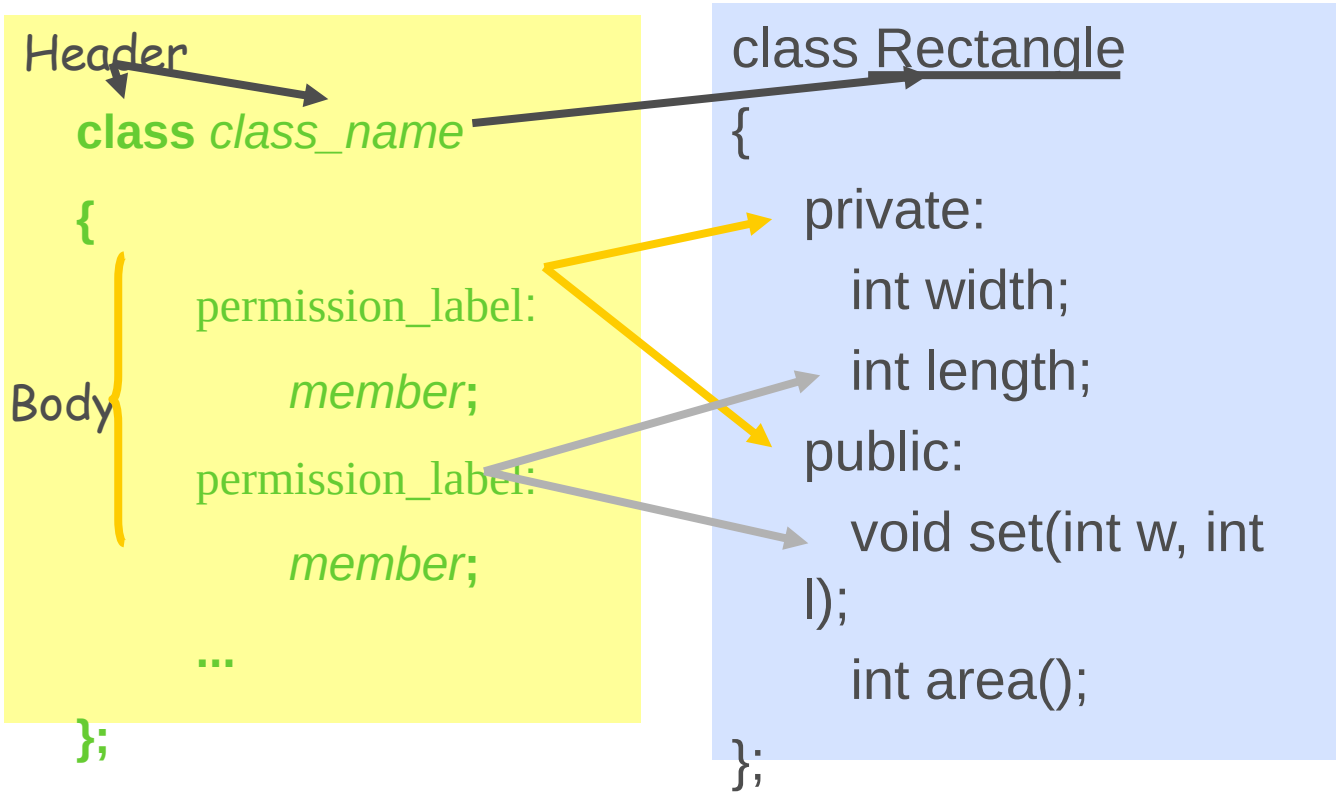
A class will look like this:

<Access-Modifier> class *MyClass* {
**   // field, constructor, and method declarations**
}

To instantiate an object we will do:

***MyClass* instance = new MyClass(<constructor params>);**

C++ class Declaration



Example for attributes and methods

Attributes:

- manufacturer's name
- model name
- year made
- color
- number of doors
- size of engine
- etc.

Methods:

- Define data items (specify manufacturer's name, model, year, etc.)
- Change a data item (color, engine, etc.)
- Display data items
- Calculate cost
- etc.

Concept: An object has behaviors

In old style programming, you had:

- data, seems to be passive and functions, which could manipulate any data

An object contains both data and methods that manipulate that data

- An object is *active,* not passive; it *does* things
- An object is *responsible* for its own data
  - But: it can *expose* that data to other objects

Concept: An object has state

An object contains both data and methods that manipulate that data

- The data represent the state of the object
- Data can also describe the relationships between this object and other objects

Example: A CheckingAccount might have

- A balance (the internal state of the account)
- An owner (some object representing a person)

Example: A “Rabbit” object

You could (in a game, for example) create an object representing a rabbit

It would have data:

- How hungry it is
- How frightened it is
- Where it is

And methods:

- eat, hide, run, dig

Concept: Classes describe objects

Every object belongs to (is an instance of) a class

An object may have fields, or variables

- The class describes those fields

An object may have methods

- The class describes those methods

A class is like a template, or cookie cutter

Concept: Classes are like Abstract Data Types

An Abstract Data Type (ADT) bundles together:

- some data, representing an object or "thing"
- the operations on that data

Example: a CheckingAccount, with operations deposit, withdraw, getBalance, etc.

Classes enforce this bundling together

Summary

- *object*

  - usually a person, place or thing (**a noun**)

- *method*

  - an action performed by an object (**a verb)**

- *attribute*

  - description of objects in a class

- *class*

  - a category of similar objects (such as *automobiles)*

  - *does not hold any values of the object's attributes*

Abstraction

Over time, data abstraction has become essential as programs became complicated.

Benefits:

1. Reduce conceptual load (minimum detail).
2. Fault containment.
3. Independent program components.

   (difficult in practice).

Code reuse possible by extending and refining abstractions.

Abstraction

- Focus only on the important facts about the problem at hand
- to design, produce, and describe so that it can be easily used without knowing the details of how it works.

Analogy:

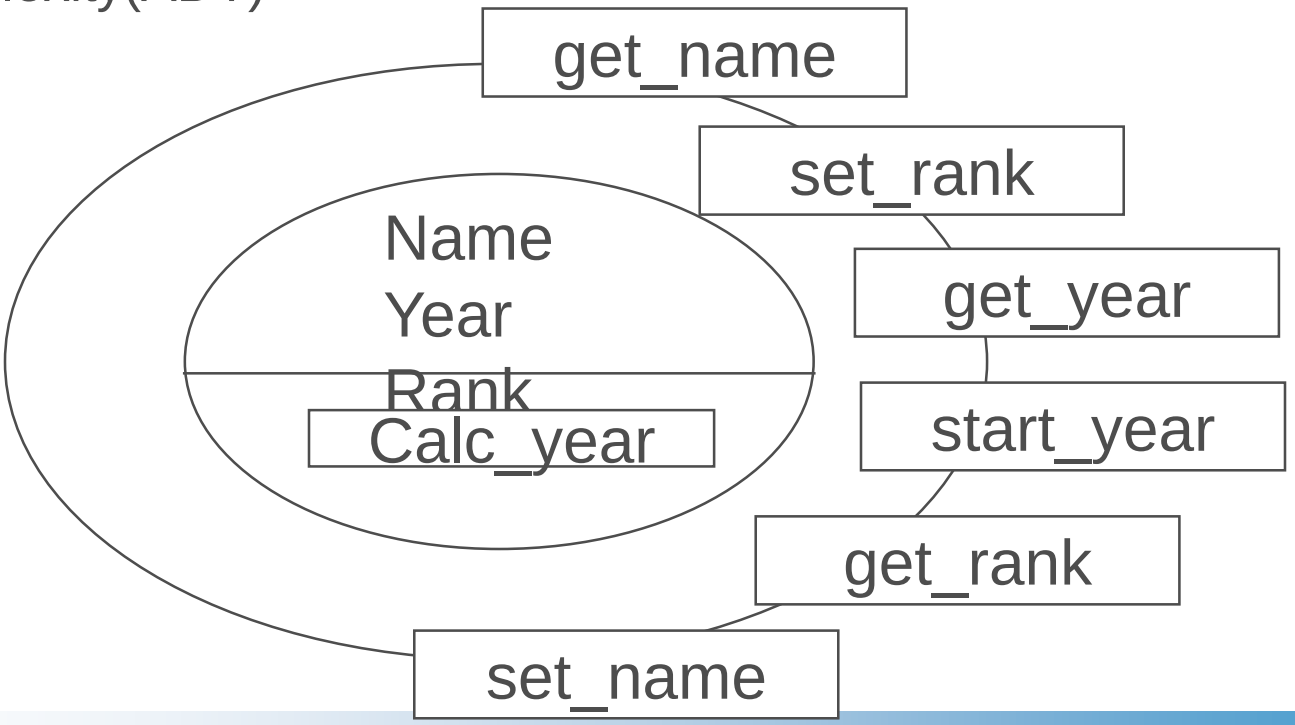
When you drive a car, you don't have to know how the gasoline and air are mixed and ignited.

Instead you only have to know how to use the controls.

Draw map

Abstraction

- Ignore details when appropriate
  - Think about what a method/object/class does, not how it does it
- One of the fundamental ways in which we handle complexity(ADT)



Encapsulation

- Also known as *data hiding*
- Only object's methods can modify information in the object.
- Is the mechanism that binds together code and the data object manipulates
- Put related things in the same place(group related data and operations in an object)

  Each object has its own data and knows how to use it
- Hide internal representation/implementation

Analogy:

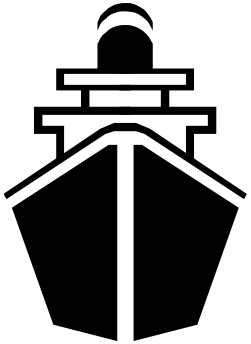
- ATM machine can only **update accounts** of one person or object only.

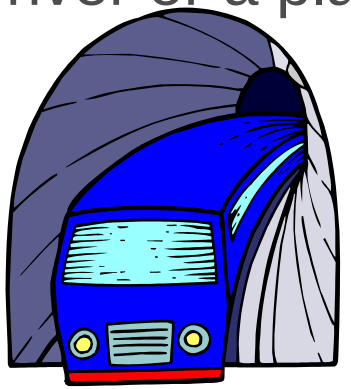
Polymorphism

- the same word or phrase can mean different things in different contexts
- In terms of the OOP, this means that a particular operation may behave differently for different sub-classes of the same class.
- Technically: many objects can implement same interface in their own way (writing to screen vs. file)

Analogy:

- In English, **bank** can mean side of a river or a place to put money
- move (in d/t way)-





Function Overloading

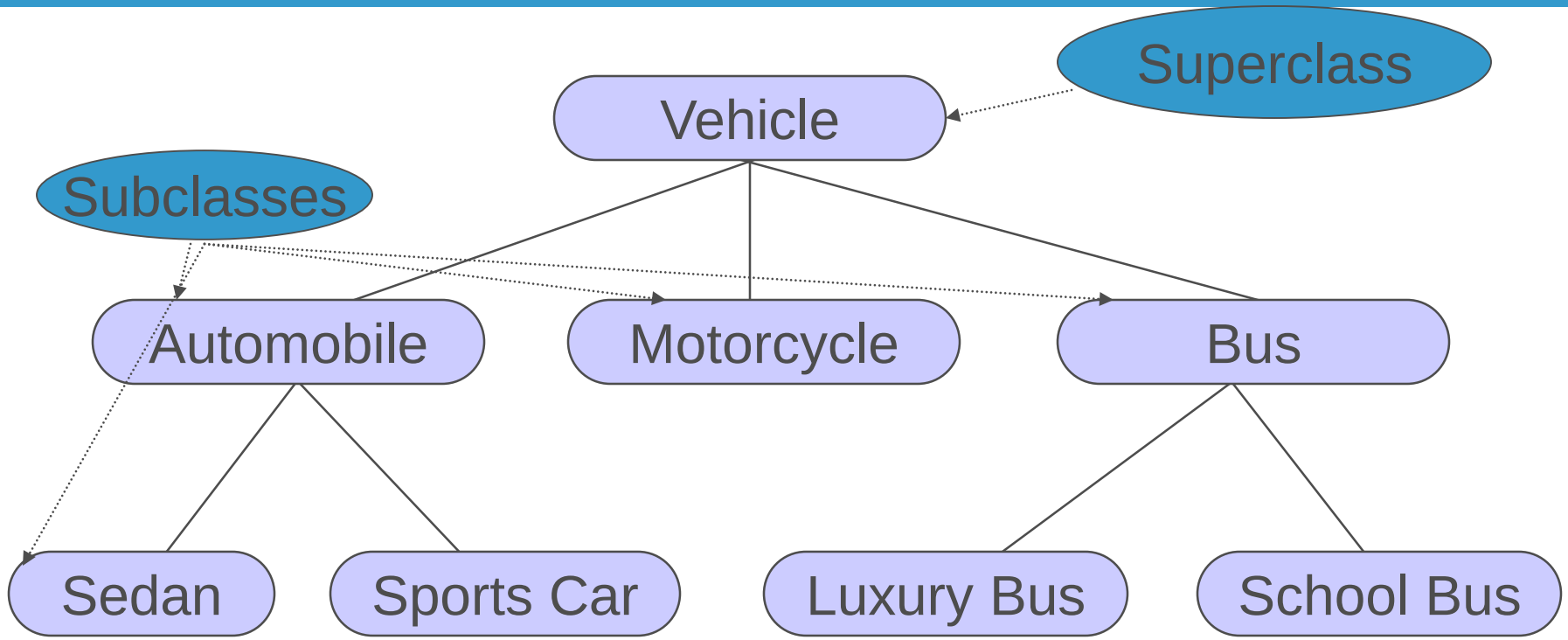
The operation of one function depends on the argument passed to it.

Example: Fly(), Fly(low), Fly(150)

Inheritance

- *Inheritance*—a way of organizing classes
- Term comes from inheritance of traits like eye color, hair color, and so on.
- Classes with properties in common can be grouped so that their common properties are only defined once.
- *Superclass* – inherit its attributes & methods to the subclass(es).
- *Subclass* – can inherit all its superclass attributes & methods besides having its own unique attributes & methods.
- Allows reuse of implementation
- Classical Inheritance: “is-a” relationship
  - **Example: fruits and types of fruit (an apple is a type of fruit)**

An Inheritance hierarchy



What properties does each vehicle inherit from the types of vehicles above it in the diagram?

Software Reusability

- Rapid application development
  - Software reusability speeds the development of powerful, high-quality software
- Good Programming Practice
- Avoid reinventing the wheel. Study the capabilities of the Java API. If the API contains a class that meets your program's requirements, use that class rather than create your own.

Metrics of Class Design

Coupling

- inheritance Vs. coupling
- Strong coupling complicates a system
- design for weakest possible coupling

Cohesion

- degree of connectivity among the elements of a single module/class
- coincidental cohesion: all elements related undesirable
- Functional cohesion: work together to provide well-bounded behavior

summary

- OOP:
  - Emphasis on data, not procedure
  - Programs are made up of objects
  - Data is hidden from external functions
  - Objects can communicate with each other through methods
- Programming Languages:
  - Pure OO Languages
    - Smalltalk, Eiffel, Actor, Java
  - Hybrid OO Languages
    - C++, Objective-C, Object-Pascal

Approximate Terminology

instance = object

field = variable

method = function

sending a message to an object =
calling a function

These are all *approximately* true

Exercise

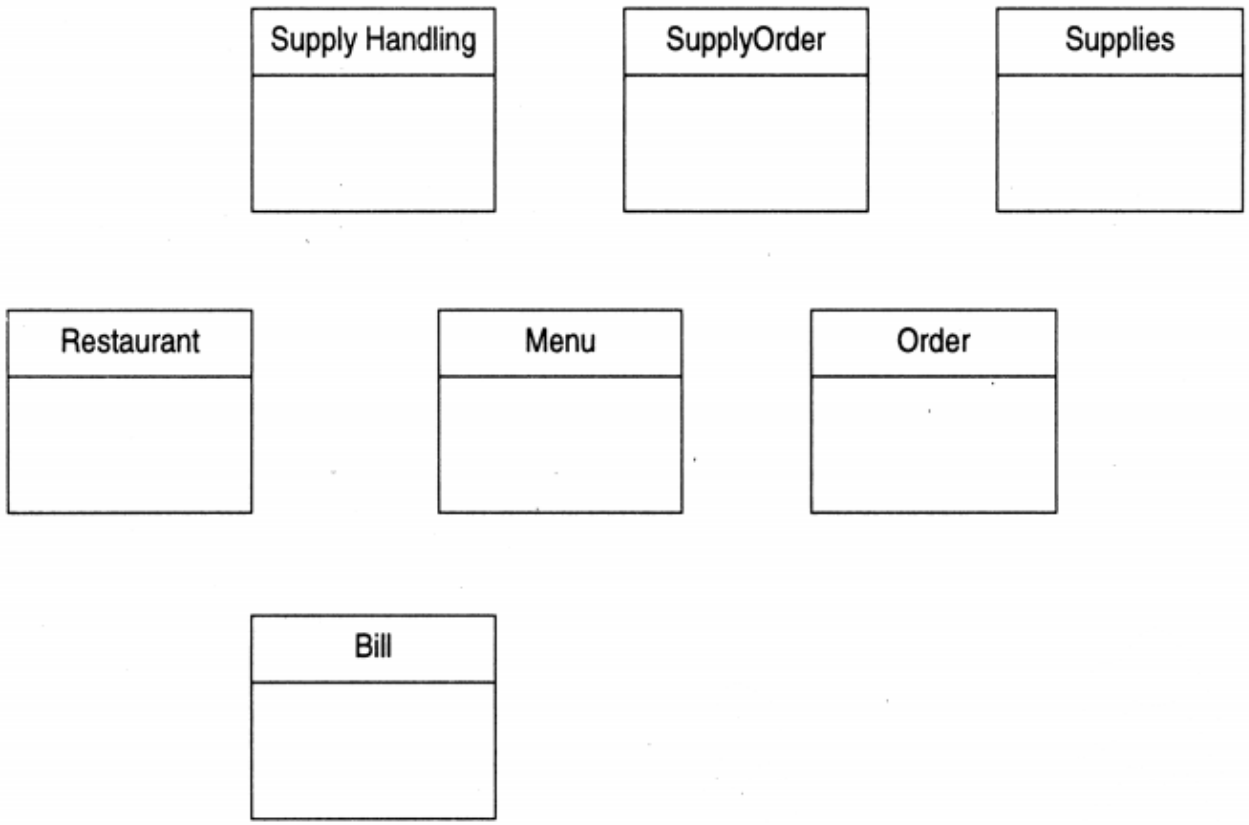
Design the following small systems using OOP

- Identify objects with its data and member methods

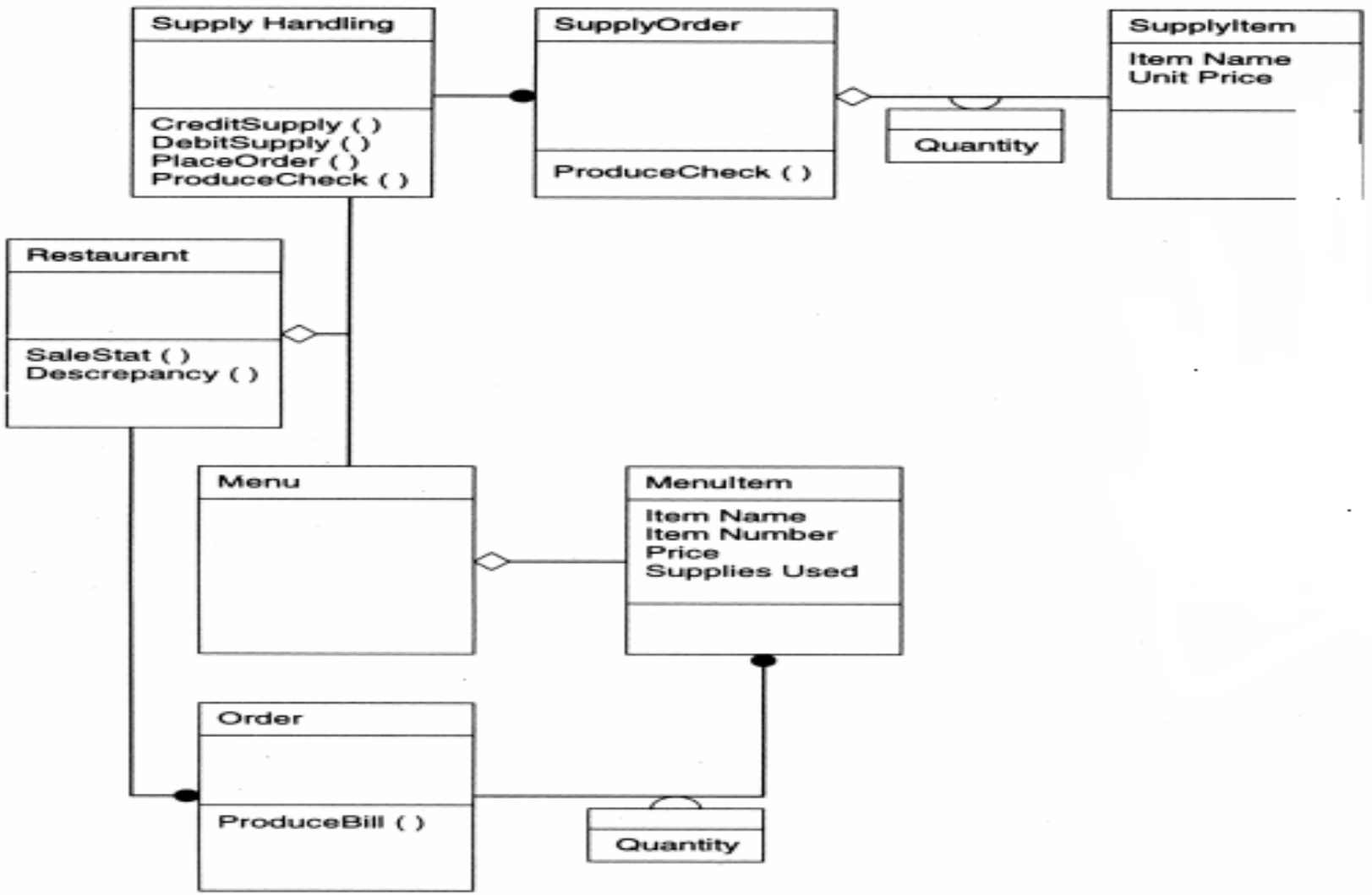
1. Small Restaurant mgmt system
2. Stock mgmt in supermarket
3. Clinic in AASTU
4. Tournament in inter-departmental Competition
5. Hotel in small town
6. Exam schedule for a college

Example

Restaurant example: Initial classes



Example


Supply Handling
CreditSupply ( )
DebitSupply ( )
PlaceOrder ( )
ProduceCheck ( )
SupplyOrder
ProduceCheck ( )
Quantity
SupplyItem
Item Name
Unit Price
Restaurant
SaleStat ( )
Descrepancy ( )
Menu
MenuItem
Item Name
Item Number
Price
Supplies Used
Order
ProduceBill ( )
Quantity

Questions?

That concludes this chapter

Thank You

Biruk M.