

Object-oriented modeling and design

UNIT 1

Introduction

- ▶ It is a new way of thinking about problems using models based on real world concepts.
- ▶ The basic construct is object which combines both data structure and behavior in a single entity.
- ▶ Rumbaugh presents an object oriented software development methodology, the Object Modeling Technique (OMT) which extends from analysis through design to implementation.

- ▶ Analysis model is built to abstract essential aspects of application domain which contains objects found in application, their properties and behavior.
- ▶ Then design model is made to describe and optimize the implementation.
- ▶ Finally the design model is implemented in a programming language, database or hardware.
- ▶ Graphical notation is used for expressing object-oriented models.

Object oriented analysis and design

The term “object- oriented” implies that the software consists of a collection of objects that include both data and methods. It further implies that such software consist of four primary characteristics:

1. Identity
2. Encapsulation
3. Inheritance
4. Polymorphism
5. Classification

Identity

- ▶ Identity means that data is organized into discrete, distinguishable entities called objects.
- ▶ Objects can be concrete or conceptual.
- ▶ In real world an object simply exist but within a programming language each object has a unique handle by which it can be uniquely referenced.
- ▶ The handle can be implemented by address, array index or unique value of an attribute.

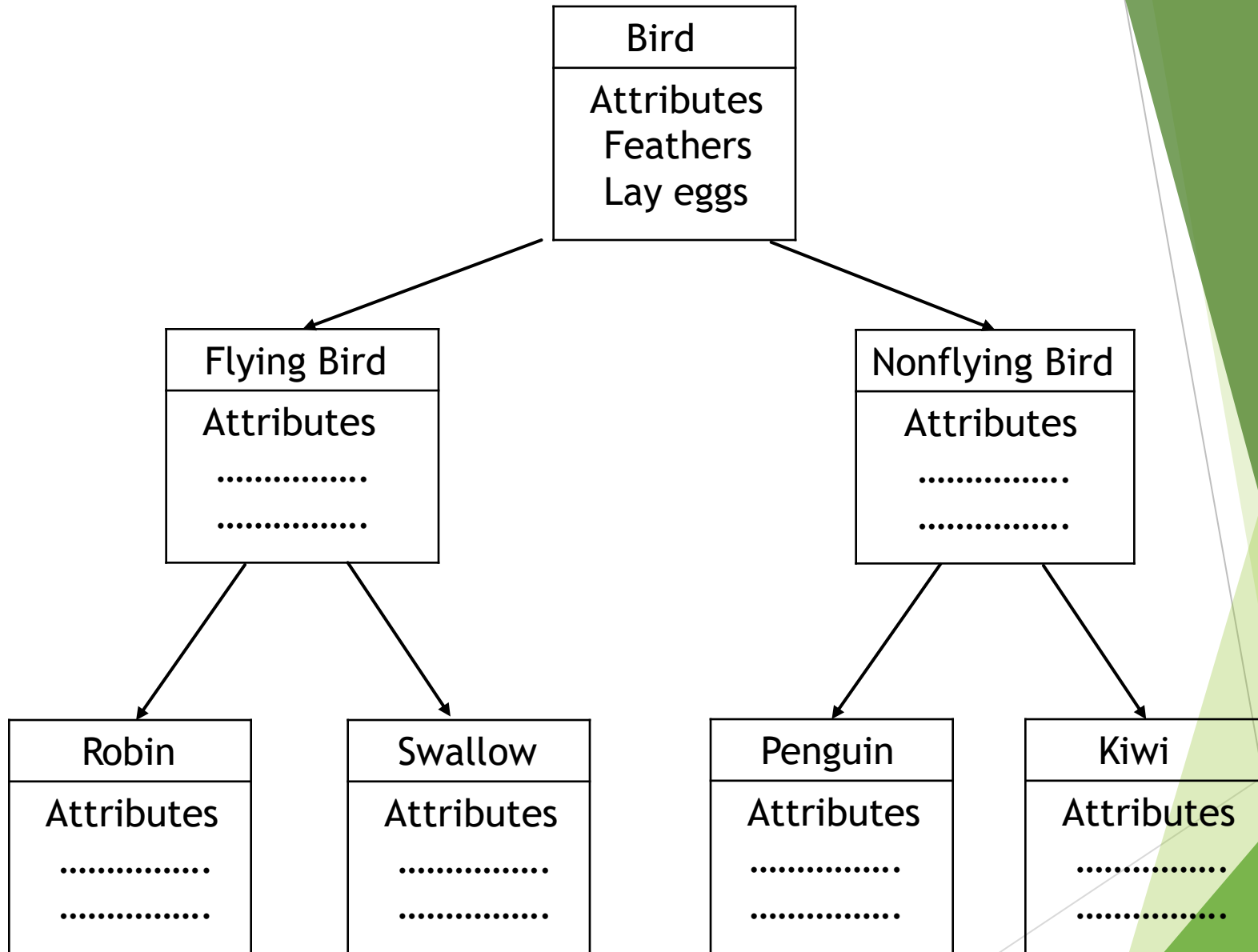
Encapsulation

- ▶ Information hiding
- ▶ Separating external aspects of an object which are accessible to other objects, from the internal implementation details of the object, which are hidden from other objects.
- ▶ The implementation of an object can be changed without affecting the applications that use it.

Inheritance

Inheritance is the process by which objects of one class acquired the property of objects of another class.

- ▶ It is the sharing of attributes and operations among classes based on a hierarchical relationship.
- ▶ Subclasses can be formed from broadly defined class.
- ▶ Each subclass incorporates or inherits all the properties of its super class and adds its own unique properties.



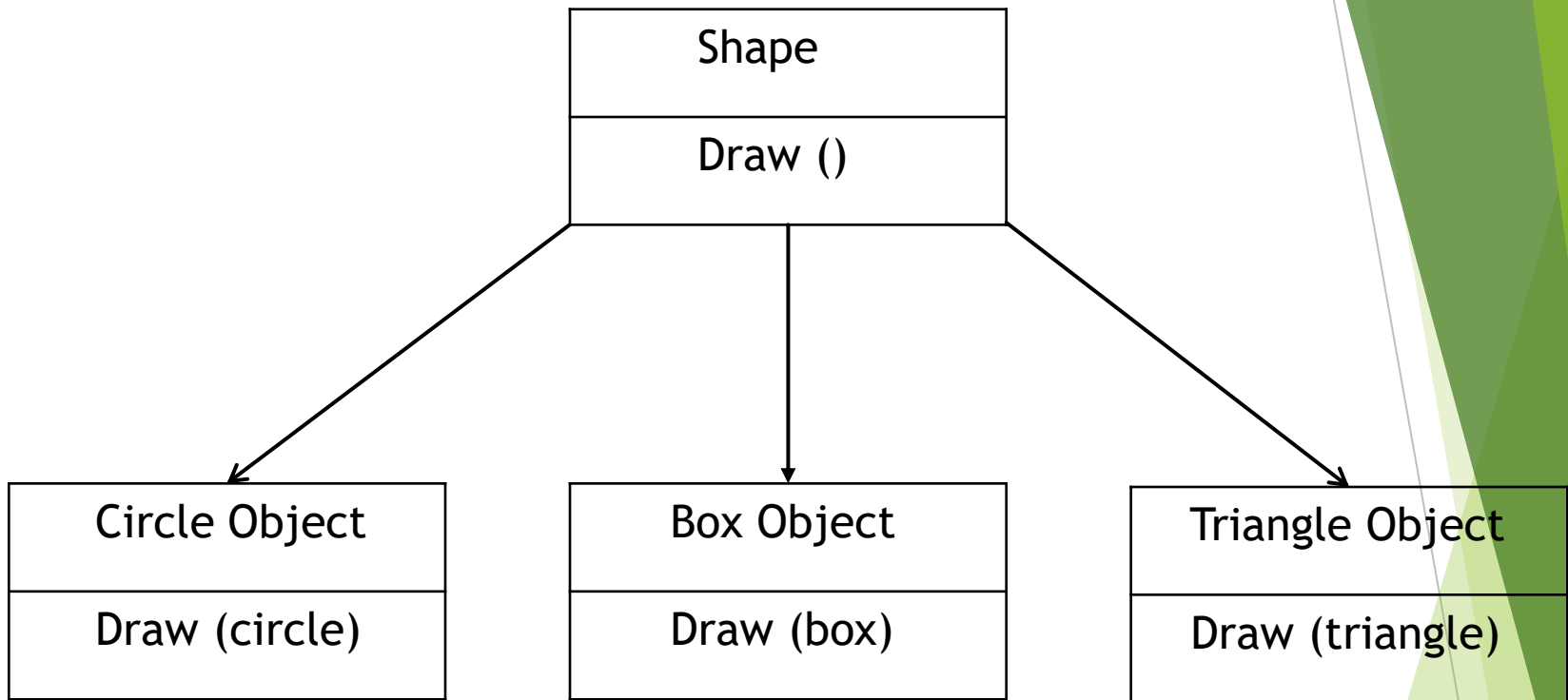
Property Inheritance

Polymorphism

It is the ability to take more than one form.

Example : $A + B$

- ▶ It means that the same operation (i.e. action or transformation that the object performs) may behave differently on different classes.
- ▶ Specific implementation of an operation by a certain class is called a method.



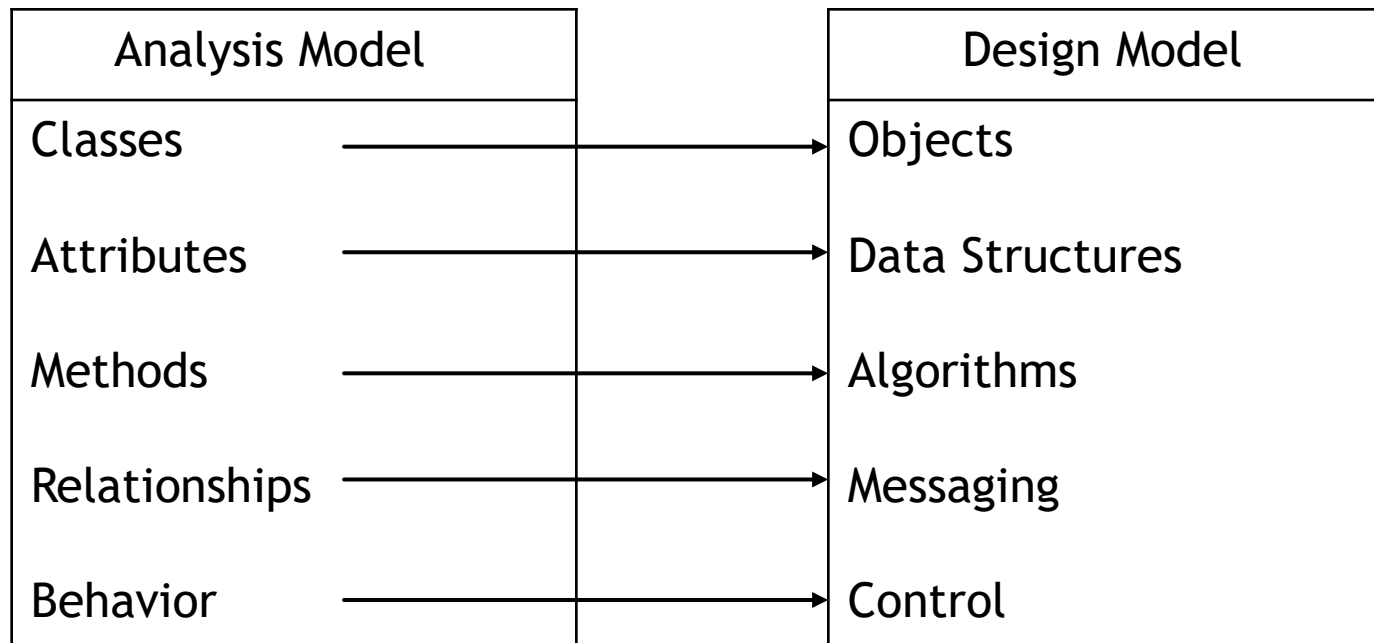
Polymorphism

Classification

- ▶ It means that objects with same data structure (attribute) and behavior (operations) are grouped into a class.
- ▶ A class is an abstraction that describes important properties and ignores the rest.

Object-oriented development

- ▶ OOD transforms the analysis model into design model that serves as a blue print for software construction.
- ▶ The theme is the identification and organization of application concepts rather than final representation in a prog. Language.
- ▶ OOD approach encourages software developers to work and think in terms of the application domain through most of the software engineering life cycle.
- ▶ It is a conceptual process independent of a programming language until the final stage.



Mapping between analysis model and
design model

Difference between OOA and OOD

Object oriented analysis (OOA) aims to model the problem domain, the problem to be solved by developing an OO system. Object oriented design (OOD) is an activity of looking for logical solution to solve a problem by using encapsulated entities called objects. We can say:

OOA: Emphasis is on finding and describing the objects or concepts of the problem domain. Focus on what the system must do. Do the right thing.

OOD: Emphasis is on defining software objects and how they collaborate to fulfil requirements. Focus on how the system will do it. Do the right thing.

Object-oriented methodology

Stages:

- ▶ Analysis: the analyst builds a model of the real world situation showing its important properties. Analyst must work with the requester to understand the problem because problem statement are rarely complete or correct.
- ▶ System design: system designer makes high level decisions about the overall architecture.
- ▶ Object design: object designer builds a design model based on the analysis model. The designer adds details to the design model in accordance.

- Implementation: object classes and relationships develop during object design are finally translated into a particular programming language, database, or hardware implementations.

Three Models

The OMT methodology uses three kinds of model to describe a system :

- ▶ Object model
- ▶ Dynamic model
- ▶ Functional model

Object model

- ▶ Describes basic structure of objects and their relationship.
- ▶ Contains object diagram.
- ▶ Object diagram is a graph whose nodes are object classes (Classes) and whose arcs are relationships among classes.

Dynamic model

- ▶ Describes the aspects of a system that change over time.
- ▶ It specifies and implement control aspects of a system.
- ▶ Contains state diagram.
- ▶ State diagram is a graph whose nodes are states and whose arcs are data-flows.

Functional Model

- ▶ Describes data value transformation within a system.
- ▶ Contains data flow diagram.
- ▶ Data Flow Diagram is a graph whose nodes are processes and whose arcs are data flows.

Introduction of UML

- ▶ UML - Unified Modeling language
- ▶ UML is a modeling language, not a methodology or process
- ▶ Fuses the concepts of the Booch, OMT, OOSE methods
- ▶ Developed by Grady Booch, James Rumbaugh and Ivar Jacobson at Rational Software.
- ▶ Accepted as a standard by the Object Management Group (OMG), in 1997.
- ▶ UML is a modeling language for visualizing, specifying, constructing and documenting the artifacts of software systems.

- ▶ Visualizing - a picture is worth a thousand words; a graphical notation articulates and unambiguously communicates the overall view of the system (problem-domain).
- ▶ Specifying - UML provides the means to model precisely, unambiguously and completely, the system in question.
- ▶ Constructing - models built with UML have a “design” dimension to it; these are language independent and can be implemented in any programming language.
- ▶ Documenting - every software project involves a lot of documentation - from the inception phase to the deliverables.

Conceptual Model of UML

- ▶ Building Blocks
 - Things
 - Relationships
 - Diagrams
- ▶ Rules
- ▶ Common Mechanisms
 - Specifications
 - Adornments
 - Common Divisions
 - Extensibility Mechanisms

UML Building Blocks

► Things

- Structural
- Behavioral
- Grouping
- Annotational

► Relationships

- Dependency
- Association
- Generalization
- Realization

► Diagrams

- Class Diagram
- Object Diagram
- Use Case Diagram
- Behavior Diagram
- Implementation Diagram

Diagrams

The graphical presentation of the model. Represented as a connected graph - vertices (things) connected by arcs (relationships).

UML includes nine diagrams - each capturing a different dimension of a software-system architecture.

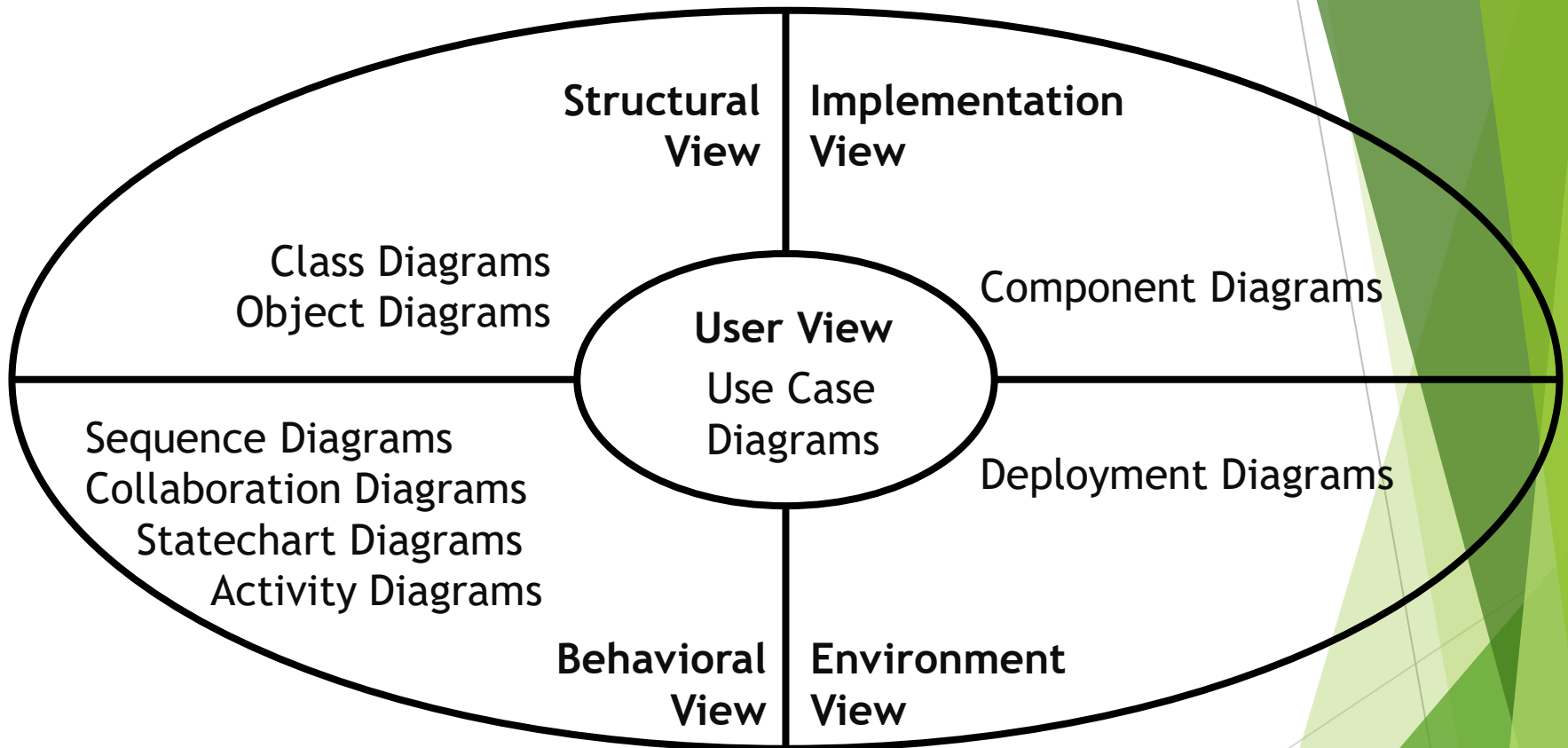
- ▶ Class Diagram
- ▶ Object Diagram
- ▶ Use Case Diagram
- ▶ Sequence Diagram
- ▶ Collaboration Diagram
- ▶ State chart Diagram

- ▶ Activity Diagram
- ▶ Component Diagram
- ▶ Deployment Diagram

- ▶ Class Diagram - the most common diagram found in OOAD, shows a set of classes, interfaces, collaborations and their relationships. Models the static view of the system.
- ▶ Object Diagram - a snapshot of a class diagram; models the instances of things contained in a class diagram.
- ▶ Use Case Diagram - shows a set of “Use Cases” (sets of functionality performed by the system), the “actors” (typically, people/systems that interact with this system[problem-domain]) and their relationships. Models WHAT the system is expected to do.

- ▶ Sequence Diagram - models the flow of control by time-ordering; depicts the interaction between various objects by of messages passed, with a temporal dimension to it.
- ▶ Collaboration Diagram - models the interaction between objects, without the temporal dimension; merely depicts the messages passed between objects.
- ▶ Statechart Diagram - shows the different state machines and the events that leads to each of these state machines. Statechart diagrams show the flow of control from state to state.
- ▶ Activity Diagram - shows the flow from activity to activity; an “activity” is an ongoing non-atomic execution within a state machine.

- ▶ Component Diagram - shows the physical packaging of software in terms of components and the dependencies between them.
- ▶ Deployment Diagram - shows the configuration of the processing nodes at run-time and the components that live on them.



Dimensions of Software Architecture

Rules

- ▶ Specify what a well-formed model should look like.
- ▶ The UML has semantic rules for:
 - Names
 - Scope
 - Visibility
 - Integrity
 - Execution

UNIT 2

Object-Oriented concepts

- ▶ Abstraction
- ▶ Encapsulation
- ▶ Combining data and behavior
- ▶ Sharing
- ▶ Object structure, not Procedure Structure
- ▶ Synergy

Abstraction

- ▶ It consists of focusing on essential aspects of an entity and ignoring accidental properties.
- ▶ The goal is to isolate those aspects that are important for some purpose and suppress those aspects that are unimportant.
- ▶ Abstraction must always for some purpose because the purpose determines what is and what is not important.
- ▶ Many different abstractions of same thing are possible, depending on the purpose for which they are made.
- ▶ In building models, one must not search for absolute truth but for adequacy for some purpose.
- ▶ A good model captures the crucial aspects of a problem and omits the others.
- ▶ Focus on what an object is and does, not how to implement.

Synergy

- ▶ Identity, Classification, polymorphism and inheritance can be used in isolation but together they complement each other synergistically.

Advantages of OOD

- ▶ Used in developing Compilers, Graphics, UI, databases, Object oriented languages, CAD systems, simulations, etc.
- ▶ Used to document existing programs that are ill-structured and difficult to understand.
- ▶ Not reduces development time; it may take more time than conventional development because it is intended to promote future reuse and reduce downstream errors and maintenance.

Modeling

- ▶ A model is an abstraction of something for the purpose of understanding it before building it.
- ▶ The word model has 2 dimensions- a view of a system (object, dynamic or functional) and a stage of development (analysis, design or implementation)

Purpose of Modelling

- ▶ Testing a physical entity before building it
- ▶ Communication with customers
- ▶ Visualization
- ▶ Reduction of complexity

Object Modeling

- ▶ Captures static structure of a system-
- ▶ Objects in the system
- ▶ Attributes
- ▶ operations

Objects

Define objects as a concept, abstraction or thing with purposeful meaning.

Two purpose of objects:

- ▶ Promote understanding of the real world
- ▶ Provide a practical basis for computer implementation.
- ▶ All objects have identity and are distinguishable.

Classes

- ▶ An object class describes a group of objects with similar properties (attributes), common behavior (operations), common relationships to other objects and common semantics.

Object diagram

- ▶ It provides a formal graphic notation for modeling objects, classes and their relationships to one another.

Types:

- ▶ Class diagram
- ▶ Instance diagram

Class diagram

- ▶ A class diagram is a schema, pattern or templates for describing many possible instances of data.
- ▶ Describes classes.

Instance diagram

- ▶ Describes how a particular set of objects relate to each other.
- ▶ Describes objects

Person

Class Diagram

(Person)
Ram

(Person)
Jai

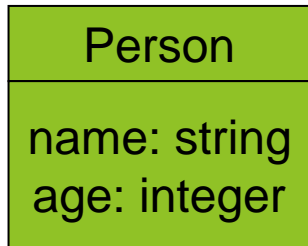
(Person)
Amit

Instance Diagram

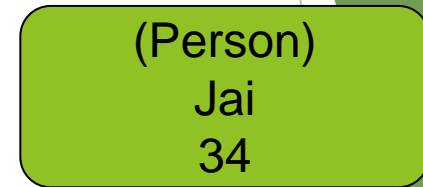
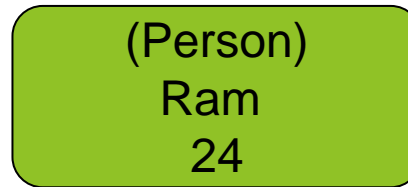
Attribute

- ▶ An attribute is a data value held by the objects in a class.
- ▶ Each attribute name is unique within a class.
- ▶ An attribute should be a pure data value, not an object.

Object Modeling Notations



Class with Attributes



Objects with Values

Operations and Methods

- ▶ An operation is a function or transformation that may be applied to or by objects in a class.
- ▶ All objects in a class share the same operations.
- ▶ The same operation may apply to many different classes.
- ▶ Such an operation is polymorphic i.e. the same operation takes on different forms in different classes.
- ▶ A method is the implementation of an operation for a class.

Person
name age
change-job change-add

File
name size
print

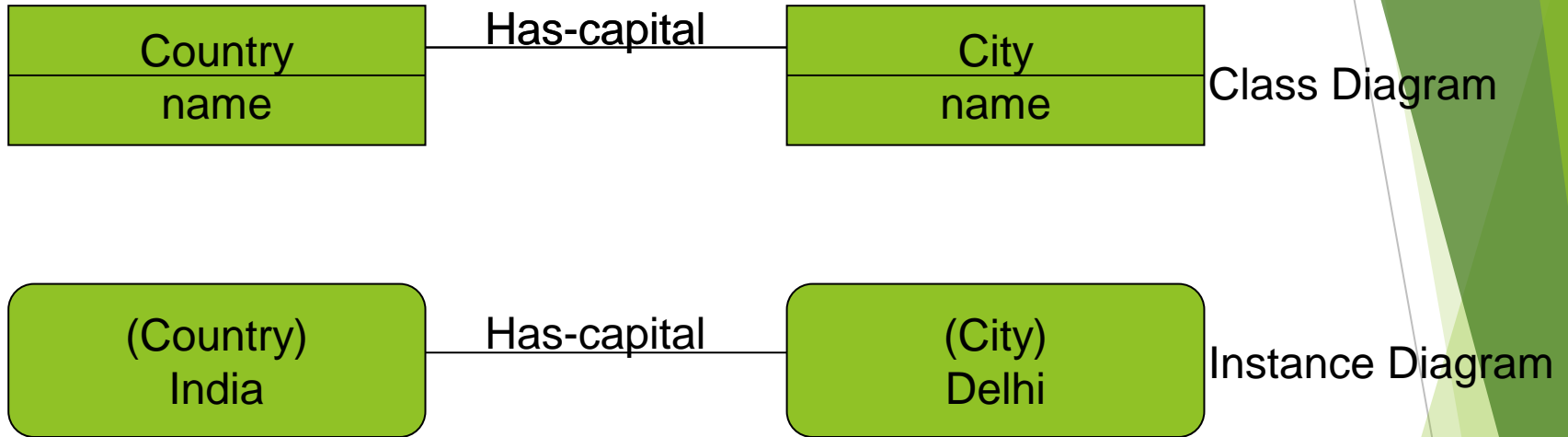
Operations

Links and Association

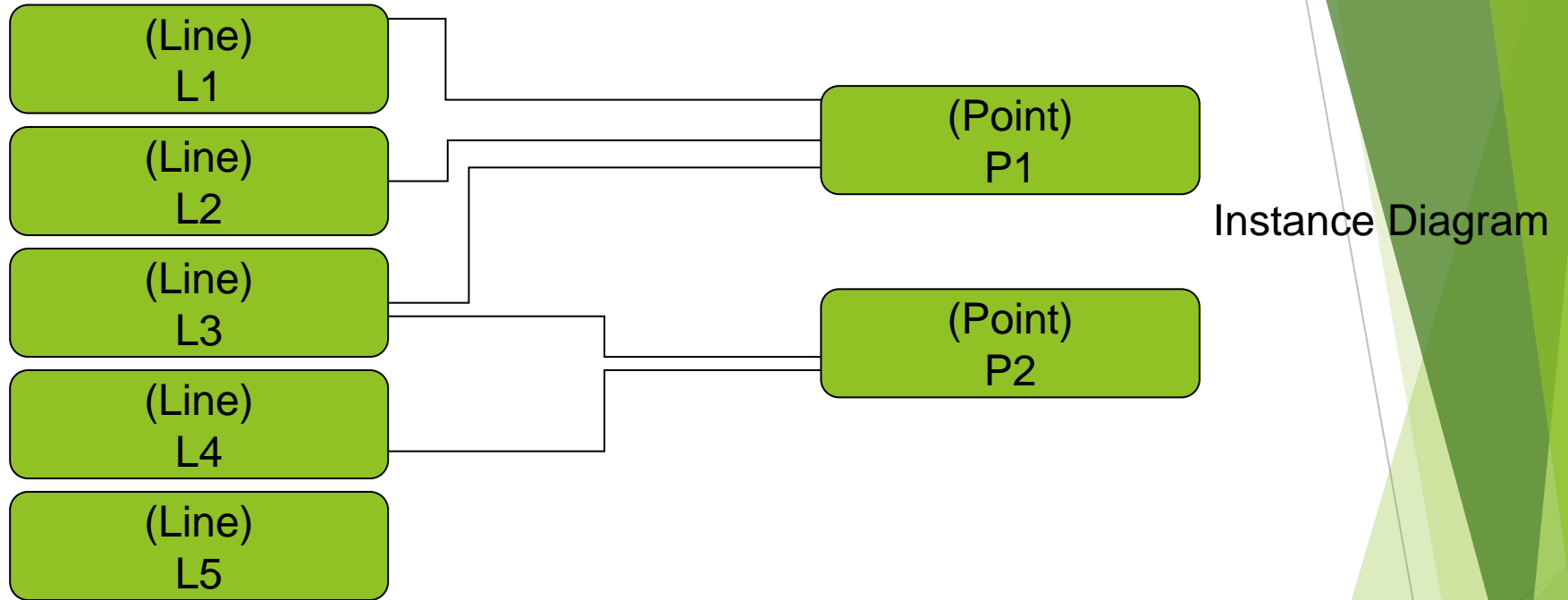
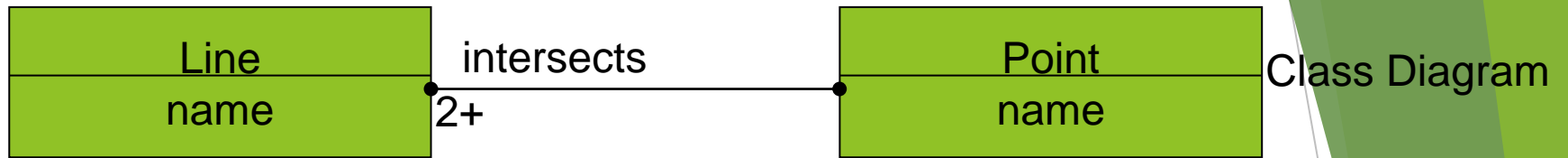
- ▶ Links and associations are the means for establishing relationships among objects and classes.
- ▶ A link is a physical or conceptual connection between objects.
- ▶ A link is an instance of an association.
- ▶ An association describes a group of links with common structure and semantics.
- ▶ All the links in an association connect objects from the same classes.
- ▶ Association and links often appear as verbs in a problem statement.
- ▶ An association describes a set of potential links in the same way that a class describes a set of potential objects.

- ▶ Associations are bidirectional.
- ▶ In real, both directions of traversal are equally meaningful and refer to same association.
- ▶ Associations are often implemented in programming languages as pointers from one object to another.
- ▶ A pointer is an attribute in one object that contains an explicit reference to another object.

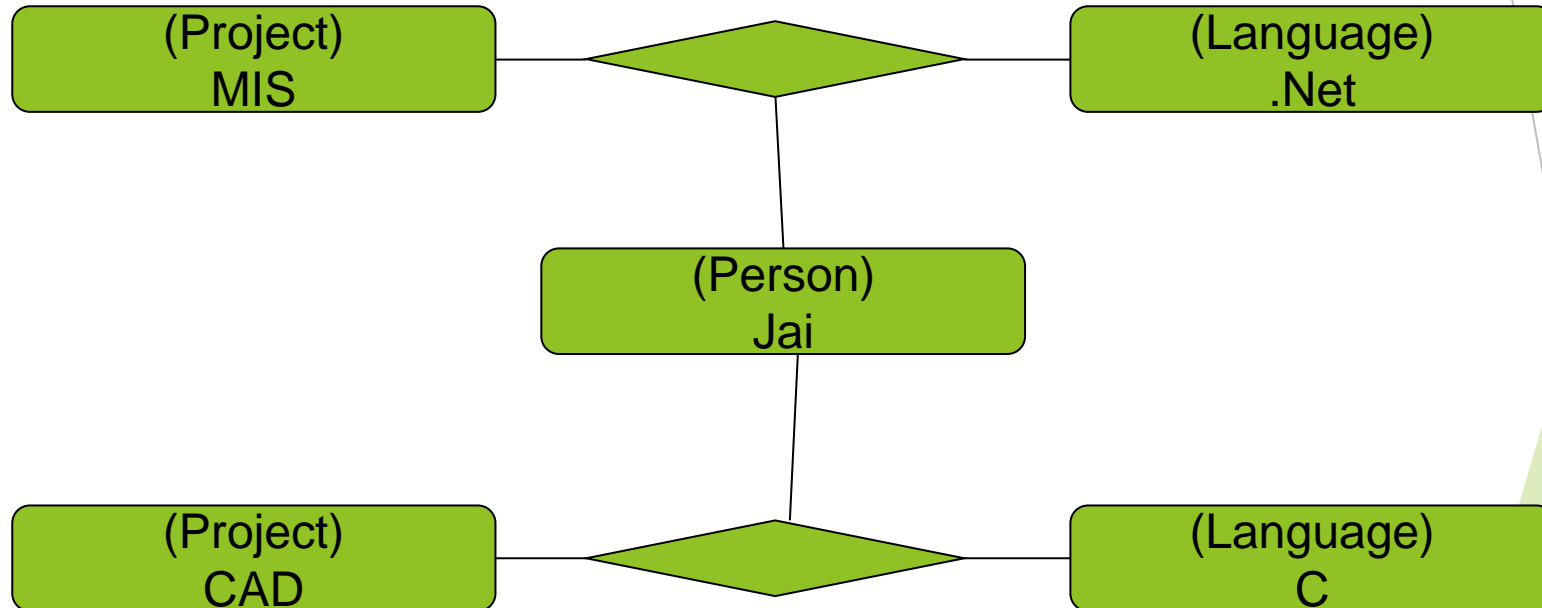
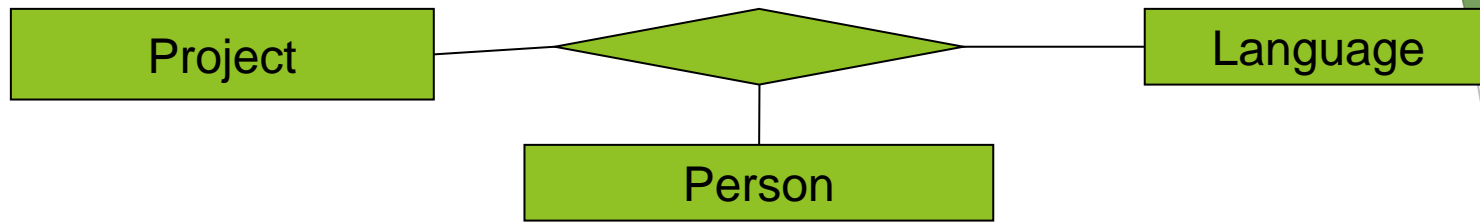
- ▶ A link shows a relationship between two or more objects.
- ▶ All connections among classes should be modeled as association.
- ▶ One-to-one association
- ▶ Many-to-many association
- ▶ Ternary association



One-to-One Association and links



Many-to-many Association and links



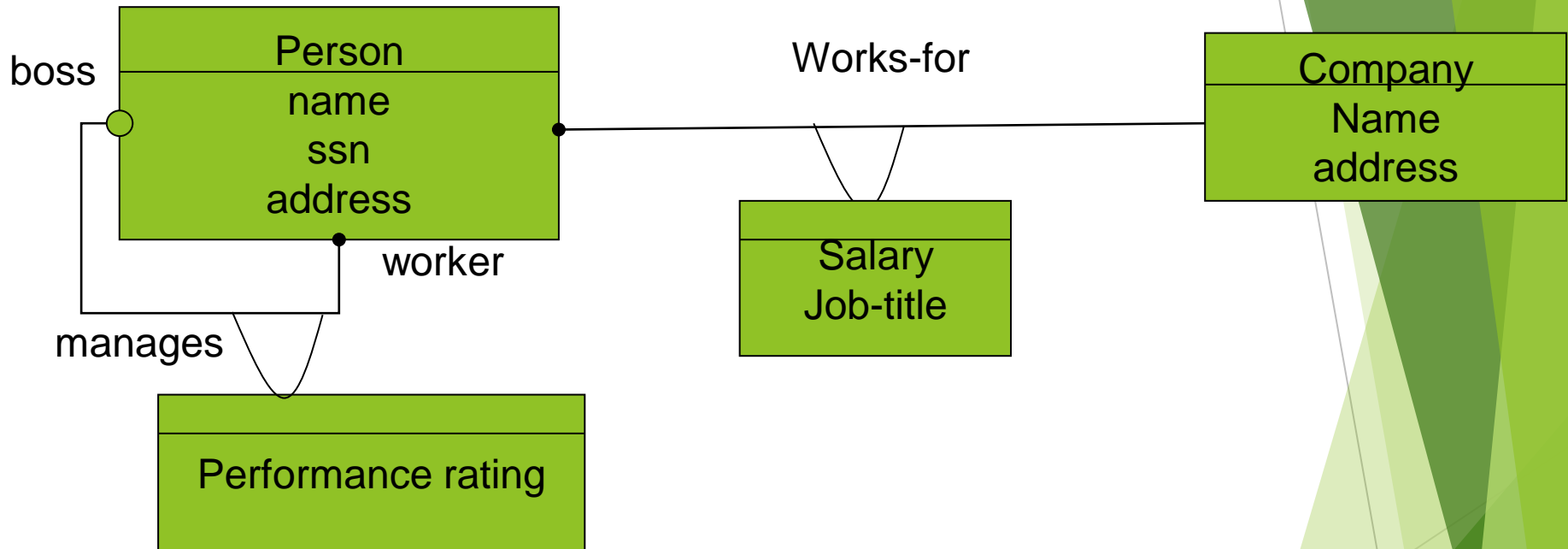
Ternary Association and Links

Multiplicity

- ▶ It specifies how many instances of one class may relate to a single instance of an associated class.
- ▶ In OMT solid ball -> zero or more
- ▶ Hollow ball -> optional (zero or one)
- ▶ Multiplicity depends upon assumptions and how you define the boundaries of a problem.

Link attributes

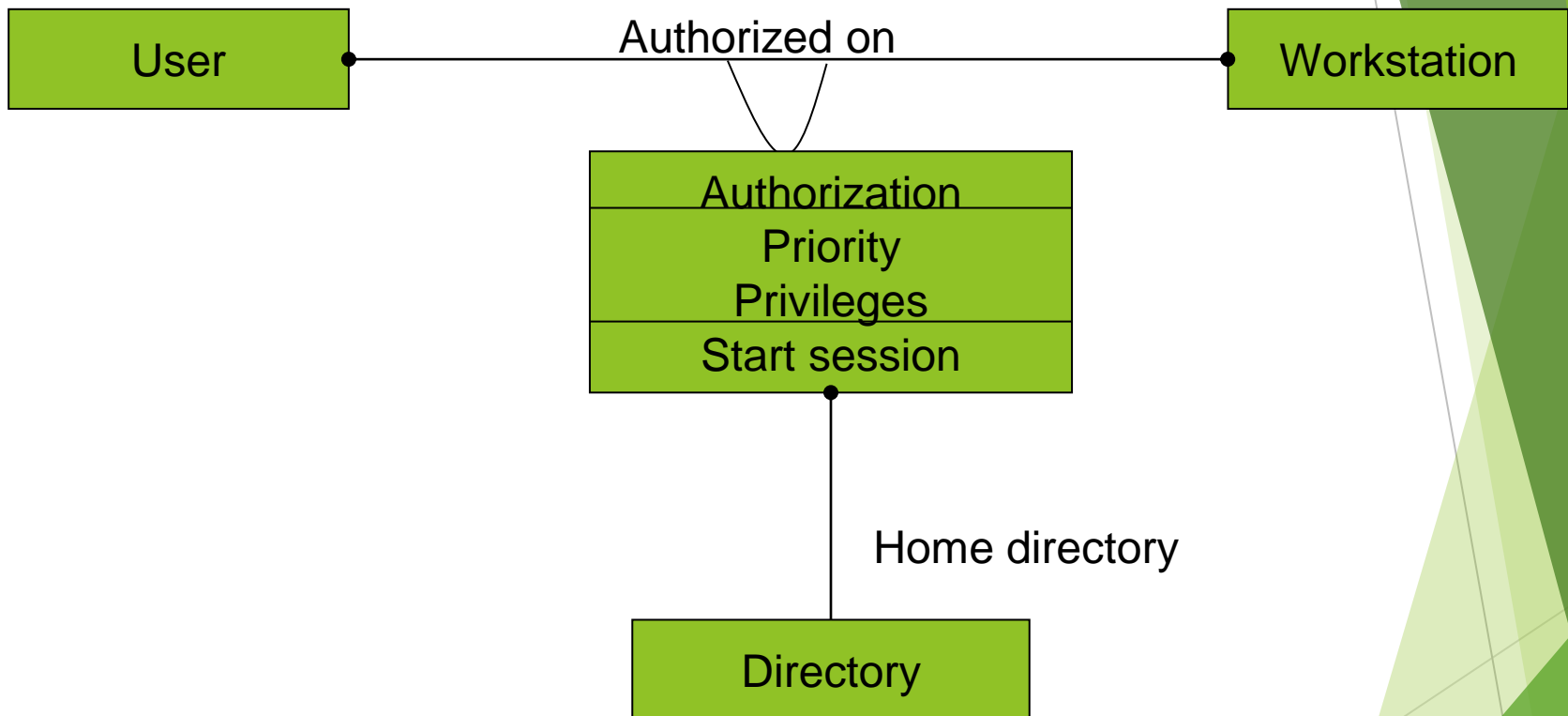
- ▶ An link attribute is a property of the links in an association.



Link attributes for one-to-many associations

Modeling an Association as a Class

- ▶ Each link becomes one instance of the class.



Modeling an association as a class

Role Names

- ▶ A role is one end of an association.
- ▶ A binary association has 2 roles, each of which may have a role name.
- ▶ A role name is a name that uniquely identifies one end of an association.
- ▶ Roles often appear as nouns in problem descriptions.
- ▶ Use of role name is optional.
- ▶ Role names are necessary for associations between two objects of the same class.



Employee	Employer
Ram	TCS
Mohan	Wipro

Role names for an association

Ordering

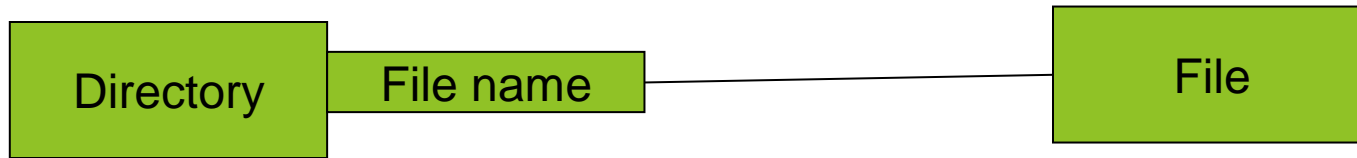
- ▶ If order of objects required
- ▶ Indicated by writing “{ordered}” next to multiplicity dot for the role.



Ordered sets in an association

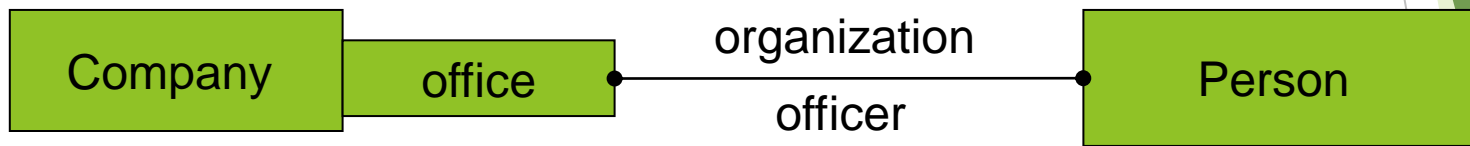
Qualification

- ▶ It relates two classes and a qualifier.
- ▶ Qualifier is a special attribute that reduces the effective multiplicity of an association.
- ▶ One-to-many or many-to-many may be qualified that can be reduced to one-to-one. (but not always)
- ▶ It distinguishes among the set of objects at the many end of an association.
- ▶ A qualified association can also be considered a form of ternary association.



A directory plus a file name yields a file

A qualified association



Aggregation

- ▶ It is a part-of relationship
- ▶ It has transitivity property i.e. A is part of B and B is part of C then A is part of C.
- ▶ It is anti symmetric i.e. if A is a part of B then B is not a part of A.
- ▶ Aggregation is a special form of association.



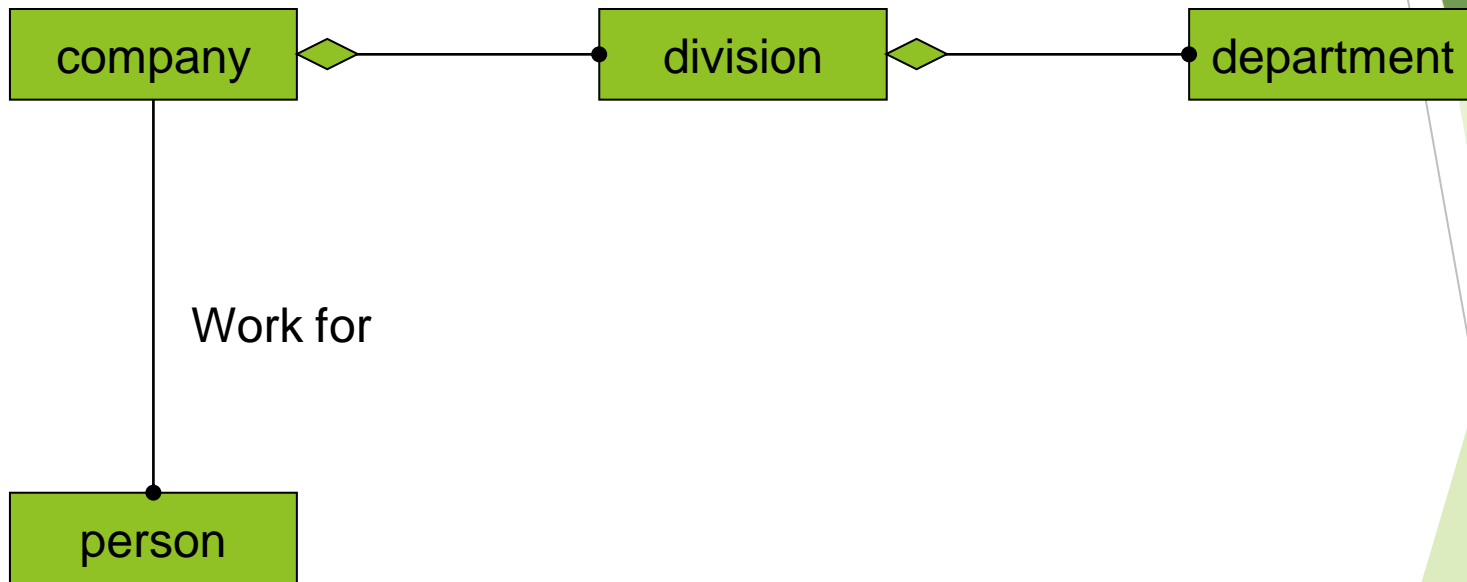
Aggregation

Aggregation

- ▶ It is a form of association in which an aggregate object is made of components.
- ▶ Components are part of aggregate.
- ▶ Aggregate is inherently transitive i.e. an aggregate has parts which may in turn have parts.

Aggregation vs. Association

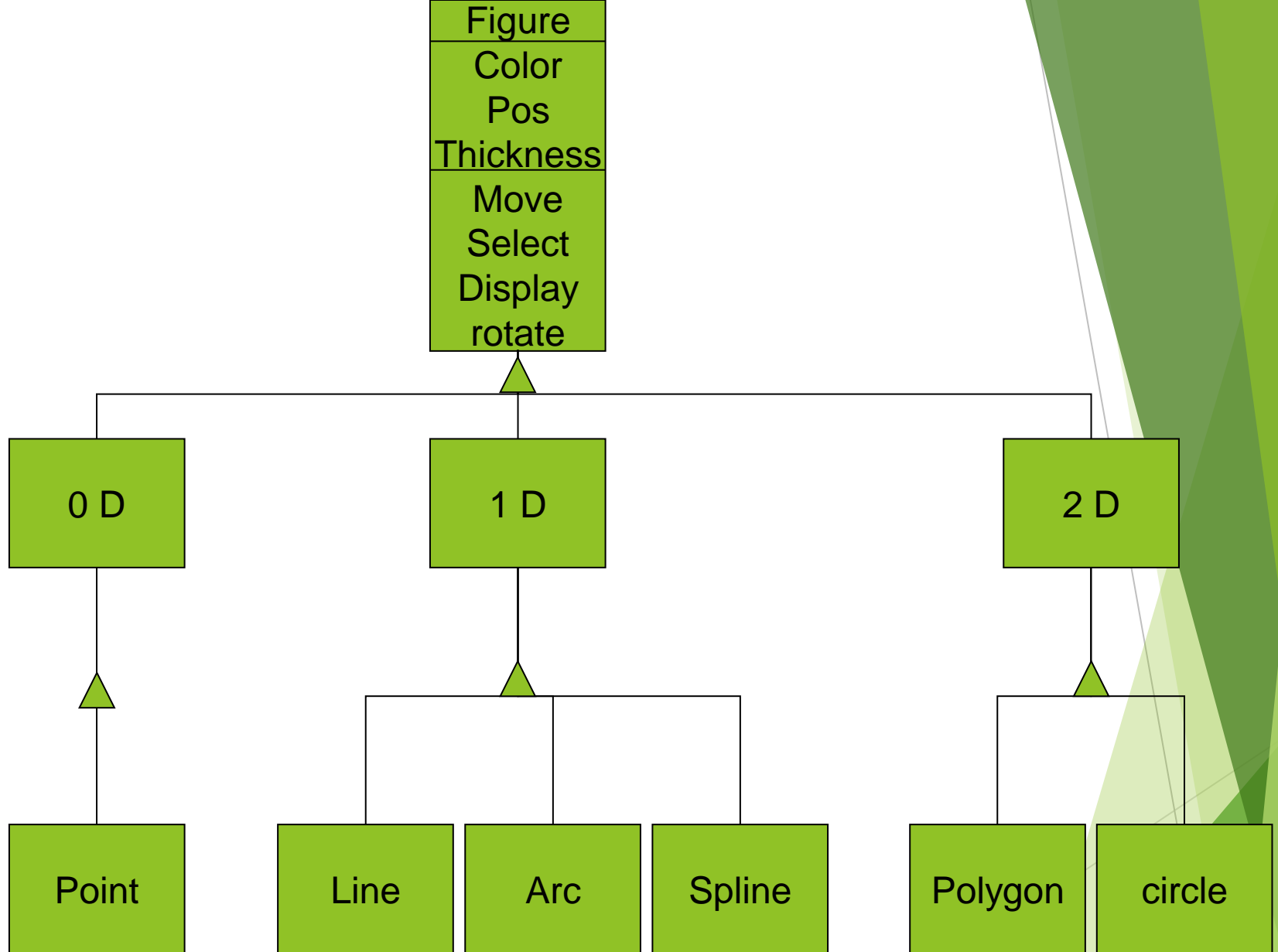
- ▶ Aggregation is a special form of association.
- ▶ If two objects are tightly bound by a part-whole relationship, it is an aggregation.
- ▶ If two objects are usually considered as independent even though they may often be linked, it is an association.
- ▶ A company is an aggregation of its divisions which are in turn aggregations of their departments; a company is indirectly an aggregation of departments.
- ▶ A company is not an aggregation of its employees since the company and person are independent objects of equal structure and status.



Aggregation and Association

Generalization

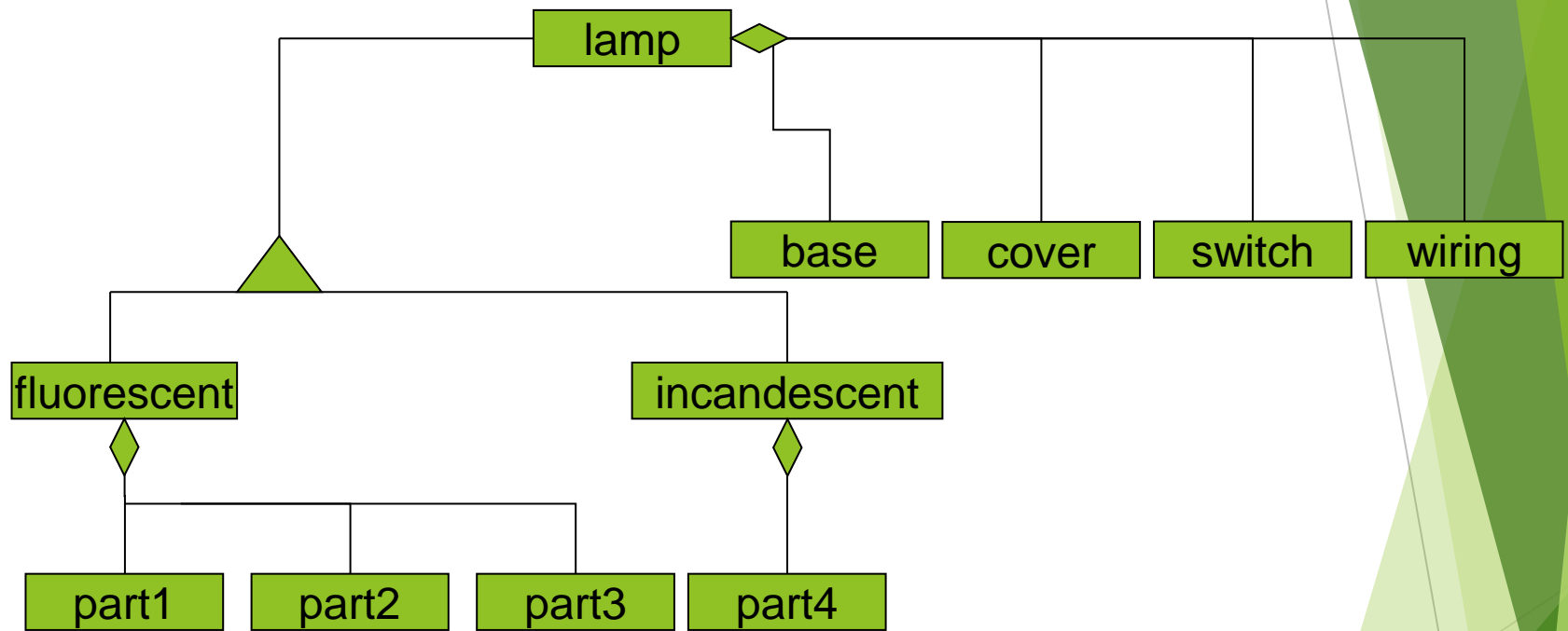
- ▶ It is a relationship between base class and sub class.
- ▶ It is “is-a” relationship because each instance of a sub class is an instance of super class as well.



Inheritance for graphic figure

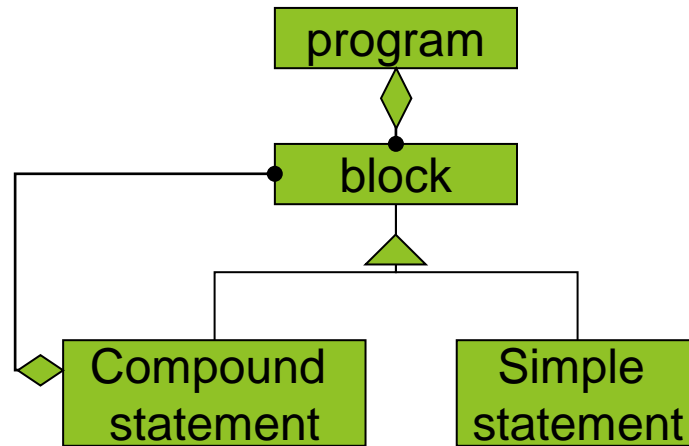
Aggregation vs. Generalization

- ▶ Aggregation relates instances.
- ▶ Two distinct objects are involved; one of them is a part of the other.
- ▶ Generalization relates classes.
- ▶ Aggregation is often called “a part of” relationship; generalization is often called “a kind-of” or “is-a” relationship.
- ▶ Aggregation is sometimes called an “and relationship” and generalization refers to “or relationship”



Aggregation and generalization

- ▶ Aggregation can be fixed, variable or recursive.
- ▶ A fixed aggregate has a fixed structure; the number and types of sub parts are predefined e.g. lamp.
- ▶ A variable aggregation has a finite number of levels but number of parts may vary e.g. company.
- ▶ A recursive aggregation contains directly or indirectly an instance of the same kind of aggregate; the number of potential levels is unlimited e.g. computer program.



Recursive aggregate

► Simple Statement:

`x=i;`

► Block(combination of more than one simple statements):

`++i;`

`x=i;`

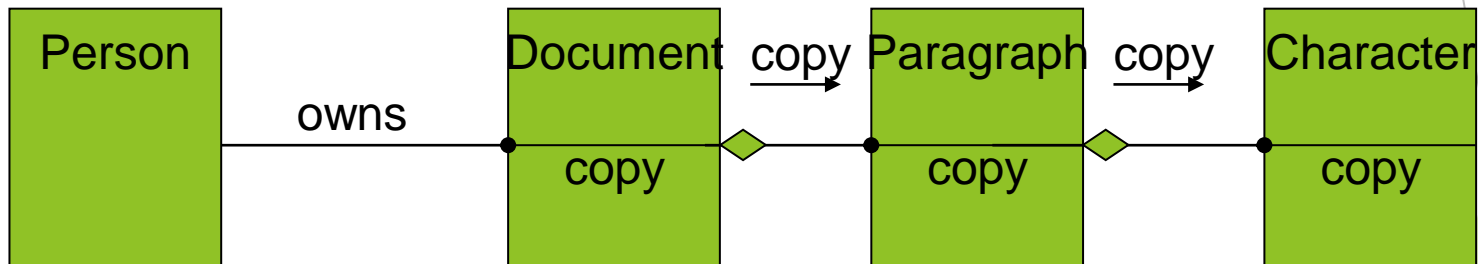
► Compound Statement(single statement combines work of multiple statement optionally blocks):

`x=++i;`

Propagation of operations

- ▶ Propagation (triggering) is automatic application of an operation to a network of objects when the operation is applied to some starting object.
- ▶ E.g. moving an aggregation moves its parts; the move operation propagates to the parts.
- ▶ A person owns multiple documents. Each document is composed of paragraphs that are in turn composed of characters. The copy operation propagates from documents to paragraphs to characters. Copying a paragraph copies all the characters in it.
- ▶ The operation does not propagate in reverse direction.

- Propagation can be possible for save, destroy, print and lock.

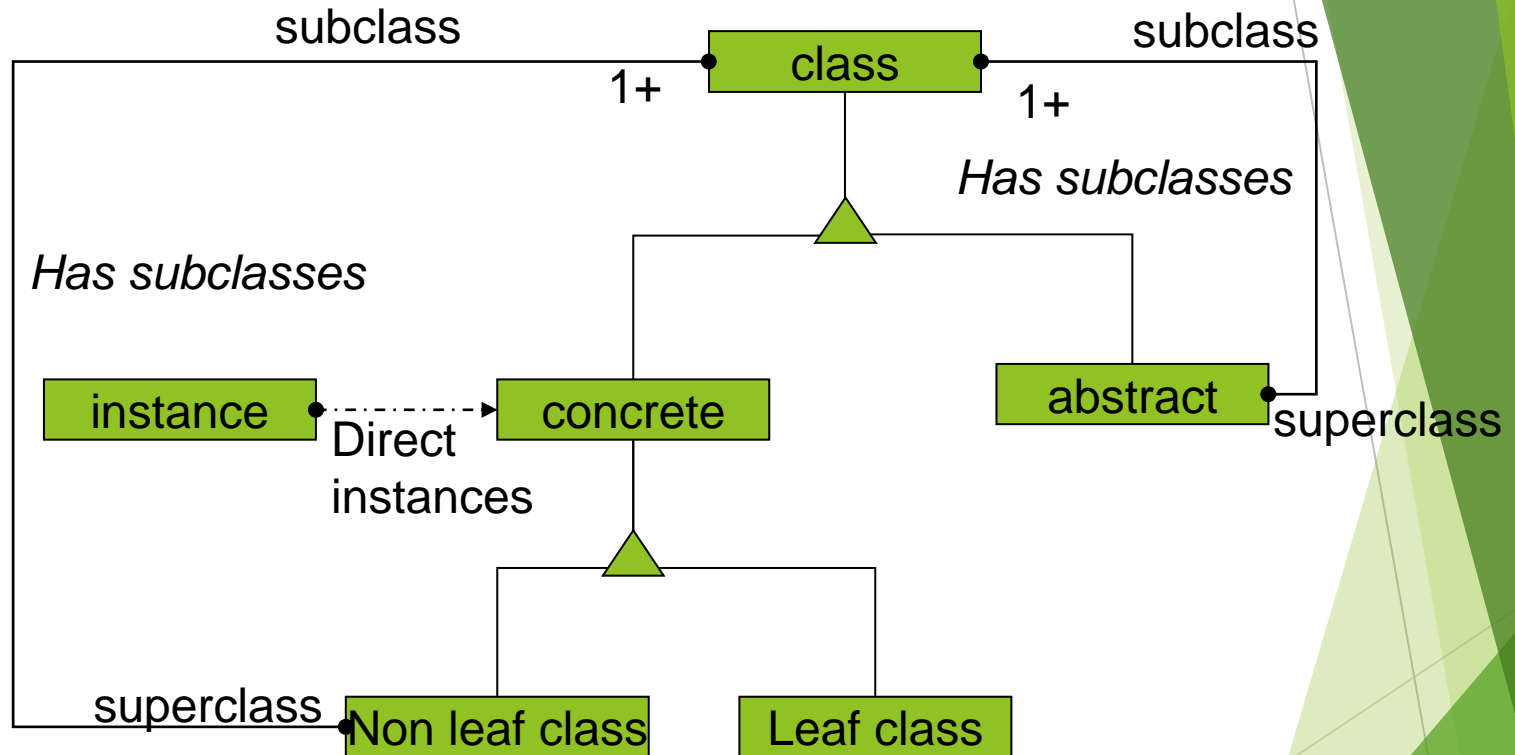


Propagation of operations

Abstract classes

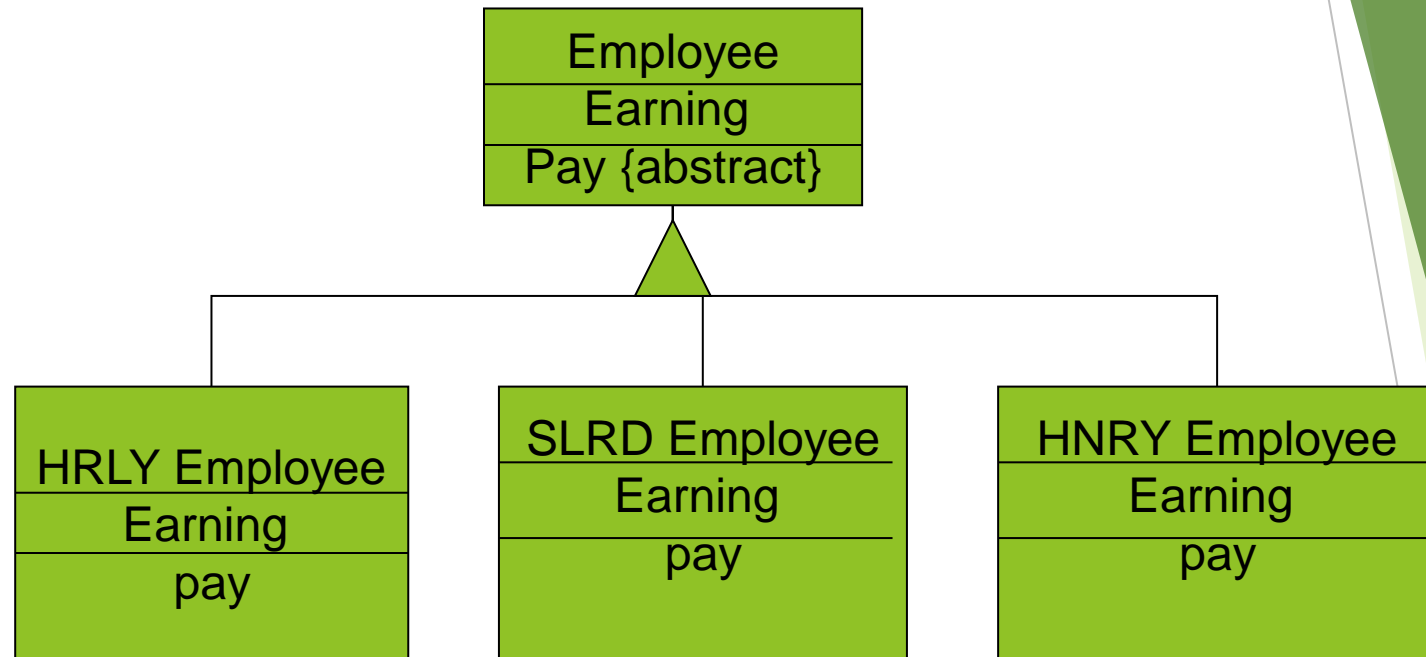
- ▶ An abstract class is a class that has no direct instances but whose descendent classes have direct instances.
- ▶ A concrete class is a class that is instantiable i.e. it can have direct instances.
- ▶ A concrete class may have abstract subclasses but they must have concrete descendents.
- ▶ Only concrete classes may be leaf classes in the inheritance tree.

Abstract & concrete class



- ▶ Abstract classes organize features common to several classes
- ▶ Abstract classes are frequently used to define methods to be inherited by subclasses.
- ▶ An abstract class can define methods to be inherited by subclasses.
- ▶ An abstract operation defines the form of an operation for which each concrete subclass must provide its own implementation.

► Abstract class & abstract operation



Overriding operations

- ▶ Reasons to use overriding-
- ▶ For extension
- ▶ For restriction
- ▶ For optimization
- ▶ For convenience



Extension

The new operation is same as inherited except it adds some behavior usually affecting new attributes of the subclass.



Restriction

The new operation restricts the inherited.



Optimization

The new method must have the same external protocol (prototype) and results as the old one but its internal representation and algorithm may differ completely.

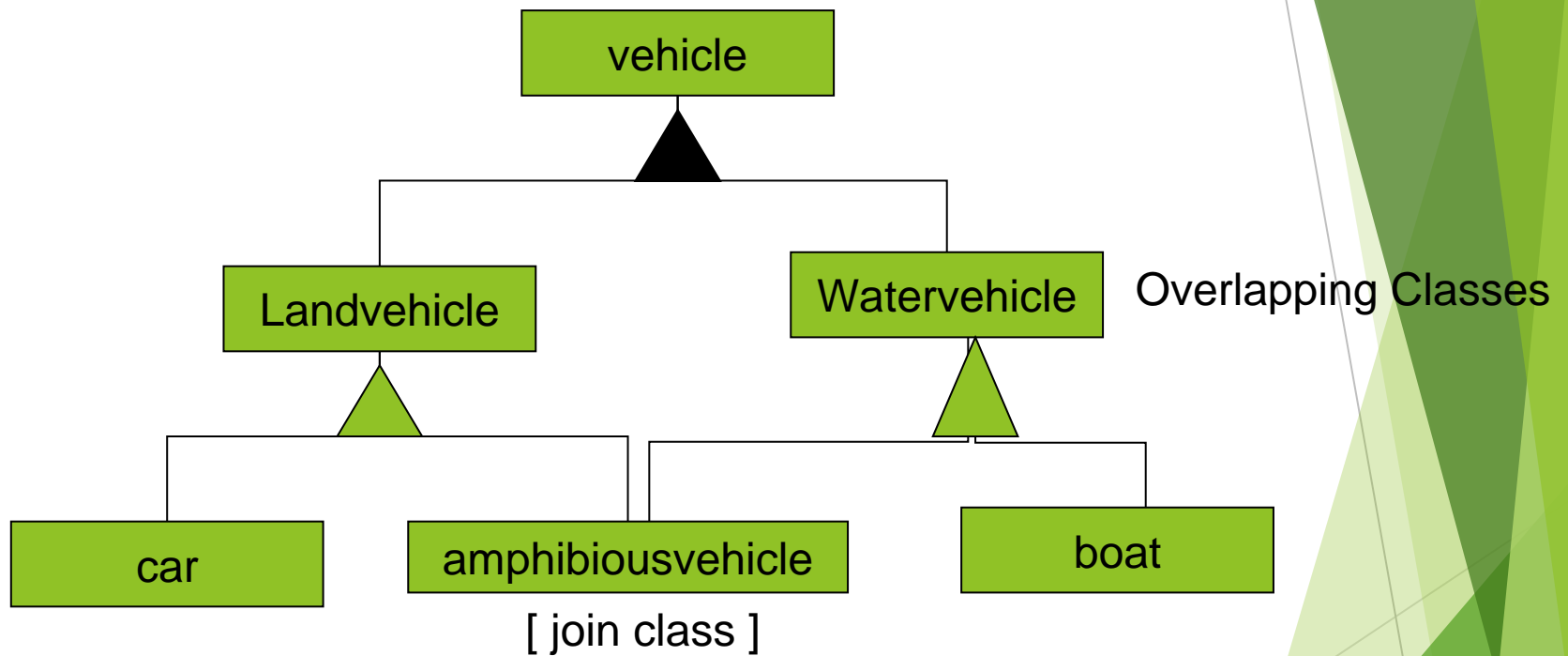


Convenience

The new class is made a subclass of the existing class and overrides the methods that are inconvenient.

Multiple inheritance

- ▶ It permits to have more than one super class and to inherit features from all parents.
- ▶ A class with more than one super class is called a join class.
- ▶ A feature from the ancestor class found along more than one path is inherited only once.
- ▶ Conflicts among parallel definitions create ambiguities that must be resolved in implementation.

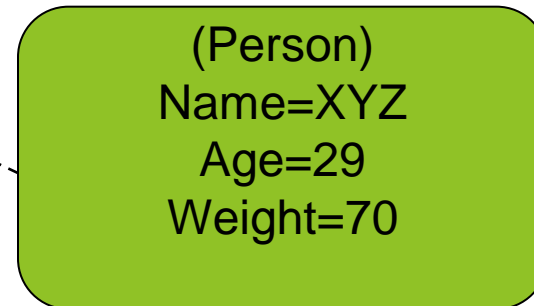
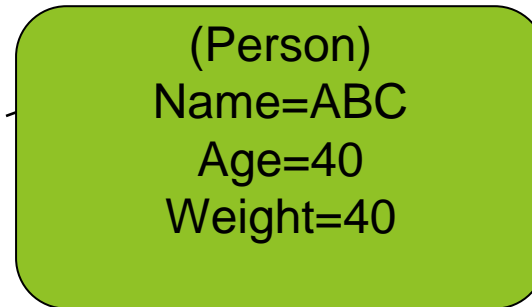
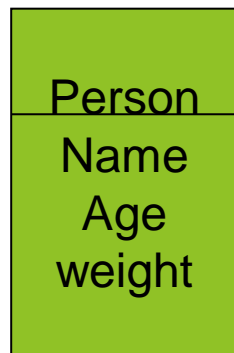


Multiple inheritance from overlapping classes

- ▶ A hollow triangle indicates disjoint subclasses while a solid triangle indicates overlapping subclasses.

Metadata

- ▶ Metadata is data that describes other data.
- ▶ The definition of a class is metadata.
- ▶ Models are metadata since they describe the things being modeled.
- ▶ RDBMS uses metadata.
- ▶ A class describes a set of instances of a given form.
- ▶ Instantiation relates a class to its instances.
- ▶ The dotted arrows connect the instances to the class.
- ▶ It is also useful for documenting examples and test cases.



Notation for instantiation

Class descriptors

- ▶ Classes are meta objects and not real world objects.
- ▶ Class descriptors have features and they have their own classes which are called meta classes.
- ▶ A class attribute describes a value common to an entire class of objects rather than data particular to each instance.
- ▶ Class attributes are useful for storing default information for creating new objects or summary information about instances of the class.
- ▶ A class operation is an operation on the class itself.
- ▶ The most common is to create new instances.
- ▶ Operations on class structure are class operations.

Class descriptors

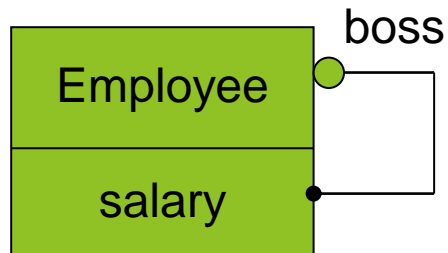
Window
Size:rectangle Visibility:boolean Default_size:rectangle
Display New_window Highest_priority_window

Candidate keys

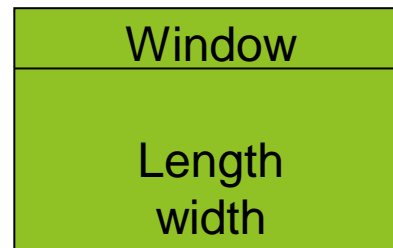
- ▶ A candidate key is a minimal set of attributes that uniquely identifies an object or link.
- ▶ A class may have one or more candidate keys each of which may have different combinations and numbers of attributes.
- ▶ Each candidate key constrains the instances in a class.
- ▶ Notation is {}

Constraints

- ▶ These are functional relation between entities of an object model.
- ▶ Entity includes objects, classes, attributes, links and associations.
- ▶ A constraint restricts the values that entities can assume.
- ▶ Simple constraints may be placed in object models and complex may be in functional model.



$\{ \text{salary} \leq \text{boss.salary} \}$

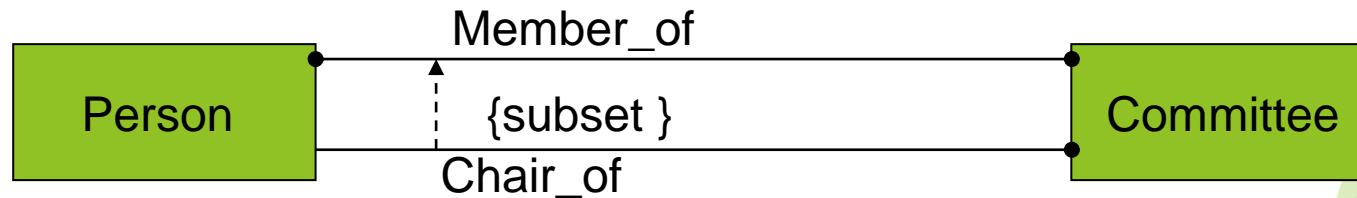


$\{ 0.8 \leq \text{length} / \text{width} \leq 1.5 \}$

Constraints on objects

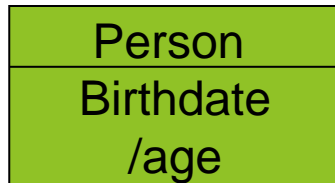
General constraints

- ▶ It must be expressed with natural languages or equations.
- ▶ Draw a dotted line between classes involved in the constraint and specify the details with comments in braces.



Derived objects, links and attributes

- ▶ The notation for a derived entity is a slash or diagonal line on the corner of a class box, on an association line or in front of an attribute.
- ▶ Show the constraint that determines the derived value. (optional)



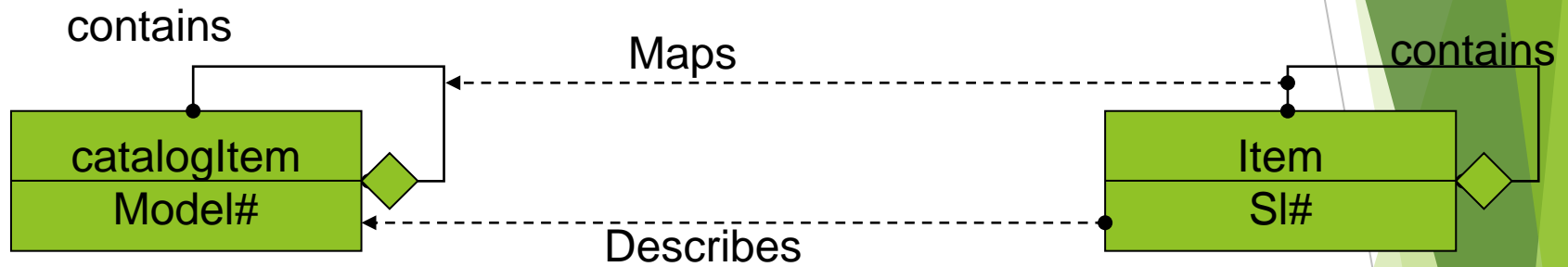
{ age = currentdate – birthdate }

Homomorphism

- ▶ A mapping between mathematical structures of the same type (eg groups or rings) that preserves the structure.
- ▶ similarity of form
- ▶ Homos = same & morphe = structure
- ▶ E.g. in a parts catalog for a automobile, a catalog item may contain other catalog items.
- ▶ Each catalog item is specified by a model number that corresponds to number of individual manufactured items, each with its own serial number.
- ▶ The individual items are also composed of sub items.
- ▶ The structure of each item's parts has the same form as the catalog item's parts.
- ▶ Thus the “contains” aggregation on catalog items is a homomorphism of the “contains” aggregation on physical items.

Homomorphism

- It maps between two associations.

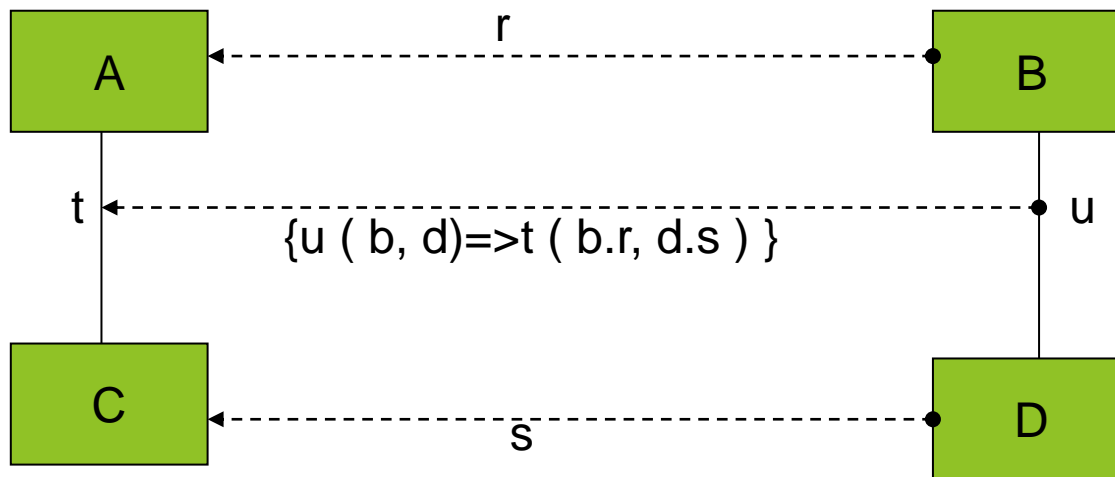


$\{ \text{item 1 contains item 2} \Rightarrow \text{item1.model contains item2.model} \}$

Homomorphism for a parts catalog

- ▶ In general, a homomorphism involves 4 relationships among 4 classes.
- ▶ The homomorphism maps links of one general association $\{u\}$ into links of another general association $\{t\}$ as a many-to-one mapping.
- ▶ Two instantiation relationships map elements of one class into another:
 - ▶ r is a many-to-one mapping from class B to A and
 - ▶ s is a many-to-one mapping from class D to C.

- ▶ In general where t is on a single class and u is on a single class then $A=C$, $B=D$ and $r=s$.
- ▶ Homomorphism is a special type of relationship between relationships.



General Homomorphism

UNIT 3

Dynamic Modeling

- ▶ First examine the system at a single moment of time.
- ▶ Then examine changes to objects and their relationships over time.
- ▶ Those aspects of a system that are concerned with time and changes are dynamic model, in contrast with static or object model.
- ▶ Control is the aspect of a system that describes the sequences of operations that occur in response to external stimuli without considering what the operation do, what they operate on, and how they are implemented.

Concepts


- ▶ Events - represent external stimuli
- ▶ States - values of objects
- ▶ State diagram

Events

- ▶ It is something that happens at a point in time
- ▶ One event may logically precede or follow another or the two events may be unrelated.
- ▶ The two events are casually related.
- ▶ The two events are casually unrelated are said to be concurrent i.e. they have no effect on each other.
- ▶ In modeling a system we do not establish an ordering between concurrent events because they can occur at any order.
- ▶ An event is a one-way transmission of information from one object to another.
- ▶ In real world, all objects exist concurrently.

- ▶ An object sending an event to another object may expect a reply but the reply is a separate event under the control of the second object which may or may not choose to send it.
- ▶ Every event is a unique occurrence, but we group them into event classes and give each event class a name to indicate common structure and behavior.
- ▶ This structure is hierarchical as class structure.
- ▶ E.g. flight 123 departs from Delhi and flight 321 departs from Rome are instances of event class airplane flight departs having attributes airline flight no, city.
- ▶ The time at which an event occurs is an implicit attribute of all events.

- ▶ An event conveys information from one object to another.
- ▶ Some classes of events may be simply signals that something has occurred while other convey data values i.e. attributes.
- ▶ Showing attributes is optional.



Airplane flight departs (airline, flight#, city)
Mouse button pushed (button, location)
Input string entered (text)
Phone receiver lifted
Digit dialed (digit)

Event classes and attributes

Scenarios and event traces

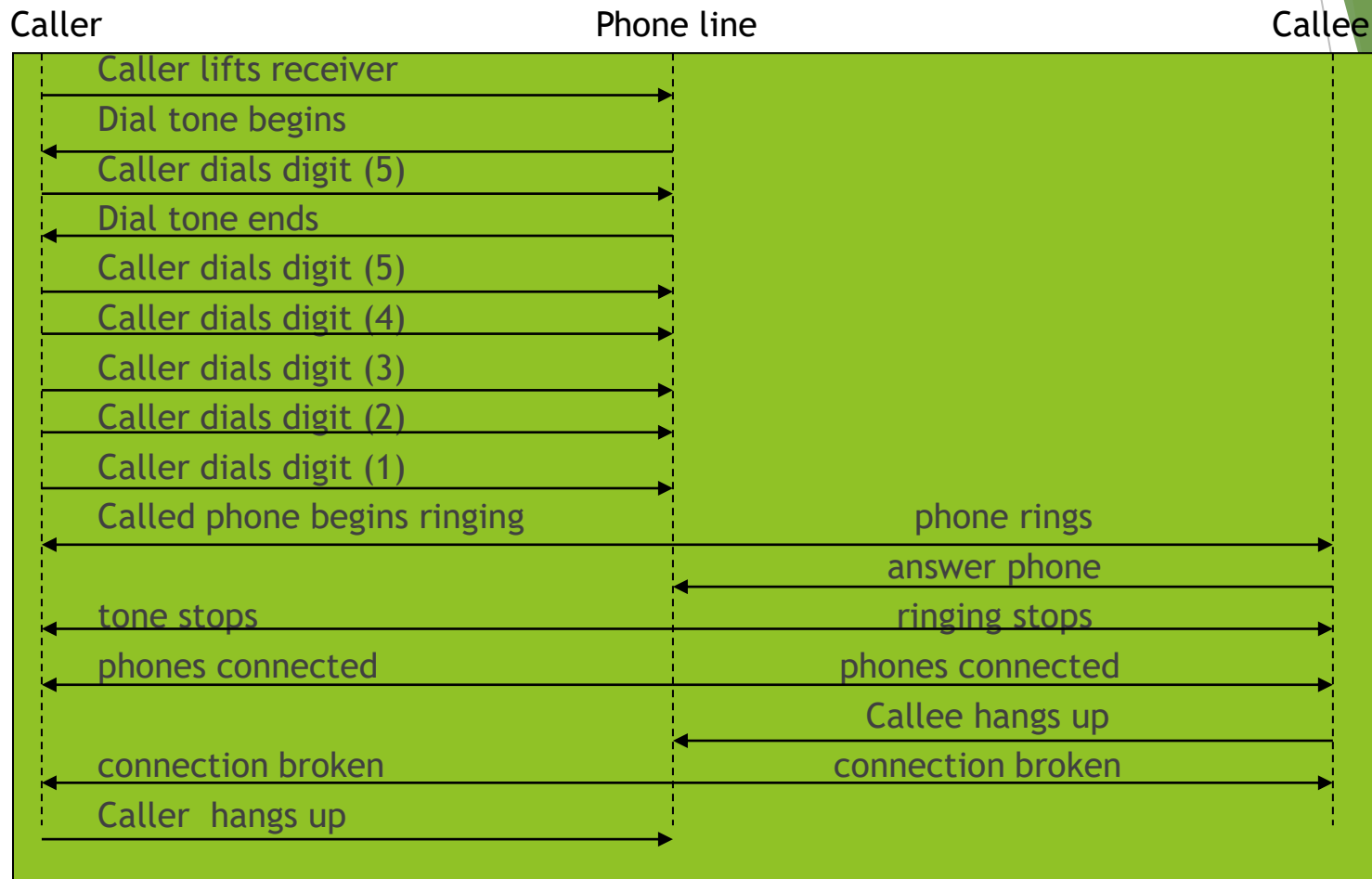
- ▶ A scenario is a sequence of events that occurs during one particular execution of a system.
- ▶ The scope of a scenario can vary; it may include all events in the system or it may include only those events generated by certain objects in the system.
- ▶ It can be the historical record of executing a system or a thought experiment of executing a proposed system.

Caller lifts receiver
Dial tone begins
Caller dials digit (5)
Dial tone ends
Caller dials digit (5)
Caller dials digit (4)
Caller dials digit (3)
Caller dials digit (2)
Caller dials digit (1)
Called phone begins ringing
Ringing tone appears in calling phone
Called party answers
Called phone stops ringing
Ringing tone disappears in calling phone
Phones are connected
Called party hangs up
Phones are disconnected
Caller hangs up

Scenario for phone call

- ▶ The next step after writing scenario is to identify the sender and receiver objects of each event.
- ▶ The sequence of events and the objects exchanging events are shown in event trace diagram.
- ▶ This diagram shows each object as a vertical line and each event as a horizontal arrow from the sender to receiver objects.
- ▶ Time increases from top to bottom.
- ▶ Sequences of events are shown not their exact timing.

Event trace for phone call



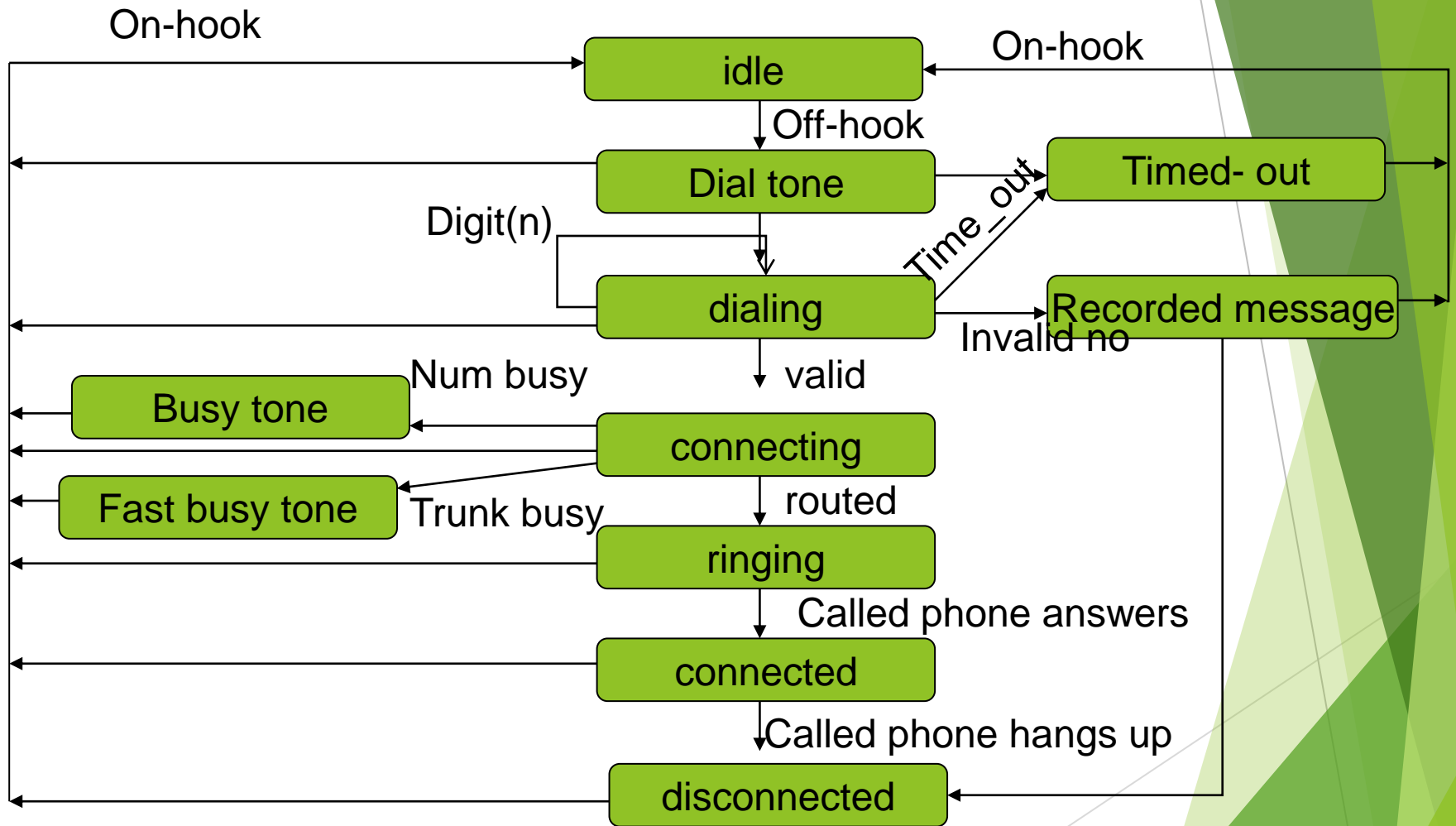
States

- ▶ It is an abstraction of the attribute values and links of an object.
- ▶ Set of values are grouped together into a state according to properties that affect the gross behavior of the object e.g. the state of bank is either solvent or insolvent whether its assets are greater than liabilities.
- ▶ A state specifies the response of the object to input events.
- ▶ Response is same for all values within the same state and may be different for values in different states.
- ▶ Response of an object to an event may include an action or change of states by the object.

- ▶ A state has duration
- ▶ An event separates two states and a state separates two events.
- ▶ In defining states we ignore those attributes that do not affect the behavior of the objects.

State diagram

- ▶ It relates events and states.
- ▶ When an event is received, the next state depends on the current state as well as the event; a change of state caused by an event is called a transition.
- ▶ State diagram is a graph whose nodes are states and whose directed arcs are transitions labeled by event names.



State diagram for phone line

- ▶ A state diagram describes the behavior of a single class.
- ▶ Since all instances of a class have same behavior they all share the same state diagram as they all share the same class features.
- ▶ But as each object has its own attribute values so each object has its own state.
- ▶ Each object is independent of the other objects and proceeds as its own pace.

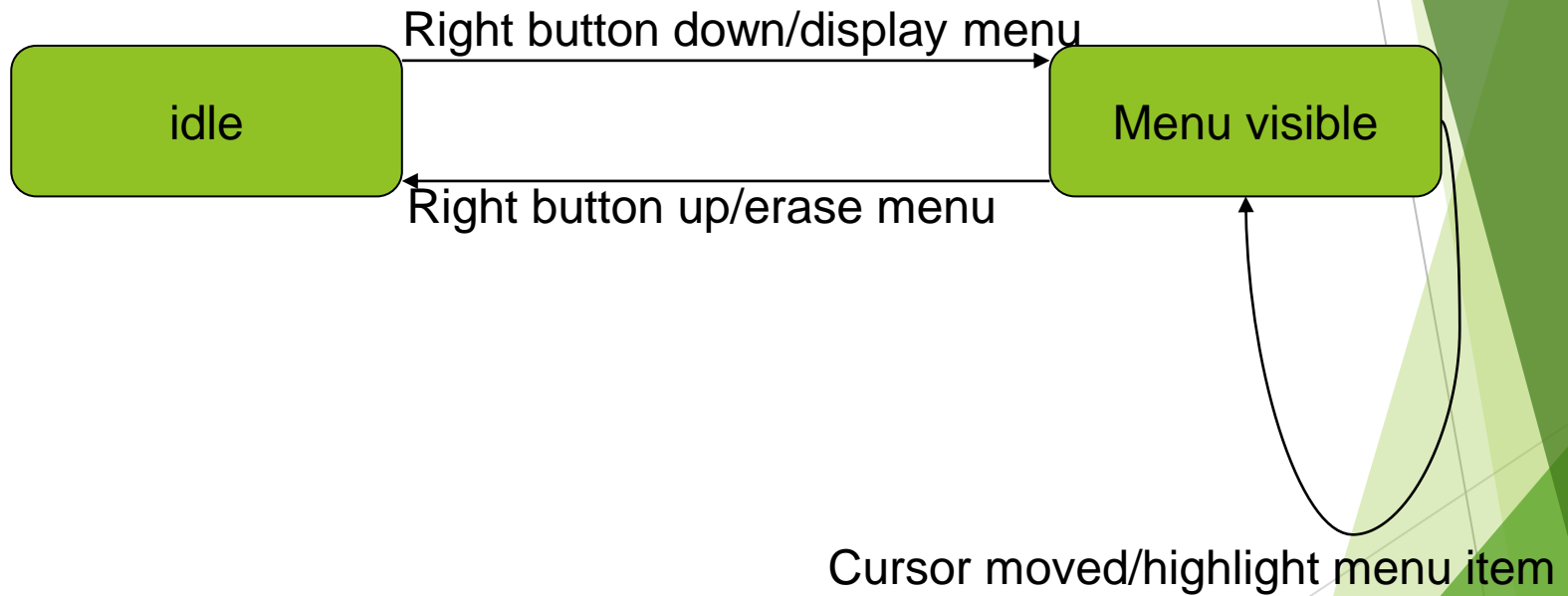
Conditions

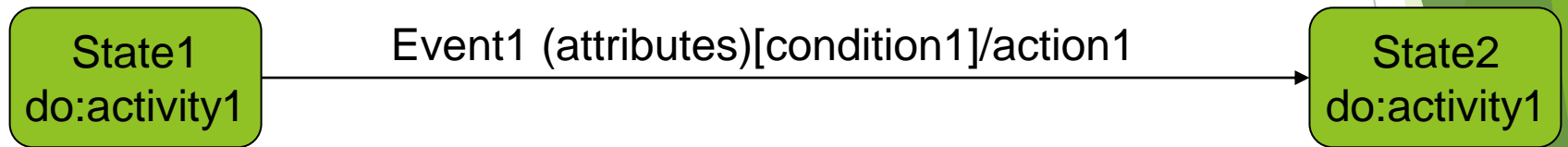
- ▶ A condition is a Boolean function of object values.
- ▶ When Ram goes out in morning(event), if the temperature is below freezing(condition), then he puts on his gloves (next state).
- ▶ Notation is [condition]

Operations

- ▶ An activity is an operation that takes time to complete.
- ▶ An activity is associated with a state.
- ▶ Notation “do:A” within a state box indicates that activity A starts on entry to the state and stops on exit.
- ▶ An action is an instantaneous operation.
- ▶ An operation is associated with an event.
- ▶ An action represents an operation whose duration is insignificant compared to the state diagram e.g. disconnect phone line might be an action in response to an on_hook event for phone line.
- ▶ Actions can also represent internal control operations such as setting attributes or generating other events.
- ▶ The notation for an action is (“/”) and the name of action, following the name of the event that causes it.

Action for pop-up menu

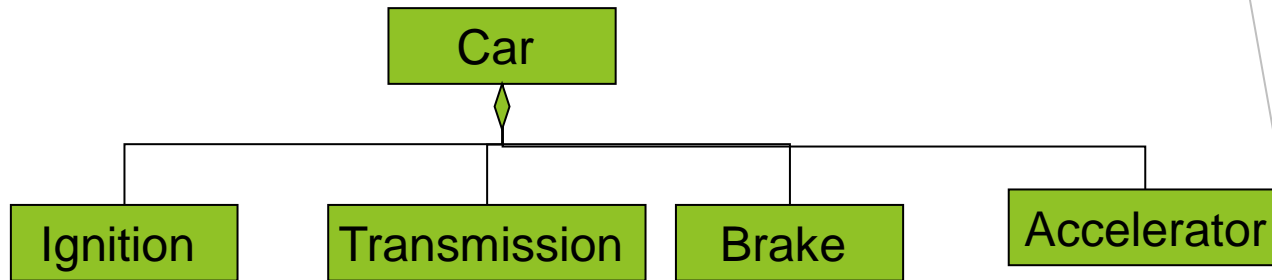




State generalization

- ▶ A nested state diagram is a form of generalization on states.
- ▶ Generalization is “or-relation”.
- ▶ States may have sub states that inherit the transitions of their super states just as classes have subclasses that inherit the attributes and operations of their super classes.

Aggregation and its concurrent state diagram



Ignition

Turn key to
start[Transmission
in Neutral]

Released Key

off

starting

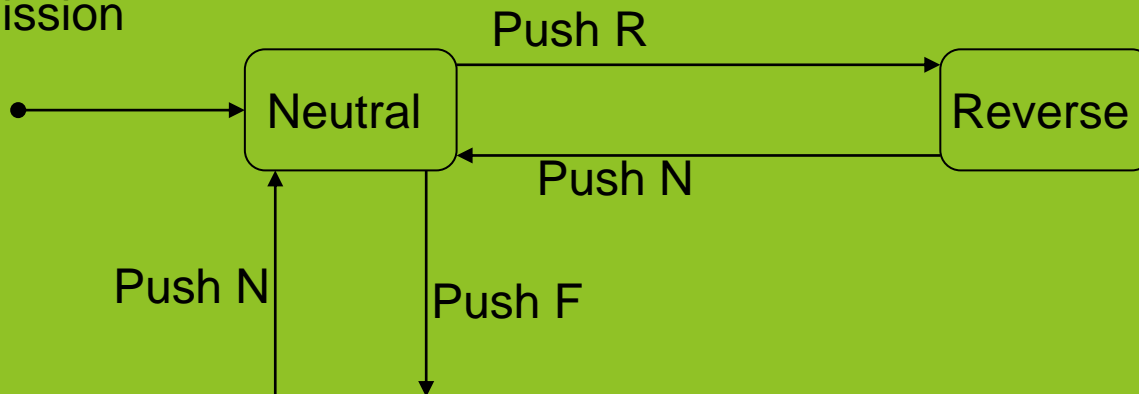
on

Turn key off

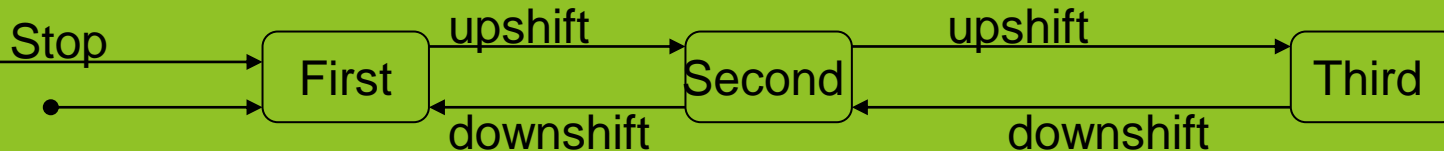


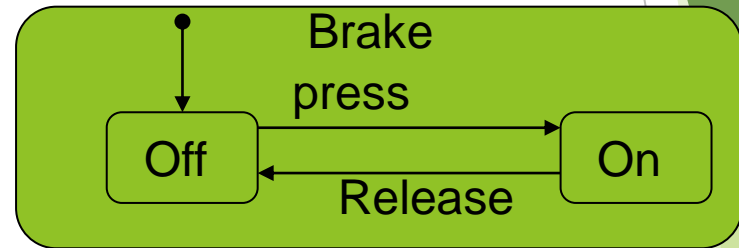
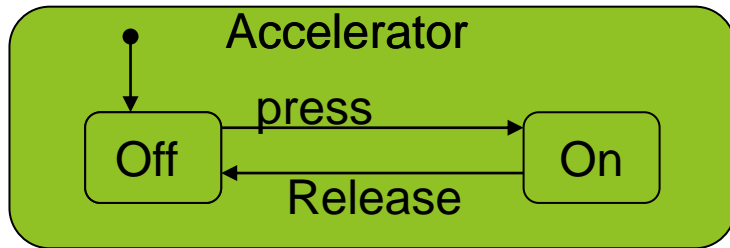
State of car transmission with generalization

Transmission



Forward





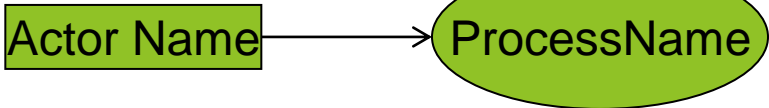
Functional Modeling

- ▶ It specifies the results of a computation without specifying how or when they are computed.
- ▶ It specifies the meaning of operations in object model and actions in dynamic model as well as constraints in object model.
- ▶ Functional model exists where transformation is a key factor.

DFD

- ▶ It shows the functional relationships of the values computed by a system, including input values, output values and internal data stores.
- ▶ A DFD is a graph showing the flow of data values from their sources in objects through processes that transform them to their destinations in other objects.
- ▶ It does not show control information like time, etc.

Elements of DFD

- ▶ Processes -> Transform data
- ▶ Data flow -> move data
- ▶ Actor -> objects that produce and consume data.


```
graph LR; Actor[Actor Name] --> Process([ProcessName]);
```
- ▶ Data store -> objects that store data passively.



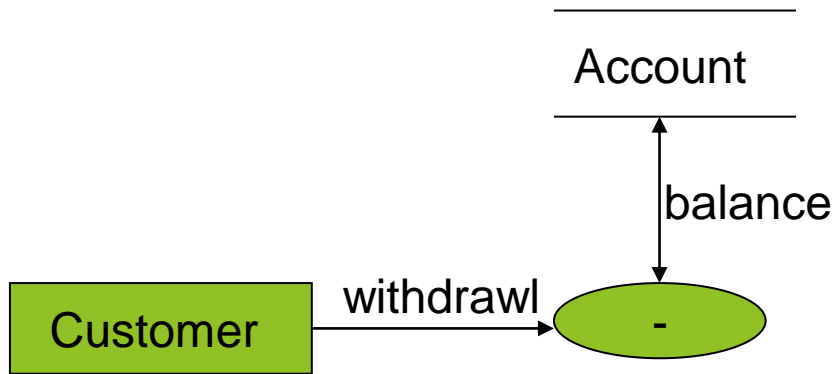
Data Stores

- ▶ It is a passive object within the graph that stores data.
- ▶ It does not generate any operation but merely responds to requests to store and access data.
- ▶ It is drawn as a pair of parallel lines containing the name of the store.
- ▶ Input arrows indicate information or operations that modify the stored data.
- ▶ Output arrows indicate information retrieved from data store.
- ▶ In general data store is implemented as a file.

Data store for temperature readings

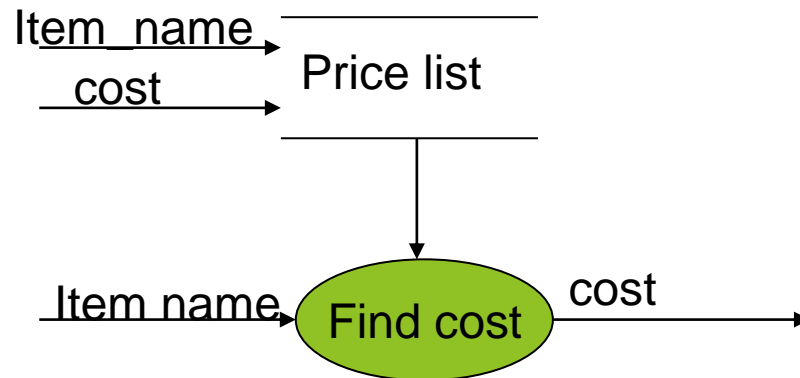


Data store for bank account

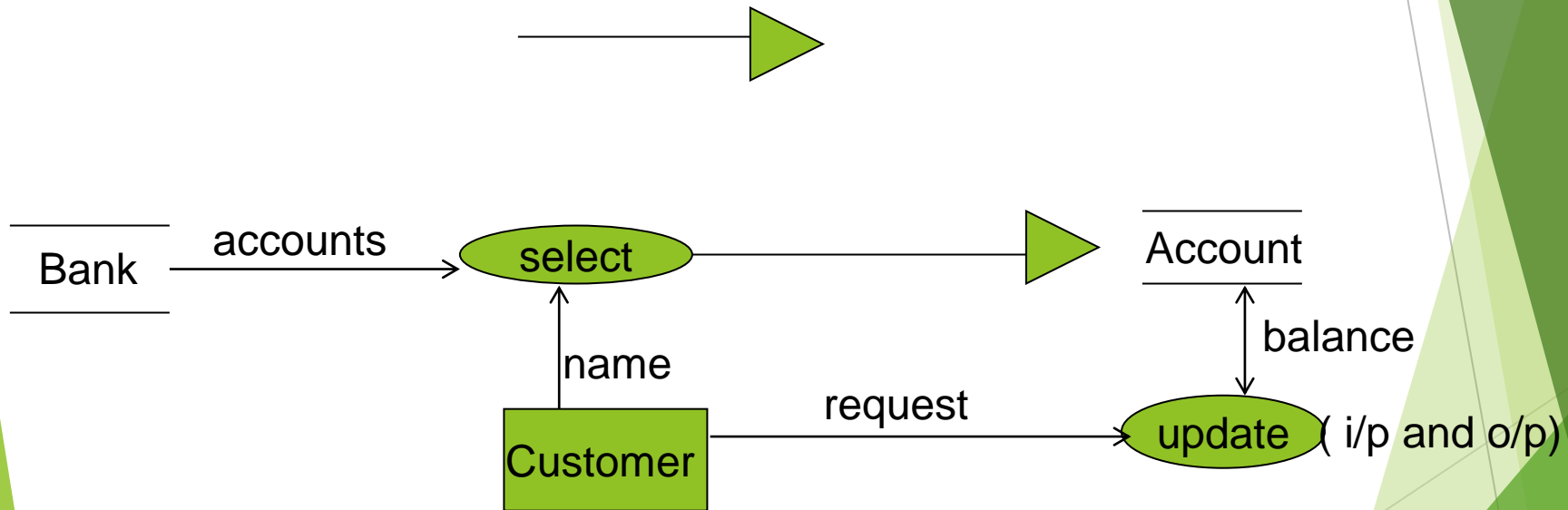


Double headed arrow indicates that balance is both an i/p and o/p of Subtract operation.

Data store for price list for items

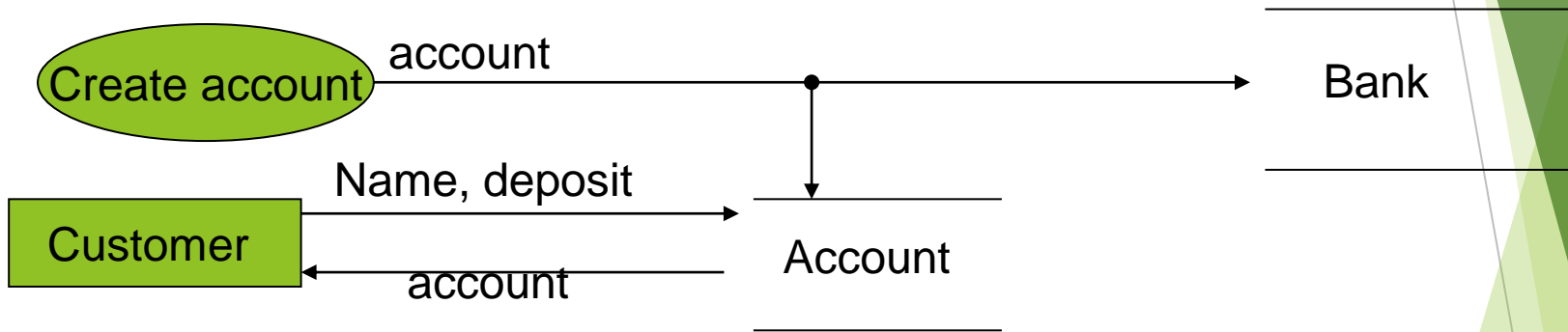


- ▶ A data flow that generates an object used as the target of another operation is indicated by a hollow triangle at the end of data flow.



Selection with an object as result

► Creation of new object

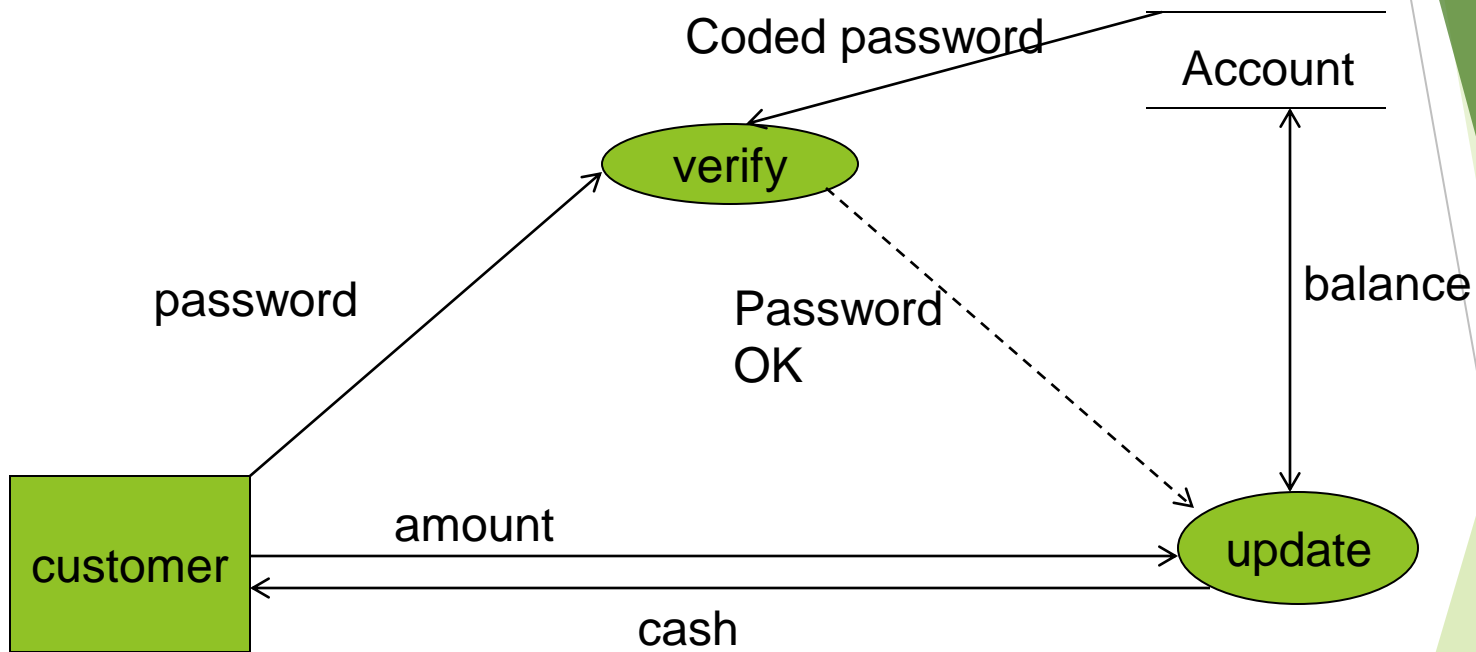


Nested DFD

- ▶ The nesting of a diagram is also called leveling.
- ▶ A process can be expanded into another DFD.

Control Flows

- ▶ A data flow diagram shows all possible computation paths for values.
- ▶ Decisions and sequencing are control issues that are part of dynamic model.
- ▶ It is sometime useful to include it in functional model so that they can not be forgotten. This is done by including control flows in DFD.
- ▶ A control flow is a Boolean value that affects whether a process is evaluated or not.
- ▶ It is shown by a dotted line from a process producing a Boolean value to the process being controlled.



Specifying Operations

Each operation may be specified as-

- ▶ Mathematical functions
- ▶ Table of input and output values.
- ▶ Equations
- ▶ Pre and post conditions
- ▶ Decisions tables
- ▶ Pseudo code, etc

- ▶ Specification of an operation includes a signature and a transformation.
- ▶ Signature defines the interface to the operation i.e. arguments, value returns.
- ▶ The transformation defines the effect of an operation i.e. the output values and the side effects of the operation on its operand objects.

Operations can be divided into three parts-

- ▶ Queries
- ▶ Actions
- ▶ Activities

Query

- ▶ A query is an operation that has no side effects on externally visible state of any object.
- ▶ An action is a transformation that has side effects on the target object or other objects in the system reachable from the target object.
- ▶ Actions can be defined by mathematical equations, decision trees, decision tables, etc.

Activity

- ▶ It is an operation to or by an object that has duration in time.
- ▶ It has inherently side effects because of its extension in time.
- ▶ Activity only make sense for actors.

Constraints

- ▶ It shows the relationship between two objects at the same time (frequency and wavelength)Or between different values of the same object at different times (no of outstanding shares of mutual fund).
- ▶ It can appear in each model.
- ▶ Object constraints specify that some objects depend entirely or partially on other objects.
- ▶ Dynamic constraints specify relationships among the states or events of different objects.
- ▶ Functional constraints specify restrictions on operations.
- ▶ A constraint between values of an object over time is called as invariant i.e. it specifies that some functions of values remain constant over time.

FM Vs OM & DM

- ▶ FM shows what has to be done by a system.
- ▶ The object model shows the “doers”- the object.
- ▶ DM shows the sequences in which the operations are performed.
- ▶ The three models come together in the implementation of methods.
- ▶ FM is a guide to the methods.
- ▶ A process is usually implemented as a method.
- ▶ Actors and data stores are objects in the object model.
- ▶ Data flows are values in object model.

- ▶ Relative to functional model the object model shows the structure of the actors, data store and flows.
- ▶ Relative to functional model the dynamic model shows the sequence in which processes are performed.
- ▶ Relative to object model the functional model shows the operations on classes and the arguments of each operations.
- ▶ Relative to object model the dynamic model shows the states of each object and the operations that are performed as it receives events and changes state.

Structured analysis/structured design (SA/SD)

- ▶ During analysis DFD, data dictionary, state transition diagram and ER diagram are used to logically describe a system.
- ▶ In the design phase, details are added to the analysis models and the DFDs are converted into structure chart describing program language code.

Data dictionary

- ▶ A paragraph is written that precisely describe each class.
- ▶ Describe the scope of class, any assumptions, restrictions.
- ▶ It also describes associations, attributes and operations.

SA/SD and OMT

- ▶ Both support object, dynamic and functional model for a system.
- ▶ OMT designs are dominated by object modeling while SA/SD stresses on functional decomposition.
- ▶ SA/SD organizes a system around procedures while OMT organizes a system around real world objects or conceptual objects that exist in user's view of the world.
- ▶ SA/SD is useful for problems where functions are more important and complex than data.

- ▶ SA/SD design has a clearly defined system boundary across which the software procedures must communicate with the real world. so it can be difficult to extend a SA/SD design to a new boundary.
- ▶ It is much easier to extend object oriented design; one merely design adds objects and relationships.
- ▶ In SA/SD the decomposition of a process into sub process is somewhat arbitrary.
- ▶ Different people will produce different decomposition.
- ▶ In OMT the decomposition is based on objects in problem domain, so developers of different programs in same domain tend to discover similar objects. This increases reusability of components from one project to the next.

- ▶ An OMT approach better integrates database with programming code.
- ▶ Object oriented database may improve the situation.
- ▶ In contrast, SA/SD is inherently awkward at dealing with databases. It is difficult to merge programming code organized about functions with database organized with data.

Reasons to use SA/SD

- ▶ Programmers have tended to think in terms of functions so data flow based method have been easier to learn.
- ▶ Another reason is historical; SA/SD was one of the first well-thought-out formal approach to software and system development.

Jackson Structured Development (JSD)

- ▶ Developed by Michael Jackson
- ▶ Popular in Europe.
- ▶ It does not distinguish between analysis and design rather lumps both phases together as specification.
- ▶ It first determine “what” and then “how”
- ▶ It is useful for application where timing is important.
- ▶ It is less graphical oriented than SA/SD and OMT.

- ▶ JSD model describes the real world in terms of entities, action and ordering of actions.
- ▶ Entities usually appear as nouns in requirements statements and actions appear as verbs.
- ▶ JSD consists of six sequential steps-

- ▶ Entity action
- ▶ Entity structure
- ▶ Initial model
- ▶ Function
- ▶ System timing
- ▶ implementation

Entity action

- ▶ The software developer lists entities and actions for part of real world.
- ▶ The purpose is to guide the choice of entities and actions for overall system.
- ▶ The input to this step is requirement statement
- ▶ The output is list of entities and actions.
- ▶ The entity structure step partially orders the actions of each entity by time.
- ▶ The initial model step states how the real world connects to the abstract model.

- ▶ The function step uses pseudo code to state outputs of actions.
- ▶ At the end of this step the developer has a complete specification of required system.
- ▶ The system timing step considers how much the model is permitted to gap the real world.
- ▶ The implementation step focuses on the problems of processes scheduling and allocates processors to processes.

JSD and OMT

- ▶ JSD is more obscure than data flow and object-oriented approaches.
- ▶ One reason for this is its reliance on pseudo codes; graphical models are easier to understand.
- ▶ It is also complex because it was specifically designed to handle difficult real time problems

JSD is useful in applications like-

- ▶ More emphasis on actions and less on attributes.
- ▶ Where processes must synchronize with each other
- ▶ Real time software
- ▶ Microcode. It is thorough and makes no assumptions about the availability of an operating system and considers synchronized processing and timing.
- ▶ Programming parallel computers.

JSD is ill suited for applications like-

- ▶ High level analysis. It is ineffective at abstraction and simplification.
- ▶ Databases. It is biased towards actions and away from entities and attributes. It is poor technique for database design.
- ▶ Conventional software running under an operating system. Its abstraction of hundreds or thousands of processes is confusing and unnecessary.

REFERENCE

- ▶ James Rumbaugh et.al, “Object Oriented Modeling and Design”, PHI
- ▶ Grady Booch, James Rumbaugh, Ivar Jacobson, “The Unified Modeling Language User Guide”, Pearson Education.

The background features abstract, overlapping green geometric shapes, primarily triangles and polygons, in various shades of green, creating a modern and dynamic visual effect.

THANK YOU