Identify Classes and Relations

- A square is a polygon
- Shyam is a student
- Every student has a name
- 100 paisa is one rupee
- Students live in hostels
- Every student is a member of the library
- A student can renew his borrowed books
- The Department has many students

Identify Classes & Relations

- A country has a capital city
- A dining philosopher uses a fork
- A file is an ordinary file or a directory file
- Files contain records
- A class can have several attributes
- A relation can be association or generalization
- A polygon is composed of an ordered set of points
- A programmer uses a computer language on a project

Exercise 1: Draw UML Diagrams

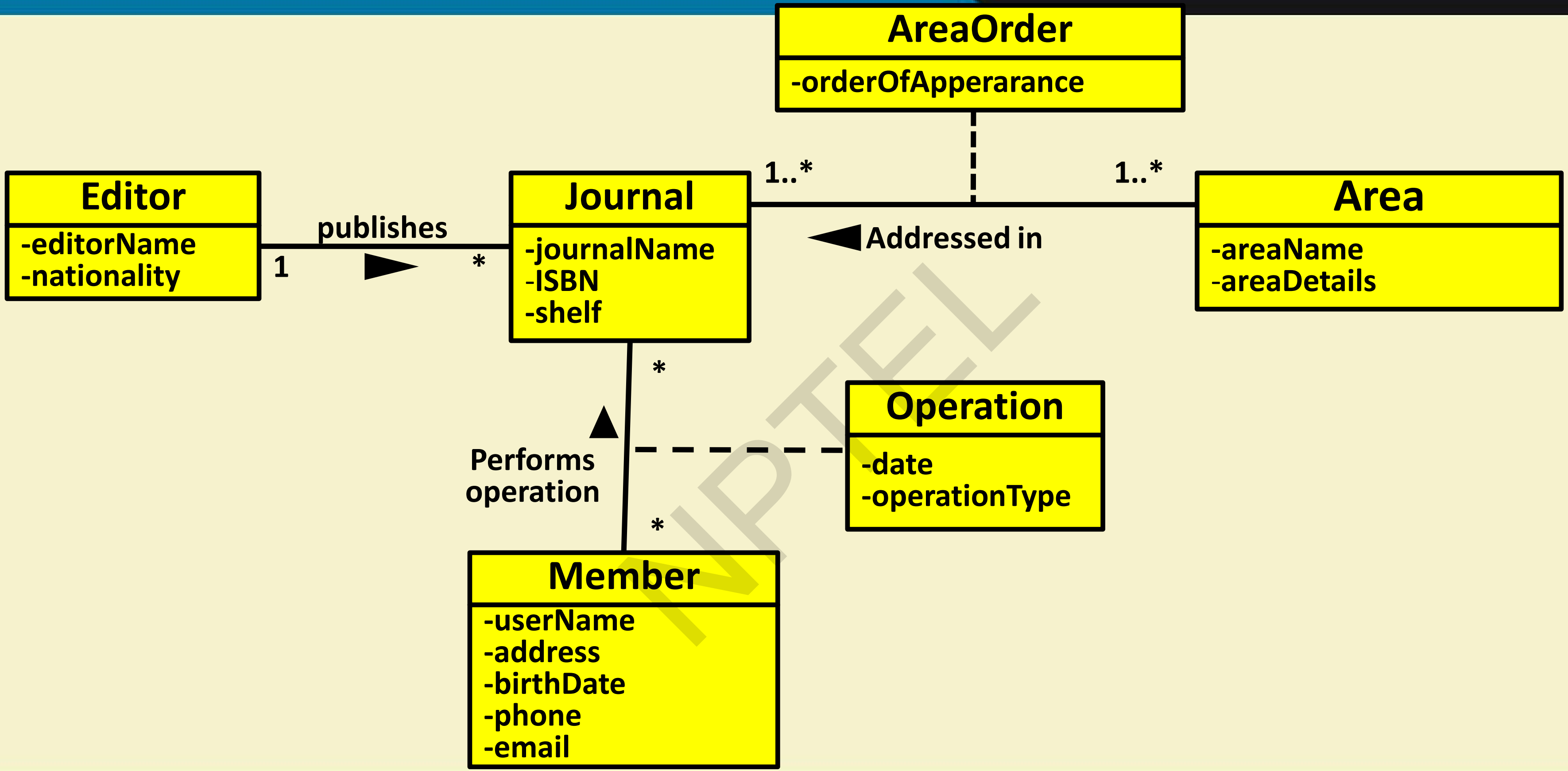
Some persons keep animals as pets.

Exercise 2: Draw UML Diagrams

A company has many employees and undertakes many projects. Each project is carried out by a team of employees.

Exercise

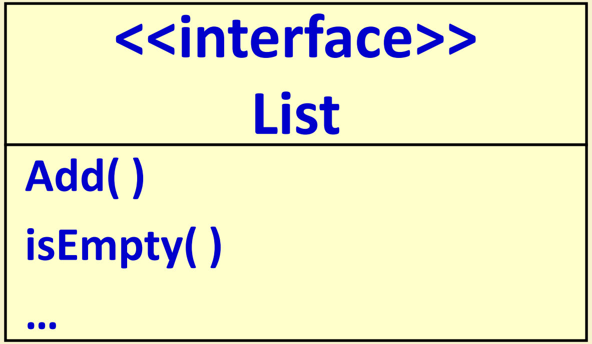
- Develop UML Class diagram for a software that we need to develop to manage a library.
- Each library member has a name, date of birth, address, phone number and email
- Each journal has a name, ISBN code, research areas, shelf in which located, and an editor. Each editor has a name and a nationality and publishes many journals.
- A journal may cover issues in several research areas. It is important to store the order of appearance of research areas, since these indicate their relative importance for the journal.
- It is important to keep track of the date on which a journal has been borrowed and returned, and the member carrying out the operation.

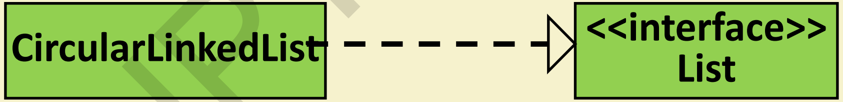


AreaOrder
-orderOfApperarance
Editor
-editorName
-nationality
publishes
1
*
Journal
-journalName
-ISBN
-shelf
1..*
1..*
Addressed in
Area
-areaName
-areaDetails
*
Operation
-date
-operationType
Performs
operation
*
Member
-userName
-address
-birthDate
-phone
-email

Interface

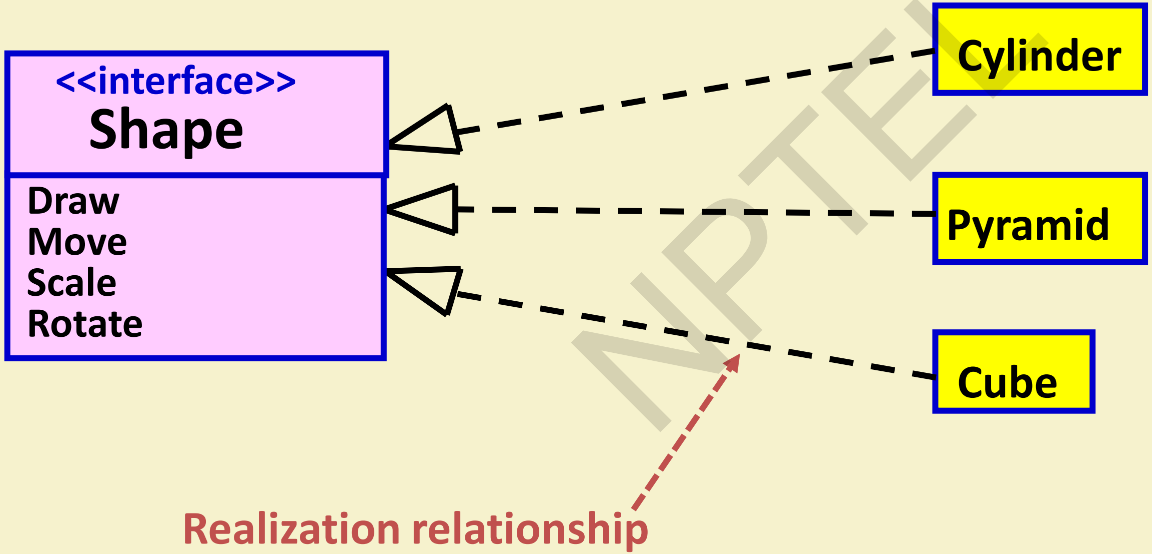
- An interface in UML is a named set of operations.
- Interfaces are used to characterize some behaviour.
  - **Shown as a stereotyped class.**
- Generalization can be defined between interfaces.





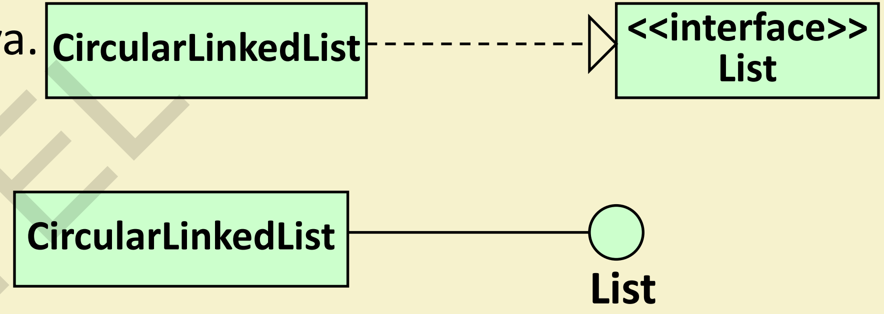
Interface Example

- An interface specifies the set of services to be offered by a class.



Realizing an Interface

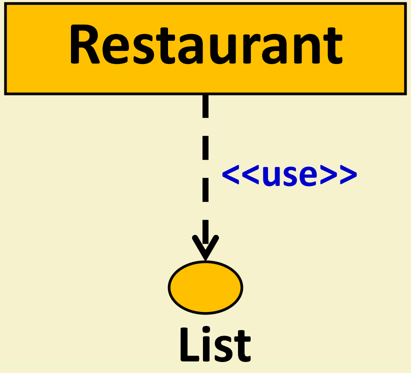
- A class *realizes* an interface if it provides implementations of all the operations.
  - Similar to the *implements* keyword in Java.
- UML provides two equivalent ways of denoting this relationship:

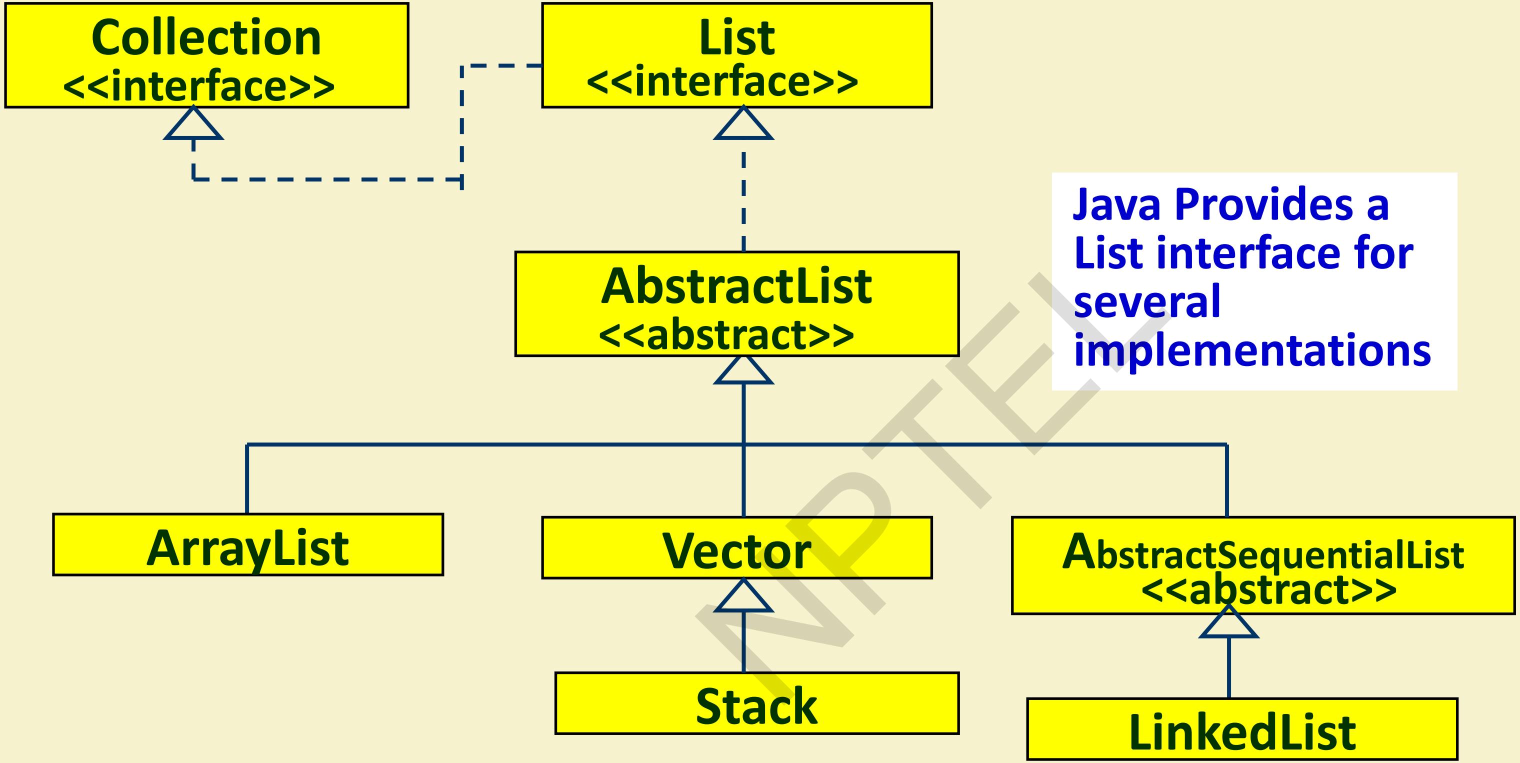


Both represent: **"CircularLinkedList implements all the operations defined by the List interface".**

Interface Dependency

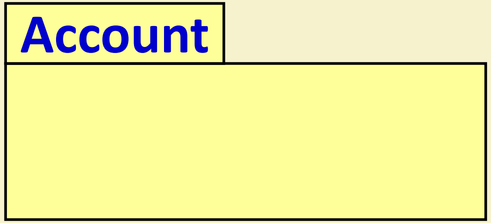
- A class can be dependent on an interface.
  - This means that it **makes use** of the operations defined in that interface.
  - E.g., the Restaurant class makes use of the **List** interface:





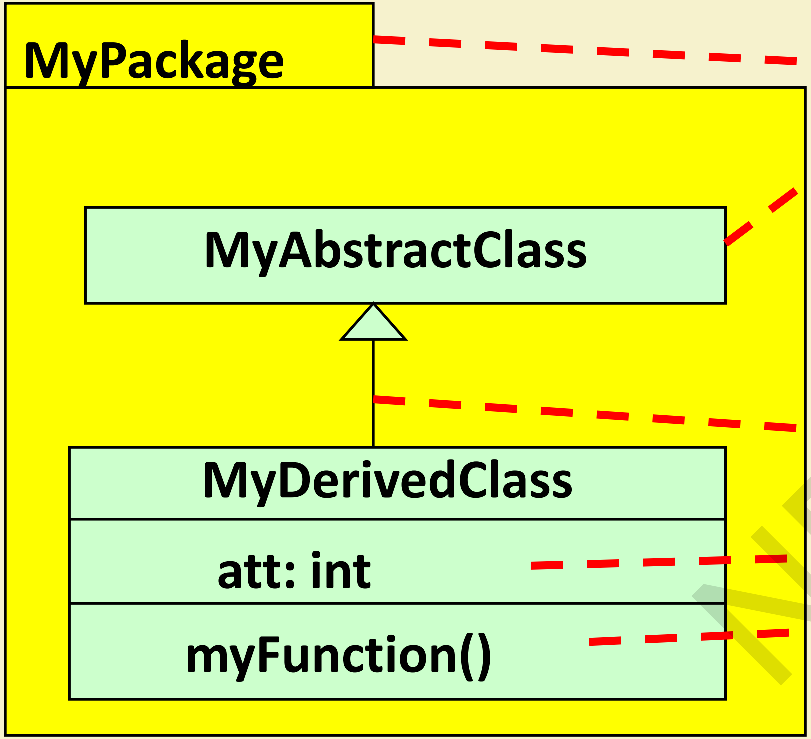
Java Provides a List interface for several implementations

UML Packages



- **A package is a general purpose grouping mechanism.**
  - Can be used to group any UML elements (e.g. use cases, actors, classes, components and other packages.)
- **Commonly used for specifying the logical grouping of classes.**
- A package does not necessarily translate into a physical sub-system.

Mapping to Code




```
package MyPackage;

abstract class MyAbstractClass{}

class MyDerivedClass extends
    MyAbstractClass{

    int att;

    void myFunction() { .. }

}
```

- The B.Tech program of IIT Computer Science Department:
  - comprises of many B.Tech batches.

Exercise

- Each B.Tech batch consists of many B.Tech students.
- CSE Department has many listed courses.
  - A course is offered in many semesters
  - A course is either listed as an elective course or a core course.
- Each B.Tech student credits between 30 to 32 course offering.

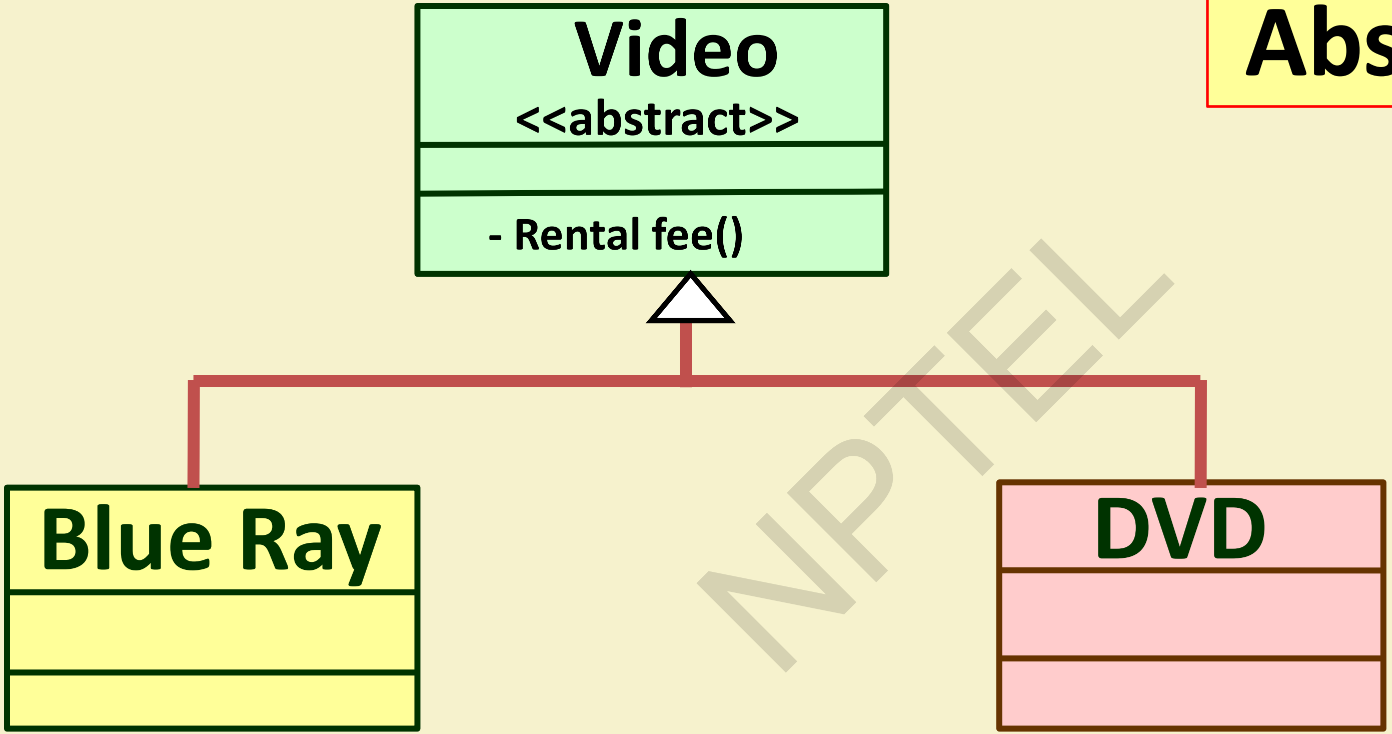
Constraints on Objects

- A constraint restricts the values that objects can take.
- **Example:** No employee's salary can not exceed the salary of his boss.

Abstract Classes

- Not allowed to have objects instantiated from it.
- Used only for inheritance purposes
- Describes common attributes and behavior for other classes.

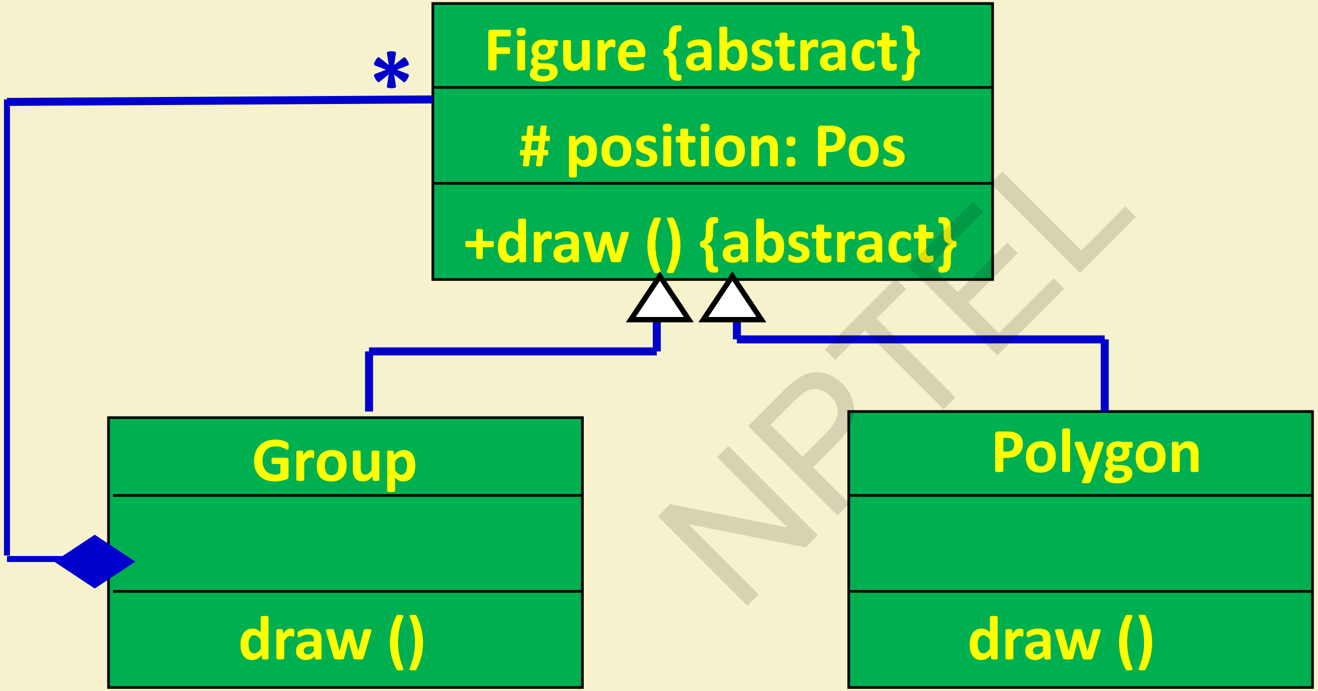
Abstract Class



Abstract Class

- Why use abstract class?
  - Reduces complexity of design
  - Enhances understandability
  - Increases productivity
- It has been observed that:
  - Productivity is inversely proportional to complexity.

Another Abstract Class Hierarchy Diagram



Java Implementation

```
Abstract public class Figure {
        abstract public void draw();
        protected Pos position;
}
public class Group extends Figure {
       private Vector <Figure> figures = new Vector <Figure> ();
       public void draw () {
     }
}
public class Polygon extends Figure {
          public void draw () {        // draw polygon code
     }
}
```

What is Polymorphism?

- Denotes poly (many) morphism (forms).
- Under different situations:
  - **Same message to the same object can result in different actions.**
- Two types:
  - Static
  - Dynamic

An Example of Static Binding

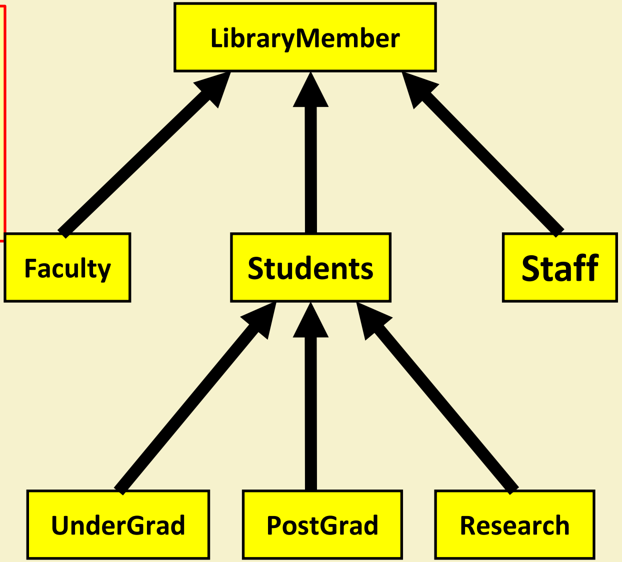
```
Class Circle{
        private float x, y,
        private int fillType;
        public create ();
        public create (float x,float y, float c
        public create (float x, float y, float c
}
```

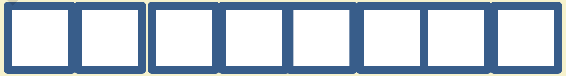
- Assume a class named Circle has three definitions for create operation
  - Without any parameter, default
  - Centre and radius as parameter
  - Centre, radius and fillType as parameter
  - Depending upon parameters, method will be invoked
  - Method **create** is said to be **overloaded**

An Example of Static Binding

- Method call to an object of an ancestor class:
  – Results in invocation of the method of an appropriate object of the derived class.
- Which principles are involved?
  – Inheritance hierarchy
  – Method overriding
  – Assignment to **compatible types**

What is Dynamic Binding?

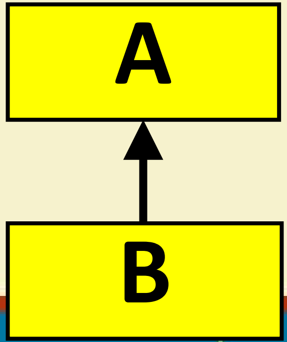




- Principle of substitutability (Liskov's substitutability principle):

Compatible Types

  - An object can be either assigned to or used in place of an object of its ancestor class, but not vice versa.



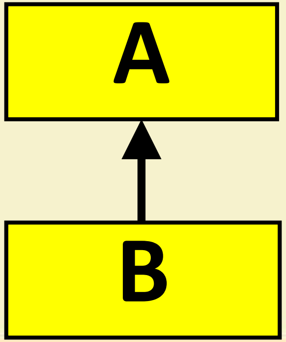
A a; B b;

a=b; (OK)

b=a; (not OK)

Liskov Substitution Principle (Barbara Liskov, 1988)

- If for an object a of type A there is an object b of type B such that A is a subtype of B:
  - **Then it is possible to use b for a.**



A a; B b;

a=b; (OK)

b=a; (not OK)

- Any subclass object should be usable in place of its parent class object.

Or in Plain English

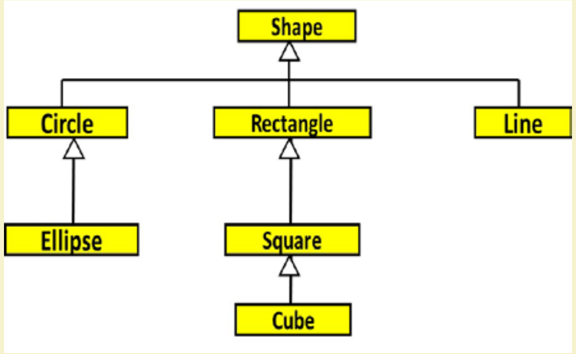
- **Corollary:**
  - All derived classes must honour the contracts of their base classes
  - **IS A = same public behavior**

Dynamic Binding

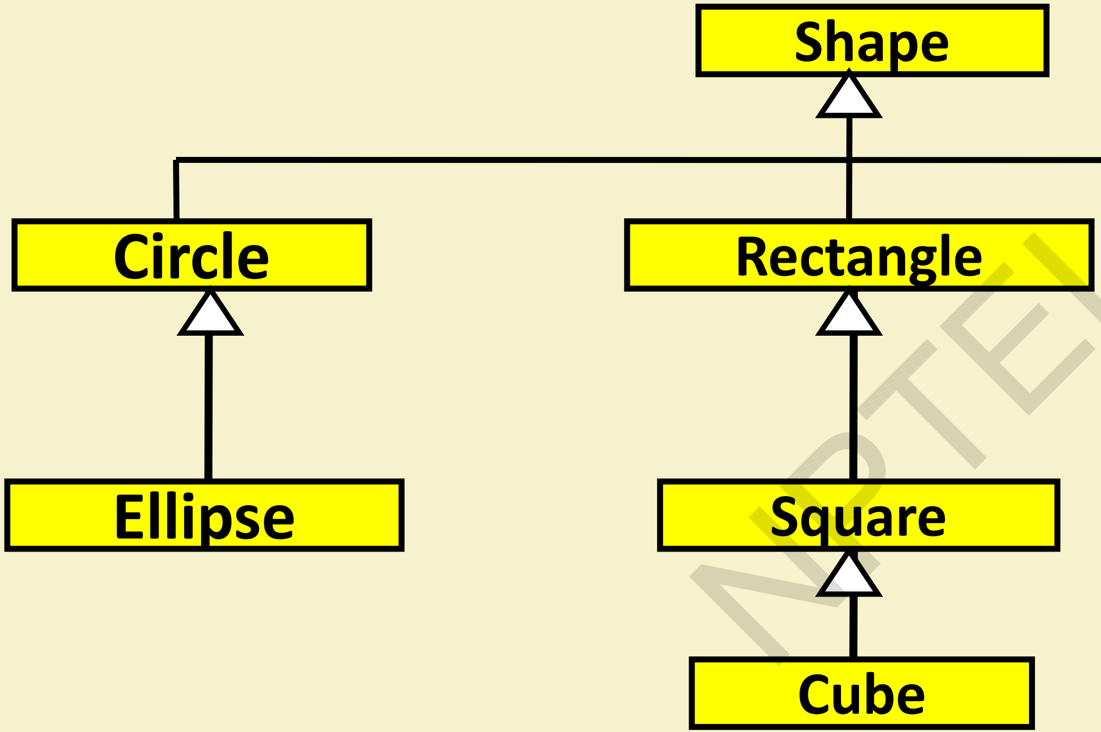
- Exact method to be bound on a method call:
  - Not possible to determine at compile time.
  - Dynamically decided at runtime.

An Example of Dynamic Binding

- Consider a class hierarchy of different geometric objects:
  - Display method is declared in the shape class and overridden in each derived class.
  - A single call to the display method would take care of displaying the appropriate element.



An Example of Dynamic Binding



Class hierarchy of geometric objects

Example Code

Traditional code

```
Shape s[1000];
for(i=0;i<1000;i++){
   If (s[i] == Circle)
         draw_circle();
    else if (s[i]== Rectangle)
           draw_rectangle();
         -
}
```

Object-oriented code using Dynamic Binding

```
Shape [] s=new Shape[1000];
for(i=0;i<s.length;i++)
   s[i].draw();

      -
-
-
-
```

```java
class Shape {
void draw() { System.out.println ("Shape");
    }
}
class Circle extends  Shape {
void draw() { System.out.println ("Circle"); }
}
class Line extends  Shape {
void draw() { System.out.println ("Line"); }
}
class Rectangle extends  Shape {
void draw() {System.out.println
    ("Rectangle"); }
}
```

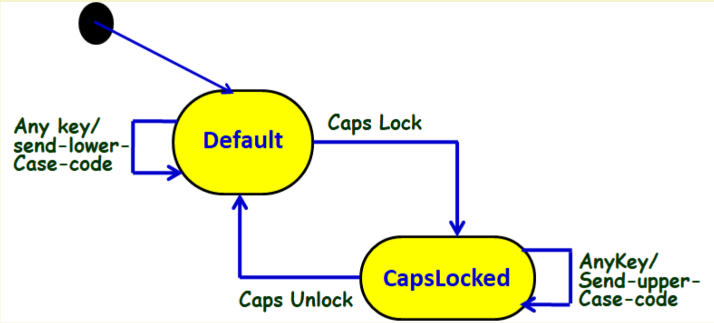
```java
public static void main(String args[]){
   Shape[] s = new Shape[3];
   s[0] = new Circle();
   s[1] = new Line();
   s[2] = new Rectangle();
   for (int i = 0; i < s.length; i++){
     s[i].draw();
    // prints Circle, Line, Rectangle
  }
}
```

State Machine Diagrams

Stateless vs. Stateful Objects

- **State-independent (modeless):**
  - Type of objects that always respond the same way to an event.
- **State-dependent (modal):**
  - Type of objects that react differently to events depending on its state or mode.

  Use state machine diagrams for modeling objects with complex state-dependent behavior.



Stateful Classes

- Give examples of some classes that have non-trivial state models:
  - **Lift controller:** Up, down, standstill,…
  - **Game software controller:** Novice, Moderate, Advanced…
  - **Gui:** Active, Inactive, clicked once, …
  - **Robot controller:** Obstacle, clear, difficult terrain…

- **Controller classes are an important class of stateful examples:**
  - A controller may change its mode depending on sensor inputs and user inputs.

Stateful Objects

- In a client-server system:
  - **Servers are stateless, clients are stateful.**
- Common stateful objects:
  - **Controllers:**
    - A game controller may put the game in expert, novice or intermediate modes.
  - **Devices:**
    - A Modem object could be dialing, sending, receiving, etc.
  - **Mutators** (objects that change state or role)
    - A RentalVideo is rented, inStore, or overDue

Event-Based Programming

- Traditional programs have single flow of control
  - Represented using flowchart or activity diagram
- Event-driven systems :
  - In contrast, depending on an event occurrence, corresponding handler is activated
  - Programming these using traditional approach often not suitable, and would cause wasteful computations.
  - **Represented using state machines.**

Why Create State Model?

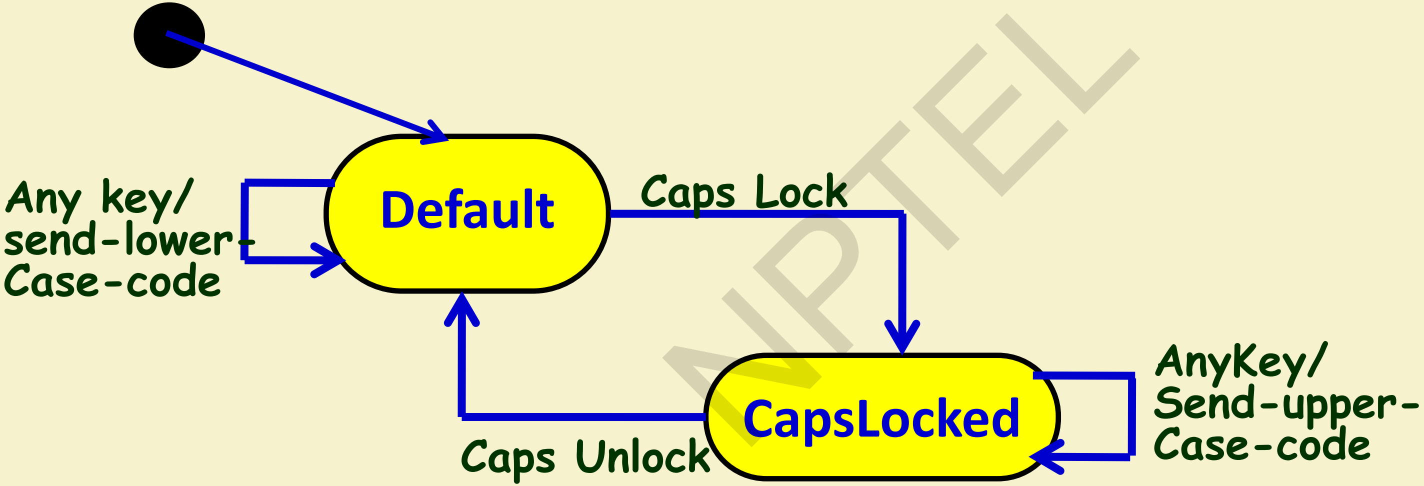
- Tackle complexity
- Document:
  - For review, explaining to others, etc.
- **Generate code automatically**
- Generate test cases

Finite State Automaton

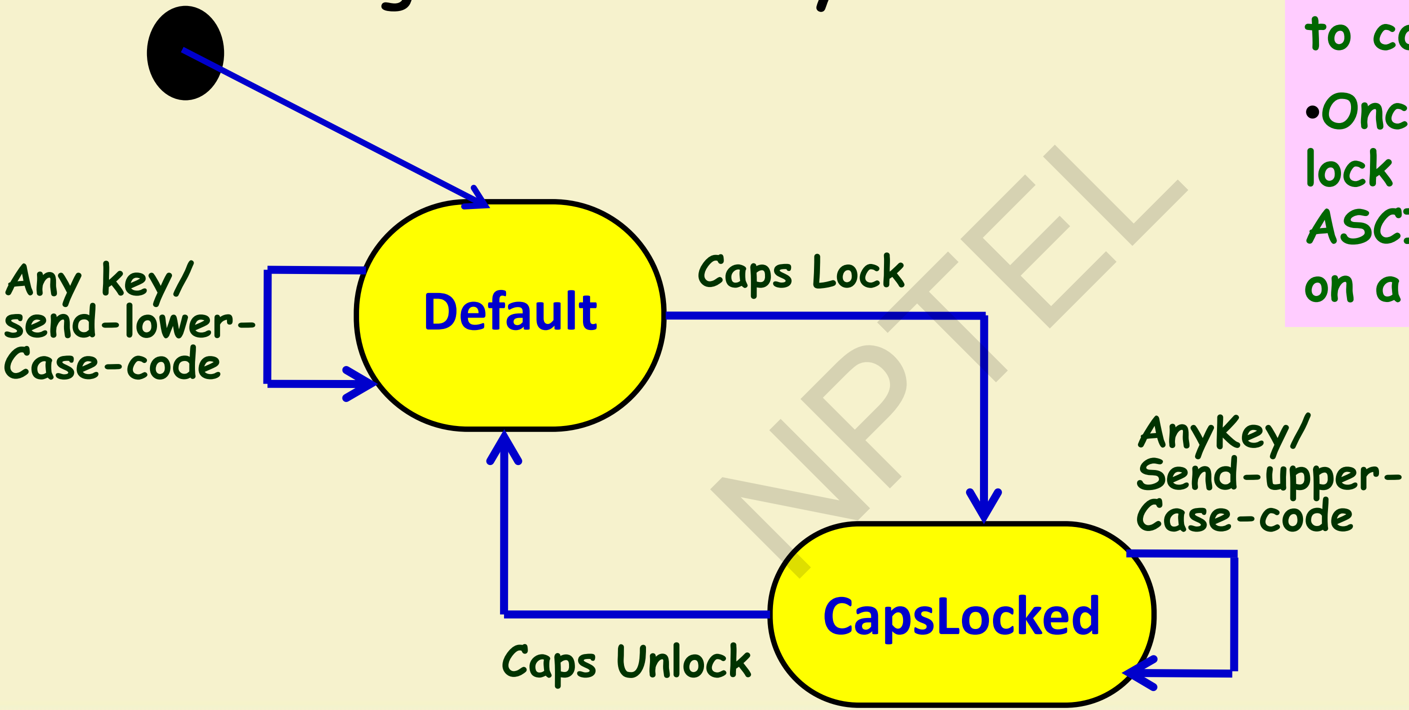
- A machine whose output behavior is not only a direct consequence of the current input,
  - But past history of its inputs
- **Characterized by an internal state which captures its past history.**

Basic State Machine Diagram

- Graphical representation of automata behavior…

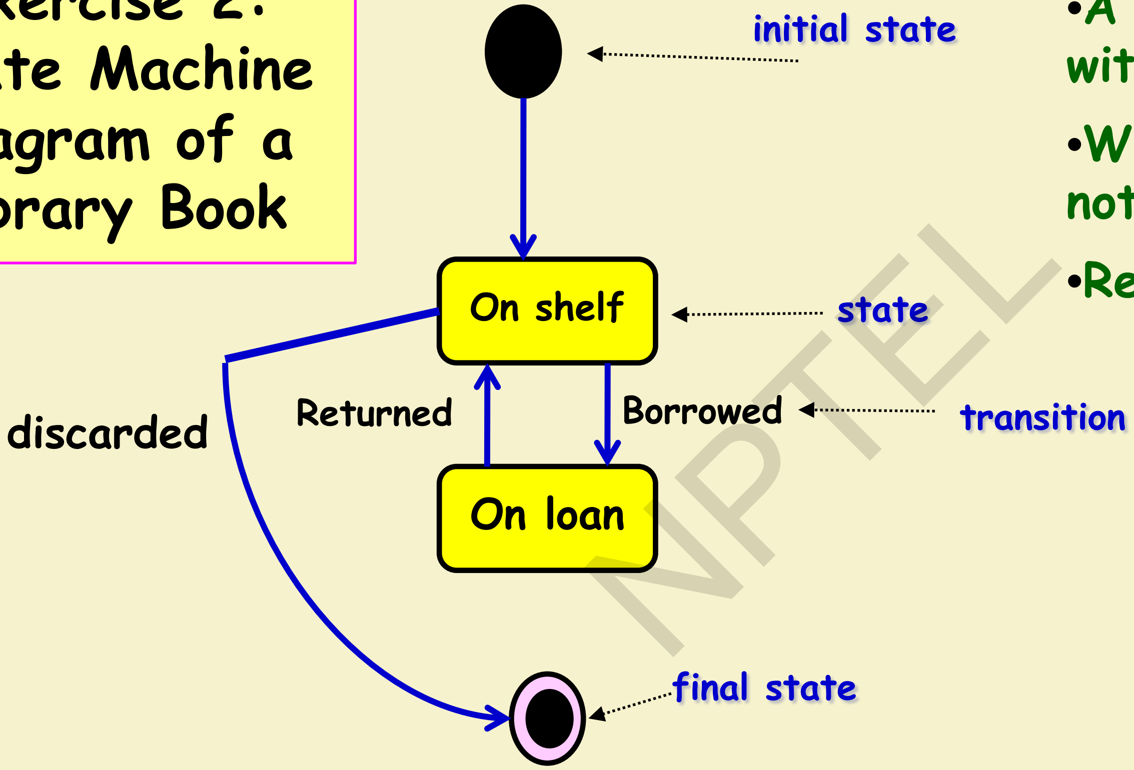


Exercise 1: Draw State Machine Diagram of a Keyboard?



- Press any key: lower case ASCII code is sent to computer…
- Once press the caps lock key: upper case ASCII code will be sent on a key press…

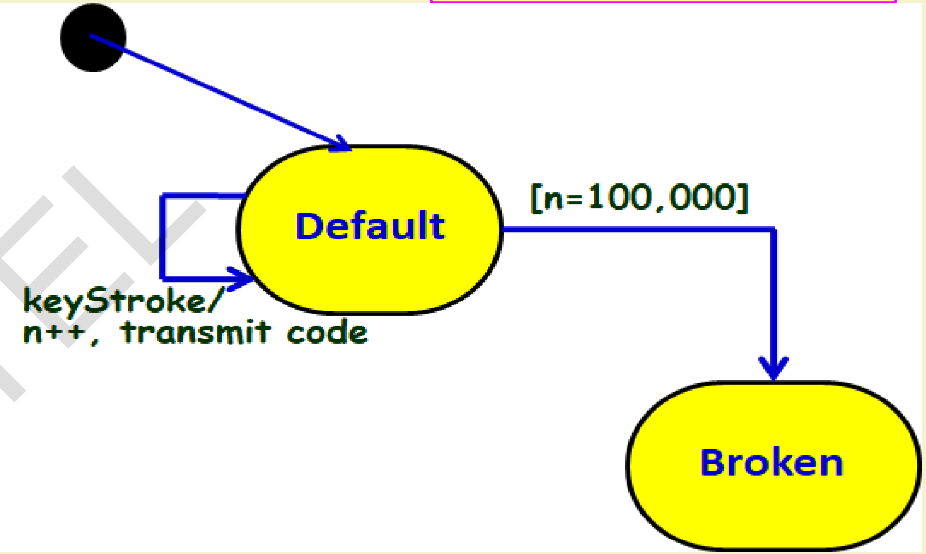
Exercise 2: State Machine Diagram of a Library Book



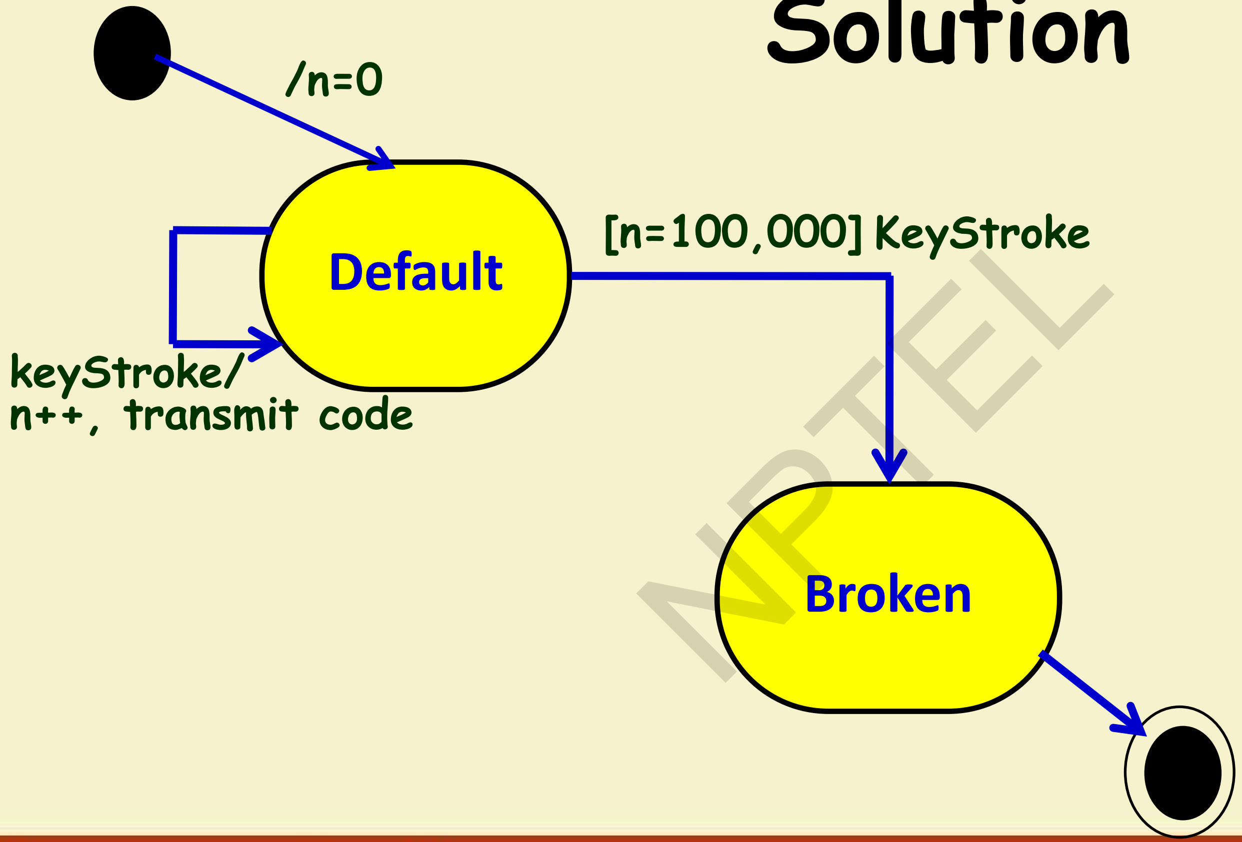
- A library book to start with, is present in a shelf…
- When borrowed out, it is not on shelf…
- Returned, on shelf…

Exercise 3

- Model a keyboard using UML state machine diagram:
  - Transmits key code on each key stroke.
  - Breaks down after entering 100,000 key strokes.



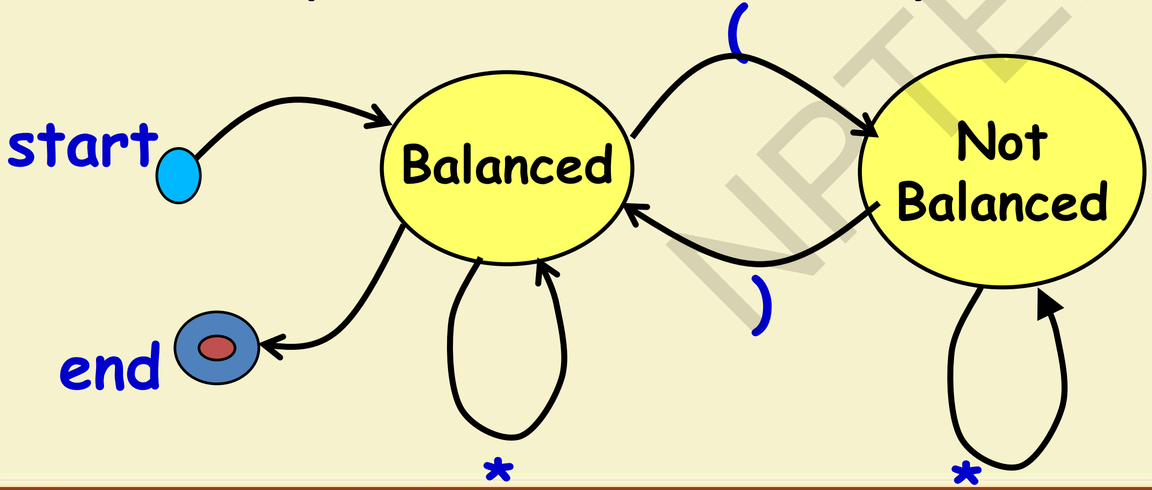
Solution



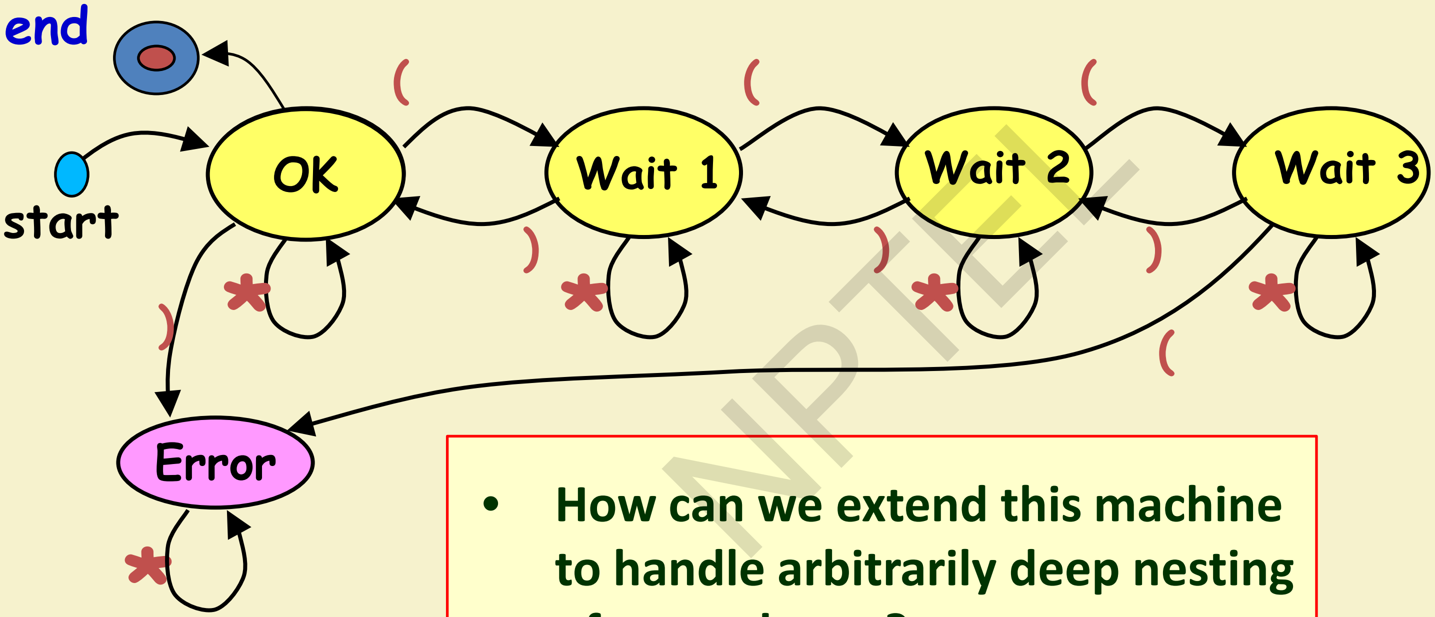
/n=0
Default
[n=100,000] KeyStroke
keyStroke/
n++, transmit code
Broken

Exercise 4: Draw State Machine: GUI Accepts only Balanced Parentheses

- Inputs are any characters
- No nesting of parentheses
- No “output” other than any state change

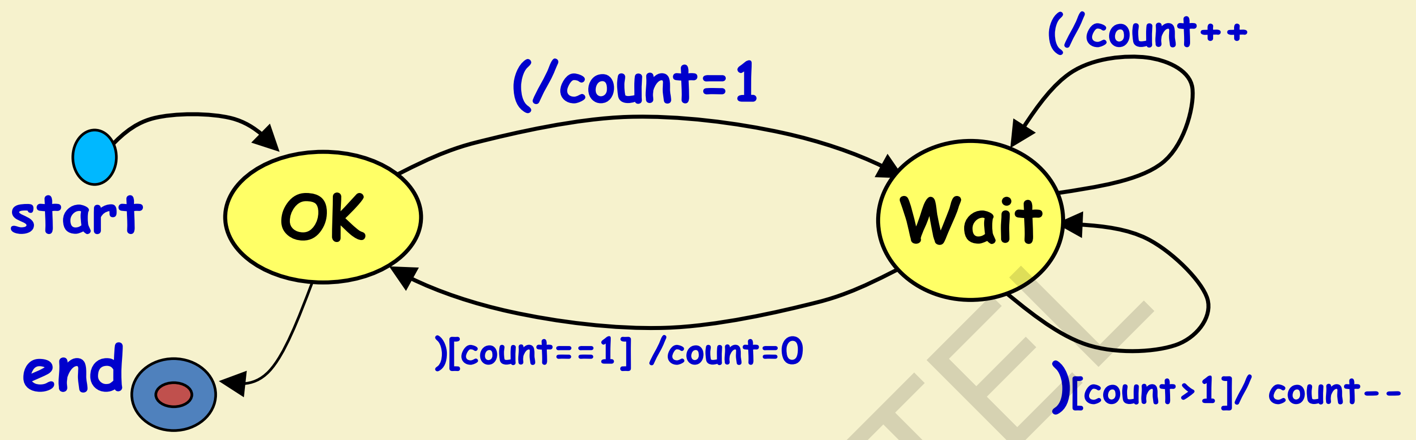


Example 5: Draw State Machine: GUI Accepts only upto 3 Nested parentheses



- How can we extend this machine to handle arbitrarily deep nesting of parentheses?

How to Model Nested parentheses?



- **A state machine, but not *just* a state machine --- an EFSM**
- Addition of variables (“extended”)

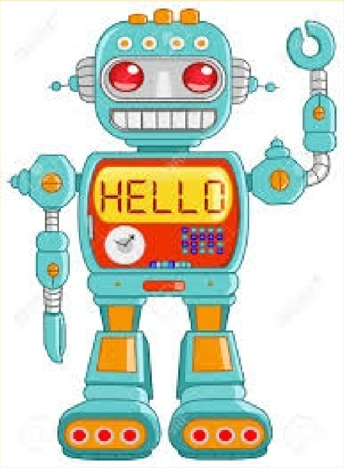
- FSMs suffer from a few severe shortcomings:
  - **What are the shortcomings of FSM?**

State Chart Diagram

- State chart is based on the work of **David Harel** [1990]
  - Overcomes important shortcomings of FSM
  - Extends FSM in 2 major ways: **Concurrent states** and **hierarchy**.

Robot: State Variables

- Power: On, OFF
- Movement: Walk, Run
- Direction: Forward, Backward, left, Right
- Left hand: Raised, Down
- Right hand: Raised, down
- Head: Straight, turned left, turned right
- Headlight: On, Off
- Turn: Left, Right, Straight



How many states in the state machine model?

FSM: exponential rise in number of states with state variables

Event	State
turnOn	Activated
turnOff	Deactivated (Idle)
stop	Stopped
walk	Walking
run	Running
raiseLeftArm	LeftArmRaised
lowerLeftArm	LeftArmLowered
lowerLeftArm	LeftArmLowered
raiseRightArm	RightArmRaised
lowerRightArm	RightArmLowered
turnHead	HeadTurned(direction)
speak	Talking(text)

- State chart avoids two problems of FSM:
  - **State explosion**
  - **Lack of support for representing concurrent states**

State Chart Diagram

- A hierarchical state model:
  - Can have **composite states** --- OR and AND states.