Software Design and Architecture

Engr. Abdul-Rahman Mahmood

DPM, MCP, QMR(ISO9001:2000)

armahmood786@yahoo.com

alphapeeler.sf.net/pubkeys/pkey.htm

pk.linkedin.com/in/armahmood

www.twitter.com/alphapeeler

www.facebook.com/alphapeeler

abdulmahmood-sss alphasecure

armahmood786@hotmail.com

http://alphapeeler.sf.net/me

alphasecure@gmail.com

http://alphapeeler.sourceforge.net

http://alphapeeler.tumblr.com

armahmood786@jabber.org

alphapeeler@aim.com

mahmood_cubix 48660186

alphapeeler@icloud.com

http://alphapeeler.sf.net/acms/

UML Components

components types

- Component:
  - Physical set of object based or functional construct that provides functionality through well defined communication mechanism.
- Types:
  - **Compile time / Link Time**: Object code libraries.
  - **Run-time**: in-memory instantiation of these build time constructs.
  - **Executable:** Example is building an application to perform a count of statements in the source code files:

    cat *.cpp |grep ";"| wc –l
    Here pipe provides a data port to the components for communication.

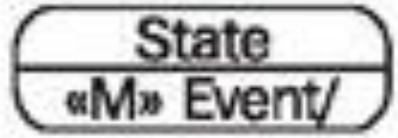
UML State Chart Diagrams

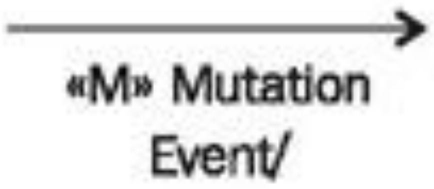
State-chart diagrams

- **State:** The state of an object is always determined by its attributes and associations. States in statechart diagrams represent a *set* of those value combinations, in which an object *behaves the same* in response to events.
- Therefore, not every modification of an attribute leads to a new state.
- **Transition:** A transition represents the change from one state to another. 
-

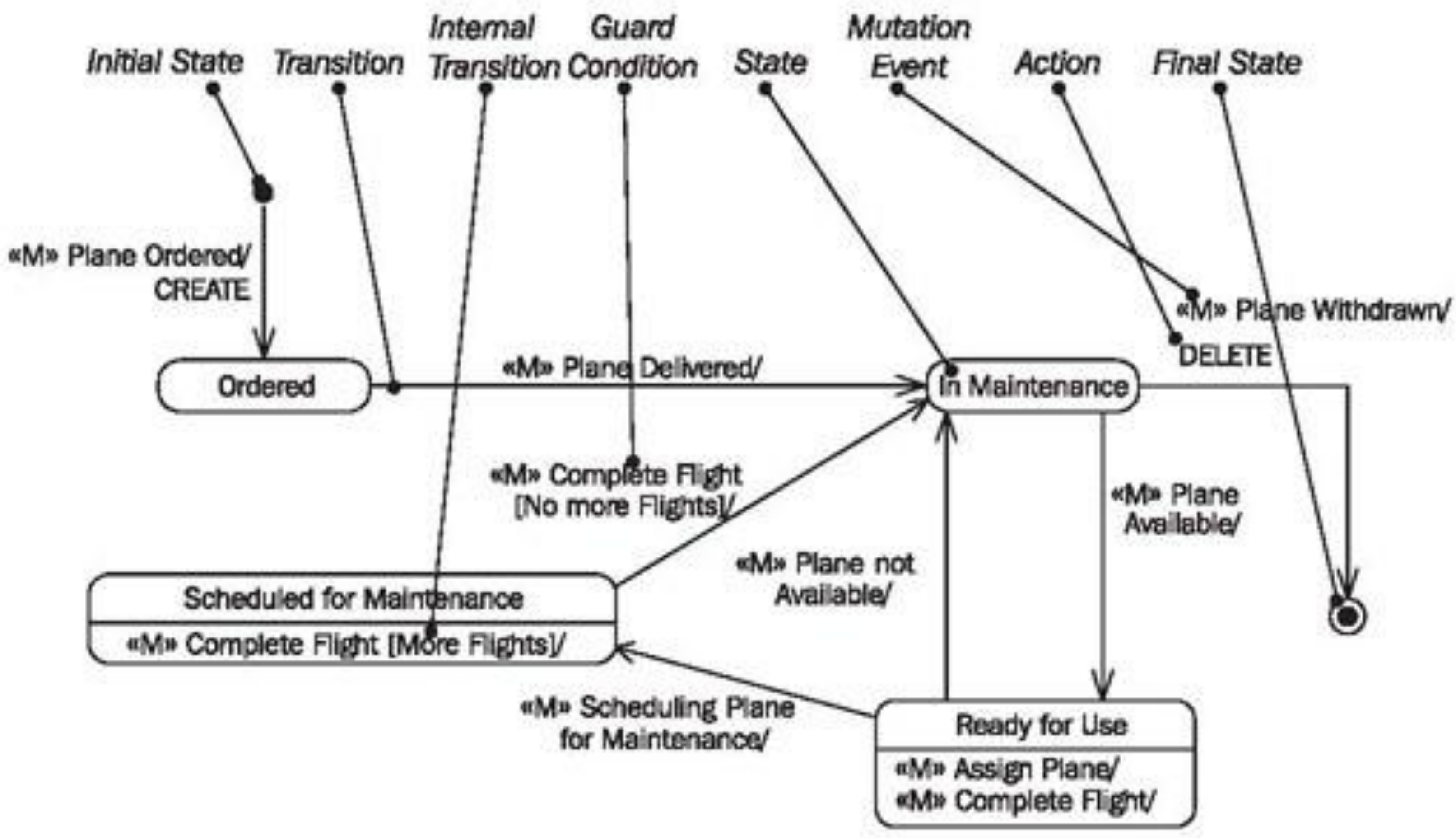
State-chart diagrams

- **Internal Transition:** transition from one state to itself. Object handles event without changing its state.



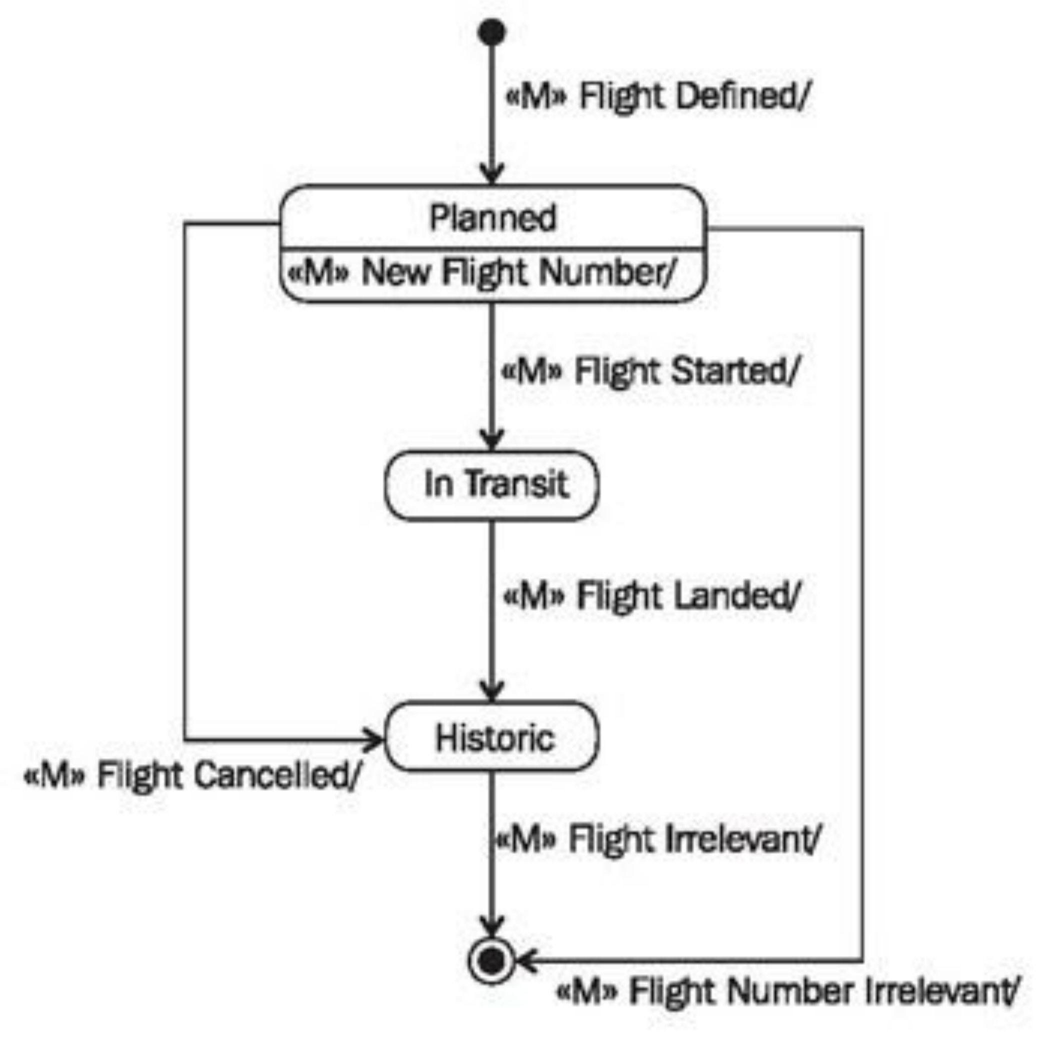
- **Mutation Event / Trigger:** The initiator of a transition from one state to another, or for an internal transition, where the state remains the same.
- 

State-chart diagrams



Elements of the statechart diagram

State-chart diagrams



Statechart diagram of the class "Flight"

State-chart diagrams

- **Checklist : Statechart Diagrams**
- Identify mutation events relevant for the object—What affects the object?
- Group relevant events chronologically—How does a normal life look?
- Model states and transitions—Which states are there?
- Add actions to the statechart diagram—What do objects do?
- Verify the statechart diagram—Is everything correct?

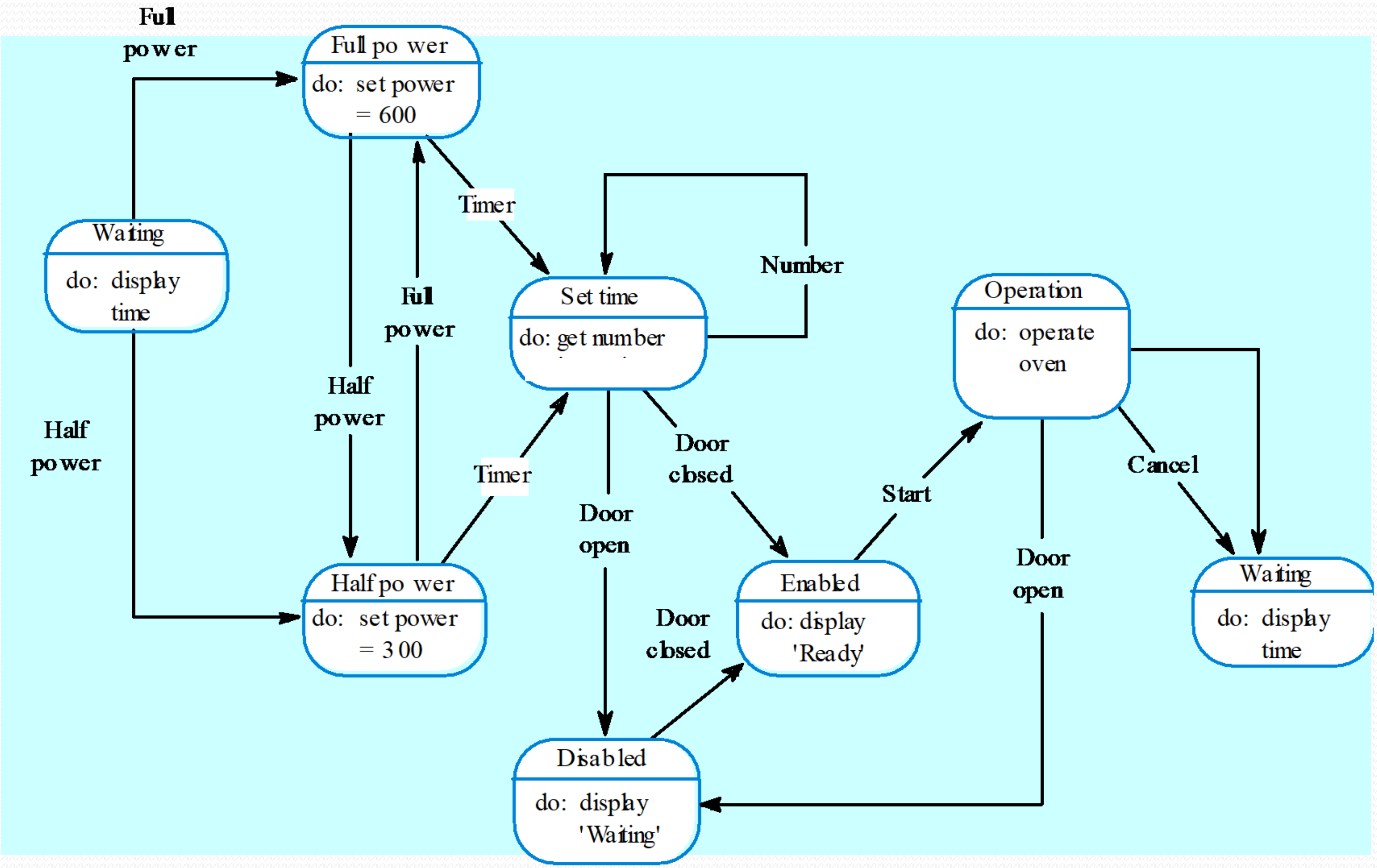
State machine models

- **These model the behaviour of the system in response to external and internal events.**
- They show the system's responses to stimuli so are often used for modelling **real-time systems**.
- State machine models **show system states as nodes** and **events as arcs** between these nodes. When an event occurs, the system moves from one state to another.
- **Statecharts** are an **integral part of the UML** and are used to represent state machine models.

Statecharts

- Allow the **decomposition of a model into sub-models** (see following slide).
- A brief description of the actions is included following the 'do' in each state.
- Can be complemented by tables describing the states and the stimuli.

Microwave oven model



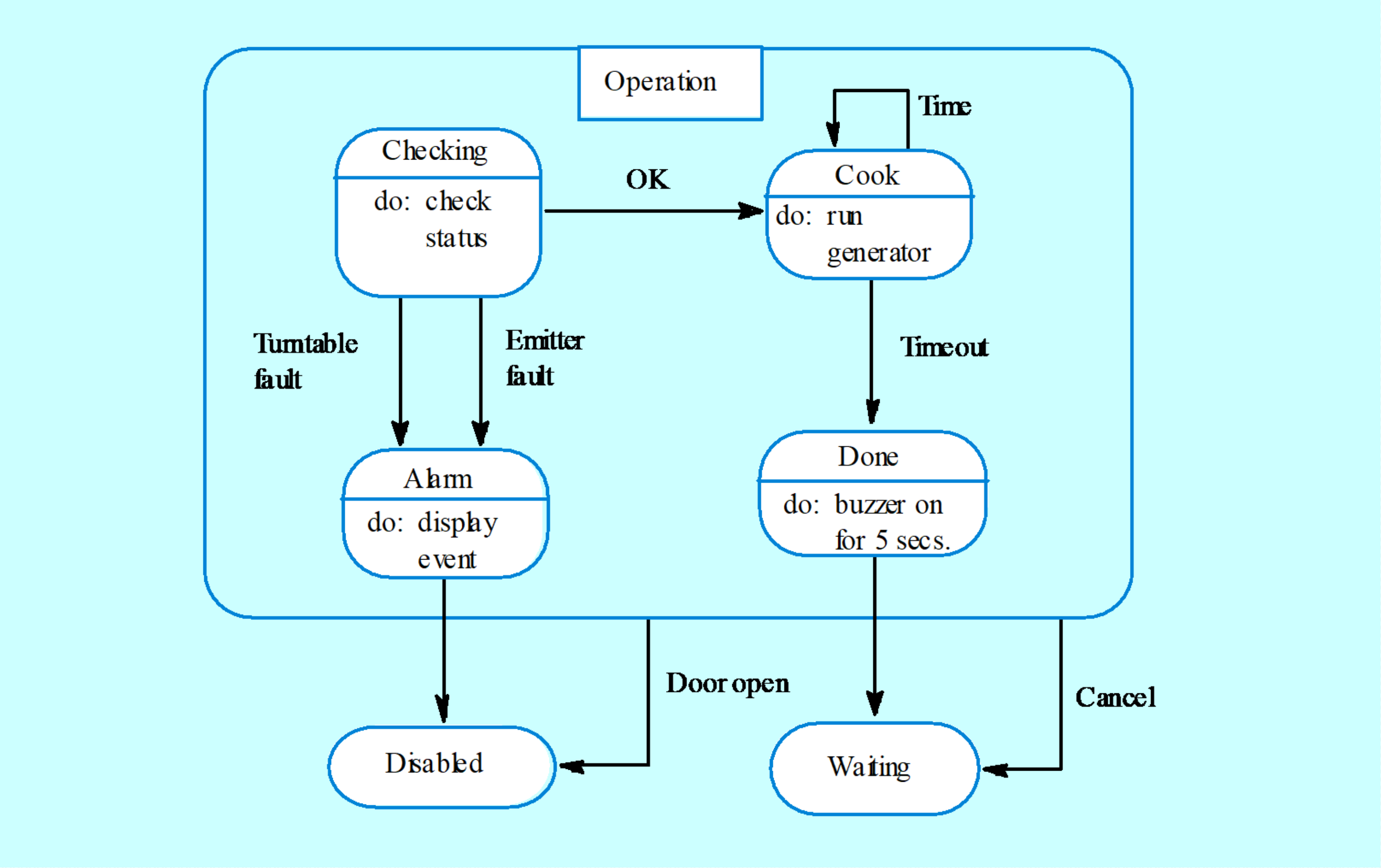
Microwave oven state description

State	Description
Waiting	The oven is waiting for input. The display shows the current time.
Half power	The oven power is set to 300 watts. The display shows ŌHalf powerÕ.
Full power	The oven power is set to 600 watts. The display shows ŌFull powerÕ.
Set time	The cooking time is set to the userÕs input value. The display shows the cooking time selected and is updated as the time is set.
Disabled	Oven operation is disabled for safety. Interior oven light is on. Display shows ŌNot readyÕ.
Enabled	Oven operation is enabled. Interior oven light is off. Display shows ŌReady to cookÕ.
Operation	Oven in operation. Interior oven light is on. Display shows the timer countdown. On completion of cooking, the buzzer is sounded for 5 seconds. Oven light is on. Display shows ŌCooking completeÕ while buzzer is sounding.

Microwave oven stimuli

Stimulus	Description
Half power	The user has pressed the half power button
Full power	The user has pressed the full power button
Timer	The user has pressed one of the timer buttons
Number	The user has pressed a numeric key
Door open	The oven door switch is not closed
Door closed	The oven door switch is closed
Start	The user has pressed the start button
Cancel	The user has pressed the cancel button

Microwave oven operation



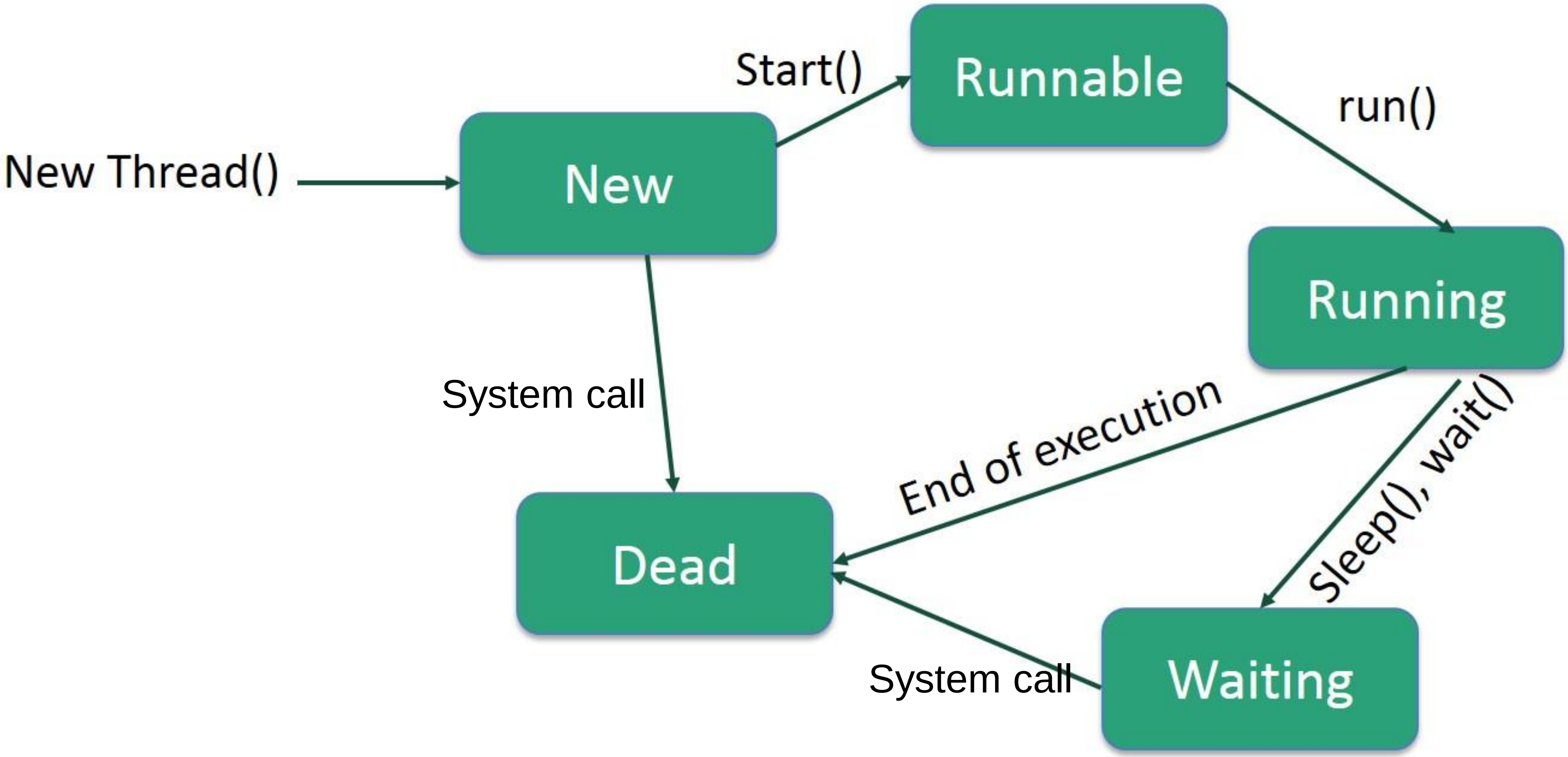
Life Cycle of a Thread

- A thread goes through various stages in its life cycle. E.g., a thread is born, started, runs, and then dies.
- **New** – A new thread begins its life cycle in the new state. It remains in this state until the program starts the thread. It is also referred to as a **born thread**.
- **Runnable** – After a newly born thread is started, the thread becomes runnable. A thread in this state is considered to be ready for executing its task.
- **Running** – executing task.
- **Waiting** – Sometimes, a thread transitions to the waiting state while the thread waits for another thread to perform a task. A thread transitions back to the runnable state only when another thread signals the waiting thread to continue executing.

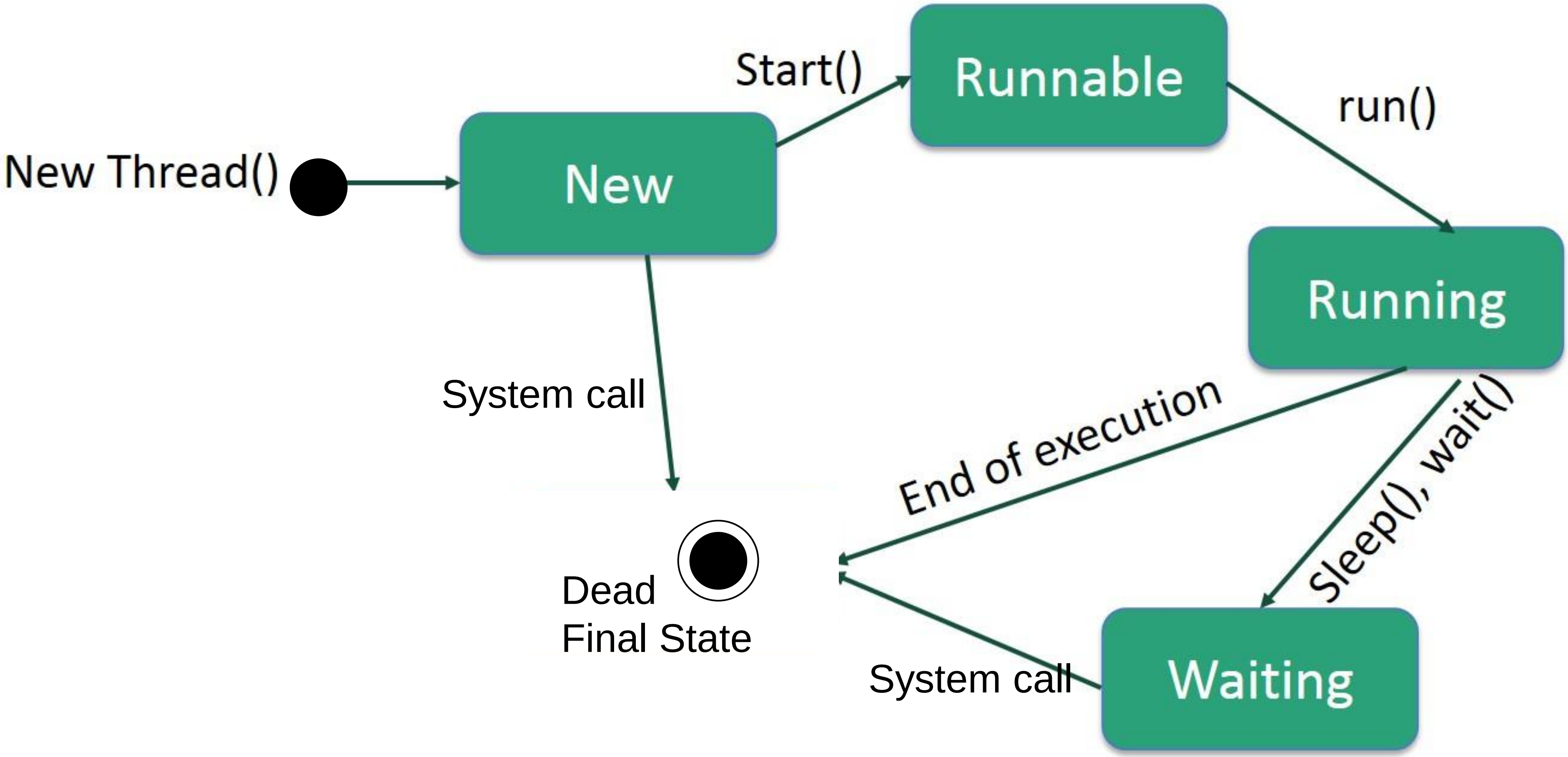
Life Cycle of a Thread

- **Timed Waiting** – A runnable thread can enter the timed waiting state for a specified interval of time. A thread in this state transitions <u>back to the runnable</u> state when that <u>time interval expires</u> or when the event it is waiting for occurs. E.g., sleep() call.
- **Terminated (Dead)** – A runnable thread enters the terminated state when it completes its task or otherwise terminates.

- The following diagram shows the complete life cycle of a thread.



- The following diagram shows the complete life cycle of a thread.



Class Exercise :

- Draw a state chart diagram for water tanker problem.
- Please draw this state transition diagram as homework for practice.

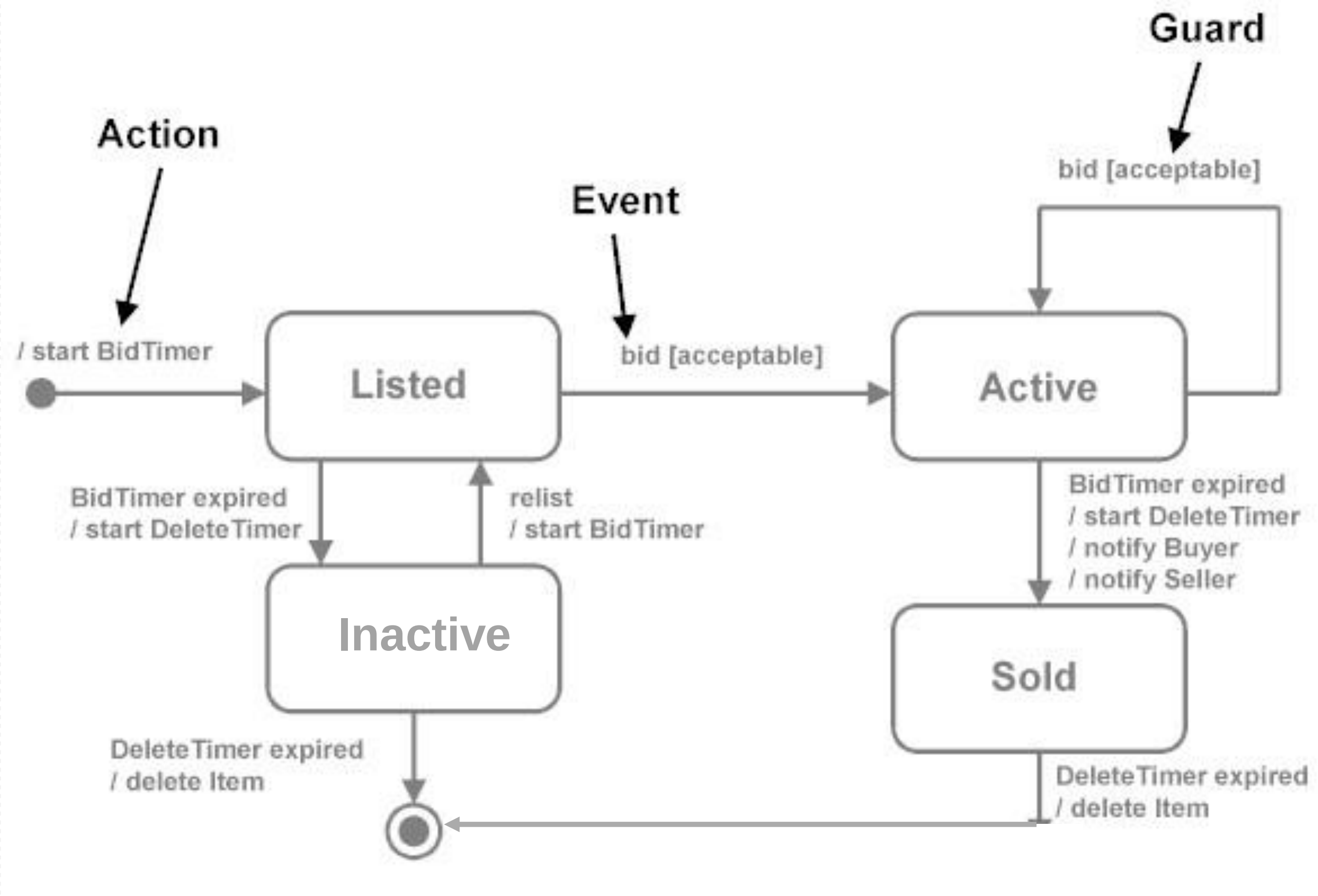
Class Exercise :

- States:
  1. Out of port
- 2. Docked
- 3. Docked & Connected
- 4. Loading
- 5. Docked & disconnected
- 6. Un-Docked
- 7. Travelling to destination
- 8. Unloading
- 9. Full / Half
- 10. Leaked
- 11. Overflow

Example – bid states

- Think ?

Example – bid states



Misconception - State chart

- Don't use control structures. Its not activity diagram. Keep it simple. UML 2.5
- No control flow in state charts like activity diagrams.
- Use atomic states do not use same states multiple times.
- Every transition / event leads to a new state? No!
- One initial and one final states.
  - But multiple initial and final states in composite state.
- Don't model activities in state transition diagrams. E,g., ATM states?
  - Insert card is ? activity
  - Out of Cash is? state
  - Authentication is ? activity
  - Logged In is ? state
  - Running is ? state

Assignment #4

- Consider the scenario in which you need to create a component model of a generalized inventory control application. There is a component called **inventoryControl** which contains following three components:
- (1) **transaction** component having two ports **stockPort** and **transPort** respectively. **inventoryControl** also have an external port called **stockPort** which is connected with **stockPort** of transaction compont.
- (2) **controlComp** component contains all the business logic of the main **inventoryControl** component, having 4 ports **transPort**, **recPort**, **DBPort**, and **catPort**. **transPort** is connected with the **transPort** of the **transaction** component, **DBport** is connected with the external **DBPort** of the external **inventoryControl, recPort** is connected with the requisition component's **recPort,** and **catPort** is connected with the external **catCtrl** port of **inventoryControl.**
- (3) **requisition** component has two ports called **reorderPort** and **recPort**. **reorderPort** is connected with the external **reorderPort** of **inventoryControl** component**.**

- Interfaces of **inventoryControl:**
- **inventoryControl** has three provided interfaces: **stockMovement**, **reorder**, **catalogueCtrl,** and one required interface called **DBAccess**. **stockMovement has** a private integer attribute called **prodID**, and two public operations called **commit()** and **rollback()**. **reorder** interface has a private integer attribute called **prodID**, and two public operations called **reorder()** and **getInvLevel()**. **DBAccess** interface has two private integer attributes called **TCPPort** and **socketID**, and three functions called **ODBC()**, **JDBC()** and **DAO()**. Finally, the **catalogueCtrl** interface has following four public attributes: **prodID** as integer, **name** as String, **qty** as integer and **price** and float.
- Datatypes various components ports:
- **stockPort**, and **reorderPort** are of type ByteArray; **recPort**, **DBPort**, **catPort**, and **catCtrl** are of type JavaObject;
- and **transPort** is of type **Date**.
- (A) Design a low level component diagram in papyrus, depicting the details of the main container component **inventoryControl** and representing the interfaces as rectangular classes showing all the details of the attributes and operations of the provided and required interfaces.
  (B) Design another low level component diagram in papyrus, depicting the details of the same container component **inventoryControl** and representing the interfaces as lollipop symbols hiding the attributes and operations of the provided and required interfaces leaving only the interface names and symbols.

(C) Crete a new component diagram in papyrus for a high-level design of inventory control system. representing the interfaces as lollipop symbols. Use the same **inventoryControl** container component (this time without showing the details of the **inventoryControl** component), with its three provided interfaces and one required interface as lollypop symbols. Connect **inventoryControl** interfaces with following components: **InventoryDatabase**, **catalogueEntry**, **stockLevel**, and **partsOrder**. **InventoryDatabase** realizes the **DBAccess** as its provided interface and **DBAccess** interface is used by the **inventoryControl** component. **catalogueEntry** component uses the provided interface **catalogueCtrl**, **stockLevel** component uses the provided interface **reorder**, and **partsOrder** component uses the **stockMovement** interface.

(D) Crete a new is component diagram in papyrus for a high-level design of inventory control system as in part (C), but this time show all the interfaces as rectangular class symbols showing all the details of their attribute and operations.

(E) After analyzing all the above four diagrams, provide a detailed summary (1 page) explaining how the system works and explain the functionality of each component in detail.