State Machine Implementation

Objectives

•Explain the ramifications of data structure during detailed design, including but not limited to numeric types, precision, and range.

•Compare and contrast absolute error and relative error.

•Explain the usage of constraints in detailed design.

•Explain visibility rules which one might use during detailed design.

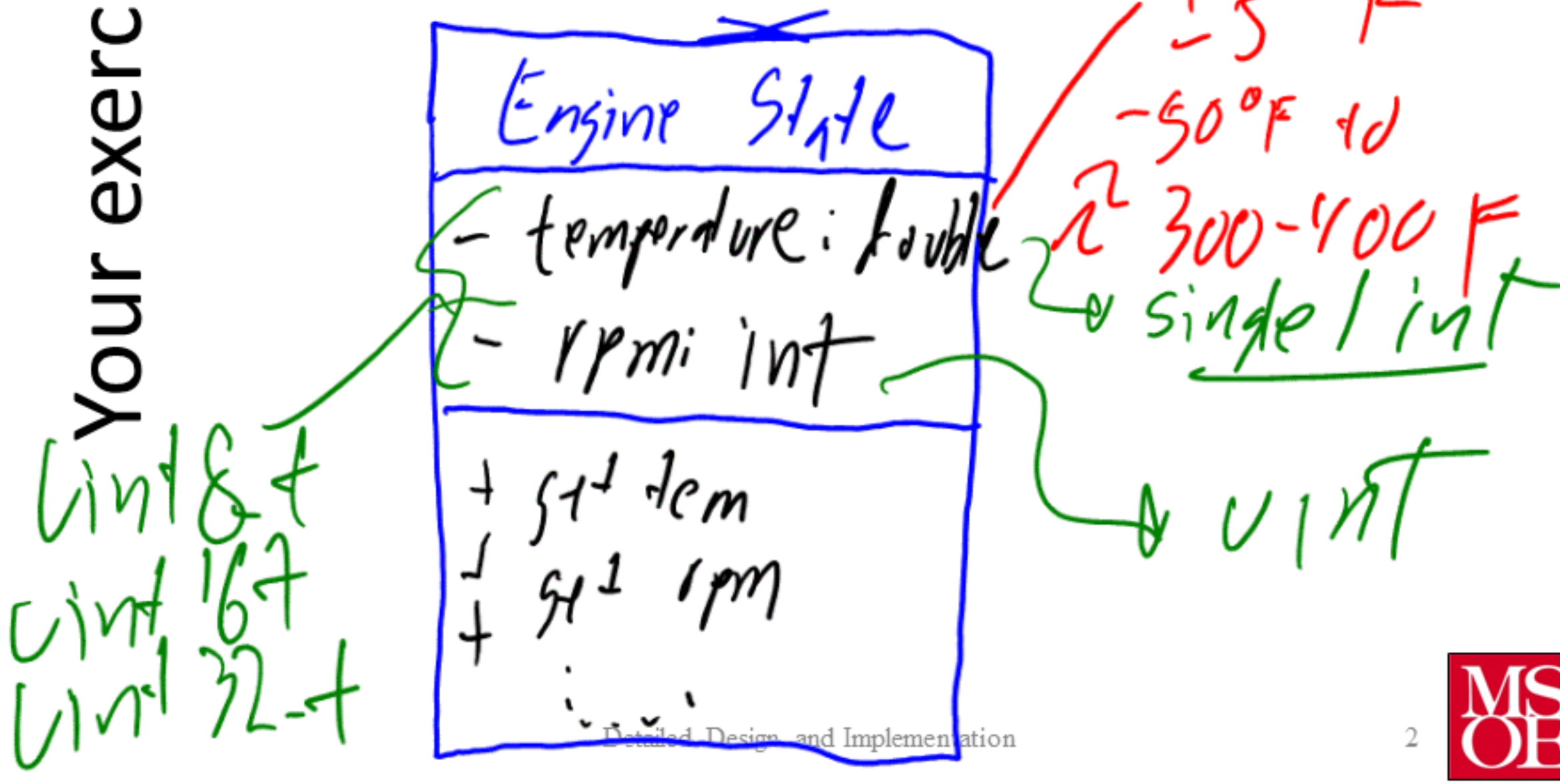
•Construct Java Interfaces matching the detailed design

•Define a generic state interface in Java including the capability to define entry action, exit action, and do action.

•Translate state charts into Java using interface patterns and state charts

Your exercise

- Take out a sheet of paper
  - Design a class to contains information about the state of an engine, including engine temperature and engine RPMs.
  - Draw it as a UML diagram.



What do we need to ask?

What do we need to ask?

- Questions to ask:
- What units?
- What is the precision?
- What is the range?
- Now we are at a point where it makes a difference

Error in calculations

```
    1
  123456.000000
+      4.567891
-------------
  123460.567891
```

6 digits of precision

Answer

Error

Small Relative error

1.2346×10^5

123456.000000

+00004.567891

1234560.567891 = .123460567891 x 10^6

Error

Six digit precision => 0.123456 x 10^6,
absolute error of 0.567891

Relative Error

$$E_R = \frac{(A-B)-(Machine(A)-Machine(B))}{A-B}$$

Actual Calculation

Machine's Calculation

Actual

Error

Small Relative Error

Relative Error

$$E_R = \frac{(A-B)-(Machine(A)-Machine(B))}{A-B}$$

$$\frac{1234560.567891 - 1234560}{1234560.567891}$$

$$4.59977 \times 10^{-8}$$

- Calculate to six digit precision
  - What is absolute error?
  - What is relative error?

```
  0.991012312
- 0.991009978
-------------
```

.000002334

Absolute error
Relative error

Very small

Error

- Calculate to six digit precision
  - What is absolute error?
  - What is relative error?

You try it

```
   0.991012312
-  0.991009978
---------------

0.2325 x 10^-5  (6 digit precision)

0.325 x 10^-6  (Absolute Error)

14% Relative Error
```

Review

- Turn to your neighbor and answer the following:
  - What methods / actions are associated with a software state?

Enter

Exit

do

  - What attributes are associated with a state transition?

Review

- Turn to your neighbor and answer the following:
  - What methods / actions are associated with a software state?
    - Entry
    - Exit
    - Do
  - What attributes are associated with a state transition?
    - Guard Conditions
    - Trigger (event)
    - Actions (Sometimes)
    - Starting State
    - Destination State

What is on a transition.

class genericstatemachinecode

«interface»
stateTransitionInterface

+ *getCurrentState() : StateInterface*
+ *getRequest() : int*
+ *getNextState() : StateInterface*
+ *conditionalLogic() : boolean*

«interface»
StateInterface

+ *entryAction() : void*
+ *exitAction() : void*
+ *getStateReport() : int*

-nextState

-currentState

«interface»
StateTableInterface

+ *findStateTransition(currentState :StateInterface, request :int) : StateInterface*
+ *addTransition(transition :stateTransitionInterface) : void*

stateTableTransition

- currentState: StateInterface
- request: int
- nextState: StateInterface

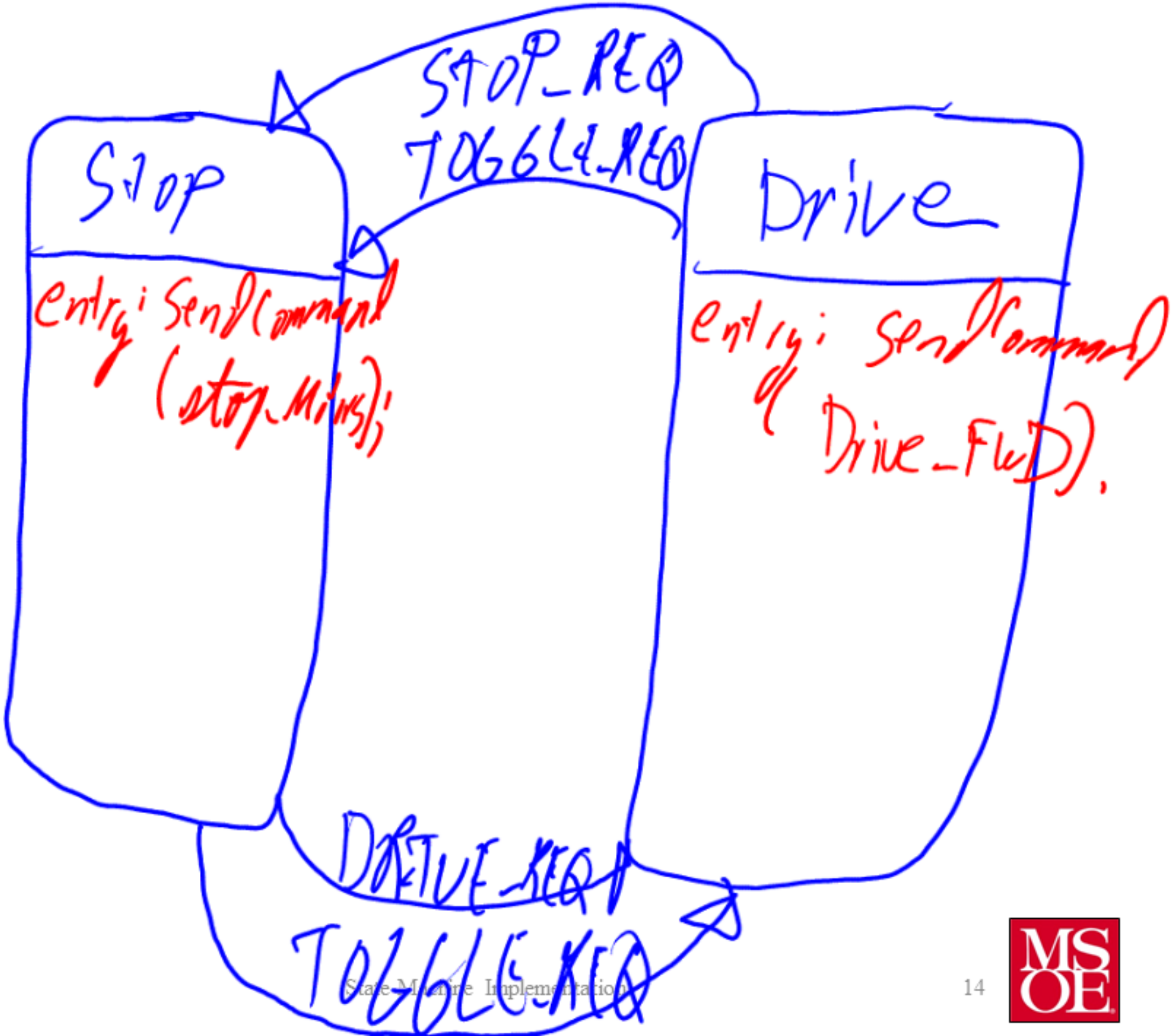
+ getCurrentState() : StateInterface
+ getRequest() : int
+ getNextState() : StateInterface
+ conditionalLogic() : boolean
+ stateTableTransition(currentState :StateInterface, request :int, nextState :StateInterface)

StateTable

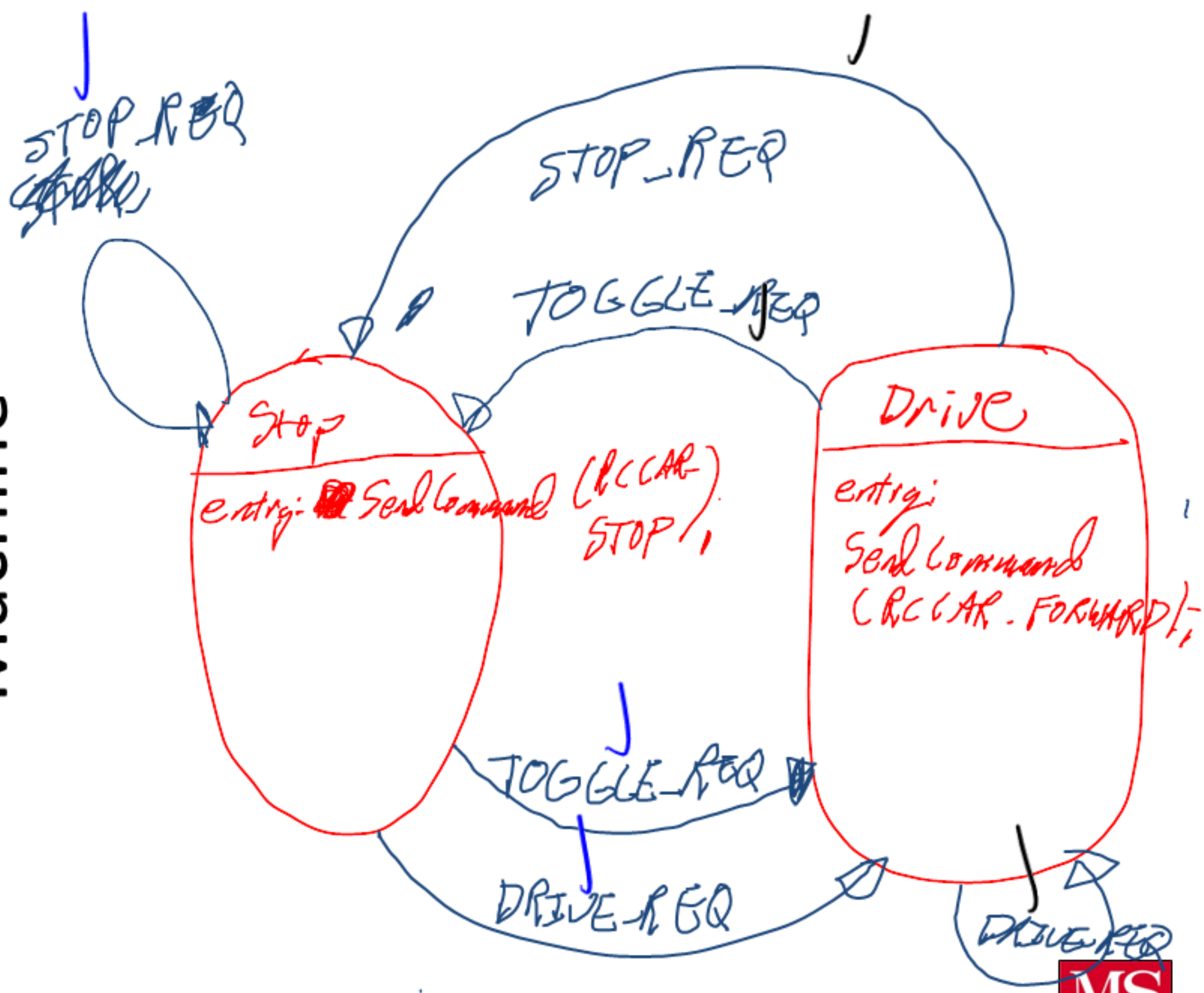
- transitions: List<stateTransitionInterface> = new ArrayList<s...

+ findStateTransition(currentState :StateInterface, request :int) : StateInterface
+ addTransition(transition :stateTransitionInterface) : void

Lets define our state machine



Lets define our state Machine



Parts of a state machine interface

Parts of a state machine implementation

- 3 parts
  - State machine command interface
    - Defines the command the state machine will respond to
  - Observer Interface
    - Allows External classes to monitor the state machine state
  - Implementation of the State Machine Interface
    - Definition of the states, entry behaviors, and transitions

class democarcontroller

«interface»
CarControllerStateMachineInterface

+ REQUEST_STOP: int = 1 {readOnly}
+ REQUEST_DRIVE: int = 2 {readOnly}
+ REQUEST_STATE_TOGGLE: int = 3 {readOnly}

+ requestCarControllerAction(request :int) : void
+ subscribe(observer :CarControllerStateMachineObserverInterface) : void
+ unsubscribe(observer :CarControllerStateMachineObserverInterface) : void

«interface»
CarControllerStateMachineObserverInterface

+ STOPPED: int = 0 {readOnly}
+ DRIVING: int = 1 {readOnly}

+ updateCarControllerState(newState :int) : void

CarController

- car: RCCarDriver
- obs: ArrayList<CarControllerStateMachineObserverInterface> = new ArrayList<C...
- stopState: StateInterface = new StateInterf...
- driveState: StateInterface = new StateInterf...
- stateTransition: StateTable = new StateTable()
- currentState: StateInterface = stopState

+ CarController(car :RCCarDriver)
+ requestCarControllerAction(request :int) : void
- updateObservers(newState :int) : void
+ subscribe(observer :CarControllerStateMachineObserverInterface) : void
+ unsubscribe(observer :CarControllerStateMachineObserverInterface) : void

Lets look at some code...

CarController.java | CarControllerStateMa | CarControllerStateMa | StopGoDisplay.java | StateChangeRequestBu | Main.java | stateTransitionInter | »4

```java
/**
 * This class implements as a state machine the logic necessary to control an RC car.
 */
public class CarController implements CarControllerStateMachineInterface {
    // This variable is the driver which the rc car is attached to.
    private RCCarDriver car;

    // This is a listing of the observers who are to be connected to the rc car controller.
    private ArrayList<CarControllerStateMachineObserverInterface> obs = new ArrayList<CarControllerStateMachineObserverInterface>();

    // This is the behavior required for the stop state.
    private StateInterface stopState = new StateInterface() {
        @Override
        public void entryAction() {
            car.sendControlCommand(RCCarDriver.STOP);
        }

        @Override
        public void exitAction() {
            // Nothing is done on exiting this state.
        }

        @Override
        public int getStateReport() {
            return CarControllerStateMachineObserverInterface.STOPPED;
        };
    };

    // This is the behavior required for the drive state.
    private StateInterface driveState = new StateInterface() {

        @Override
        public void entryAction() {
            car.sendControlCommand(RCCarDriver.FORWARD);
        }

        @Override
        public void exitAction() {
            // Nothing is done on exiting this state.
        }

        @Override
        public int getStateReport() {

            return CarControllerStateMachineObserverInterface.DRIVING;
        };
    };

    /**
```

```java
    /**
     * This table stores all of the state transitions.
     */
    private StateTable stateTransition = new StateTable();

    /**
     * This variable holds the current state of the RC Car state machine.
     */
    private StateInterface currentState = stopState;

    /**
     *
     * @param car
     */
    public CarController(RCCarDriver car) {
        super();
        this.car = car;
        // Stop state behavior.
        stateTransition.addTransition(new stateTableTransition(stopState, CarControllerStateMachineInterface.REQUEST_DRIVE, driveState));
        stateTransition.addTransition(new stateTableTransition(stopState, CarControllerStateMachineInterface.REQUEST_STOP, stopState));
        stateTransition.addTransition(new stateTableTransition(stopState, CarControllerStateMachineInterface.REQUEST_STATE_TOGGLE, driveState));

        // Drive State Behavior
        stateTransition.addTransition(new stateTableTransition(driveState,  CarControllerStateMachineInterface.REQUEST_DRIVE, driveState));
        stateTransition.addTransition(new stateTableTransition(driveState,  CarControllerStateMachineInterface.REQUEST_STOP, stopState));
        stateTransition.addTransition(new stateTableTransition(driveState,  CarControllerStateMachineInterface.REQUEST_STATE_TOGGLE, stopState));
    }

    /* (non-Javadoc)
     * @see edu.msoe.se2890.democarcontroller.CarControllerStateMachineInterface#requestCarControllerAction(int)
     */
    @Override
    public void requestCarControllerAction(int request) {
        // Find the appropriate transition, if one exists.
        StateInterface newState = this.stateTransition.findStateTransition(
                currentState, request);

        if (newState != null) {
            newState.entryAction();
            currentState = newState;
            updateObservers(currentState.getStateReport());
        }
    }

    /**
     * This method will update the observers with the new current state.
     *
     * @param newState
     *            This is the new state to announce.
```

```java
    /* (non-Javadoc)
     * @see edu.msoe.se2890.democarcontroller.CarControllerStateMachineInterface#requestCarControllerAction(int)
     */
    @Override
    public void requestCarControllerAction(int request) {
        // Find the appropriate transition, if one exists.
        StateInterface newState = this.stateTransition.findStateTransition(
                currentState, request);

        if (newState != null) {
            newState.entryAction();
            currentState = newState;
            updateObservers(currentState.getStateReport());
        }
    }

    /**
     * This method will update the observers with the new current state.
     *
     * @param newState
     *            This is the new state to announce.
     */
    private void updateObservers(int newState) {
        for (CarControllerStateMachineObserverInterface c : this.obs) {
            c.updateCarControllerState(newState);
        }
    }

    /* (non-Javadoc)
     * @see edu.msoe.se2890.democarcontroller.CarControllerStateMachineInterface#subscribe(edu.msoe.se2890.democarcontroller.CarControllerStateMachineOb
     */
    @Override
    public void subscribe(CarControllerStateMachineObserverInterface observer) {
        this.obs.add(observer);

    }

    /* (non-Javadoc)
     * @see edu.msoe.se2890.democarcontroller.CarControllerStateMachineInterface#unsubscribe(edu.msoe.se2890.democarcontroller.CarControllerStateMachine
     */
    @Override
    public void unsubscribe(CarControllerStateMachineObserverInterface observer) {
        this.obs.remove(observer);

    }

}
```