NC STATE UNIVERSITY

Finite State Machines
State Pattern

Reading: Finite-State Machines Packet

2

NC STATE UNIVERSITY

FSM REVIEW

3

FSM

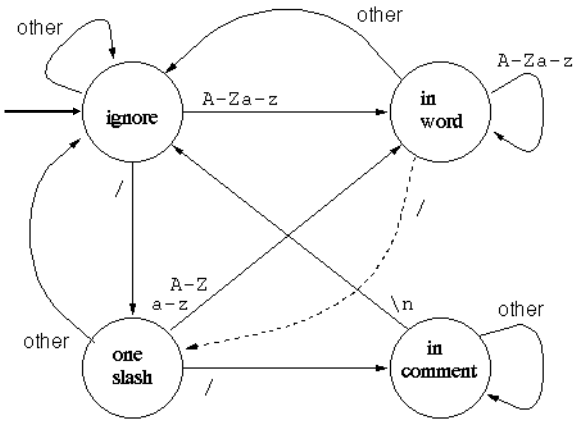
We can model a process lifecycle using a finite stat machine

- Abstract model of a system (Physical, biological, mechanical, electronic, or software)
- Consists of **inputs**, **states**, and **transitions**
- *Initial* and *final* states

Purpose of FSM?

- Trace through the states of the following inputs

```
if (ch == '/')

//Comment

x=0; //Comment
```



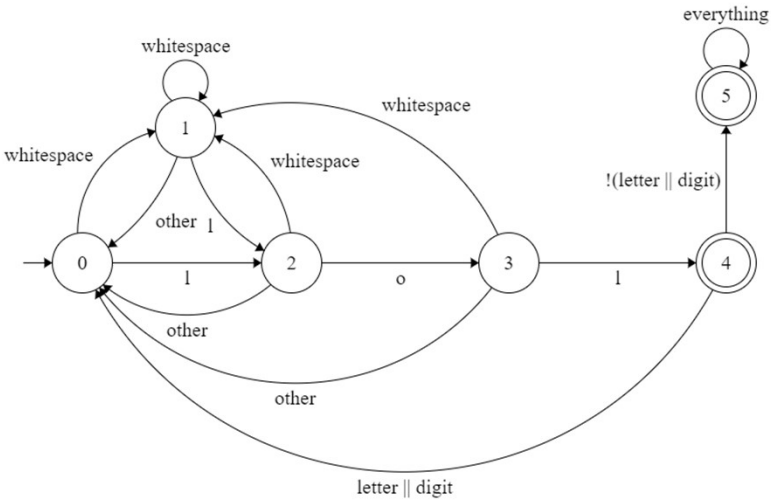
NC STATE UNIVERSITY

Practice FSM

- Create an FSM that would recognize the acronym "lol" (case insensitive) in a String of input (like a text message)
  - By word, I mean that "lol" would be surrounded by whitespace, end with punctuation (non-letter and non-number), or be the only text in a String
    - Lol - YES
    - Lollipop - NO
    - LOL. - YES
    - Alol - NO
- Implement your FSM in a method containsLol() that accepts a String of text and returns true if the String contains "lol" (based on definition above).
- We'll copy method implementations into a test program and see if they pass

NC STATE UNIVERSITY

Abbreviation FSM



EXAMPLE: HORNER'S RULE

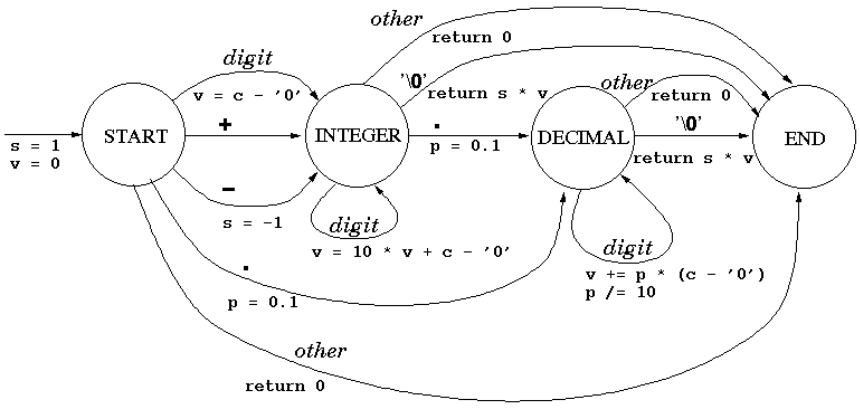
Horner's Rule

- Horner's Rule is a method for evaluating polynomials
  - $5x^4 – 3x^3 + x^2 – 4x – 7 = ((((0 + 5)x – 3)x + 1)x – 4)x -7$
  - What's the result if x = 3?
- Integers are polynomials evaluated at x = 10
  - $1092 = 1x^3 + 0x^2 + 9x + 2$, when x = 10
  - Using Horner's Rule, we can convert a string of digits to a number
    - "1092" = (((0 + 1)x + 0)x + 9)x + 2 = 1092, when x = 10
- Convert a character digit to its value
  - 'd' – '0' = d
- Digits to the right of the decimal point are multiplied by appropriate negative power of 10
- Use an FSM to convert a String of ACSII characters to a real number
  - `java.lang.Float.parseFloat(String s)`

Horner's Rule FSM

- If we want to convert the string "3.14" to the double 3.14, what are our inputs, states, and transitions for using the Horner's Rule algorithm as a finite state machine?

Horner's Rule FSM

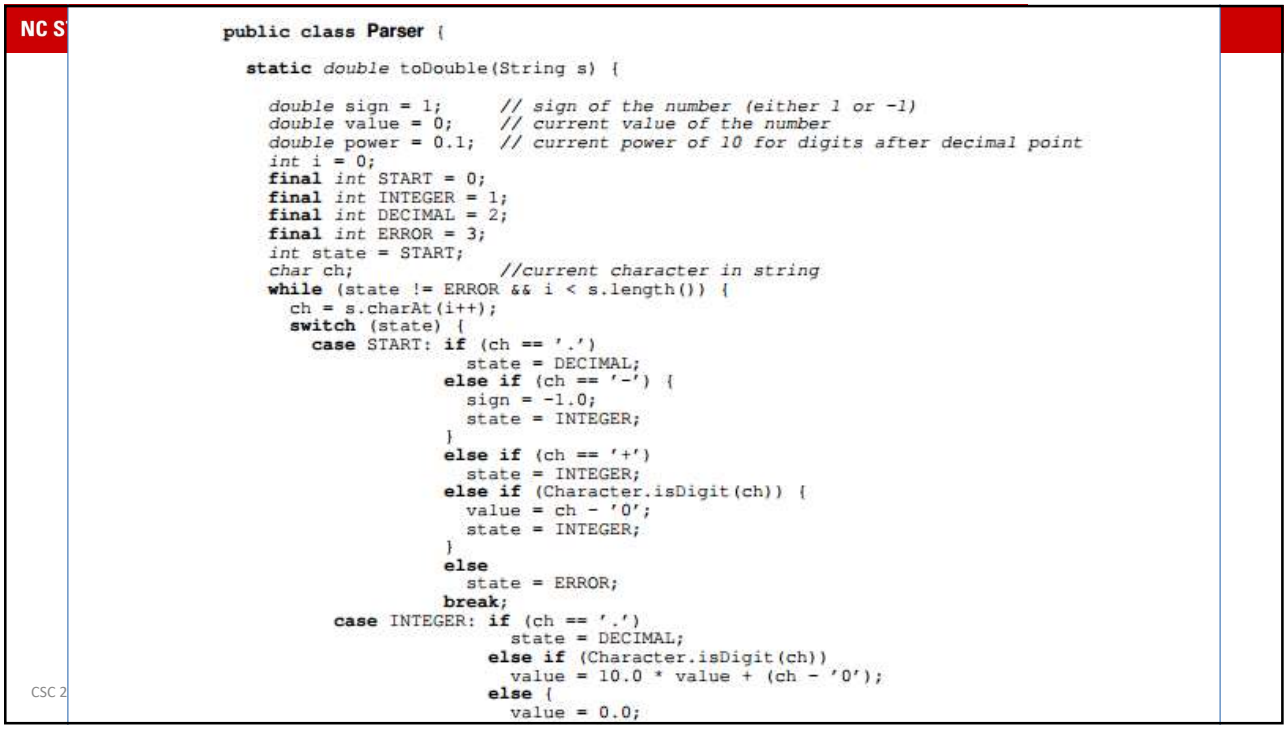


Input symbols are written in **bold** above the transitions.

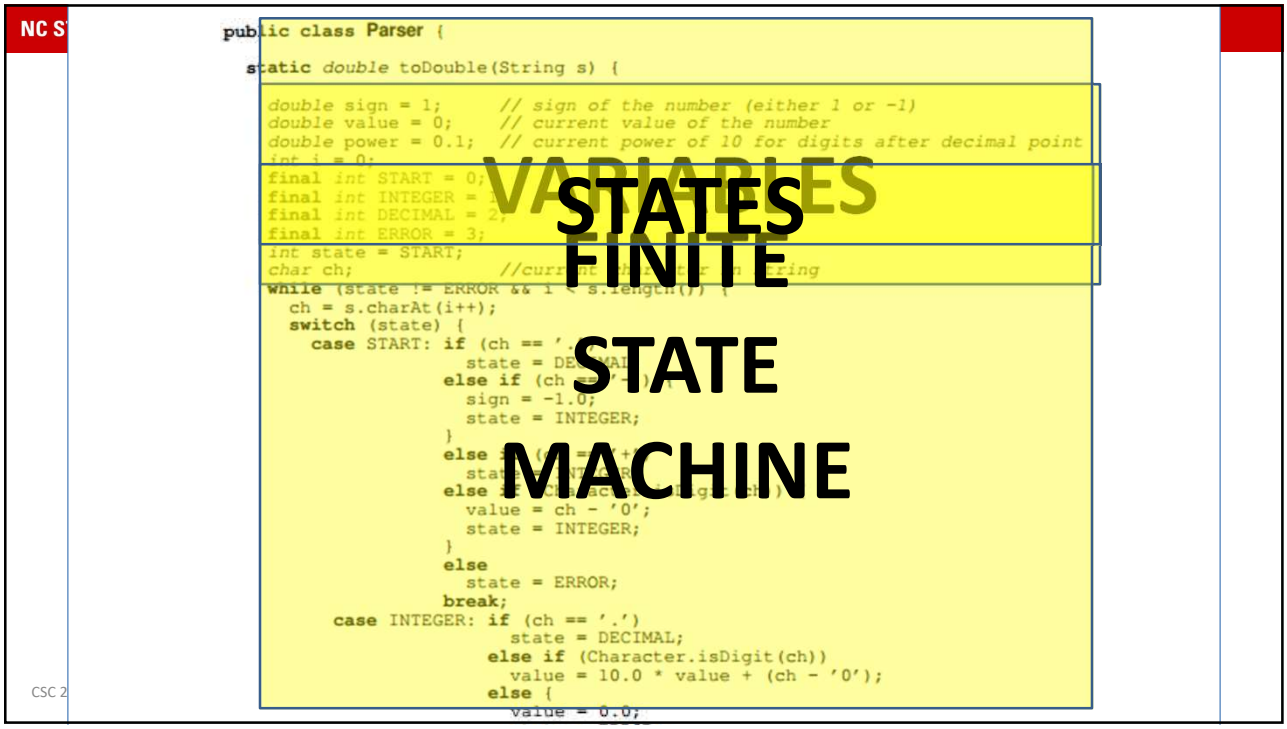
Symbol classes (such as *digit*) are written in *italic*.

Actions performed by the program are in `courier` below the transitions.

The variable `c` denotes the input character.


NC S
public class Parser {
static double toDouble(String s) {
double sign = 1; // sign of the number (either 1 or -1)
double value = 0; // current value of the number
double power = 0.1; // current power of 10 for digits after decimal point
int i = 0;
final int START = 0;
final int INTEGER = 1;
final int DECIMAL = 2;
final int ERROR = 3;
int state = START;
char ch; //current character in string
while (state != ERROR && i < s.length()) {
ch = s.charAt(i++);
switch (state) {
case START: if (ch == '.')
state = DECIMAL;
else if (ch == '-') {
sign = -1.0;
state = INTEGER;
}
else if (ch == '+')
state = INTEGER;
else if (Character.isDigit(ch)) {
value = ch - '0';
state = INTEGER;
}
else
state = ERROR;
break;
case INTEGER: if (ch == '.')
state = DECIMAL;
else if (Character.isDigit(ch))
value = 10.0 * value + (ch - '0');
else {
value = 0.0;
CSC 2

12


NC S
public class Parser {
static double toDouble(String s) {
double sign = 1; // sign of the number (either 1 or -1)
double value = 0; // current value of the number
double power = 0.1; // current power of 10 for digits after decimal point
final int START = 0;
final int INTEGER = 1;
final int DECIMAL = 2;
final int ERROR = 3;
int state = START;
char ch; //current character in string
while (state != ERROR && i < s.length()) {
ch = s.charAt(i++);
switch (state) {
case START: if (ch == '.')
state = DECIMAL;
else if (ch == '-') {
sign = -1.0;
state = INTEGER;
}
else if (ch == '+')
state = INTEGER;
else if (Character.isDigit(ch)) {
value = ch - '0';
state = INTEGER;
}
else
state = ERROR;
break;
case INTEGER: if (ch == '.')
state = DECIMAL;
else if (Character.isDigit(ch))
value = 10.0 * value + (ch - '0');
else {
value = 0.0;
VARIABLES
STATES
FINITE
STATE
MACHINE
CSC 2

13

NC STATE UNIV

```
while (state != ERROR && i < s.length()) {
  ch = s.charAt(i++);
  switch (state) {
    case START: if (ch == '.')
                  state = DECIMAL;
                else if (ch == '-') {
                  sign = -1.0;
                  state = INTEGER;
                }
                else if (ch == '+')
                  state = INTEGER;
                else if (Character.isDigit(ch)) {
                  value = ch - '0';
                  state = INTEGER;
                }
                else
                  state = ERROR;
                break;
      case INTEGER: if (ch == '.')
                      state = DECIMAL;
                    else if (Character.isDigit(ch))
                      value = 10.0 * value + (ch - '0');
                    else {
                      value = 0.0;
                      state = ERROR;
                    }
                    break;
```


14

NC STATE UNIV

```
while (state != ERROR && i < s.length()) {
  ch = s.charAt(i++);
  switch (state) {
    case START: if (ch == '.')
                  state = DECIMAL;
                else if (ch == '-') {
                  sign = -1.0;
                  state = INTEGER;
                }
                else if (ch == '+')
                  state = INTEGER;
                else if (Character.isDigit(ch)) {
                  value = ch - '0';
                  state = INTEGER;
                }
                else
                  state = ERROR;
                break;
      case INTEGER: if (ch == '.')
                      state = DECIMAL;
                    else if (Character.isDigit(ch))
                      value = 10.0 * value + (ch - '0');
                    else {
                      value = 0.0;
                      state = ERROR;
                    }
                    break;
```

Switch over STATE

Each case is a STATE value

Each transition as an if()

Each transition as an if()

Each transition as an if()

Each transition as an if()

Each transition as an if()

BREAK!!!

Each case is a STATE value

15

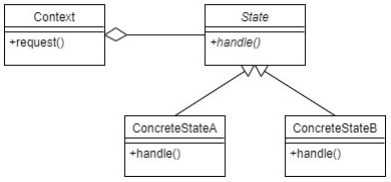
STATE PATTERN

Designing Object-Oriented FSMs

- How can we implement an FSM using objects instead of a switch statement?
  - What code is similar in the while-switch version of the FSM?
  - What objects could we use?
  - What behaviors could the objects have?

NC STATE UNIVERSITY

State Pattern



- Previous examples of FSMs are not object-oriented
  - Focus is on a method that returns a result based on traversing an FSM (while-switch idiom) for a given input
- In some cases the state of something may influence the actions of the entire program
  - We can use the State Pattern as an OO solution to a state-based application
  - An object's behavior (the context, or what the client interacts with) depends on state, and the object must change its behavior at run time based on state
  - Alternative to conditionals with redundant code – treat state as an object that can vary independently from other objects
    - Inputs are passed to the current state and are handled appropriately for the given state.

18

NC STATE UNIVERSITY

IMPLEMENTING THE STATE PATTERN

19

NC STATE UNIVERSITY

Implementing the State Pattern

1. **Define a `State` Interface/Abstract Class**
   - Method for each transition (action) that encapsulates the behavior associated with the `Context`'s state
   - Could be a single method that handles an input.
2. **Implement concrete State classes for all possible states in the FSM**
   - Provide behavior of the FSM when in the given state
3. **Implement the Context**
   - Defines interface that clients will use
   - Maintains state instances that are used to delegate behavior to the appropriate state class

Let's implement Horner's Rule using the State Pattern

NC STATE UNIVERSITY

State Interface

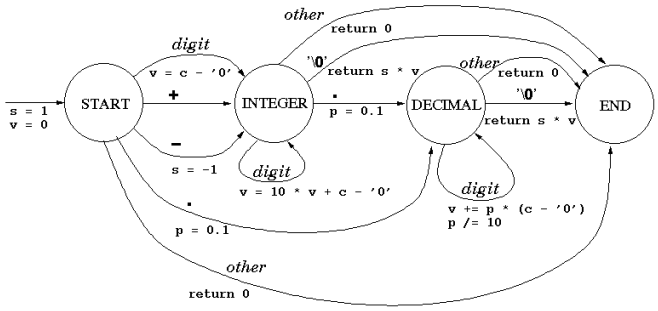
- Transition Table
  - Rows represent current state
  - Columns represent behaviors/inputs that may change state
- What could our `State` interface look like?

State	+ or -	.	Digit	Other
START	INTEGER	DECIMAL	INTEGER	ERROR
INTEGER	ERROR	DECIMAL	INTEGER	ERROR
DECIMAL	ERROR	ERROR	DECIMAL	ERROR

State Interface (2)

```
public interface State {
    void onPlus();
    void onMinus();
    void onPoint();
    void onDigit();
    void onOther();
}
```

State	+ or -	.	Digit	Other
START	INTEGER	DECIMAL	INTEGER	ERROR
INTEGER	ERROR	DECIMAL	INTEGER	ERROR
DECIMAL	ERROR	ERROR	DECIMAL	ERROR



Input symbols are written in **bold** above the transitions,
Symbol classes (such as *digit*) are written in *italic*.
Actions performed by the program are in courier below the transitions.
The variable c denotes the input character.

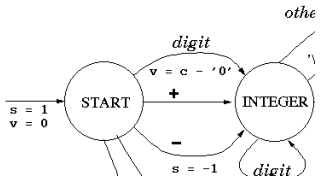
Implementing the State Pattern

1. **Define a `State` Interface/Abstract Class**
   - Method for each transition (action) that encapsulates the behavior associated with the `Context`'s state
   - Could be a single method that handles an input.
2. **Implement concrete State classes for all possible states in the FSM**
   - Provide behavior of the FSM when in the given state
3. **Implement the Context**
   - Defines interface that clients will use
   - Maintains state instances that are used to delegate behavior to the appropriate state class

Let's implement Horner's Rule using the State Pattern

`Start` State Implementation

```
/**
  * Implementation of the Start state
  * for the Horner's Rule FSM as inner
  * class of HornersRule
  * @author SarahHeckman
  */
private class Start implements State {

  @Override
  public void onDigit() {
    value = ch - '0';
    state = integer;
  }

  @Override
  public void onMinus() {
    sign = -1.0;
    state = integer;
  }

  @Override
  public void onOther() {
    throw new NumberFormatException();
    //Throw exception rather than
    //have an error state
  }

  @Override
  public void onPlus() {
    state = integer;
  }

  @Override
  public void onPoint() {
    state = decimal;
  }

}
```



Implementing the State Pattern

1. **Define a `State` Interface/Abstract Class**
   - Method for each transition (action) that encapsulates the behavior associated with the `Context`'s state
   - Could be a single method that handles an input.
2. **Implement concrete State classes for all possible states in the FSM**
   - Provide behavior of the FSM when in the given state
3. **Implement the Context**
   - Defines interface that clients will use
   - Maintains state instances that are used to delegate behavior to the appropriate state class

Let's implement Horner's Rule using the State Pattern

NC STATE UNIVERSITY

HornersRule Implementation

```
/**
 * State Pattern implementation of
 * Horner's Rule
 */
public class HornersRule {
  private final State start = new
   Start();
  private final State integer = new
   Integer();
  private final State decimal = new
   Decimal();

  private State state = start;

  private double sign = 1;
  private double value = 0;
  private double power = 0.1;
  private double multi = 10.0;

  private char ch;
```

```
/** Returns a double for the given string
 * @throws NumberFormatException if
 * invalid character.
 */
public double toDouble(String s) {
  //reset double fields - left off slide
  int i = 0;
  while (i < s.length()) {
    ch = s.charAt(i++);
    if (ch == '.')
      state.onPoint();
    else if (ch == '+')
      state.onPlus();
    else if (ch == '-')
      state.onMinus();
    else if (Character.isDigit(ch))
      state.onDigit();
    else
      state.onOther();
  }
  return sign * value;
}
```


26

NC STATE UNIVERSITY

HornersRule Implementation

Define the States

```
/**
 * State Pattern implementation of
 * Horner's Rule
 */
public class HornersRule {
  private final State start = new
   Start();
  private final State integer = new
   Integer();
  private final State decimal = new
   Decimal();

  private State state = start;

  private double sign = 1;
  private double value = 0;
  private double power = 0.1;
  private double multi = 10.0;

  private char ch;
```

```
/** Returns a double for the given string
 * @throws NumberFormatException if
 * invalid character.
 */
le toDouble(String s) {
  //reset double fields - left off slide
  int i = 0;
  while (i < s.length()) {
    ch = s.charAt(i++);
    if (ch == '.')
      state.onPoint();
    else if (ch == '+')
      state.onPlus();
    else if (ch == '-')
      state.onMinus();
    else if (Character.isDigit(ch))
      state.onDigit();
    else
      state.onOther();
  }
  return sign * value;
}
```


27

HornersRule Implementation

```
/**
 * State Pattern implementation of
 * Horner's Rule
 */
public class HornersRule {
  private final State start = new
    Start();
  private final State integer = new
    Integer();
  private final State decimal = new
    Decimal();

  private State state = start;

  private double sign = 1;
  private double value = 0;
  private double power = 0.1;
  private double multi = 10.0;

  private char ch;
```

Set the Initial State

```
/** Returns a double for the given string
 * @throws NumberFormatException if
 * invalid character.
 */
public double toDouble(String s) {
  //reset double fields - left off slide
  int i = 0;
  while (i < s.length()) {
    ch = s.charAt(i++);
    if (ch == '.')
      state.onPoint();
    else if (ch == '+')
      state.onPlus();
    else if (ch == '-')
      state.onMinus();
    else if (Character.isDigit(ch))
      state.onDigit();
    else
      state.onOther();
  }
  return sign * value;
}
```


28

HornersRule Implementation

```
/**
 * State Pattern implementation of
 * Horner's Rule
 */
public class HornersRule {
  private final State start = new
    Start();
  private final State integer = new
    Integer();
  private final State decimal = new
    Decimal();

  private State

  private double sign = 1;
  private double value = 0;
  private double power = 0.1;
  private double multi = 10.0;

  private char ch;
```

Process input character by character

```
/** Returns a double for the given string
 * @throws NumberFormatException if
 * invalid character.
 */
public double toDouble(String s) {
  //reset double fields - left off slide
  int i = 0;
  while (i < s.length()) {
    ch = s.charAt(i++);
    if (ch == '.')
      state.onPoint();
    else if (ch == '+')
      state.onPlus();
    else if (ch == '-')
      state.onMinus();
    else if (Character.isDigit(ch))
      state.onDigit();
    else
      state.onOther();
  }
  return sign * value;
}
```


29

HornersRule Implementation

```
/**
 * State Pattern implementation of
 * Horner's Rule
 */
public class HornersRule {
  private final State start = new
    Start();
  private final State integer = new
    Integer();
  private final State decimal = new
    Decimal();

  private St

  private do
  private do
  private do
  private double multi = 10.0;

  private char ch;
```

An input signals a possible transition. Delegate to the current state's transition method! (Polymorphism in action!)

```
/** Returns a double for the given string
 * @throws NumberFormatException if
 * invalid character.
 */
public double toDouble(String s) {
  //reset double fields - left off slide
  int i = 0;
  while (i < s.length()) {
    ch = s.charAt(i++);
    if (ch == '.')
      state.onPoint();
    else if (ch == '+')
      state.onPlus();
    else if (ch == '-')
      state.onMinus();
    else if (Character.isDigit(ch))
      state.onDigit();
    else
      state.onOther();
  }
  return sign * value;
}
```

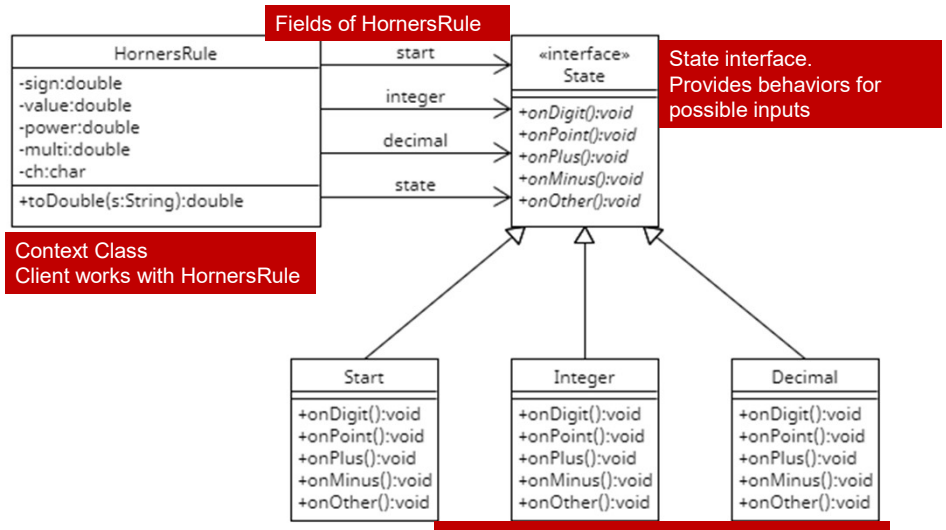

HornersRule Implementation

```
/**
 * State Pattern implementation of
 * Horner's Rule
 */
public class HornersRule {
  private final State start = new
    Start();
  private final State integer = new
    Integer();
  private final State decimal = new
    Decimal();

  private State state = start;

  private double sign = 1;
  private double value = 0;
  private double power = 0.1;
  private double multi = 10.0;

  private ch
```

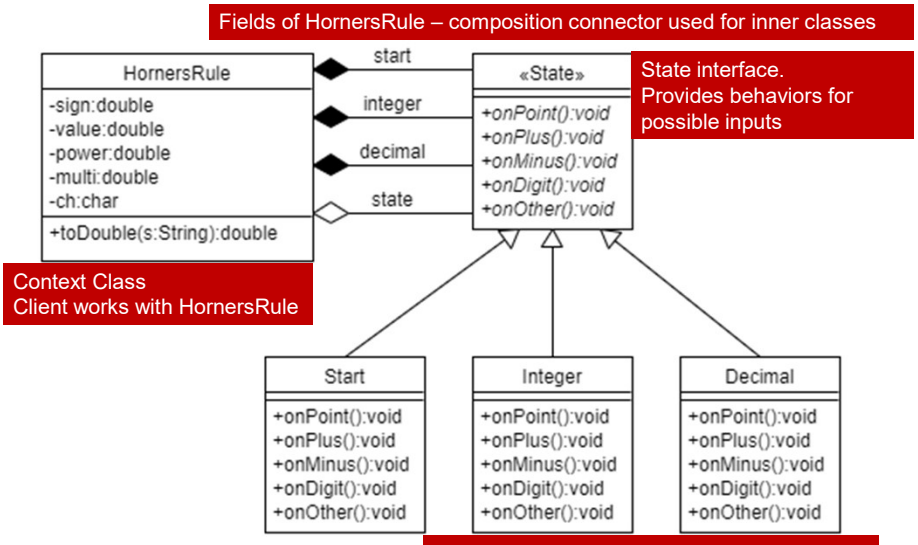
Return when done. Exceptions are thrown from the States.

```
/** Returns a double for the given string
 * @throws NumberFormatException if
 * invalid character.
 */
public double toDouble(String s) {
  //reset double fields - left off slide
  int i = 0;
  while (i < s.length()) {
    ch = s.charAt(i++);
    if (ch == '.')
      state.onPoint();
    else if (ch == '+')
      state.onPlus();
    else if (ch == '-')
      state.onMinus();
    else if (Character.isDigit(ch))
      state.onDigit();
    else
      state.onOther();
  }
  return sign * value;
```

Horner's Rule UML Diagram



Horner's Rule UML Diagram



NC STATE UNIVERSITY

`HornersRule` Implementation

```
/**
 * State Pattern implementation of
 * Horner's Rule
 */
public class HornersRule {
  private final State start = new
    Start();
  private final State integer = new
    Integer();
  private fin
    Decimal();

  private Sta

  private dou
  private dou
  private dou
  private dou

  private char ch;
```

Private inner classes for the concrete states!

They have access to the fields of HornersRule!

```
/** Returns a double for the given string
 * @throws NumberFormatException if
 * invalid character.
 */
public double toDouble(String s) {
   …
}

private class Start implements State {
   …
}

private class Integer implements State {
   …
}

private class Decimal implements State {
   …
}
```


NC STATE UNIVERSITY

`Integer` and `Decimal` States?

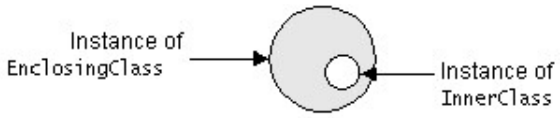
```
private class Integer implements State {

   public void onDigit() { }

   public void onMinus() { }

   public void onOther() { }

   public void onPlus() { }

   public void onPoint() { }

}
```

```
private class Decimal implements State {

   public void onDigit() { }

   public void onMinus() { }

   public void onOther() { }

   public void onPlus() { }

   public void onPoint() { }

}
```

INNER CLASSES

Inner classes

- **inner class**: A class defined inside of another class.
  - we will focus on standard "nested" inner classes
- usefulness:
  - inner classes (and their public fields) are hidden from other classes (encapsulated)
  - inner objects can access/modify the fields of the outer object (if the inner class is not static)



Inner class syntax

```
// outer (enclosing) class
public class OuterName {
    ...

    // inner (nested) class
    private class InnerName {
        ...
    }
}
```

- Only this outer class/object can see the inner class or make objects of it.
- Each inner object is associated with the outer object that created it, so it can access/modify that outer object's methods/fields.
  - If necessary, can refer to outer object as **OuterName**`.this`

Adding Inner Classes

- Inner classes can be declared in a method or within an entire enclosing class
- You are telling code inspectors that the class is *only* of interest to the enclosing body.

```
public class HornersRule {
   ...
   private class Start implements State {...}
   private class Integer implements State {...}
   ...
}
```

What Happens…

- When we compile code containing inner classes?
  - Class files are made for each inner class, but the naming convention is different
  - `HornersRule.class`
  - `HornersRule$Decimal.class`
  - `HornersRule$Integer.class`
  - `HornersRule$Start.class`

TESTING FSM

NC STATE UNIVERSITY

Horner's Rule Testing

- Testing State Machines
  - Test each transition from each state
  - See `HornersRuleTest` and Transition Table for example

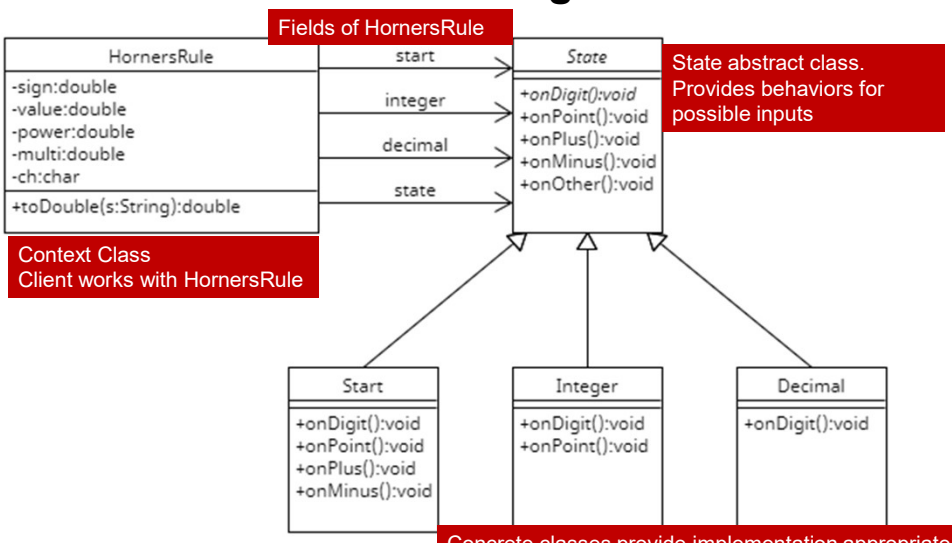
State	+ or -	.	Digit	Other
START	INTEGER	DECIMAL	INTEGER	ERROR
INTEGER	ERROR	DECIMAL	INTEGER	ERROR
DECIMAL	ERROR	ERROR	DECIMAL	ERROR

NC STATE UNIVERSITY

ALTERNATIVE HORNER'S RULE DESIGNS

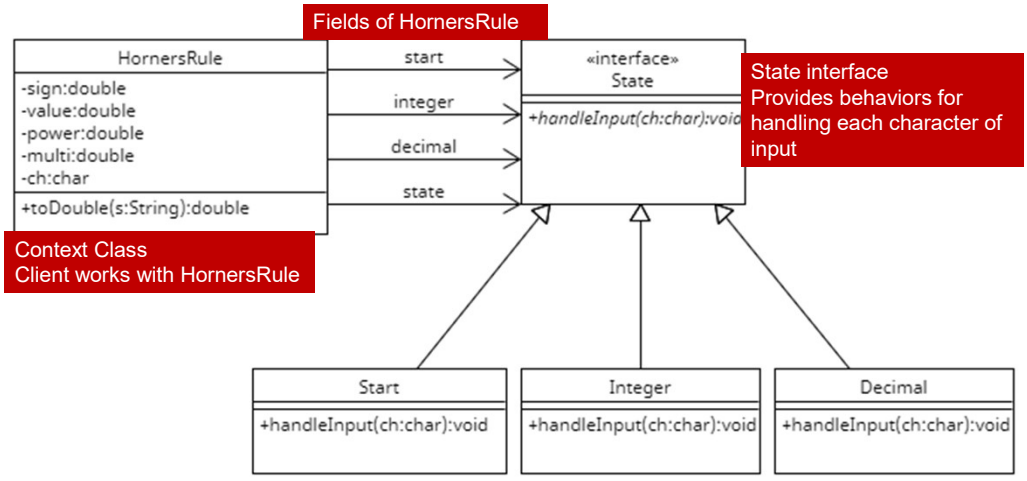
NC STATE UNIVERSITY

Alternative Horner's Rule Designs – Abstract Class



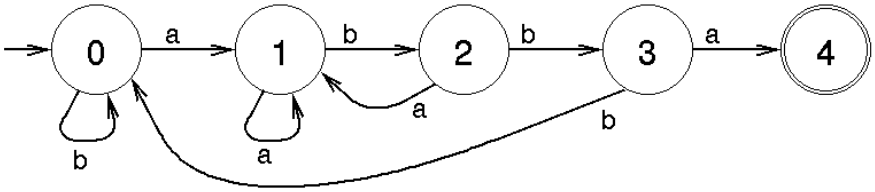
NC STATE UNIVERSITY

Alternative Horner's Rule Design - handleInput



ANOTHER STATE PATTERN EXAMPLE

Designing an OO FSM for "abba"



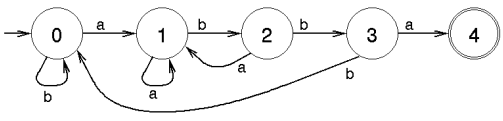
1. Define your State interface or abstract class
2. Implement concrete states
3. Implement context

State Interface for 'abba' FSM

```
public interface State {
   public void onA();
   public void onB();
   public void onOther();
}
```


Inner Classes

```
public class ABBAChecker {
…
    private class Start implements State {
      @Override
      public void onA() {
          state = STATE1;
      }

      @Override
      public void onB() {
          state = START;
      }

      @Override
      public void onOther() {
         throw new IllegalArgumentException("Invalid input");
      }
    }
}
```



NC STATE UNIVERSITY

```
public class ABBAChecker {
 private final int START = 0;
 private final int STATE1 = 1;
 private final int STATE2 = 2
 private final int STATE3 = 3;

 private int state;
 private boolean foundABBA;

 public boolean isInWord(String s){
   foundABBA = false;
   state = START; // initial state
   int i = 0; // traversing index
   char ch; // current character
```

FSM

```
   return false;
 }

}
```


50

NC STATE UNIVERSITY

```
public class ABBAChecker {
 private final int START = 0;
 private final int STATE1 = 1;
 private final int STATE2 = 2
 private final int STATE3 = 3;

 private int state;
 private boolean foundABBA;

 public boolean isInWord(String s){
   foundABBA = false;
   state = START; // initial state
   int i = 0; // traversing index
   char ch; // current character
```

FSM

```
   return false;
 }

}
```

```
while(i < s.length()){
    ch = s.charAt(i);
    switch(state){
        case START:
           if(ch == 'a') {
                state = STATE1;
           } else if(ch == 'b'){
                state = START;
           } break;
        case(STATE1):
           if(ch == 'a'){
                state = STATE1;
           } else if(ch == 'b') {
                state = STATE2;
           } break;
        case(STATE2):
           if(ch == 'a'){
                state = STATE1;
           } else if(ch == 'b'){
                state = STATE3;
           } break;
        case(STATE3):
           if(ch == 'a'){
                 return true;
           } else if(ch == 'b'){
                state = START;
           } break;
    }
    i++;
}
```


51

NC STATE UNIVERSITY

```
public class ABBAChecker {
 private final int START = 0;
 private final int STATE1 = 1;
 private final int STATE2 = 2
 private final int STATE3 = 3;

 private int state;
 private boolean foundABBA;

 public boolean isInWord(String s){
   foundABBA = false;
   state = START; // initial state
   int i = 0; // traversing index
   char ch; // current character

   return false;
 }
}
```

FSM

Inner Classes

```
while(i < s.length()){
   ch = s.charAt(i);
   switch(state){
       case START:
          if(ch == 'a') {
               state = STATE1;
          } else if(ch == 'b'){
               state = START;
          } break;
       case(STATE1):
          if(ch == 'a'){
               state = STATE1;
          } else if(ch == 'b') {
               state = STATE2;
          } break;
       case(STATE2):
          if(ch == 'a'){
               state = STATE1;
          } else if(ch == 'b'){
               state = STATE3;
          } break;
       case(STATE3):
          if(ch == 'a'){
                return true;
          } else if(ch == 'b'){
               state = START;
          } break;
   }
   i++;
}
```


52

NC STATE UNIVERSITY

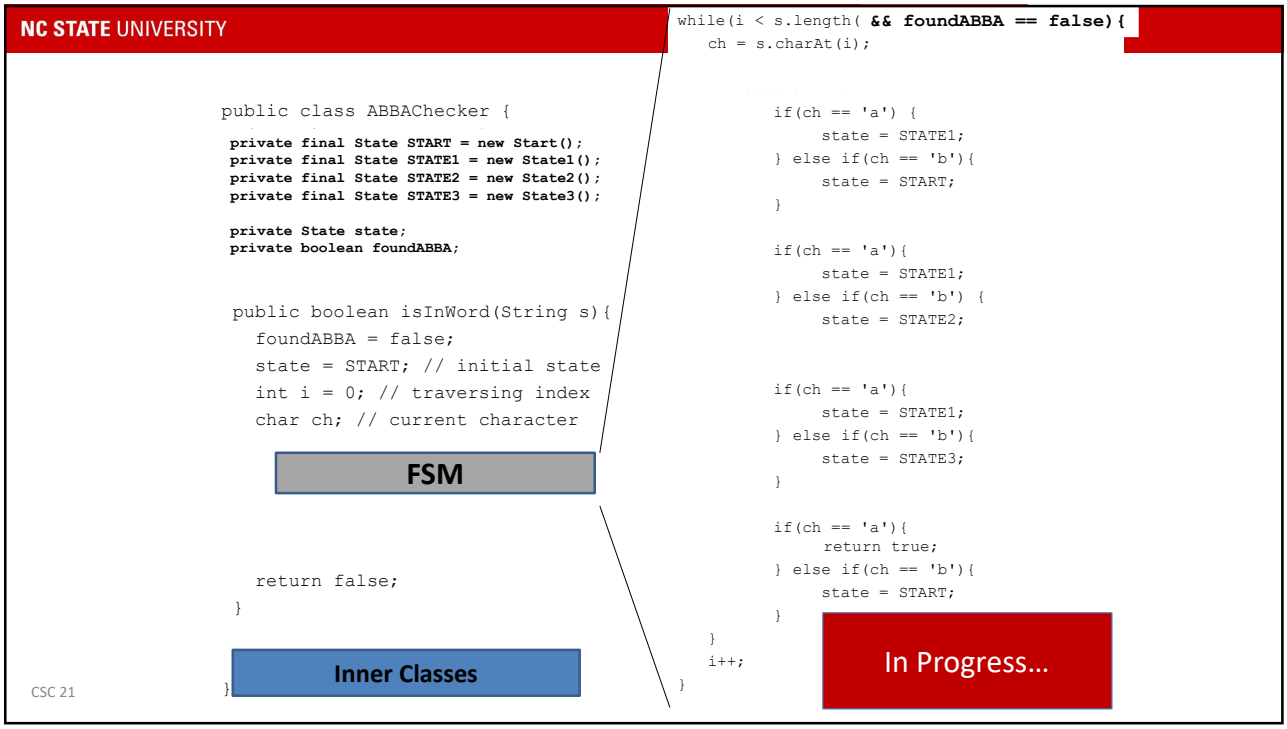
```
public class ABBAChecker {
 private final State START = new Start();
 private final State STATE1 = new State1();
 private final State STATE2 = new State2();
 private final State STATE3 = new State3();

 private State state;
 private boolean foundABBA;

 public boolean isInWord(String s){
   foundABBA = false;
   state = START; // initial state
   int i = 0; // traversing index
   char ch; // current character

   return false;
 }
}
```

FSM

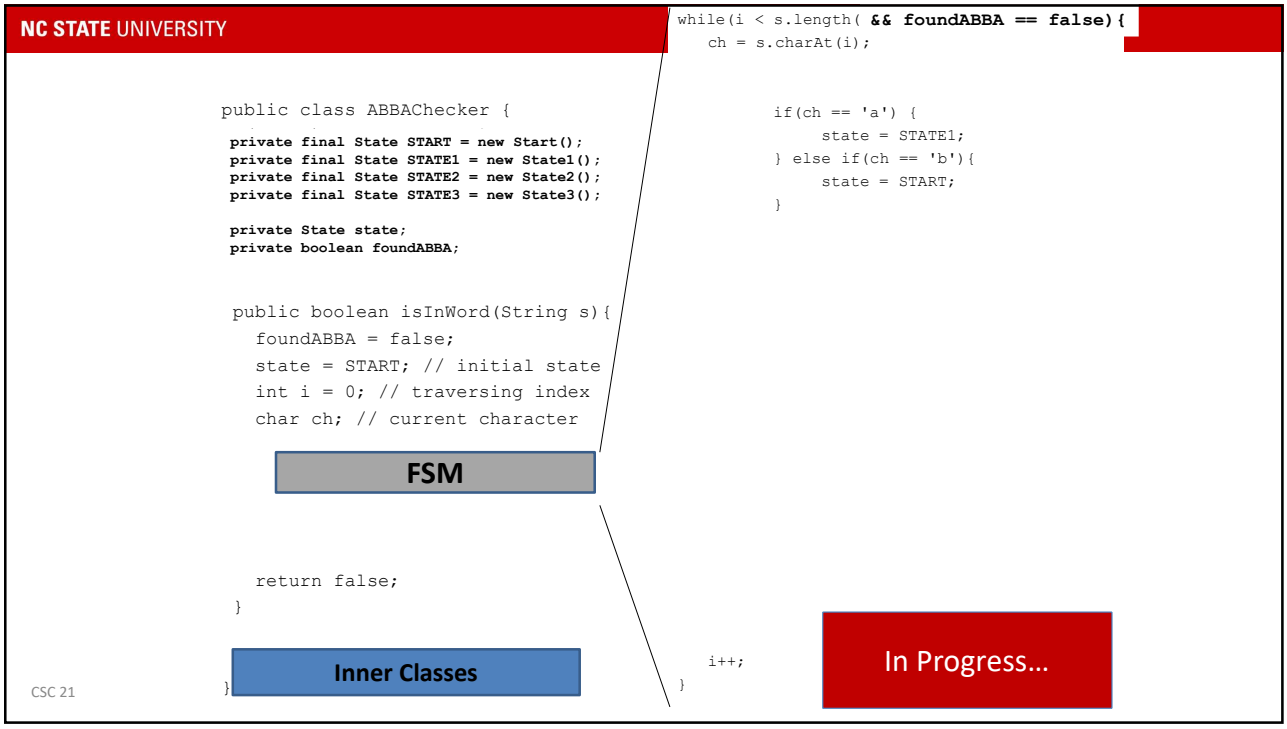
Inner Classes

```
while(i < s.length( && foundABBA == false){
   ch = s.charAt(i);
   switch(state){
       case START:
          if(ch == 'a') {
               state = STATE1;
          } else if(ch == 'b'){
               state = START;
          } break;
       case(STATE1):
          if(ch == 'a'){
               state = STATE1;
          } else if(ch == 'b') {
               state = STATE2;
          } break;
       case(STATE2):
          if(ch == 'a'){
               state = STATE1;
          } else if(ch == 'b'){
               state = STATE3;
          } break;
       case(STATE3):
          if(ch == 'a'){
                return true;
          } else if(ch == 'b'){
               state = START;
          } break;
   }
   i++;
}
```

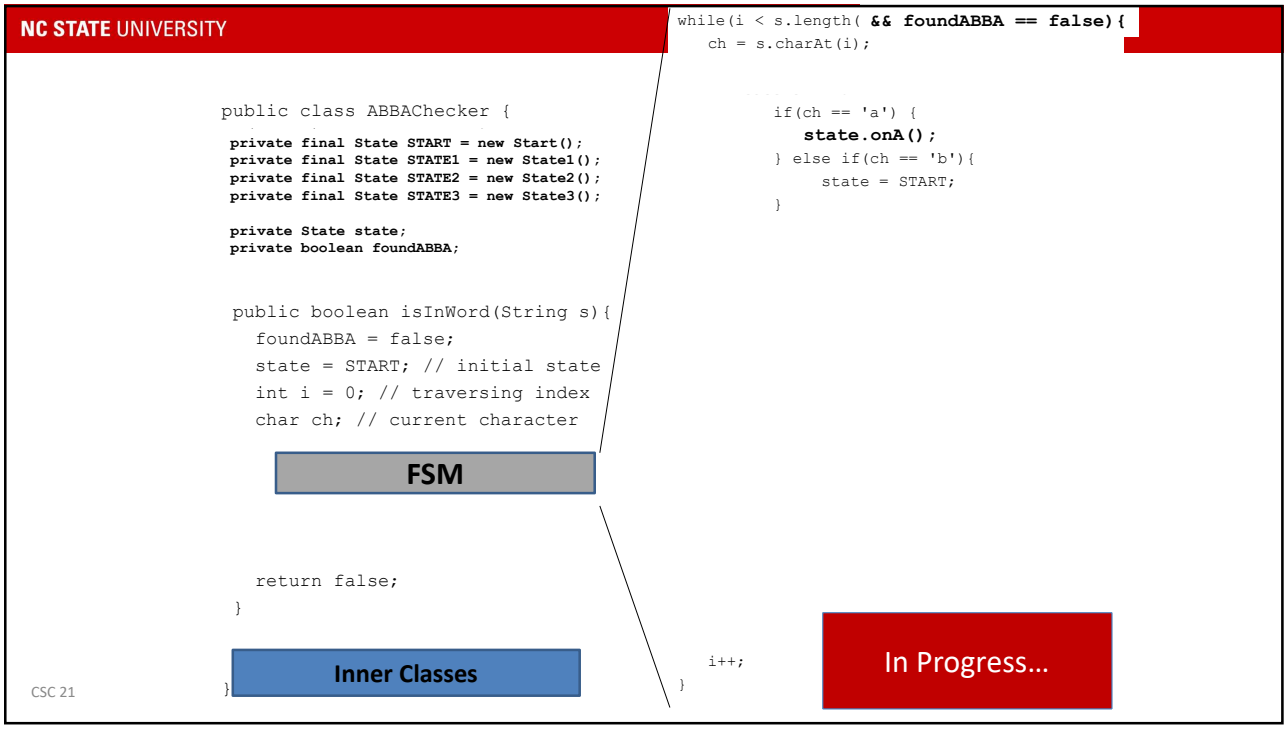

53


NC STATE UNIVERSITY
public class ABBAChecker {
private final State START = new Start();
private final State STATE1 = new State1();
private final State STATE2 = new State2();
private final State STATE3 = new State3();
private State state;
private boolean foundABBA;
public boolean isInWord(String s){
foundABBA = false;
state = START; // initial state
int i = 0; // traversing index
char ch; // current character
FSM
return false;
}
Inner Classes
}
while(i < s.length(&& foundABBA == false){
ch = s.charAt(i);
if(ch == 'a') {
state = STATE1;
} else if(ch == 'b'){
state = START;
}
if(ch == 'a'){
state = STATE1;
} else if(ch == 'b') {
state = STATE2;
if(ch == 'a'){
state = STATE1;
} else if(ch == 'b'){
state = STATE3;
}
if(ch == 'a'){
return true;
} else if(ch == 'b'){
state = START;
}
}
i++;
}
In Progress...
CSC 21

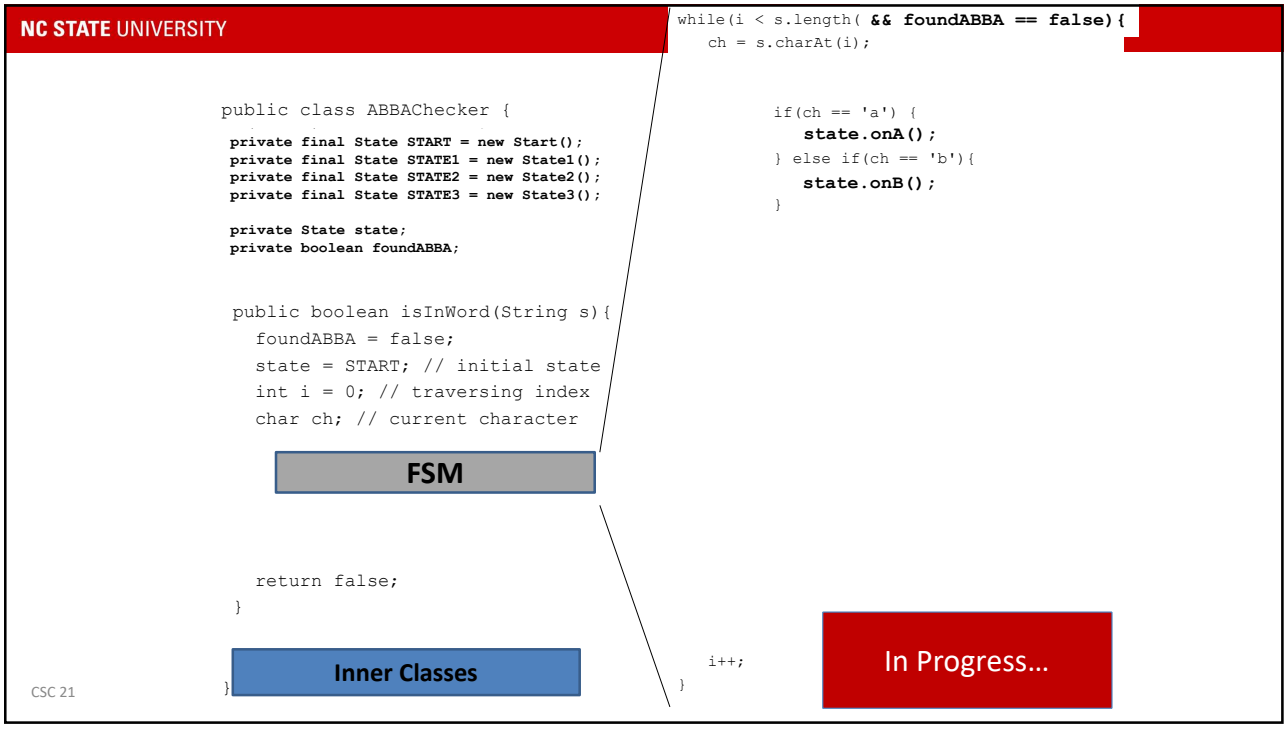
54


NC STATE UNIVERSITY
public class ABBAChecker {
private final State START = new Start();
private final State STATE1 = new State1();
private final State STATE2 = new State2();
private final State STATE3 = new State3();
private State state;
private boolean foundABBA;
public boolean isInWord(String s){
foundABBA = false;
state = START; // initial state
int i = 0; // traversing index
char ch; // current character
FSM
return false;
}
Inner Classes
}
while(i < s.length(&& foundABBA == false){
ch = s.charAt(i);
if(ch == 'a') {
state = STATE1;
} else if(ch == 'b'){
state = START;
}
i++;
}
In Progress...
CSC 21

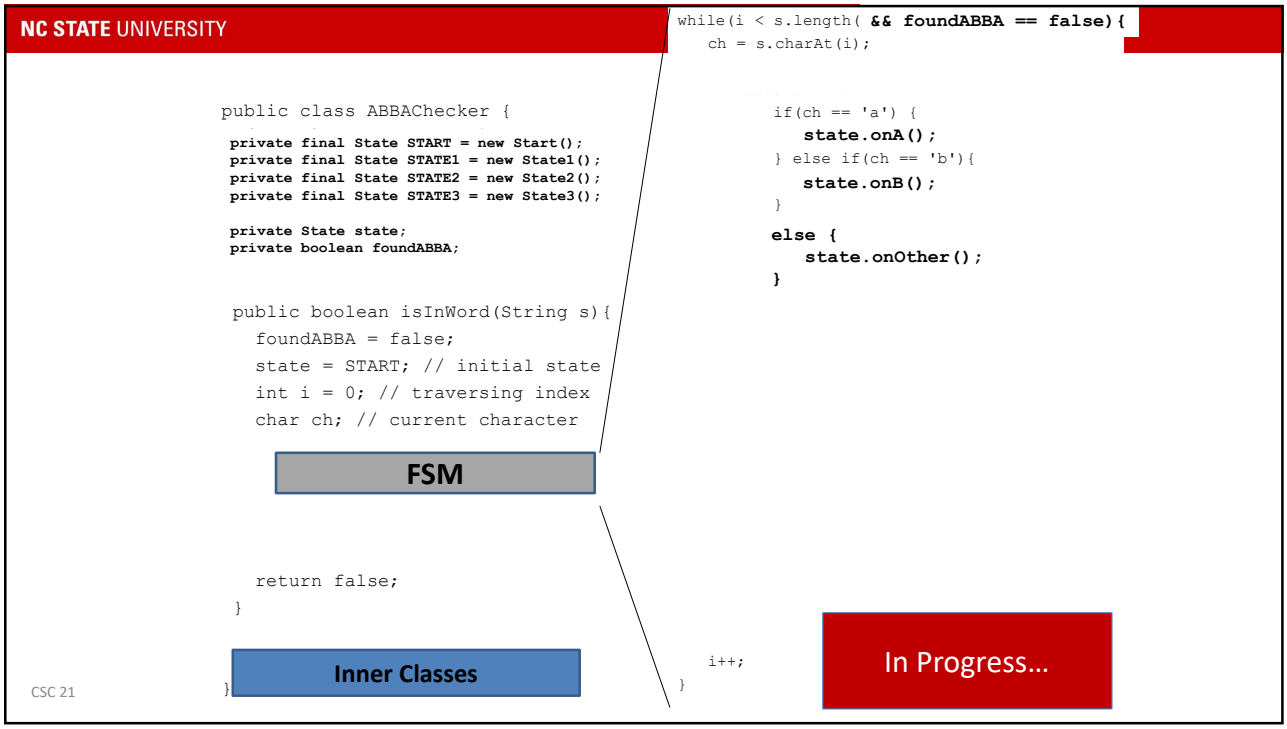
55


NC STATE UNIVERSITY
public class ABBAChecker {
private final State START = new Start();
private final State STATE1 = new State1();
private final State STATE2 = new State2();
private final State STATE3 = new State3();
private State state;
private boolean foundABBA;
public boolean isInWord(String s){
foundABBA = false;
state = START; // initial state
int i = 0; // traversing index
char ch; // current character
FSM
return false;
}
Inner Classes
}
while(i < s.length(&& foundABBA == false){
ch = s.charAt(i);
if(ch == 'a') {
state.onA();
} else if(ch == 'b'){
state = START;
}
i++;
}
In Progress...
CSC 21

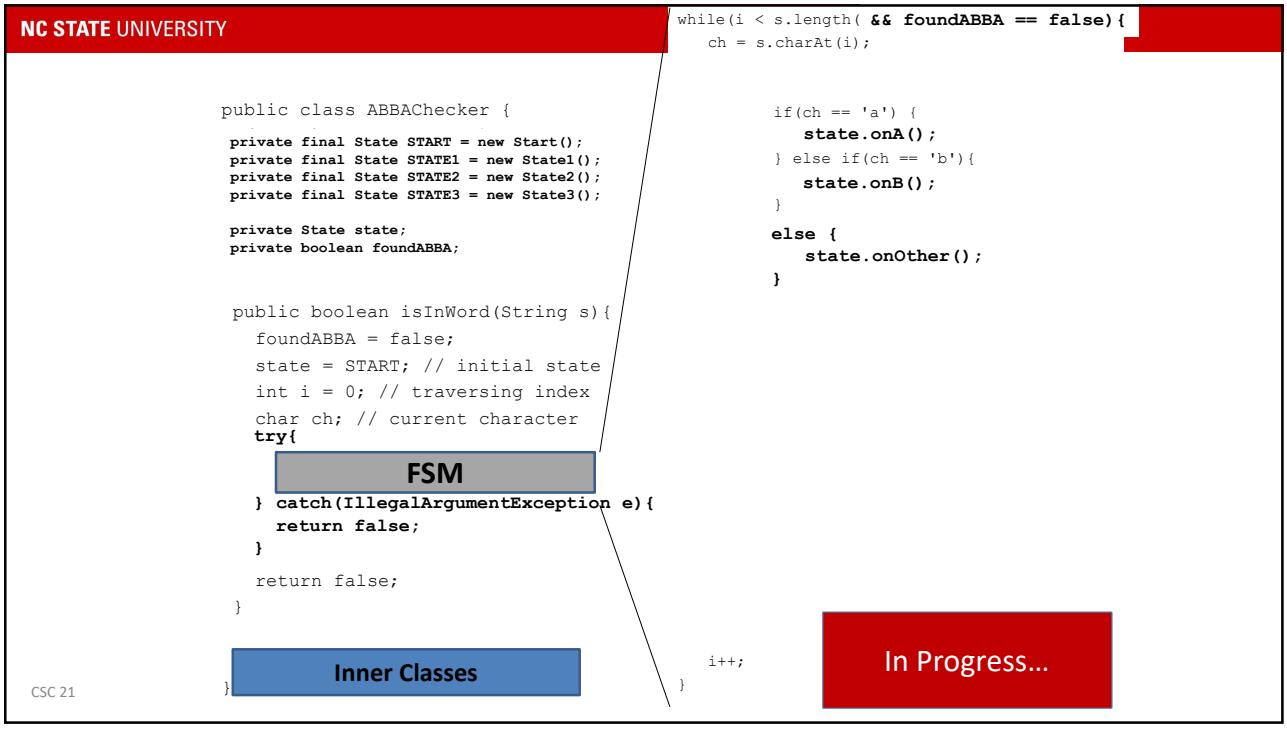
56


NC STATE UNIVERSITY
public class ABBAChecker {
private final State START = new Start();
private final State STATE1 = new State1();
private final State STATE2 = new State2();
private final State STATE3 = new State3();
private State state;
private boolean foundABBA;
public boolean isInWord(String s){
foundABBA = false;
state = START; // initial state
int i = 0; // traversing index
char ch; // current character
FSM
return false;
}
Inner Classes
}
while(i < s.length(&& foundABBA == false){
ch = s.charAt(i);
if(ch == 'a') {
state.onA();
} else if(ch == 'b'){
state.onB();
}
i++;
}
In Progress...
CSC 21

57


NC STATE UNIVERSITY
public class ABBAChecker {
private final State START = new Start();
private final State STATE1 = new State1();
private final State STATE2 = new State2();
private final State STATE3 = new State3();
private State state;
private boolean foundABBA;
public boolean isInWord(String s){
foundABBA = false;
state = START; // initial state
int i = 0; // traversing index
char ch; // current character
FSM
return false;
}
Inner Classes
}
while(i < s.length(&& foundABBA == false){
ch = s.charAt(i);
if(ch == 'a') {
state.onA();
} else if(ch == 'b'){
state.onB();
}
else {
state.onOther();
}
i++;
}
In Progress...
CSC 21

58


NC STATE UNIVERSITY
public class ABBAChecker {
private final State START = new Start();
private final State STATE1 = new State1();
private final State STATE2 = new State2();
private final State STATE3 = new State3();
private State state;
private boolean foundABBA;
public boolean isInWord(String s){
foundABBA = false;
state = START; // initial state
int i = 0; // traversing index
char ch; // current character
try{
FSM
} catch(IllegalArgumentException e){
return false;
}
return false;
}
Inner Classes
}
while(i < s.length(&& foundABBA == false){
ch = s.charAt(i);
if(ch == 'a') {
state.onA();
} else if(ch == 'b'){
state.onB();
}
else {
state.onOther();
}
i++;
}
In Progress...
CSC 21

59

```
public class ABBAChecker {

 private final State START = new Start();
 private final State STATE1 = new State1();
 private final State STATE2 = new State2();
 private final State STATE3 = new State3();

 private State state;
 private boolean foundABBA;


 public boolean isInWord(String s){
   foundABBA = false;
   state = START; // initial state
   int i = 0; // traversing index
   char ch; // current character
   try{
   } catch(IllegalArgumentException e){
     return false;
   }
   return foundABBA;
 }
}
```

FSM

Inner Classes

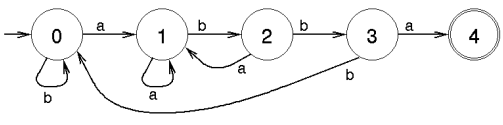
```
while(i < s.length( && foundABBA == false){
   ch = s.charAt(i);

   if(ch == 'a') {
      state.onA();
   } else if(ch == 'b'){
      state.onB();
   }
   else {
      state.onOther();
   }

   i++;
}
```


Testing "abba"

- Test "leaving" *each* state on *each* transition

	On A	On B	On Other
START	STATE1	START	*exception*
STATE1	STATE1	STATE2	*exception*
STATE2	STATE1	STATE3	*exception*
STATE3	***ACCEPT***	START	*exception*

- assertFalse("a");
- assertFalse("ab");
- assertFalse("abb");
- assertTrue("abba");
- …



Alternative Solution

- Let's use an abstract class instead of interface
  - Why?

Abstract class

```
public abstract class State
{
    public abstract void onA();
    public abstract void onB();
    public void onOther() {
        throw new IllegalArgumentException("Invalid Input");
    }
}
```

NC STATE UNIVERSITY

Concrete Inner Classes: Start

```
public class ABBAChecker {
…
    private class Start extends State
    {
        public void onA()
        {
            state = STATE1;
        }

        public void onB()
        {
            state = START;
        }
    }
}
```


NC STATE UNIVERSITY

References

- "Finite-State Machines," by Suzanne Balik and Matthias Stallmann
- Freeman and Freeman, *Head First Design Patterns*, O'Reily, 2004
  - Chapter 10: The State PatternGamma, Helm, Johnson, Vlissides, *Design Patterns*, Addison-Wesley, 1995.
  - State