State Machines

Spring 2013
A Final Word on Specs & Testing

Glass box specs don’t mean testing behavior not in specification

// requires: a, b nonempty strings
// effects: returns a new string that is concatenation of a and b
String myConcat(String a, String b) {
    if a >> b {
        // do something
    } else {
        // do something else
    }
}

Glass box testing points to having testcases for both a>>b and for !(a>>b).

➢ But not tests for empty a or empty b!
Today’s Topics

Snapshot Diagrams
Immutability
State Machines
Snapshot Diagrams: Why?

Used to show the internal state of a program during runtime

We will use them to talk about

- Illustrate concepts such as immutability, primitive types, new object types
- Explaining designs to other project members, your TAs, etc
Snapshot Diagrams

**Primitive data**

10.0 → 5 → 'k'

**Object values**

Point

Point

x → 2

y → 11
Snapshot Diagrams: Immutability

Immutable references (aka “final”)

Immutable objects
➢ These are objects that the designer intends to be immutable
Immutable Objects

Java String objects are immutable

String a = "foo";
a = a.concat("bar");
Immutable Objects

Java String objects are immutable

String a = "foo";
a = a.concat("bar");
Mutable Strings

**StringBuilder is Java’s mutable string class**

```java
StringBuilder sb = new StringBuilder("foo");
sb.append("bar");
```

sb

StringBuilder "foo"
Mutable Strings

**StringBuilder is Java’s mutable string class**

```java
StringBuilder sb = new StringBuilder("foo");
sb.append("bar");
```

(sb

StringBuilder
“foobar”

© 6.005 Course Staff 07-13)
Mutable vs Immutable: So what?

String a = "foo";
a = a.concat("bar");
StringBuilder sb = new StringBuilder("foo");
sb.append("bar");

String b = a;
b.concat("baz");

StringBuilder tb = sb;
tb.append("baz");

String "foobar"
Mutable vs Immutable: So what?

String a = "foo";
a = a.concat("bar");
StringBuilder sb = new StringBuilder("foo");
sb.append("bar");

String b = a;
b.concat("baz");

StringBuilder tb = sb;
tb.append("baz");
Mutable vs Immutable: So what?

String a = "foo";
a = a.concat("bar");
StringBuilder sb = new StringBuilder("foo");
sb.append("bar");

String b = a;
b.concat("baz");

StringBuilder tb = sb;
tb.append("baz");
Mutable vs Immutable: So what?

String a = "foo";
  a = a.concat("bar");
StringBuilder sb = new StringBuilder("foo");
  sb.append("bar");

String b = a;
  b.concat("baz");

StringBuilder tb = sb;
  tb.append("baz");
Mutable vs Immutable: So what?

Immutable strings can result in many unnecessary copies
- E.g. constructing a large string by concatenating many smaller strings
- Temporary copies must later be garbage-collected

Mutable strings can avoid copies
- StringBuilder doesn’t do copies until toString() is called on it
- Clever data structures hidden underneath
Aliasing

Aliasing = multiple references to the same mutable object

Mutable objects & aliasing can introduce risks
- Can be confusing what is happening internally
- Harder to enforce contracts: behavior depends on the behavior of all locations that have references to the object
- Result: specifications may need to make assumptions over all users of an object
- Worse when it comes to concurrency
Mutable Example: Iterators

List\<String\> words = ...;
\textbf{for} (String s: words) {
    System.out.println(s);
}

List\<String\> words = ...;
Iterator iter = words.iterator();
\textbf{while} (iter.hasNext()) {
    String s = iter.next();
    System.out.println(s);
}

\textbf{Iterator methods: hasNext()} and next()

- next() method \textit{mutates} the iterator---advances internal state
Mutable Example: MyIterator
MyIterator Snapshot Diagram

Reference to ArrayList<String> is final

- But that doesn’t mean the list is immutable!
- Can still change elements in the ArrayList<String>
- What happens if the list is changed before the iterator is finished?
State Machines

One way of thinking about mutable objects

State Machine

- A set of states
- Rules for transition between states

![State Machine Diagram]
Simple State Machine Example

Does the string have an even number of a’s?
State Machines: Formalism

Formally, a state machine consists of

- A set of states, State = {...}
- A set of initial states, Init = {...}
- A set of events, Event = {...}
- A transition relation trans ⊆ State × Event × State
- A set of traces (derived from trans and Init)
Simple State Machine Example

Does the string have an even number of a’s?

- A set of states, State = {Start, Odd, Even}
- A set of initial states, Init = {Start}
- A set of events, Event = {a, !a}
- A transition relation trans ⊆ State × Event × State
  \{(Start, !a, Start), (Start, a, Odd), (Odd, !a, Odd), (Odd, a, Even), (Even, a, Odd), (Even, !a, Even)\}
- A set of traces (derived from trans and Init)
  \{<>, <!a>, <!a,!a>, <a>, …\}
MyIterator State Machine

**Mutable object can be modeled by state machine**

- State is represented by instance variables of the object
- Change in state = changing instance variables
- Events are operations that can be performed on the object—its methods
- Mutators change state
MylIterator State Machine

Mutable object can be modeled by state machine

- State is represented by instance variables of the object
- Change in state = changing instance variables
- Events are operations that can be performed on the object—its methods
- Mutators change state

State machine is nondeterministic, but the iterator behaves in a deterministic manner.
**Myliterator State Machine**

**Mutable object can be modeled by state machine**

- State is represented by instance variables of the object
- Change in state = changing instance variables
- Events are operations that can be performed on the object—its methods
- Mutators change state

![State Machine Diagram]

- INPROGRESS
  - hasNext/true
  - next/element
- DONE
  - hasNext/false
MylIterator State Machine

**Mutable object can be modeled by state machine**

- State is represented by instance variables of the object
- Change in state = changing instance variables
- Events are operations that can be performed on the object—its methods
- Mutators change state

![Diagram showing state transitions of a MylIterator state machine]

**INPROGRESS**
- hasNext/true
- next/element
- next/element

**DONE**
- hasNext/false
- next/NotFoundException
State Machines: So What?

Models behavior of complex systems

➢ Simplifies complexity of code, graphically
➢ Useful for more than software (e.g. widely used in networking protocols, HW development, etc)

State machines focus on behavior, not mechanism

➢ Good starting point for design
➢ Mechanism design follows behavior
➢ But gap between mechanism & behavior is non-trivial!

State machines give us a simple model for objects.
State Machines: Motivating Example

Afghanistan, December 2001

- US soldier uses PLGR* to mark Taliban position for air-strike
- Notices battery-low warning, so replaces battery and calls in coordinates
- Resulting strike kills user and two comrades and wounds 20 others

What happened?

- MYCOORD
- ENEMY COORD
- NOCOORD

locate
reset
adjust
State Machines: Motivating Example

Afghanistan, December 2001

- US soldier uses PLGR* to mark Taliban position for air-strike
- Notices battery-low warning, so replaces battery and calls in coordinates
- Resulting strike kills user and two comrades and wounds 20 others

What happened?

- NOCOORD
- MYCOORD
- ENEMY COORD

locate -> adjust
reset
Combining State Machines

Two machines can form a single product machine
- states are tuples
- one state from each machine

“State explosion”
- $k$ machines of $N$ states
- product machine has $N^k$ states

Ex:
Common Misconceptions

State machine != flowchart
- No logic or behavior inside a state

State machines are not perfect models
- Simplification of behavior
- Assist in design, but don’t necessarily reflect all behavior

State machines do not have “decision edges”
- Transitions may be non-deterministic
- Logic may make a deterministic decision, but that’s not part of state machine

Change doesn’t happen within a state
Testing A State Machine

Mutable objects can’t be tested by single inputs

Coverage of a state machine
- All-actions: every event is included in test suite
- All-states: every state of the machine is visited in some test
- All-transitions: every legal transition in the machine is tested
- All-paths: every possible path of transitions is tested

Relative strength of tests
- All-actions & all-states are weaker than all-transitions
- All-transitions is weaker than all-paths
Testing MyIterator

Coverage examples

- **All-actions**
  - Create a MyIterator for a 2-element list
    - Invoke hasNext()
    - Invoke next()

- **All-states**
  - Create a MyIterator for a single-element list
    - Invoke next()

- **All-transitions**
  - Create a MyIterator for a 2-element list
    - Invoke hasNext()
    - Invoke next(), next()
    - Invoke hasNext()
State Machines Summary

Mutable objects can be useful, but introduce difficulties in modeling & testing

State machines are simple model suitable for mutable objects

Testing strategies for state machines can give credible tests for mutable objects