Non-binary trees and visitor pattern

CS201
Spring 2005
Week 11
So far

• we've seen binary trees with uniform nodes:
Binary trees

- Traversal is easy: e.g. first left subtree, then this node, then right subtree
Binary trees

- In lab: we used callbacks to process data but structure is encapsulated in node

CS201 – Spring 2005
public void inOrderCallBack(Callback cb) {
    if (left != null) {
        left.inOrderCallBack(cb);
    }
    cb.visit(data);
    if (right != null) {
        right.inOrderCallBack(cb);
    }
}

public class PrintCallback implements Callback{
    private Stack callStack = new Stack();
    public String toString();
    public void visit(Object obj) {
        callStack.push(obj.toString());
    }
}

CS201 – Spring 2005
Binary trees

- In lab: we also used visitors to process data
- structure is no longer encapsulated in node

CS201 – Spring 2005
public void bstVisit(Visitor v) {
    v.visit(this);
}

public class PrintInOrderVisitor
    implements Visitor {
private Stack callStack = new Stack();
public void visit(BinaryTreeNode btn) {
    // visit the left subtree
    if (btn.getLeft() != null) {
        btn.getLeft().bstVisit(this);
    } else {
        // if left subtree is null, do nothing
        callStack.push(""");
    }
    String leftTree = (String) callStack.pop();
    // visit the data
    if (btn.getData() != null) {
        callStack.push(btn.getData().toString());
    } else {
        callStack.push(" data==null ");
    }
    String Data = (String) callStack.pop
    // visit the right subtree
    if (btn.getRight() != null) {
        btn.getRight().bstVisit(this);
    } else {
        // if right subtree null, do nothing
        callStack.push(""");
    }
    String rightTree =
        (String) callStack.pop();
    // put them together and push on
    // the stack
    callStack.push(
        leftTree + Data + rightTree
    );
}

CS201 – Spring 2005
Simulator setup

- a text file
  - you will attach a reader using a file dialog
- this text file will specify environment
  - robot
    - motors
    - sensors
    - heading
  - walls
  - floormarkings
Partial example

Robotsim "example1"
Author = "cs302 tester"
Clock:
mpt = 50
Start = 0
End = 50000
ROBOT:
Name = "robot1"
color=GREEN
heading=0.0
POSITION = (50.0,50.0)
Sensor
S1
LIGHT
PCT
ACTIVE
OFFSET = (15.0,15.0)
POINTS = (0.0,0.0) (5.0,0.0) (5.0,5.0) (0.0,5.0) (0.0)

Sensor
S2
TOUCH
EDGE
ACTIVE
OFFSET = (-15.0,-15.0)
POINTS = (0.0,0.0) (5.0,0.0) (5.0,5.0) (0.0)

Motor
A
POWER = 4
Direction = forward

Motor
B
POWER = 4
Direction = forward

Wheelbase =
axlelength = 5.0
offset = (5.0,5.0)
robotcim dlm = (0.0) (0.20) (20.20) (20.0) (0.0)

CS201 – Spring 2005
Partial example: hierarchy

Robotsim "example1"
Author = "cs302 tester"
Clock:
  mpt = 50
  Start = 0
  End = 50000

ROBOT:
  Name = "robot1"
  color=GREEN
  heading=0.0
  POSITION = (50.0, 50.0)

Sensor
  S1
  LIGHT
  PCT
  ACTIVE
  OFFSET = (15.0, 15.0)
  POINTS = (0.0, 0.0), (5.0, 0.0), (5.0, 5.0), (0.0, 5.0), (0.0, 0.0)

Sensor
  S2
  TOUCH
  EDGE
  ACTIVE
  OFFSET = (-15.0, -15.0)
  POINTS = (0.0, 0.0), (5.0, 0.0), (5.0, 5.0), (0.0, 5.0), (0.0, 0.0)

Motor
  A
  POWER = 4
  Direction = forward

Motor
  B
  POWER = 4
  Direction = forward

Wheelbase =
axiellength = 5.0
offset = (5.0, 5.0)
robotcim dim = (0.0, 0.20), (20.20, 20.0), (0.0, 0.0)
Partial example: hierarchy

Robotsim "example1"
Author = "cs302 tester"
Clock:
  mpt = 50
  Start = 0
  End = 50000

ROBOT:
  Name = "robot1"
  color = GREEN
  heading = 0.0
  POSITION = (50.0, 50.0)

Sensor:
  S1
  LIGHT
  PCT
  ACTIVE
  OFFSET = (15.0, 15.0)
  POINTS = (0.0, 0.0), (5.0, 0.0), (5.0, 5.0), (0.0, 5.0)

Sensor:
  S2
  TOUCH
  EDGE
  ACTIVE
  OFFSET = (-15.0, -15.0)
  POINTS = (0.0, 0.0), (5.0, 0.0), (5.0, 5.0), (0.0, 5.0), (0.0, 0.0)

Motor:
  A
  POWER = 4
  Direction = forward

  B
  POWER = 4
  Direction = forward

Wheelbase =
  axielen = 5.0
  offset = (5.0, 5.0)
  robotsim dim = (0.0, 0.20), (20.0, 20.0), (20.0, 0.0)

CS201 – Spring 2005
Partial example: hierarchy

RobotSim "example1"
Author = "cs302 tester"

Clock:
- mpt = 50
- Start = 0
- End = 50000

ROBOT:
- Name = "robot1"
- color = GREEN
- heading = 0.0
- POSITION = (50.0, 50.0)

Sensor:
- S1
  - LIGHT
  - PCT
  - ACTIVE
    - OFFSET = (15.0, 15.0)
    - POINTS = (0.0, 0.0) (5.0, 0.0) (5.0, 5.0) (0.0, 5.0)

Sensor:
- S2
  - TOUCH
  - EDGE
  - ACTIVE
    - OFFSET = (-15.0, -15.0)
    - POINTS = (0.0, 0.0) (5.0, 0.0) (5.0, 5.0) (0.0, 5.0)

Motor:
- A
  - POWER = 4
  - Direction = forward

Motor:
- B
  - POWER = 4
  - Direction = forward

Wheelbase:
- axlelength = 5.0
- offset = (5.0, 5.0)

robotSim dim = (0.0) (0.20) (20.20) (20.0) (0.0)
Partial example: hierarchy

public class Clock {
    private int mpt;
    private int startTick;
    private int endTick;
    public Clock(int millisPT, int start, int end) {
        mpt = millisPT;
        startTick = start;
        endTick = end;
    }
}

CS201 – Spring 2005
Partial example: hierarchy

```java
public class RobotDef{
    private Position pos;
    private List sensorList;
    ...

    ROBOT:
    Name = "robot1"
    color=GREEN
    heading=0.0
    POSITION = (50.0,50.0)
    Sensor
    S1
    LIGHT
    PCT
    ACTIVE
    OFFSET = (15.0,15.0)
    POINTS = (0.0,0.0) (5.0,0.0) (5.0,5.0) (0.0, 5.0)
    Motor
    A
    POWER = 4
    Direction = forward
    #no stopped spec
    Motor
    B
    POWER = 4
    Direction = forward
    #no stopped spec
    Wheelbase =
    axielen = 5.0
    offset = (5.0,5.0)
    robotdim = (0.0) (0.20) (20.20) (20.0) (0.0)
```
The constructor for the RobotDefinition

```
RobotDef(NameSpec name,
    ColorSpec colorspec,
    HeadingSpec headingspec,
    PositionSpec positionspec,
    SensorSpecs sensorspecs,
    MotorSpecs motorspecs,
    WheelbaseSpec wheelbasespec,
    RobotDimensionSpec robotdimensionspec)
```
Each class is a tree node

- public RobotDef(NameSpec name, ColorSpec colorspec, HeadingSpec headingspec, PositionSpec positionspec, SensorSpecs sensorspecs, MotorSpecs motorspecs, WheelbaseSpec wheelbasespec, RobotDimensionSpec robotdimensionspec)
Revisit Visitor – need more visit methods: one per node

public interface Visitor {
    void visit(Author author);
    void visit(BaseSpec spec);
    void visit(Clock clock);
    void visit(Color color);
    void visit(ColorSpec spec);
    void visit(DimensionSpec spec);
    etc.
}

CS201 – Spring 2005
You will be given a tree builder class

- CS302 (theory) is writing a parser
- you'll pass a FileReader to the parser
- the parser will return a tree of the environment
- e.g.
  SimulationTree simTree =
    Parser.parse(FileReader("file.txt");
SimBuildVisitor sbv = new SimBuildVisitor();
simTree.visit(sbv);
Environment e = sbv.getEnvironment();

CS201 – Spring 2005
Lab 13 – Simulation Tree Visito

- You will write visitors for each type of node
- The visitor create parts of your simulator environment
  - at a motor node:
    - construct motor and push it on stack
  - at a sensor node:
    - construct sensor and push it on stack
  - at a robot node:
    - first get motors, then get sensors, etc.
    - pop the motors and sensors, etc
    - then construct a robot using these motors and sensors

CS201 – Spring 2005
Example: Clock node

```java
public class Clock {
    private int mpt;
    private int startTick;
    private int endTick;
    public Clock(int millisPT, int start, int end) {
        mpt = millisPT,
        startTick = start;
        endTick = end;
    }
    public void accept(Visitor visitor) {
        visitor.visit(this);
    }
}

public void visit(Clock clock) {
    // visit mpt, start and end
    clock.getM().accept(this);
    clock.getSt().accept(this);
    clock.getEt().accept(this);
    Integer end =
        (Integer) stack.pop();
    Integer start =
        (Integer) stack.pop();
    Integer mpt =
        (Integer) stack.pop();
    Clock tmpClock =
        new Clock(mpt, start, end);
    stack.push(tmpClock);
}
```
Visitor pros and cons

- **Pro:**
  - flexible – allows new traversals rather than just those thought up at first
  - handles differing structure of tree – not just binary trees

- **Con:**
  - must write one for each node class
  - need detailed knowledge of structure of each node – makes it hard to change structure later
  - cannot add new node types without changing Visitor interface first

CS201 – Spring 2005
On Friday

- Joe will lecture on Streams and Exceptions
- Please read Chapter 10
Next week

- Wednesday – Review
- Friday – In class exam
- Will post topics this evening