Object-Oriented Programming with Java

Recitation No. 7

The Routing Table Example

```
public class RoutingTable {
    /**
     * Adds a key-value pair to the routing table, where the key is a binary string
     * specified by the n high-order bits of address
     **/
    public void add(int address, int n, String value) {…}

    /**
     * @return the value associated with the longest prefix in the routing
     * table that matches the argument.
     * @throws NoRouteToHostException if no prefix matches address
     **/
    public String route(int address) throws NoRouteToHostException {…}
}
```

Abstraction Function

$$A: RT \rightarrow 2^{\{(b_1 \ldots b_n, s) | b_i \in \{0,1\}, 1 \leq n \leq 32, s \text{ is a string} \neq null\}}$$

where:

$$RT = \{rt \mid rt \text{ is a } RoutingTable \text{ state}\}$$

No clues about the implementation

Example:

$$A(rt) = \{(000, wifi1), (001, wifi2), (1, eth), (11, ir)\}$$

The Contract

- public void add(int address, int n, String value):
  - Requires:

$$(1 \le n \le 32) \;\; \&\& \;\; (value \ne null)$$

  - Ensures:

$$if\ \exists (a_1 ... a_n, s) \in A(old):$$

$$A(new) = A(old) / \{(a_1 ... a_n, s)\} \cup \{(a_1 ... a_n, value)\}$$

$$otherwise:$$

$$A(new) = A(old) \cup \{(a_1 ... a_n, value)\}$$

where:

$a_1, \ldots, a_n$ are the n high-order bits of address

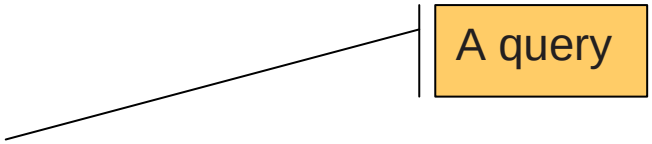
The Contract (cont.)

- public String route(int address)

  throws NoRouteToHostException:

  - Requires: nothing
  - Ensures:

    1. $A(new) = A(old)$

    2. *if a NoRouteToHostException is thrown*:

       $\forall (b_1 ... b_n, s) \in A(this) : b_1 ... b_n \neq a_1 ... a_n$

       otherwise:

       $\exists (a_1 ... a_n, returned\_value) \in A(this)$ &&

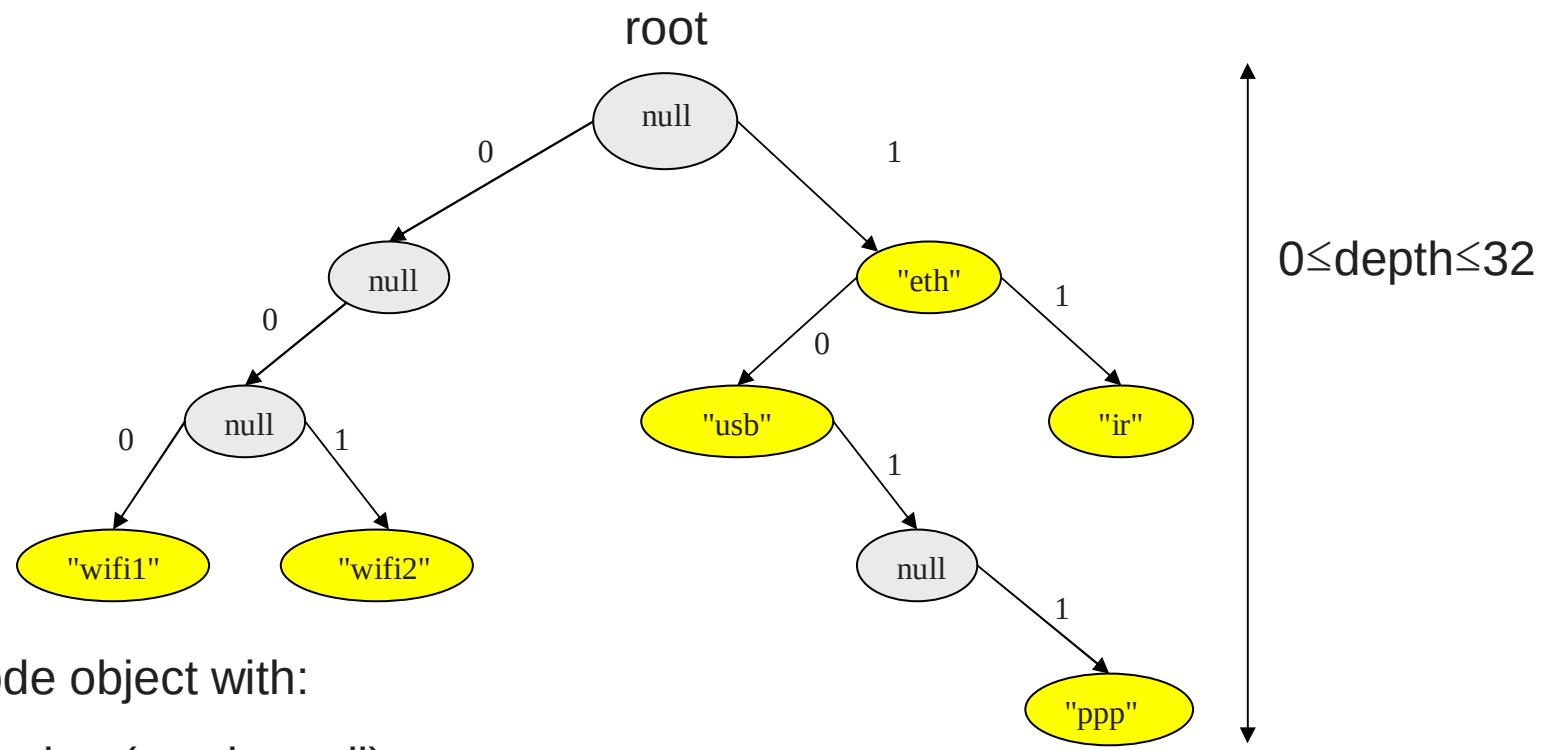
       $\forall (b_1 ... b_{m>n}, s) \in A(this) : b_1 ... b_m \neq a_1 ... a_m$

       where:

       $a_1 ... a_{32}$ *is the binary representation of address*

Implementation

- Based on a binary trie:



is a Node object with:

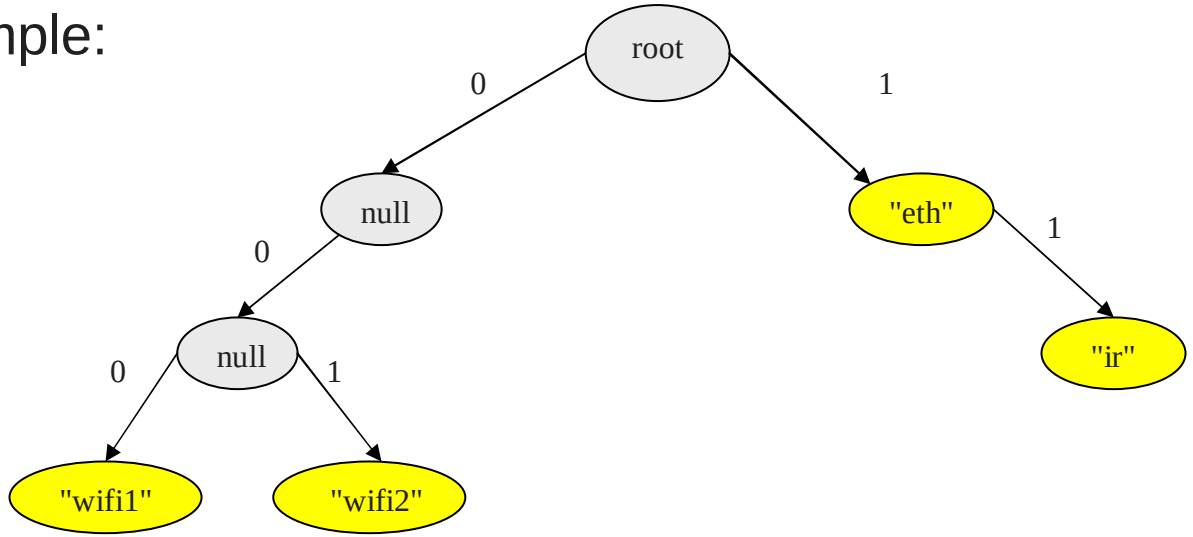
String value (can be null)

Node one,zero (can be null)

Representation Invariant

1. $root \neq null$
2. $0 \leq depth(trie) \leq 32$
3. $A(this) = \{(b_1 ... b_n, s) \mid root \xrightarrow{b_1 ... b_n} v_n \text{ with value } s \neq null \text{ in a trie}\}$

Example:



$$A(this) = \{(000, wifi1), (001, wifi2), (1, eth), (11, ir)\}$$

Class Correctness Proof

- Invariant Verification at initial state:

1. root = new Node()	(field initialization)
2. root != null	(1) ■
3. (root.zero == null) and (root.one == null)	(1, Node's initialization)
4. depth(trie)==0	(3) ■
5. A(this)=Ø	(4) ■
6. All three invariants are satisfied at initial state	(2,4,5)

Class Correctness Proof (cont.)

- add:

add(address, n, value) extracts the n high-order bits of address: $a_{1\ldots}a_n$ = key[0,…,n-1].
The method tries to traverse a path: $root = node_0 \xrightarrow{key[0,\ldots,i<n]} node_{i+1\le n} \in A(old)$
such that: $i = n-1$ *or* $node_i \xrightarrow{key[i]} null$
Then:
if i=n-1: $node_n.value = value$, that is $(key[0,\ldots,n-1], s)$ *is replaced with* $(key[0,\ldots,n-1], value)$

⬇

post-cond. $A(new) = A(old)/\{(a_1\ldots a_n, s)\} \cup \{(a_1\ldots a_n, value)\}$ and inv. 3 are satisfied

otherwise: the path $node_i \xrightarrow{key[i,\ldots,n-1]} node_n$ is added, where:

$$\forall i+1 \le j \le n-1 \; node_j.value = null \; and \; node_n.value = value$$

⬇

post condition $A(new) = A(old) \cup \{(a_1\ldots a_n, value)\}$ and invariant 3 are satisfied

Class Correctness Proof (cont.)

- add (cont.):
  - the method does not modify the reference to root $\Rightarrow$

    root != null and invariant 1 is satisfied
  - $1 \leq n \leq 32$ (pre-condition) $\Rightarrow$

    a path no longer than 32 nodes (excluding the root) is added $\Rightarrow$

    $1 < depth(trie) \leq 32$ and invariant 2 is satisfied

Class Correctness Proof (cont.)

- route:
  - the method does not modify the reference to root $\Rightarrow$
    root != null and invariant 1 is satisfied
  - The trie is unchanged $\Rightarrow$
    A(new)=A(old), 0≤depth(trie) ≤32 $\Rightarrow$
    post condition 1 and invariants 2 and 3 are satisfied

Class Correctness Proof (cont.)

- route (cont.):

The method represents address as a 32-bit array: $a_{1\ldots}a_{32}$ = key[0,…,32].

If a NoRouteToHostException is thrown:

$$\forall 1 \le n \le 32: \ \exists root \xrightarrow{key[0,\ldots,n-1]} node_n \Rightarrow node_n.value = null \quad \text{(code)}$$

$$\downarrow$$

$$\neg\exists (b_1 \ldots b_n, s) \in A(this): \ b_1 \ldots b_n = key[0,\ldots,n-1] \quad \text{(invariant 3)}$$

$$\downarrow$$

$$\forall (b_1 \ldots b_n, s) \in A: b_1 \ldots b_n \neq a_1 \ldots a_n$$

$$\downarrow$$

post condition 2 is satisfied

Class Correctness Proof (cont.)

- route (cont.):

Otherwise:

$returned\_value \neq null$ (code)

$\Downarrow$

1. $\exists\ 1 \leqslant n \leq 32 :\ root \xrightarrow{key[0,\ldots n-1]} node_n$ and $node_n.value = returned\_value \neq null$
2. $n = 32$ or $node_n \xrightarrow{key[n]} null$ (code)

$\Downarrow$

$\exists n : (key[0,\ldots,n-1], returned\_value) \in A(this)$ &&

$\forall (b_1 \ldots b_{m>n}, s) \in A : b_1 \ldots b_n \neq key[0,\ldots,m-1]$

$\Downarrow$

post condition 2 is satisfied

Class Correctness Proof (cont.)

- Other Code Verification:

root is private.

No other code can change the invariant.

The Contract – Different Format

```
/**
 * @pre (1 <= n <= 32) && (value != null)
 * @post get(address,n) == value
 */
 public void add(int address, int n, String value)

/**
 * @pre (1 <= n <= 32)
 * @post nothing
 */
 public String get(int address, int n)
```

The Contract – Different Format

/**

* @pre nothing

* @post if an exception is thrown: $\forall 1 \le i \le 32 : \; get(address, i) == null$

* else, $\exists\, 1 \le i \le 32 : \; get(address, i) = returned\_value \;\&\&\; \forall i < j \le 32 : \; get(address, j) == null$

**/

public String route(int address) throws
java.net.NoRouteToHostException

Visibility

```
package A;

public class Molecule {
    …
    protected void calculateWeight() {
        …
    }
    …
}
```

```
package B;

public class Protein extends Molecule {
    private int bar = 5;
    void foo(Protein p, Molecule m) {
        calculateWeight();
        p.calculateWeight();
        m.calculateWeight();
        bar = p.bar;
        …
    }
    …
}
```

Illegal

Legal

Exceptions

```
int i=1, j=1;
try {
    i++;
     j--;
     if (i/j > 1)
          i++;
} catch(ArithmeticException e) {
    System.out.println(1);
} catch(Exception e) {
    System.out.println(2);
} finally {
    System.out.println(3);
}
```

The output is:
1
3

By-Value/Reference Arguments

```
public class Test {
  private static class Value { int v = 1; }

  public static void main(String[] args) {
    Test test = new Test();
    int v = 2;
    Value value = new Value();
    value.v = 3;
    foo(value, v);
    System.out.println(value.v + " " + v);
  }

  private static void foo(Value value, int v) {
    v= 4;
    value.v = 5;
    value =  new Value();
    System.out.println(value.v + " " + v);
  }
}
```

The output is:

1 4

5 2

Interfaces

```
public interface Foo {
        public void bar()
                throws Exception;
}

public class FooImpl implements Foo {
        public void bar() {
                System.out.println("An exception is not thrown");
        }

        public static void main(String args[]) {
                Foo foo = new FooImpl();
                foo.bar();
        }
}
```

Compilation Error:
"Unhandled exception type Exception"

Interfaces

```
public interface Foo {
        public void bar()
                throws Exception;
}

public class FooImpl implements Foo {
        public void bar() {
                System.out.println("No exception is thrown");
        }

        public static void main(String args[]) {
                FooImpl foo = new FooImpl();
                foo.bar();
        }
}
```

Output:
No exception is thrown