public class MinHeap {
    public Node root;
}
public class MinHeap {
    public Node root;
}
public class MinHeap {
    private Node root;

    public Node root() {
        return root;
    }
}
public class MinHeap {
    private Node root;

    public Node root() {
        return root;
    }
}
public class MinHeap {
    private Node root;

    public Node root() {
        return root;
    }

    public void insert(Node root, String element) {
        blah
    }
}

MinHeap test = new MinHeap();
Node root = test.root();
test.insert(root, "cat");
public class MinHeap {
    private Node root;

    public void insertNode(Node newElement) {
        blah, blah, etc
    }
}

MinHeap test = new MinHeap();
Node element = new Node("cat");
test.insertNode(element);
public class MinHeap {
    private Node root;

    public void insertNode(Node newElement) {
        // blah, blah, etc
    }

    MinHeap test = new MinHeap();
    Node element = new Node("cat");
    test.insertNode(element);
public class MinHeap {
    private Node root;

    public void insertNode(String element) {
        blah, blah, etc
    }
}
public class MinHeap {
    private Node root;

    public void insertNode(String element) {
        blah, blah, etc
    }
}
What is the name of the method used to add something to a collection in Java?

What does the method return?
public class MinHeap {

    public int sizeOfHeap() {
        blah
        return result;
    }

    What name should the method have?
}
What type of collection is a heap?
Why does it only hold Strings?
public class MinHeap {
    private static Node root;
    MinHeap first = new MinHeap();
    MinHeap second = new MinHeap();
    first.add("cat");
    second.add("dog");
    int wrong = second.size();

    What value does wrong have?
    What value should it be
public class MinHeap {
    private static Node root;

    MinHeap first = new MinHeap();
    MinHeap second = new MinHeap();
    first.add("cat");
    second.add("dog");
    int wrong = second.size();

    What value does wrong have?
    What value should it be
public class MinHeap {

    public void display() {
        blah;
        System.out.println( currentNode.value);
        blah;
    }
}
public class MinHeap {

    public void display() {
        blah;
        System.out.println( currentNode.value);
        blah;
    }
}
public class MinHeap {

    public String display() {
        blah;
        blah;
        return result;
    }
}

How is this displaying anything?
public class MinHeap {

    public String display() {
        blah;
        blah;
        return result;
    }

    What do we call the method that returns a string representation of the object?
public class MinHeap {

    public String toString() {
        blah;
        blah;
        blah;
        return result;
    }
}
public class MinHeap {

    public ArrayList display() {
        blah;
        blah;
        blah;
        return result;
    }
}
public class MinHeap {

    public <T> T[] toArray(T[] a) {
        blah;
        blah;
        return result;
    }

    The Java collection standard

Abstraction?

public class MinHeap {
    public void displayWordsEndingInIng() {
        blah;
        blah;
    }
}
public class MinHeap {
    private Node root;
    private String ingWords;  // Not part of class state

    public void display() {
        blah;
        ingWords = ingWords + currentNode.value;
        blah;
    }

    public String getIngWords() {
        return ingWords;
    }
}
public class MinHeap {
    public Iterator iterator() {
        blah;
        blah;
    }
}

MinHeap test = new MinHeap();
//add elements

for (Iterator elements = test.iterator; elements.hasNext();)
    String item = elements.next();
    if (some condition on item) {
        }
    }
public class MinHeap {

    //Declare fields
    private Node root;

    //constructor
    public MinHeap() { blah }

    // return the root
    public Node getRoot() { blah }
}

Duh Comments
What have we lost?

public class MinHeap {

    private Node root;

    public MinHeap() { blah }

    public Node getRoot() { blah }

}
Data Class

public Node {
    public String value;
    public Node left;
    public Node right;
}

Where are the Operations?

```java
public Node {
    private String value;
    private Node left;
    private Node right;

    public String getValue() { return value;}
    public void setValue(String x) {value = x;}

    etc.
}
```
Find all the Helper methods in MinHeap

public class MinHeap {

    private int getHeight(Node aNode) {
        if (aNode.left == null & aNode.right == null)
            return 1;
        if (aNode.left == null)
            return 1;
        return 1 + Math.max(getHeight(aNode.left()), getHeight(aNode.right()));
    }
}
Why not put it in Node class?

public class Node {

    public int height() {
        if (left == null & right == null)
            return 1;
        if (left == null)
            return 1;
        return 1 + Math.max(left.height(), right.height());
    }
}
Store the value

public class Node {
    private int height;

    public int height() {
        return height();
    }
}
public class Node {
    private static final NOT_SET = -1;
    private int height;

    public boolean add(String value) {
        height = NOT_SET; // adding may change height
        blah blah
    }

    public int height() {
        if (height == NOT_SET)
            height = computeHeight();
        return height;
    }

    private int computeHeight() { blah blah }
}
Strategy Pattern
Favor Composition over Inheritance
Orderable List

Sorted
Reverse Sorted
Random
One size does not fit all
Issue 1 - Orthogonal Features

Order
Sorted
Reverse Sorted
Random

Threads
Synchronized
Unsynchronized

Mutability
Mutable
Non-mutable
OrderableList

SortedList
  - Mutable
    - Synch
    - Not Synch
  - NonMutable
    - Synch
    - Not Synch

ReverseList
  - Mutable
    - Synch
    - Not Synch
  - NonMutable
    - Synch
    - Not Synch

RandomList
  - Mutable
    - Synch
    - Not Synch
  - NonMutable
    - Synch
    - Not Synch
Issue 2 - Flexibility
Change behavior at runtime

OrderableList x = new OrderableList();
x.makeSorted();
x.add(foo);
x.add(bar);
x.makeRandom();
Configure objects behavior at runtime
class OrderableList {
    private Object[] elements;
    private Algorithm orderer;

    public OrderableList(Algorithm x) {
        orderer = x;
    }

    public void add(Object element) {
        elements = ordered.add(elements, element);
    }
}
Structure

Context
contextInterface()

Strategy
algorithmInterface()

ConcreteStrategyA
algorithmInterface()

ConcreteStrategyB
algorithmInterface()
The algorithm is the operation

Context contains the data

How does this work?
Prime Directive
Data + Operations
How does Strategy Get the Data?

Pass needed data as parameters in strategy method

Give strategy object reference to context
Strategy extracts needed data from context
Example - Java Layout Manager

```java
import java.awt.*;

class FlowExample extends Frame {

    public FlowExample( int width, int height ) {
        setTitle( "Flow Example" );
        setSize( width, height );
        setLayout( new FlowLayout( FlowLayout.LEFT) );

        for ( int label = 1; label < 10; label++ )
            add( new Button( String.valueOf( label ) ) );

        show();
    }

    public static void main( String args[] ) {
        new FlowExample( 175, 100 );
        new FlowExample( 175, 100 );
    }
}
```
Example - Smalltalk Sort blocks

| list |
list := #( 1 6 2 3 9 5 ) asSortedCollection.
Transcript
    print: list;
    cr.
list sortBlock: [:x :y | x > y].
Transcript
    print: list;
    cr;
    flush.
Costs

Clients must be aware of different Strategies

Communication overhead between Strategy and Context

Increase number of objects
Benefits

Alternative to subclassing of Context

Eliminates conditional statements

Replace in Context code like:

```java
switch ( flag ) {
  case A: doA(); break;
  case B: doB(); break;
  case C: doC(); break;
}
```

With code like:

```java
strategy.do();
```

Gives a choice of implementations
Refactoring: Replace Conditional Logic with Strategy

Conditional logic in a method controls which of several variants of a calculation are executed

so

Create a Strategy for each variant and make the method delegate the calculation to a Strategy instance
Replace Conditional Logic with Strategy

class Foo {
    public void bar() {
        switch ( flag ) {
            case A: doA(); break;
            case B: doB(); break;
            case C: doC(); break;
        }
    }
}

class Foo {
    private strategy;
    public void bar() {
        strategy.do(data);
    }
}