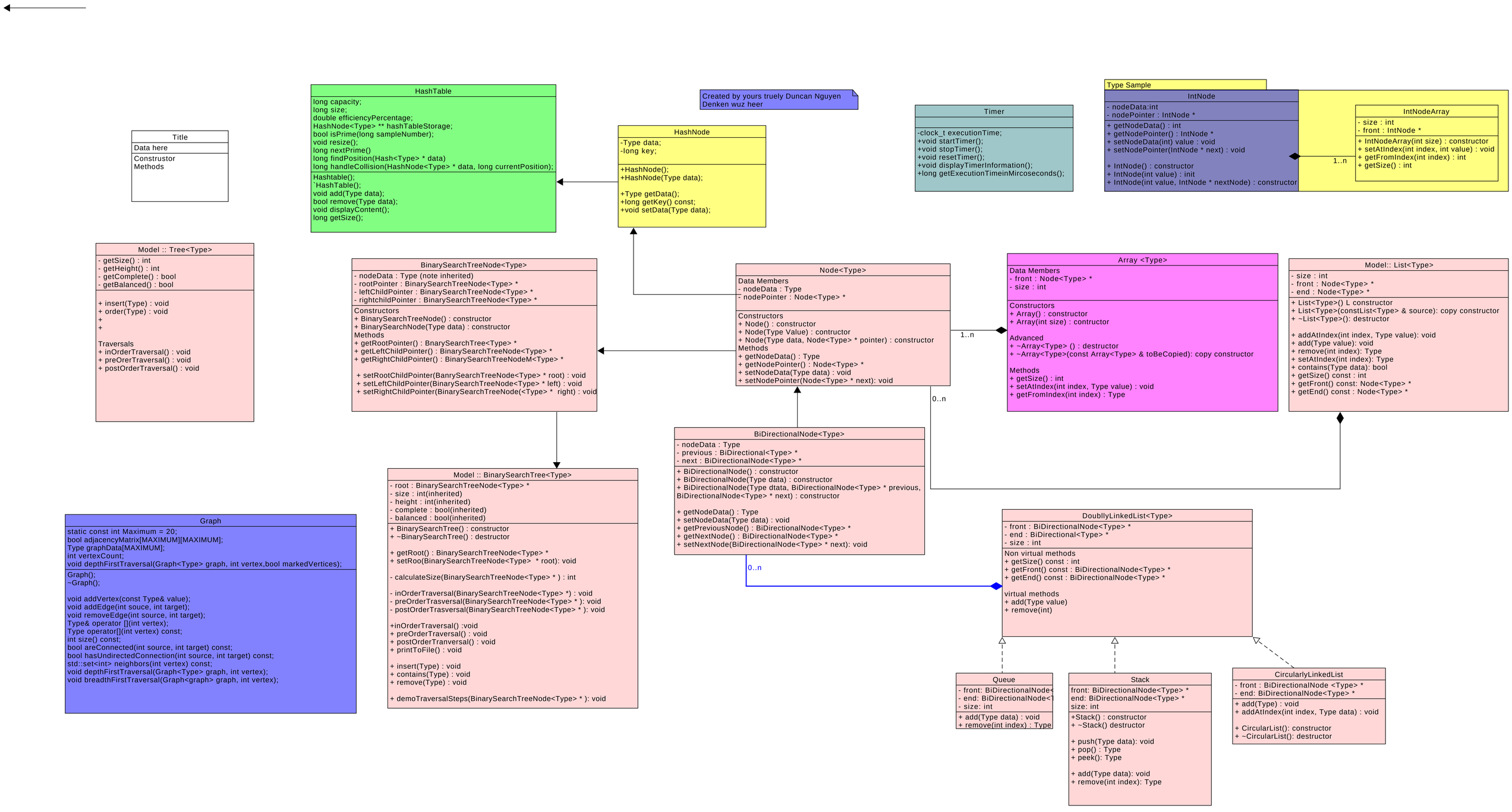
Title
Data here
Construstor
Methods
HashTable
long capacity;
long size;
double efficiencyPercentage;
HashNode<Type> ** hashTableStorage;
bool isPrime(long sampleNumber);
void resize();
long nextPrime()
long findPosition(Hash<Type> * data)
long handleCollision(HashNode<Type> * data, long currentPosition);
Hashtable();
`HashTable();
void add(Type data);
bool remove(Type data);
void displayContent();
long getSize();
Created by yours truely Duncan Nguyen
Denken wuz heer
HashNode
-Type data;
-long key;
+HashNode();
+HashNode(Type data);
+Type getData();
+long getKey() const;
+void setData(Type data);
Timer
-clock_t executionTime;
+void startTimer();
+void stopTimer();
+void resetTimer();
+void displayTimerInformation();
+long getExecutionTimeinMircoseconds();
Type Sample
IntNode
- nodeData:int
- nodePointer : IntNode *
+ getNodeData() : int
+ getNodePointer() : IntNode *
+ setNodeData(int) value : void
+ setNodePointer(IntNode * next) : void
+ IntNode() : constructor
+ IntNode(int value) : init
+ IntNode(int value, IntNode * nextNode) : constructor
1..n
IntNodeArray
- size : int
- front : IntNode *
+ IntNodeArray(int size) : constructor
+ setAtIndex(int index, int value) : void
+ getFromIndex(int index) : int
+ getSize() : int
Model :: Tree<Type>
- getSize() : int
- getHeight() : int
- getComplete() : bool
- getBalanced() : bool
+ insert(Type) : void
+ order(Type) : void
+
+
Traversals
+ inOrderTraversal() : void
+ preOrerTraversal() : void
+ postOrderTraversal() : void
BinarySearchTreeNode<Type>
- nodeData : Type (note inherited)
- rootPointer : BinarySearchTreeNode<Type> *
- leftChildPointer : BinarySearchTreeNode<Type> *
- rightchildPointer : BinarySearchTreeNode<Type> *
Constructors
+ BinarySearchTreeNode() : constructor
+ BinarySearchNode(Type data) : constructor
Methods
+ getRootPointer() : BnarySearchTree<Type> *
+ getLeftChildPointer() : BinarySearchTreeNode<Type> *
+ getRightChildPointer() : BinarySearchTreeNodeM<Type> *
+ setRootChildPointer(BanrySearchTreeNode<Type> * root) : void
+ setLeftChildPointer(BinarySearchTreeNode<Type> * left) : void
+ setRightChildPointer(BinarySearchTreeNode(<Type> * right) : void
Node<Type>
Data Members
- nodeData : Type
- nodePointer : Node<Type> *
Constructors
+ Node() : constructor
+ Node(Type Value) : contructor
+ Node(Type data, Node<Type> * pointer) : constructor
Methods
+ getNodeData() : Type
+ getNodePointer() : Node<Type> *
+ setNodeData(Type data) : void
+ setNodePointer(Node<Type> * next): void
1..n
Array <Type>
Data Members
- front : Node<Type> *
- size : int
Constructors
+ Array() : constructor
+ Array(int size) : contructor
Advanced
+ ~Array<Type> () : destructor
+ ~Array<Type>(const Array<Type> & toBeCopied): copy constructor
Methods
+ getSize() : int
+ setAtIndex(int index, Type value) : void
+ getFromIndex(int index) : Type
Model:: List<Type>
- size : int
- front : Node<Type> *
- end : Node<Type> *
+ List<Type>() L constructor
+ List<Type>(constList<Type> & source): copy constructor
+ ~List<Type>(): destructor
+ addAtIndex(int index, Type value): void
+ add(Type value): void
+ remove(int index): Type
+ setAtIndex(int index): Type
+ contains(Type data): bool
+ getSize() const : int
+ getFront() const: Node<Type> *
+ getEnd() const : Node<Type> *
0..n
BiDirectionalNode<Type>
- nodeData : Type
- previous : BiDirectional<Type> *
- next : BiDirectionalNode<Type> *
+ BiDirectionalNode() : constructor
+ BiDirectionalNode(Type data) : constructor
+ BiDirectionalNode(Type dtata, BiDirectionalNode<Type> * previous, BiDirectionalNode<Type> * next) : constructor
+ getNodeData() : Type
+ setNodeData(Type data) : void
+ getPreviousNode() : BiDirectionalNode<Type> *
+ getNextNode() : BiDirectionalNode<Type> *
+ setNextNode(BiDirectionalNode<Type> * next): void
0..n
Model :: BinarySearchTree<Type>
- root : BinarySearchTreeNode<Type> *
- size : int(inherited)
- height : int(inherited)
- complete : bool(inherited)
- balanced : bool(inherited)
+ BinarySearchTree() : constructor
+ ~BinarySearchTree() : destructor
+ getRoot() : BinarySearchTreeNode<Type> *
+ setRoo(BinarySearchTreeNode<Type> * root): void
- calculateSize(BinarySearchTreeNode<Type> *) : int
- inOrderTraversal(BinarySearchTreeNode<Type> *) : void
- preOrderTrasversal(BinarySearchTreeNode<Type> *): void
- postOrderTrasversal(BinarySearchTreeNode<Type> *): void
+inOrderTraversal() :void
+ preOrderTraversal() : void
+ postOrderTranversal() : void
+ printToFile() : void
+ insert(Type) : void
+ contains(Type) : void
+ remove(Type) : void
+ demoTraversalSteps(BinarySearchTreeNode<Type> *): void
Graph
static const int Maximum = 20;
bool adjacencyMatrix[MAXIMUM][MAXIMUM];
Type graphData[MAXIMUM];
int vertexCount;
void depthFirstTraversal(Graph<Type> graph, int vertex,bool markedVertices);
Graph();
~Graph();
void addVertex(const Type& value);
void addEdge(int souce, int target);
void removeEdge(int source, int target);
Type& operator [](int vertex);
Type operator[](int vertex) const;
int size() const;
bool areConnected(int source, int target) const;
bool hasUndirectedConnection(int source, int target) const;
std::set<int> neighbors(int vertex) const;
void depthFirstTraversal(Graph<Type> graph, int vertex);
void breadthFirstTraversal(Graph<graph> graph, int vertex);
DoublyLinkedList<Type>
- front : BiDirectionalNode<Type> *
- end : BiDirectional<Type> *
- size : int
Non virtual methods
+ getSize() const : int
+ getFront() const : BiDirectionalNode<Type> *
+ getEnd() const : BiDirectionalNode<Type> *
virtual methods
+ add(Type value)
+ remove(int)
Queue
- front: BiDirectionalNode<
- end: BiDirectionalNode<T
- size: int
+ add(Type data) : void
+ remove(int index) : Type
Stack
front: BiDirectionalNode<Type> *
end: BiDirectionalNode<Type> *
size: int
+Stack() : constructor
+ ~Stack() destructor
+ push(Type data): void
+ pop() : Type
+ peek(): Type
+ add(Type data): void
+ remove(int index): Type
CircularlyLinkedList
- front : BiDirectionalNode <Type> *
- end: BiDirectionalNode<Type> *
+ add(Type) : void
+ addAtIndex(int index, Type data) : void
+ CircularList(): constructor
+ ~CircularList(): destructor