Food Ordering Simulator

Presented By Creative Coders:

Samanvitha Basole, Yanpeng Li, Aidan Nguyen, Katherine Soohoo

Food Ordering Simulator

- Java GUI Application
- Demonstrates use of data structures in restaurant businesses

Menu & Ordering

Kitchen

Pick Up

PROGRAM RUN

Data Structures

HashMap

BST

Queue

AVLTree

Stack

LinkedList

HashMap

Implementation

Features

Application

HashMap.java

Implementation

Features

Application

- Problem: We needed a data structure to store the menu items by categorizing each item
- Solution: By using a Hashmap, we are able to add a category and a tree of items under each category
- Able to use the hash function to transform the key into the index of an array element where corresponding value is to be sought

Implementation

Features

Application

- Menu has a category name
- Under each category - a list of items
- Example: Drinks category consists of Pepsi, Coffee, Iced Tea...
- Menu class “is a” HashMap
  String as Key
  AVLTree as Value

Menu Class

Implementation

Features

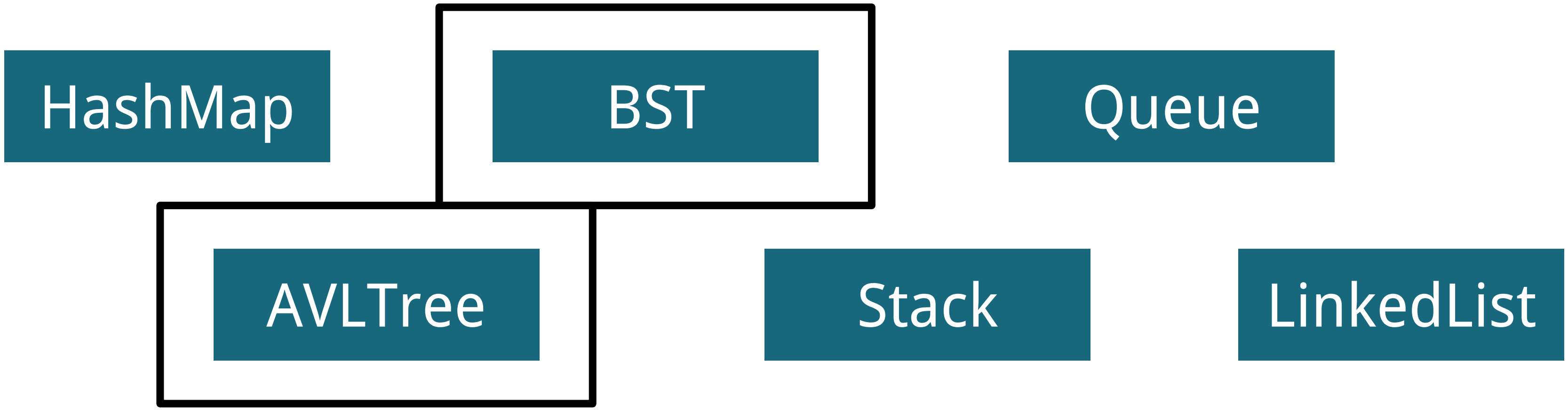
Application

```
public class Menu extends HashMap<String, AVLTree> {

    void addEntry(String category, AVLTree itemList)
{
        put(category, itemList);
    }

    void removeEntry(String category) {
        remove(category);
    }
}
```

Data Structures



HashMap

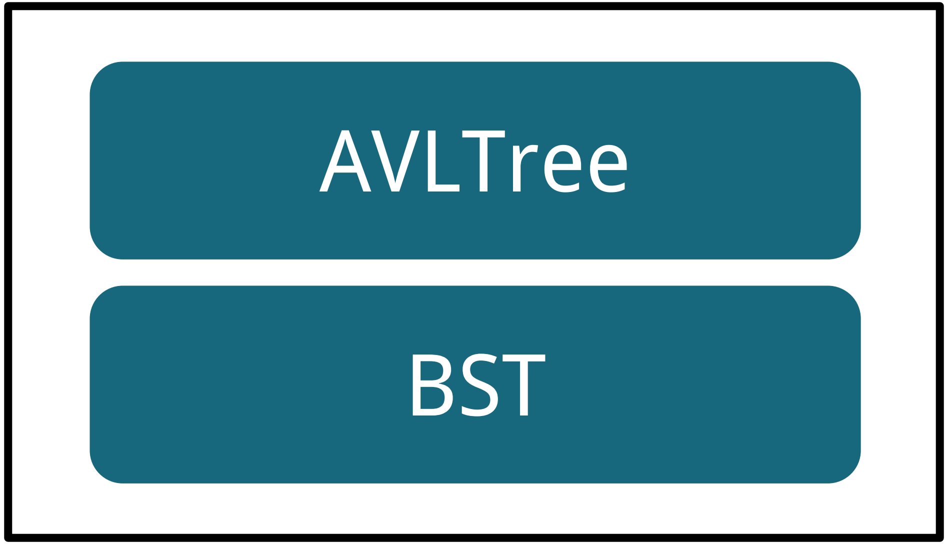
BST

Queue

AVLTree

Stack

LinkedList



AVLTree

BST

Implementation

Features

Application

BinarySearchTree.java

AVLTree.java

Implementation

Features

Application

- Problem: Menu needs to hold a list of items based on category that can be easily added to and accessed.
- Solution: BST provides the best time complexity for adding, removing, and accessing. AVLTree improves on worst case complexities.

Implementation

Features

Application

- List of items in a category are added to an AVLTree before being added to the menu.

```
// loop through lines in file
AVLTree itemList = new AVLTree();
while (in.hasNextLine())
{
    String st = in.next();
    double price = Double.parseDouble(in.next());
    itemList.insert(new Item(st, price));
}

// add list of items to menu
menu.addEntry(food.getName(), itemList);
```

Implementation

Features

Application

- AVLTree iterator used to display all items in a category.

```
// add each item under category
Iterator<Item> iter = entry.getValue().iterator();
while(iter.hasNext())
{
    Item item = iter.next();
    JPanel foodPanel = new JPanel(
          new FlowLayout(FlowLayout.LEFT));
    JLabel itemLabel = new JLabel(
          item.getName() + ", $" + item.getPrice());
    JButton more = new JButton("+");
    JButton less = new JButton("-");
    JLabel count = new JLabel(item.getQuantity() + "");
    ...
}
```

Data Structures

HashMap

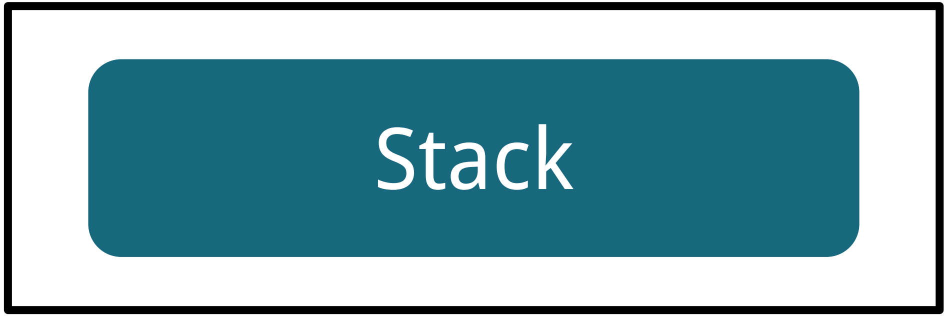
BST

Queue

AVLTree

Stack

LinkedList



Stack

Implementation

Features

Application

[Stack.java]()

Implementation

Features

Application

- Problem: Needed an efficient way to implement an iterator for a BST.

- Solution: A stack provides an easy and efficient way to store and retrieve nodes while iterating through a BST.

Implementation

Features

Application

- When BSTIterator is created, the root and its left nodes are pushed to the stack.

```
public BSTIterator()
{
    stack = new Stack<>();

    // push all left nodes to stack
    while (theRoot != null) {
            stack.push(theRoot);
            theRoot = theRoot.left;
    }
}
```

Implementation

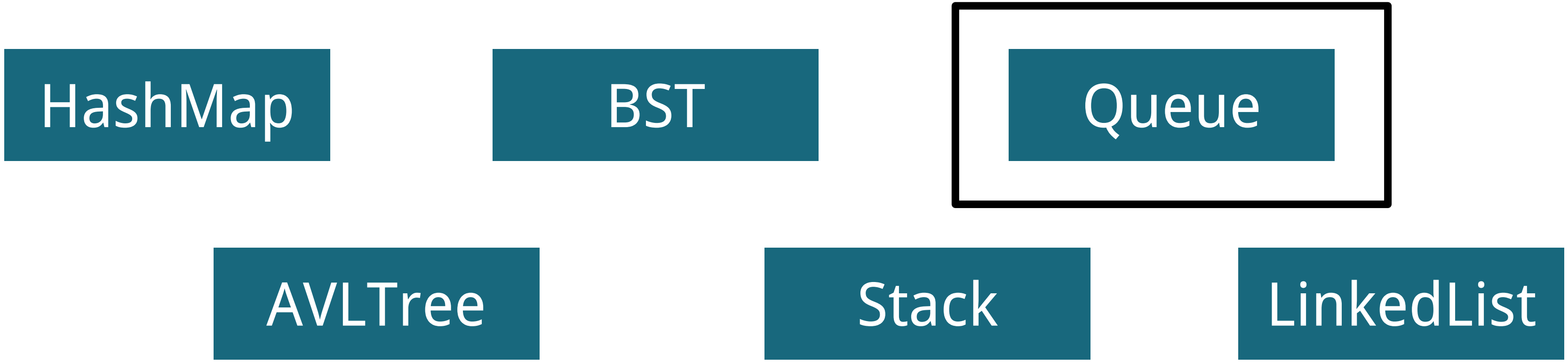
Features

Application

- Node at top gets returned. The node to the right and its left nodes are pushed on to stack.

```
public T next() {
    BinaryNode<T> node = stack.pop();
    T result = node.data;
    if (node.right != null) {
        node = node.right;
        while (node != null) {
            stack.push(node);
            node = node.left;
        }
    }
    return result;
}
```

Data Structures



HashMap

BST

Queue

AVLTree

Stack

LinkedList

Queue

Implementation

Features

Application

Queue.java

Implementation

Features

Application

- Problem: We need a data structure that will most efficiently serve customers once their order is placed to notify the kitchen staff to cook.

- Solution: Out of every data structure thus far, queues let us easily serve customers in the **fairest** order. However, heaps (priority queues) were also discussed, but eventually we went with queues.

Implementation

Features

Application

- Once an order has been placed from the customer, the kitchen cooks the meal from first customer in, to first customer out.
- By using a queue, we take advantage of FIFO approach.
- After the order has been completed, the “remove order” button dequeues the order, and puts it into the PickUpList.

Kitchen Class

Implementation

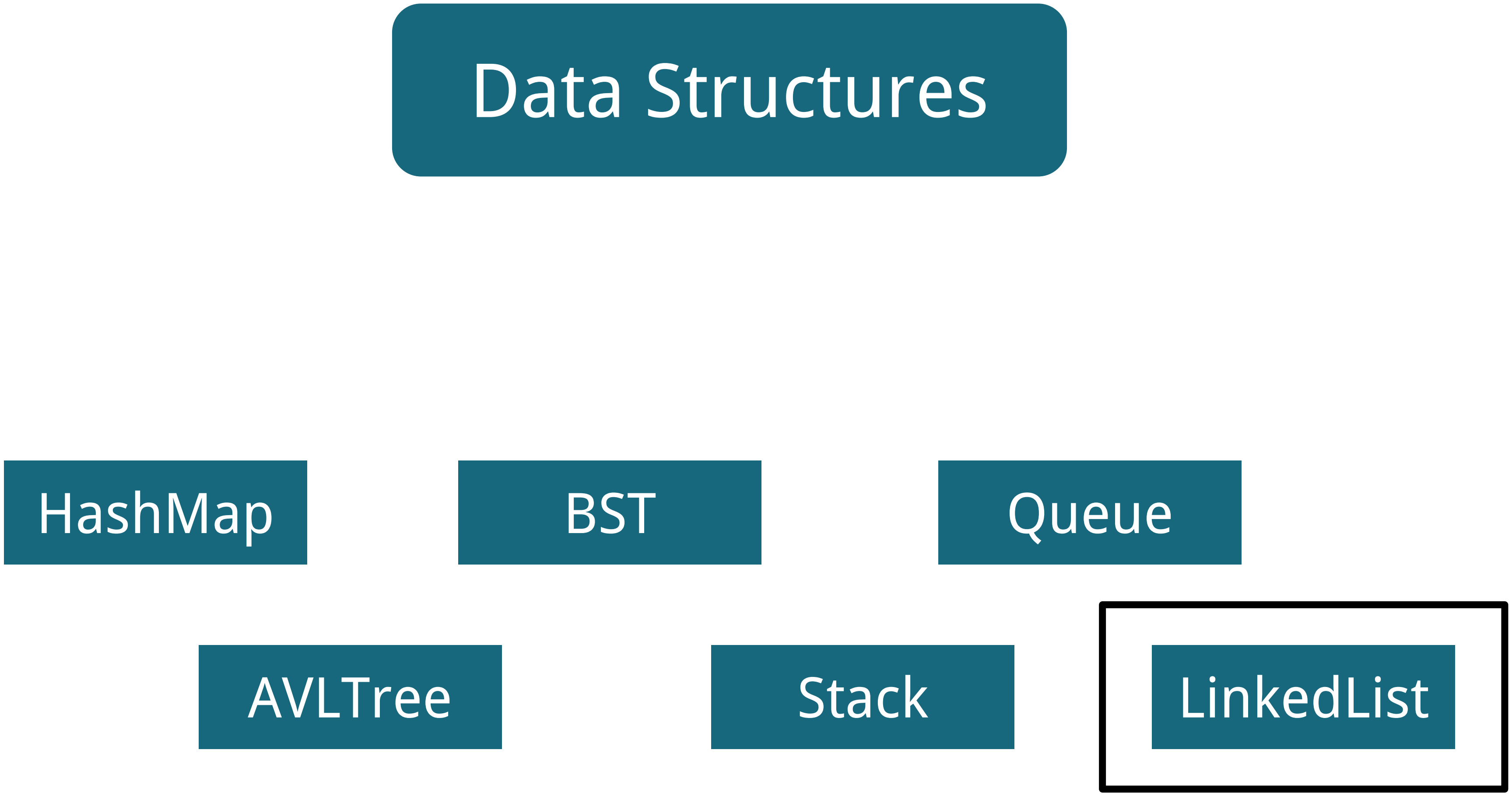
Features

Application

```
public class Kitchen extends Queue<Order>
{
    public void addOrder(Order order) //Enqueue
    {...}

    public Order removeOrder() //Dequeue
    {...}

    ...
}
```



Data Structures

HashMap

BST

Queue

AVLTree

Stack

LinkedList

LinkedList

Implementation

Features

Application

DoublyLinkedList.java

Implementation

Features

Application

- Problem: When a meal is ready to pick up, its customer may not be around. The customer may show up 15 mins late. In a Singly Linked Linear List, it is not possible to reach the previous node.

- Solution: By using a Doubly Linked List, we are able to go to any node we want to.

Implementation

Features

Application

- Through a Doubly Linked List, customers can check real-time information on a digital board and pick up their orders once they show up.
- After an order has been picked up, server can remove the order from the data structure by simply clicking it on the user interface.

PickUpList Class

Implementation

Features

Application

```
public class PickUpList extends
DoublyLinkedList<Order>
{
    //Adds order to pick up list
    public void addOrder(Order order) {...}

    //Removes order from pick up list
    public Order removeOrder(int position) {...}

    //Attaches change listener
    public void attach(ChangeListener c) {...}

    //Updates change listener
    public void update() {...}

    ...
}
```

Thank You