COMP 2080

Assignment 2

Due Date: 9th April 2021

First Name: Aleksandr Last Name: Kudin Student ID: 101258693	First Name: Matthew Last Name: Campbell Student ID: 101289518	First Name: Michael Last Name: Sirna Student ID: 101278670

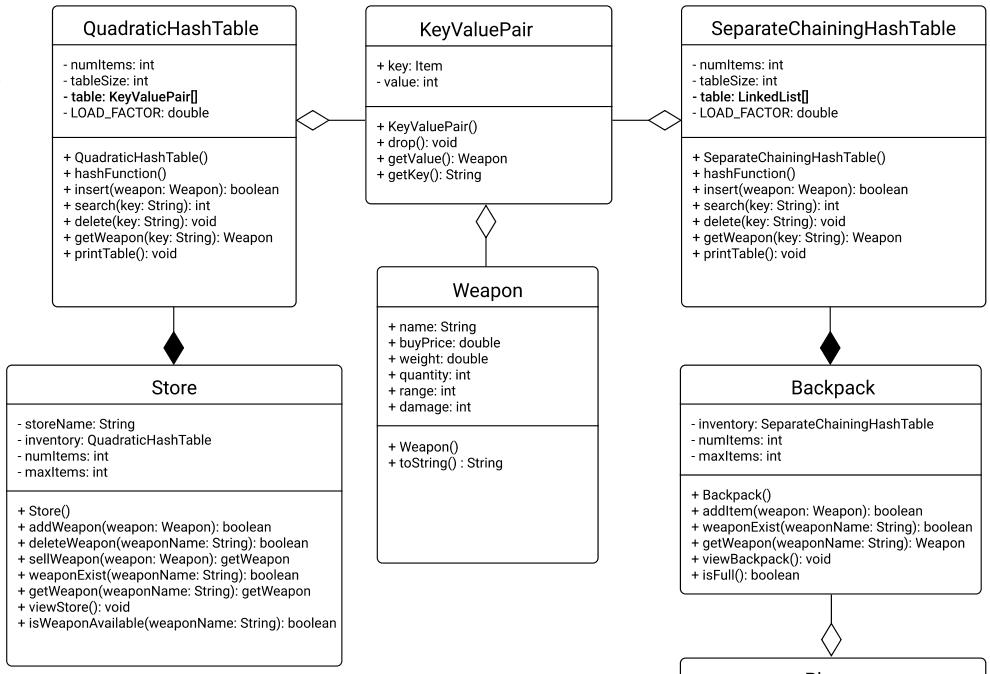
	First Name: Stephen Last Name: Davis Student ID: 101294116	

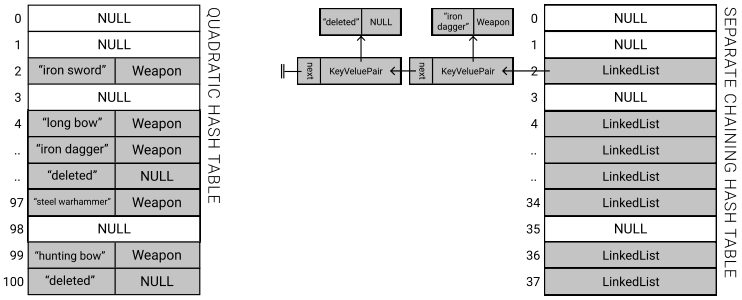
Winter 2021 Final Group Assignment – UML Classes Diagram / Hash Tables Visualization

GEORGE BROWN COLLEGE

COURSE: COMP 2080

GROUP OF: Aleksandr Kudin, Matthew Campbell, Michael Sirna, Stephen Davis





Paste your code for each class after this point:

1 Weapon.java

2 KeyValuePair.java

3 QuadraticHashTable.java

4 Store.java

5 Node.java

6 LinkedList.java

7 SeparateChainingHashTable.java

8 Backpack.java

9 Player.java

10 Validation.java

11 Program.java

1 Weapon.java

```
// Aleksandr Kudin, 101258693
// Matthew Campbell, 101289518
// Michael Sirna, 101278670
// Stephen Davis, 101294116
public class Weapon {
    public String name;
    public double buyPrice;
    public double weight;
    public int quantity;
    public int range;
    public int damage;

    public Weapon(String name, double weight, double buyPrice, int range, int damage) {
        this.name = name;
        this.buyPrice = buyPrice;
        this.weight = weight;
        this.range = range;
        this.damage = damage;
        this.quantity = 0;
    }

    public Weapon(String name, double weight, double buyPrice, int range, int damage, int
quantity) {
        this.name = name;
        this.buyPrice = buyPrice;
        this.weight = weight;
        this.range = range;
        this.damage = damage;
        this.quantity = quantity;
    }

    @Override
    public String toString() {
        return String.format("%-35s %-8.2f %.2f", name + " (" + quantity + ")", buyPrice,
weight);
    }
}
```

2 KeyValuePair.java

```java
// Aleksandr Kudin, 101258693
// Matthew Campbell, 101289518
// Michael Sirna, 101278670
// Stephen Davis, 101294116
public class KeyValuePair {
    public String key;
    private Weapon value;

    public KeyValuePair(String key, Weapon weapon) {
        this.key = key;
        this.value = weapon;
    }

    // Drop Function. Implements when the Weapon is being deleted from the hash table.
    public void drop(){
        key = null;
        value = null;
    }

    public Weapon getValue(){ return value; }

    public String getKey() {
        if (key != null){
            return key;
        }
        return "DELETED"; // If the Weapon is deleted.
    }
}
```

3 QuadraticHashTable.java

```java
// Aleksandr Kudin, 101258693
// Matthew Campbell, 101289518
// Michael Sirna, 101278670
// Stephen Davis, 101294116
public class QuadraticHashTable {
    private int numItems;
    private int tableSize;
    private KeyValuePair[] table;
    private final double LOAD_FACTOR = 0.80;

    // Constructor. Table size is hardcoded.
    public QuadraticHashTable(){
        numItems = 0;
        tableSize = 101; // 101 is a prime number suitable for the hash table.
        table = new KeyValuePair[101];
    }

    // Hash Method. Returns the hash value of the Weapon key (weapon name).
    private int hashFunction(String key){
        key = key.toLowerCase();
        int value = 0, weight = 1;
        for (int x = 0; x < key.length(); x++){
```

```
            value += (key.charAt(x) -'a'+ 1) * weight;
            weight++;
        }
        if (value < 0) { value *= -1; } // If the value is negative, make it positive.
        return value % tableSize;
    }

    // Insert Method. Insert Weapon object in the location based on the value of the hash
method.
    public boolean insert (Weapon weapon){
        if ((double)numItems/tableSize < LOAD_FACTOR){
            int count = 1;
            int loc = hashFunction(weapon.name);
            // Loop while it's not NULL AND the Key Value Pair value is not deleted.
            while (table[loc] != null && table[loc].key != null){
                loc = (loc + count * count) % tableSize; // Quadratic probing.
                count++;
            }
            table[loc] = new KeyValuePair(weapon.name, weapon);
            numItems++;
            return true;
        }
        return false; // Load Factor Error.
    }

    // Search Method. Search for a Weapon object in the hash table by its key (weapon
name) and returns its location.
    public int search(String key){
        int loc = hashFunction(key);
        int count = 1;
        // Loop while it's not NULL AND the key value in the Key Value Pair does not
match.
        while (table[loc] != null && !table[loc].getKey().equalsIgnoreCase(key)){
            loc = (loc + count * count) % tableSize; // Quadratic probing.
            count++;
        }
        if (table[loc] == null){ // Weapon not found.
            return -1;
        }
        return loc;
    }

    // Delete Method. Search for a Weapon object in the hash table by its key (weapon
name) and reset the Key Value Pair object.
    public void delete(String key){
        int loc = search(key);
        if (loc == -1){ return; } // Weapon not found.
        table[loc].drop(); // Reset the Key Value Pair object with the Weapon object in
it.
        numItems--;
    }

    // Get Method. Search for a Weapon object in the hash table by its key (weapon name)
and returns the object if found.
    public Weapon getWeapon(String key){
        int loc = search(key);
        if (loc != -1){ // Weapon is found.
            return table[search(key)].getValue(); // Return the Weapon object.
        }
        return null; // Weapon not found.
    }

    // Print Method. Check for existing in the hash table Weapon objects and print their
```

```
value.
    public void printTable(){
        for (int x = 0; x < tableSize; x++){
            if (table[x] != null && table[x].getValue() != null){
                System.out.println(table[x].getValue().toString());
            }
        }
    }
}
```

4 Store.java

```
// Aleksandr Kudin, 101258693
// Matthew Campbell, 101289518
// Michael Sirna, 101278670
// Stephen Davis, 101294116
public class Store {
    private String storeName;
    private QuadraticHashTable inventory;
    private int numItems;
    private int maxItems;

    // Constructor. Set the store name and inventory limit.
    public Store(String storeName, int maxSlots){
        this.storeName = storeName;
        this.inventory = new QuadraticHashTable();
        this.numItems = 0;
        this.maxItems = maxSlots;
    }

    // Add Method. Add NEW TYPE of Weapon in the inventory.
    public boolean addWeapon(Weapon weapon){
        if (numItems != maxItems){ // Check for the items limit.
            numItems++; // Increment the number of TYPES of weapon in the inventory.
            return inventory.insert(weapon);
        }
        return false; // Maximum items limit is hit.
    }

    // Delete Method. Delete the selected TYPE of Weapon in the inventory.
    public boolean deleteWeapon(String weaponName) {
        if (weaponExist(weaponName)){
            numItems--; // Decrement the number of TYPES of weapon in the inventory.
            inventory.delete(weaponName); // Calls the delete method in the hash table by
passing the item key (weapon name);
            return true;
        }
        return false; // Weapon not found.
    }

    // Sell Method. Decrements the quantity of the Weapon TYPE and returns a singe copy
of that Weapon.
    public Weapon sellWeapon(Weapon weapon){
        Weapon soldItem;
        weapon.quantity--; // Decrement the quantity of the Weapons in store storage.
        // Create an instance of the weapon with single quantity.
        soldItem = new Weapon(weapon.name, weapon.weight, weapon.buyPrice, weapon.range,
```

```
weapon.damage, 1);
        return soldItem; // Return the singe instance copy.
    }

    // Exist Method. Check whether the Weapon is exist in the store.
    public boolean weaponExist(String weaponName){
        int loc = inventory.search(weaponName);
        return loc != -1;
    }

    // Get Method. Return the Weapon object from the store by its name.
    public Weapon getWeapon(String weaponName){
        return inventory.getWeapon(weaponName);
    }

    // View Store Method. Format and print the store information and all the items in the
inventory.
    public void viewStore(){
        System.out.println("\n=====================================================");
        System.out.printf("%-38s %s %d/%d\n", storeName.toUpperCase(), "ITEMS:",
numItems, maxItems);
        System.out.println("=====================================================");
        System.out.printf("%-35s %-8s %-8s\n", "NAME", "PRICE", "WEIGHT");
        inventory.printTable();
        System.out.println("=====================================================\n");
    }

    // The method checks whether the Weapon is exist and in stock. (For the Game Logic.)
    public boolean isWeaponAvailable(String weaponName) {
        if (weaponExist(weaponName)){
            return inventory.getWeapon(weaponName).quantity > 0; // Whether the quantity
in stock is more than zero.
        }
        return false; // Weapon not found.
    }
}
```

5 Node.java

```
// Aleksandr Kudin, 101258693
// Matthew Campbell, 101289518
// Michael Sirna, 101278670
// Stephen Davis, 101294116
public class Node {
    public KeyValuePair data;
    public Node next;

    public Node(KeyValuePair keyValuePair) {
        this.data = keyValuePair;
        this.next = null;
    }
}
```

6 LinkedList.java

```java
// Aleksandr Kudin, 101258693
// Matthew Campbell, 101289518
// Michael Sirna, 101278670
// Stephen Davis, 101294116
public class LinkedList {
    private Node head;

    public LinkedList() {
        this.head = null;
    }

    public void addFront(KeyValuePair keyValuePair){
        Node newNode = new Node(keyValuePair);
        if (head == null){ // Check whether the Linked List is empty.
            head = newNode; // Create first node.
            return;
        }
        newNode.next = head;  // Insert current head of the Linked List in the new node.
        head = newNode; // Insert the new node in the front of Lhe linked List.
    }

    // Exist Function. Checks whether the item exist in the current Link List.
    public boolean itemExist(String key){
        Node curr = head;
        while (curr != null){ // Checking each node in the list.
            if (curr.data.key.equalsIgnoreCase(key)){ // Match the key.
                return true;
            }
            curr = curr.next; // Go to the next node.
        }
        return false;
    }

    // Get Function. Returns the node data value (Weapon) by the KeyValuePair key (Weapon
name).
    public Weapon getItem(String key){
        Node curr = head;
        while (curr != null){ // Checking each node in the list.
            if (curr.data.key.equalsIgnoreCase(key)){ // Match the key.
                return curr.data.getValue(); // Return the Weapon object.
            }
            curr = curr.next; // Go to the next node.
        }
        return null; // The weapon not found.
    }

    // Print Function. Prints all nodes and their data (KeyValuePairs).
    public void printList(){
        Node curr = head; // Start from the head (the first) node.
        while (curr != null){ // Go through each node in the list.
            System.out.println(curr.data.getValue().toString()); // Print KeyValuePair
values information.
            curr = curr.next; // Go to the next node.
        }
    }
}
```

7 SeparateChainingHashTable.java

```java
// Aleksandr Kudin, 101258693
// Matthew Campbell, 101289518
// Michael Sirna, 101278670
// Stephen Davis, 101294116
public class SeparateChainingHashTable {
    private int numItems;
    private int tableSize;
    private LinkedList[] table;
    private final double LOAD_FACTOR = 0.80;

    // Constructor. Takes maximum number of items needed and calculates the table size.
    public SeparateChainingHashTable(int maxItems){
        numItems = 0;
        tableSize = (int)Math.ceil(maxItems/LOAD_FACTOR); // Multiply the maximum number
of items needed by the load factor.
        table = new LinkedList[tableSize];
    }

    // Hash Method. Returns the hash value of the weapon key (weapon name).
    private int hashFunction(String key){
        key = key.toLowerCase();
        int value = 0, weight = 1;
        for (int x = 0; x < key.length(); x++){
            value += (key.charAt(x) -'a'+ 1) * weight;
            weight++;
        }
        if (value < 0) { value *= -1; } // If the value is negative, make it positive.
        return value % tableSize;
    }

    // Insert Method. Insert Weapon object in the location based on the value of the hash
method.
    public void insert (Weapon weapon){
        if ((double)numItems/tableSize < LOAD_FACTOR){
            int loc = hashFunction(weapon.name);
            if (table[loc] == null){ // Check if Linked List object is created.
                table[loc] = new LinkedList(); // Create new Linked ist object.
                numItems++; // Increment the number of items in the hash table.
            }
            // Add the Weapon object in the front of the Linked List object.
            table[loc].addFront(new KeyValuePair(weapon.name, weapon));
        }
    }

    // Search Method. Search for a Weapon object in the hash table by its key (weapon
name) and returns its location.
    public int search(String key){
        int loc = hashFunction(key);
        if (table[loc] == null){ // Check if the Linked List object exist.
            return -1; // Linked List object not exist. Weapon cannot be found.
        }
        if (!table[loc].itemExist(key)){ // Check if the Weapon object exist in the
Linked List object.
            return -1; // Weapon object does not exist.
        }
        return loc;
    }
```

```
    // Get Method. Search for a Weapon object in the hash table by its key (weapon name)
and returns the object if found.
    public Weapon getWeapon(String key){
        int loc = search(key);
        if (loc != -1){ // Weapon found.
            return table[loc].getItem(key); // Return the Weapon object.
        }
        return null; // Weapon not found.
    }

    // Print Method. Calls the printList method of each Linked List object in the hash
table.
    public void printTable(){
        for (int x = 0; x < tableSize; x++){
            if (table[x] != null){
                table[x].printList();
            }
        }
    }
}
```

8 Backpack.java

```
// Aleksandr Kudin, 101258693
// Matthew Campbell, 101289518
// Michael Sirna, 101278670
// Stephen Davis, 101294116
public class Backpack {
    private SeparateChainingHashTable inventory;
    private int numItems;
    private int maxItems;

    // Constructor.
    public Backpack(int maxItems) {
        this.inventory = new SeparateChainingHashTable(maxItems);
        this.numItems = 0;
        this.maxItems = maxItems;
    }

    // Add Method.
    public boolean addItem(Weapon weapon){
        if (isFull()){ // Check if the number of items hit the maximum limit.
            System.out.println("The backpack is full!");
            return false; // Backpack is full.
        }
        if (weaponExist(weapon.name)) { // Check if the Weapon already exist.
            getWeapon(weapon.name).quantity += weapon.quantity; // Increment the Weapon
quantity.
            numItems += weapon.quantity; // Increment the number of current number of
items in the backpack.
            return true;
        }
        inventory.insert(weapon); // Insert the new type of Weapon in the Backpack (in
the HashTable).
        numItems += weapon.quantity; // Increment the number of current number of items
in the backpack.
        return true;
    }
```

```
    // Weapon Exist Method. Check whether the Weapon object exists in the backpack
hashtable or not.
    public boolean weaponExist(String weaponName){
        int loc = inventory.search(weaponName);
        return loc != -1;
    }

    // Get Method. Returns the Weapon object from the backpack hash table.
    public Weapon getWeapon(String weaponName){
        return inventory.getWeapon(weaponName);
    }

    // View Backpack Method. Print all the information about the back pack and the list
of weapons in it.
    public void viewBackpack(){
        System.out.println("\n=====================================================");
        System.out.printf("%-38s %s %d/%d\n", "My Backpack".toUpperCase(), "ITEMS:",
numItems, maxItems);
        System.out.println("=====================================================");
        if (numItems == 0) {
            System.out.println("Your backpack is empty..");
        }else{
            System.out.printf("%-35s %-8s %-8s\n", "NAME", "PRICE", "WEIGHT");
        }
        inventory.printTable();
        System.out.println("=====================================================");
    }

    // The method check whether the backpack hit the maximum items limit.
    public boolean isFull() {
        return numItems == maxItems;
    }
}
```

9 Player.java

```
// Aleksandr Kudin, 101258693
// Matthew Campbell, 101289518
// Michael Sirna, 101278670
// Stephen Davis, 101294116
public class Player {
    private String name;
    private Backpack backpack;
    private double coins;
    private double currWeight;
    private double maxWeight;

    // Constructor.
    public Player(String name, int maxItems) {
        this.name = name;
        this.coins = 45;
        this.backpack = new Backpack(maxItems);
        this.currWeight = 0;
        this.maxWeight = 90;
    }
```

```
    // View Player Method. Display all the information about the player (Amount of money
and the weight carried) and the backpack,
    public void viewPlayer() {
        System.out.println("\n=================================================");
        System.out.printf("%s %-10s %s %.2f   %s %.1f/%.1f\n", "PLAYER:",
name.toUpperCase(), "MONEY:", coins, "WEIGHT:", currWeight, maxWeight);
        System.out.println("=================================================");
        viewBackpack();
    }

    public void viewBackpack() {
        backpack.viewBackpack();
    }

    // Can Afford Method. The method checks whether the player has meet all the
requirement in order to get an item from the store.
    public boolean canAfford(Weapon weaponToBuy) {
        if (coins - weaponToBuy.buyPrice < 0){
            System.out.println("You don't have enough coins.");
            return false; // Not enough money.
        }
        if (currWeight + weaponToBuy.weight > maxWeight){
            System.out.println("You don't have enough weight to carry.");
            return false; // Too much weight.
        }
        if (backpack.isFull()) {
            System.out.println("You don't have enough space in your backpack.");
            return false; // No space in the backpack.
        }
        return true; // All good! Can afford!
    }

    // Buy Item Method. Implements the Buy Weapon Logic.
    public void buyItem(Weapon weaponToBuy) {
        coins -= weaponToBuy.buyPrice; // Removes amount of money the new Weapon cost.
        currWeight += weaponToBuy.weight; // Adds amount of weight the new Weapon weight.
        backpack.addItem(weaponToBuy); // Finally, add the new item into the back pack.
    }
}
```

10 Validation.java

```
import java.util.Scanner;
// Aleksandr Kudin, 101258693
// Matthew Campbell, 101289518
// Michael Sirna, 101278670
// Stephen Davis, 101294116
public class Validation {
    public static int getInteger(String message){
        Scanner sc = new Scanner(System.in);
        System.out.print(message);
        while (!sc.hasNextInt())
        {
            sc.nextLine(); //clear the invalid input ...
            System.out.print(message);
        }
        return sc.nextInt();
```

```
    }

    public static String getString(String message){
        Scanner sc = new Scanner(System.in);
        String string;
        System.out.print(message);
        string = sc.nextLine();
        return string;
    }

    public static double getDouble(String message){
        Scanner sc = new Scanner(System.in);
        System.out.print(message);
        while (!sc.hasNextDouble())
        {
            sc.nextLine(); //clear the invalid input ...
            System.out.print(message);
        }
        return sc.nextDouble();
    }

    public static int getValidChoice(int max, String menu)
    {
        int choice;
        System.out.println(menu);
        choice = Validation.getInteger("Menu Option: ");
        while (choice < 1 || choice > max)
        {
            System.out.println(menu);
            choice = Validation.getInteger("Menu Option: ");
        }
        return choice;
    }
}
```

11 Program.java

```
// Aleksandr Kudin, 101258693
// Matthew Campbell, 101289518
// Michael Sirna, 101278670
// Stephen Davis, 101294116
public class Program {

    static Store blacksmithStore;
    static Player bravePlayer;

    public static void viewPlayer() {
        bravePlayer.viewPlayer();
        Validation.getString("\nPress ENTER to continue return to the main menu.");
    }

    public static void viewBackpack() {
        bravePlayer.viewBackpack();
        Validation.getString("\nPress ENTER to continue return to the main menu.");
    }

    public static void addStoreItem() {
        String iname;
```

```
        double iweight, icost;
        int irange, idamage;
        blacksmithStore.viewStore();
        System.out.println("-----------Add Item----------");
        iname = Validation.getString("Please enter the item's name(0 to exit): ");
        if (iname.equalsIgnoreCase("0")){
            System.out.println("Return to the menu..");
            return;
        }
        if (blacksmithStore.weaponExist(iname)){
            System.out.println("The item is already exist..");
            addItemInStock(iname);
            return;
        }
        iweight = Validation.getDouble("Please enter the item's weight: ");
        icost = Validation.getDouble("Please enter the item's cost: ");
        irange = Validation.getInteger("Please enter the item's range: ");
        idamage = Validation.getInteger("Please enter the item's damage: ");
        if (blacksmithStore.addWeapon(new Weapon(iname, iweight, icost, irange,
idamage))){
            System.out.println("Item successfully added..");
            addItemInStock(iname);
        }
        else{
            System.out.println("Item was not added..");
        }
    }

    public static void addItemInStock(String weaponName){
        int iquantity;
        iquantity = Validation.getInteger("Please enter the item's quantity: ");
        blacksmithStore.getWeapon(weaponName).quantity =
blacksmithStore.getWeapon(weaponName).quantity + iquantity;
        System.out.println(iquantity + " " + weaponName.toUpperCase() + " have/has been
added in stock.");
        Validation.getString("\nPress ENTER to continue return to the main menu.");
    }

    public static void deleteStoreItem() {
        String itemName;
        blacksmithStore.viewStore();
        itemName = Validation.getString("Please enter an item's name to delete(0 to
exit): ");
        if (itemName.equalsIgnoreCase("0")){
            System.out.println("Return to the menu..");
            return;
        }
        if (blacksmithStore.deleteWeapon(itemName)) {
            System.out.println("Item " + itemName.toUpperCase() + " deleted..");
        }
        else {
            System.out.println("Item " + itemName.toUpperCase() + " was not found..");
        }
        Validation.getString("\nPress ENTER to continue return to the main menu.");
    }

    // Business Logic of buying weapon.
    public static void buyItem() {
        String itemName;
        Weapon soldWeapon;
        bravePlayer.viewPlayer();
        blacksmithStore.viewStore();
        System.out.println("-----------Buy Weapon----------");
```

```
        itemName = Validation.getString("Please enter a weapon name to buy(0 to exit):
");
        if (itemName.equalsIgnoreCase("0")){
            System.out.println("Return to the main menu..");
            return;
        }
        if (!blacksmithStore.isWeaponAvailable(itemName)){
            System.out.println("The weapon is not available in the store...");
            return;
        }
        if (!bravePlayer.canAfford(blacksmithStore.getWeapon(itemName))){
            System.out.println("The item cannot be added in your backpack...");
            return;
        }
        soldWeapon = blacksmithStore.sellWeapon(blacksmithStore.getWeapon(itemName));
        bravePlayer.buyItem(soldWeapon);
        System.out.println("Thank you for purchasing " + itemName.toUpperCase());
        System.out.println(itemName.toUpperCase() + " has been added in your backpack");
        bravePlayer.viewBackpack();
        Validation.getString("\nPress ENTER to continue return to the main menu.");
    }

    public static String storeManageMenu() {
        String s = "\nPlease select a choice from the menu below:\n";
        s += "1: Add Store Item \n";
        s += "2: Delete Item \n";
        s += "3: Return to the main menu.";
        return s;
    }

    public static String mainMenu() {
        String s = "\nPlease select a choice from the menu below:\n";
        s += "1: Buy Item \n";
        s += "2: View Player \n";
        s += "3: View Backpack \n";
        s += "4: Store Manage Options \n";
        s += "5: Exit";
        return s;
    }

    public static void runStoreManageMenu() {
        String menu = storeManageMenu();
        blacksmithStore.viewStore();
        int choice = Validation.getValidChoice(3, menu);
        while (choice != 3)
        {
            if (choice == 1) { addStoreItem(); }
            if (choice == 2) { deleteStoreItem(); }
            blacksmithStore.viewStore();
            choice = Validation.getValidChoice(3, menu);
        }
    }

    public static void runProgram() {
        String menu = mainMenu();
        int choice = Validation.getValidChoice(5, menu);
        while (choice != 5)
        {
            if (choice == 1) { buyItem(); }
            if (choice == 2) { viewPlayer(); }
            if (choice == 3) { viewBackpack(); }
            if (choice == 4) { runStoreManageMenu(); }
```

```
            choice = Validation.getValidChoice(5, menu);
        }
    }

    public static void createPlayer(){
        String playerName = Validation.getString("Enter the player name: ");
        bravePlayer = new Player(playerName, 30);
    }

    public static void main(String[] args) {
        blacksmithStore = new Store("Blacksmith's Store", 80);
        blacksmithStore.addWeapon(new Weapon("Iron Sword", 11, 10, 1, 7, 4));
        blacksmithStore.addWeapon(new Weapon("Steel Sword", 15, 15, 1, 8, 1));
        blacksmithStore.addWeapon(new Weapon("Orcish Sword", 11, 10, 1, 9));
        blacksmithStore.addWeapon(new Weapon("Iron War Axe", 11, 10, 1, 8, 1));
        blacksmithStore.addWeapon(new Weapon("Glass War Axe", 13, 50, 1, 16, 2));
        blacksmithStore.addWeapon(new Weapon("Steel War Axe", 11, 15, 1, 9, 2));
        blacksmithStore.addWeapon(new Weapon("Iron Dagger", 2, 4, 1, 4, 5));
        blacksmithStore.addWeapon(new Weapon("Steel Dagger", 2.5, 5, 1, 5));
        blacksmithStore.addWeapon(new Weapon("Steel Battleaxe", 21, 30, 1, 18));
        blacksmithStore.addWeapon(new Weapon("Steel Warhammer", 25, 20, 1, 20, 2));
        blacksmithStore.addWeapon(new Weapon("Wizard Necronomicon", 2.5, 35, 1, 12, 1));
        blacksmithStore.addWeapon(new Weapon("Hunting Bow", 7, 25, 8, 7, 6));
        blacksmithStore.addWeapon(new Weapon("Long Bow", 5, 10, 7, 6));
        createPlayer();
        bravePlayer.viewPlayer();
        runProgram();
    }
}
```