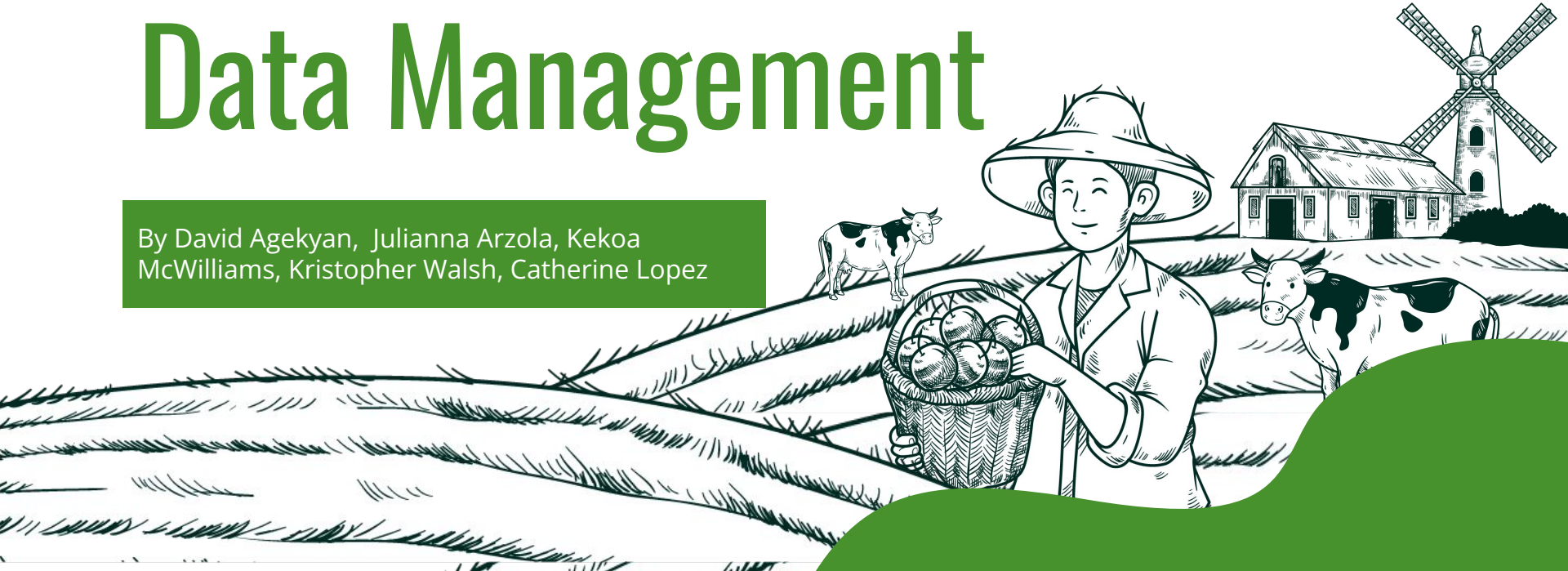
SRC Project

Data Management

By David Agekyan, Julianna Arzola, Kekoa McWilliams, Kristopher Walsh, Catherine Lopez



Significance of Systems programming in Farms

- Problem: Resource management
- Solution: Data management tool that informs farmers which resources are most necessary to purchase
- Goal: Promote operational efficiency by alleviating stress of meticulous resource management

Our Code - Data Structures

```
import java.util.ArrayList;
import java.util.Comparator;
import java.util.HashMap;
import java.util.InputMismatchException;
import java.util.List;
import java.util.Map;
import java.util.Scanner;
```

Imported classes and interfaces:

- ArrayList (class)
- Hashmap (class)
- List (interface)
- LinkedLists (class)

```java
public class FarmMainClass {
    public ResizableArrayBag<String> resourceBag;
    public HashMap<String, Double> resources; //resource name, prices
    public HashMap<String, Double> resourceQuantityCurrent; //resource name, current quantity
    public HashMap<String, Double> resourceQuantityNeeded; //resource name, quantity needed
    public HashMap<String, Double> resourceTotalPrice; //resource name, total price for quantity needed
    public HashMap<String, Double> resourceImportance; //resource name, importance
    public LinkedList<String> resourcePriorityList; //list of resources in order of importance
    public int budget; //total budget
public FarmMainClass() {
    this.resources = new HashMap<String, Double>();
    this.resourceQuantityCurrent = new HashMap<String, Double>();
    this.resourceQuantityNeeded = new HashMap<String, Double>();
    this.resourceTotalPrice = new HashMap<String, Double>();
    this.resourceImportance = new HashMap<String, Double>();
    this.resourcePriorityList = new LinkedList<String>();
    this.resourceBag = new ResizableArrayBag<String>();
    this.budget = 0;
}
```

- Defines a class FarmMainClass for managing resources on a farm.
- Includes various data structures for handling resources
- Initializes the class with default values in the constructor

```java
public void addResource(String resourceName, double price, double currentQuantity, double quantityNeeded) {
    resources.put(resourceName, price);
    resourceQuantityCurrent.put(resourceName, currentQuantity);
    resourceQuantityNeeded.put(resourceName, quantityNeeded);
    calculateTotalPrice(resourceName);
    calculateImportance(budget);
    resourceBag.add(resourceName);
}

/**
 * Remove a resource from the resource bag
 * @param resourceName
 */
public void removeResource(String resourceName) {
    if(!resourceBag.contains(resourceName)){
        System.out.println("Resource doesn't exist.");
        return;
    }
    resources.remove(resourceName);
    resourceQuantityCurrent.remove(resourceName);
    resourceQuantityNeeded.remove(resourceName);
    resourceTotalPrice.remove(resourceName);
    resourceImportance.remove(resourceName);
    resourceBag.remove(resourceName);
}
```

```java
public boolean checkAdd(String resourceName) {
    for (String resource : resourceBag.toArray(new String[0])) {
        if (resource.equalsIgnoreCase(resourceName)) {
            System.out.println("Resource already exists.");
            return false;
        }
    }   return true;
}
```

Methods:

- checkAdd
- addResource
- removeResource

```java
public boolean checkEdit(String resourceName) {
    for (String resource : resourceBag.toArray(new String[0])) {
        if (!resource.equalsIgnoreCase(resourceName)) {
            System.out.println("Resource doesn't exist.");
            return false;
        }
    }   return true;
}
```

Methods:

- checkEdit
- editResource

```java
public void editResource(String resourceName, double price, double currentQuantity, double quantityNeeded) {
    resources.put(resourceName, price);
    resourceQuantityCurrent.put(resourceName, currentQuantity);
    resourceQuantityNeeded.put(resourceName, quantityNeeded);
    calculateTotalPrice(resourceName);
    calculateImportance(budget);
}
```

```java
public void calculateTotalPrice(String resourceName) {
    double price = resources.get(resourceName);
    double quantityNeeded;
    double quantityNeedCheck = resourceQuantityNeeded.get(resourceName) - resourceQuantityCurrent.get(resourceName);
    if (quantityNeedCheck < 0) {
        quantityNeeded = 0;
    }   else {
        quantityNeeded = quantityNeedCheck;
    }
    double totalPrice = price * quantityNeeded;
    resourceTotalPrice.put(resourceName, totalPrice);
}
```

Method

- calculateTotalPrice

```java
public void calculateImportance(double budget) {
    // Calculate the total purchase cost for each resource
    Map<String, Double> totalPurchaseCost = new HashMap<>();
    for (Map.Entry<String, Double> entry : resourceQuantityNeeded.entrySet()) {
        String resourceName = entry.getKey();
        double quantityNeeded = entry.getValue();
        double quantityCurrent = resourceQuantityCurrent.get(resourceName);
        double price = resources.get(resourceName);
        double purchaseQuantity = Math.max(0, quantityNeeded - quantityCurrent);
        totalPurchaseCost.put(resourceName, purchaseQuantity * price);
    }

    // Sort resources based on total purchase cost in descending order
    List<Map.Entry<String, Double>> sortedResources = new ArrayList<>(totalPurchaseCost.entrySet());
    sortedResources.sort(Map.Entry.comparingByValue(Comparator.reverseOrder()));

    double remainingBudget = budget;
    double totalCostWithinBudget = 0;
    for (Map.Entry<String, Double> entry : sortedResources) {
        double totalCost = entry.getValue();
        if (totalCost <= remainingBudget) {
            totalCostWithinBudget += totalCost;
            remainingBudget -= totalCost;
        }
    }
```

```java
    for (Map.Entry<String, Double> entry : sortedResources) {
        String resourceName = entry.getKey();
        double totalCost = entry.getValue();

        double importance;
        if (totalCost <= budget) {
            importance = (totalCost / totalCostWithinBudget) * 100;
        } else {
            importance = (remainingBudget / totalCost) * 100;
            remainingBudget = 0;
        }

        resourceImportance.put(resourceName, importance);
    }
```

Method:

- calculateImportance (double budget)

```java
public void printResources() {
    for (String resourceName : resources.keySet()) {
        System.out.println("Resource Name: " + resourceName +
         ", Price: " + resources.get(resourceName) + ", Current Quantity: "
         + resourceQuantityCurrent.get(resourceName) + ", Quantity Needed: "
         + resourceQuantityNeeded.get(resourceName));
    }
}
public void printResourceTotalPrice() {
    for (String resourceName : resourceTotalPrice.keySet()) {
        System.out.println("Resource Name: " + resourceName + ", Total Purchase Price: "
        + resourceTotalPrice.get(resourceName));
    }
}
```

```java
public void updateResourcePriorityList() {
```

```java
public void printResourcePriorityList() {

    for (String resourceName : resourcePriorityList) {
        double quantityNeeded;
        double quantityNeedCheck = resourceQuantityNeeded.get(resourceName) -
        resourceQuantityCurrent.get(resourceName);
        if (quantityNeedCheck < 0) {
            quantityNeeded = 0;
        }   else {
            quantityNeeded = quantityNeedCheck;
        }
        if(resourceImportance.get(resourceName)>=50){
            System.out.println("Resource Name: " + resourceName + ", Buying is suggested.");
        }
        else{
            System.out.println("Resource Name: " + resourceName + ", Buying is not suggested.");
        }
        System.out.println("Resource Name: " + resourceName + ", Units Needed: " + (quantityNeeded) +
        ", Price per Unit: " + resources.get(resourceName) + ", Total Price: " +
        resourceTotalPrice.get(resourceName));
    }
}
```

Methods

- updateResourcePriorityList()
- printResources()
- printResourceTotalPrice()
- printResourcePriorityList()

Improvements

- Allow users to categorize budget and resources
- Include price analysis in importance method
- Implementation of user interface (UI)

THANK YOU!

Any questions?