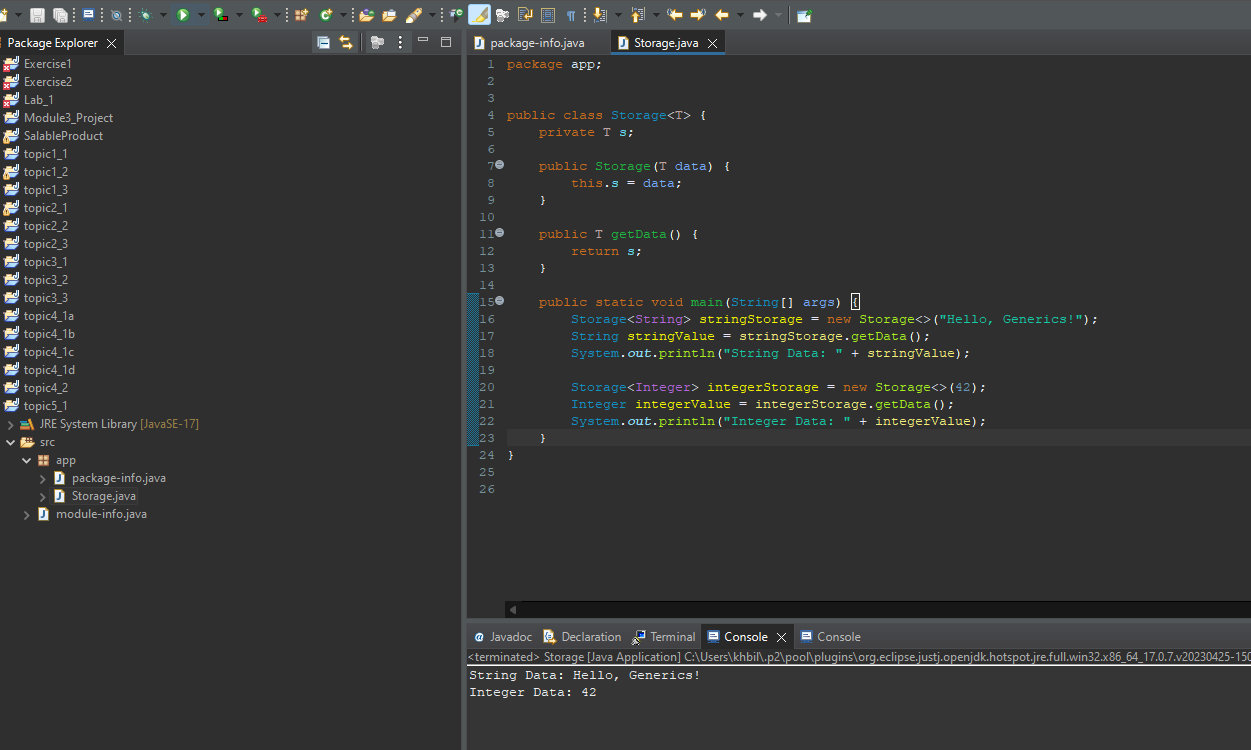
Topic 5

KariAnn Harjo

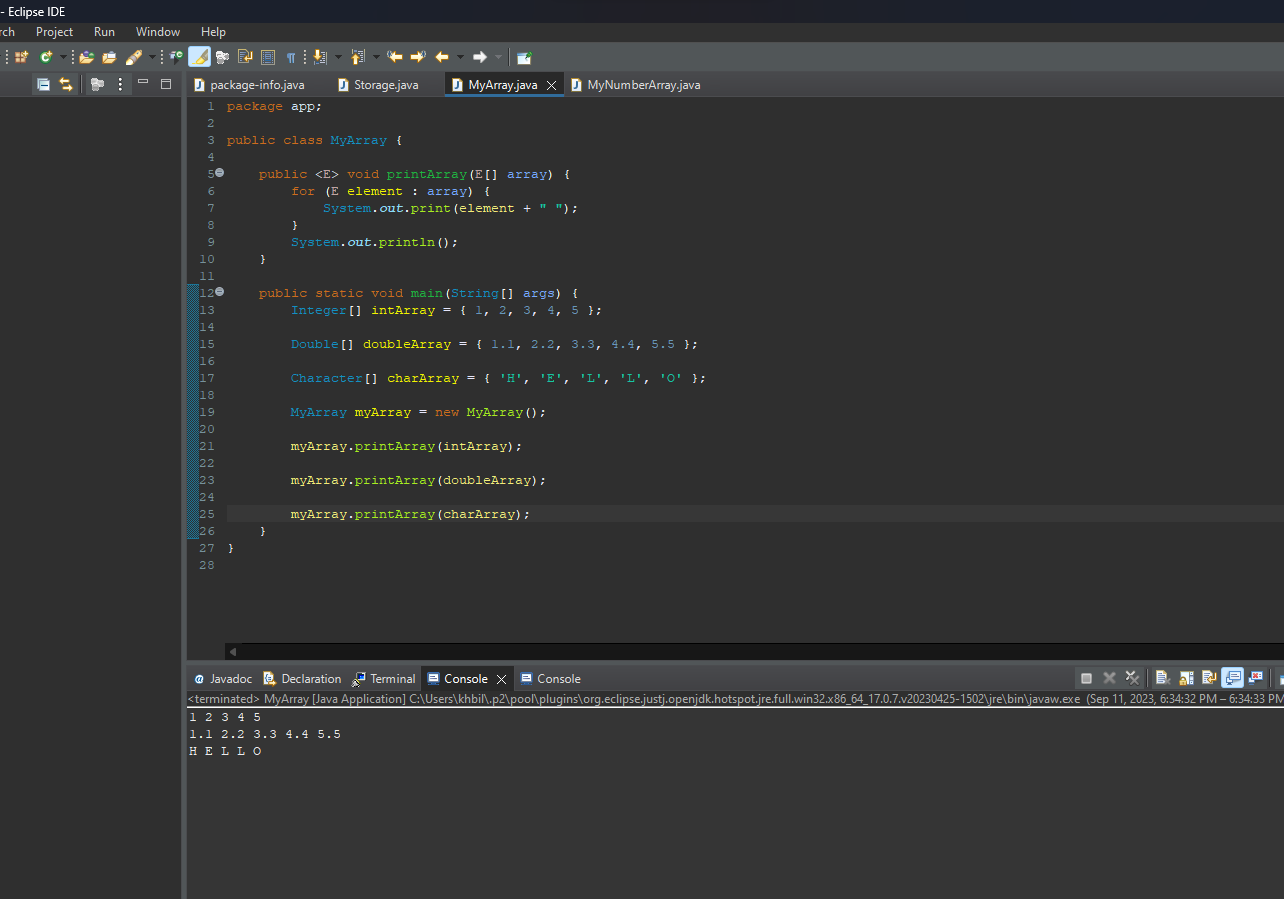
09/17/2023

Part 1: Java Generics

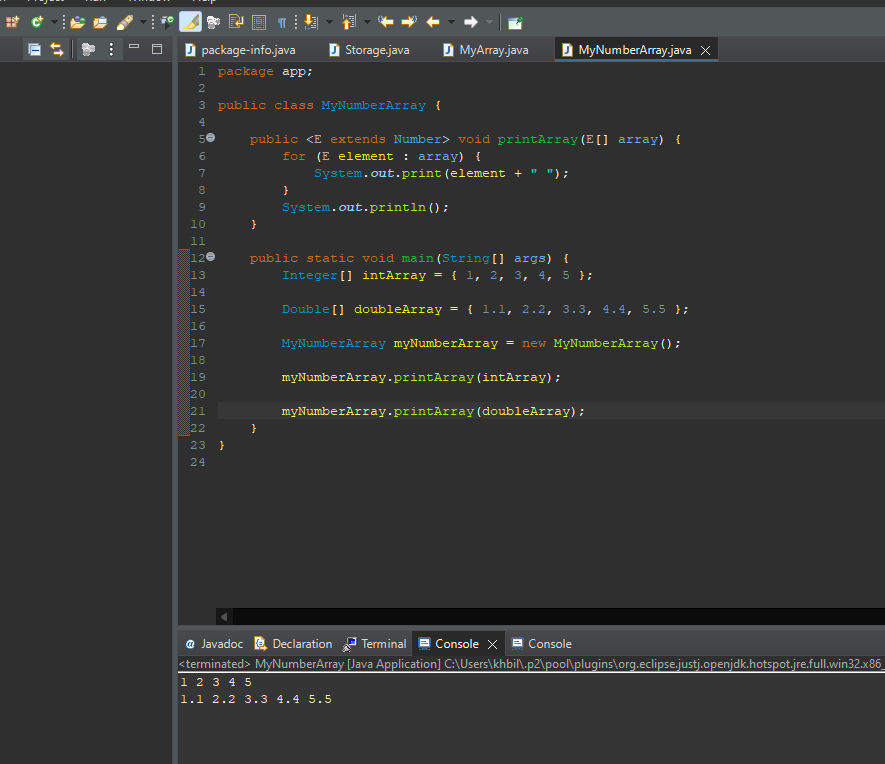
#2



#3a

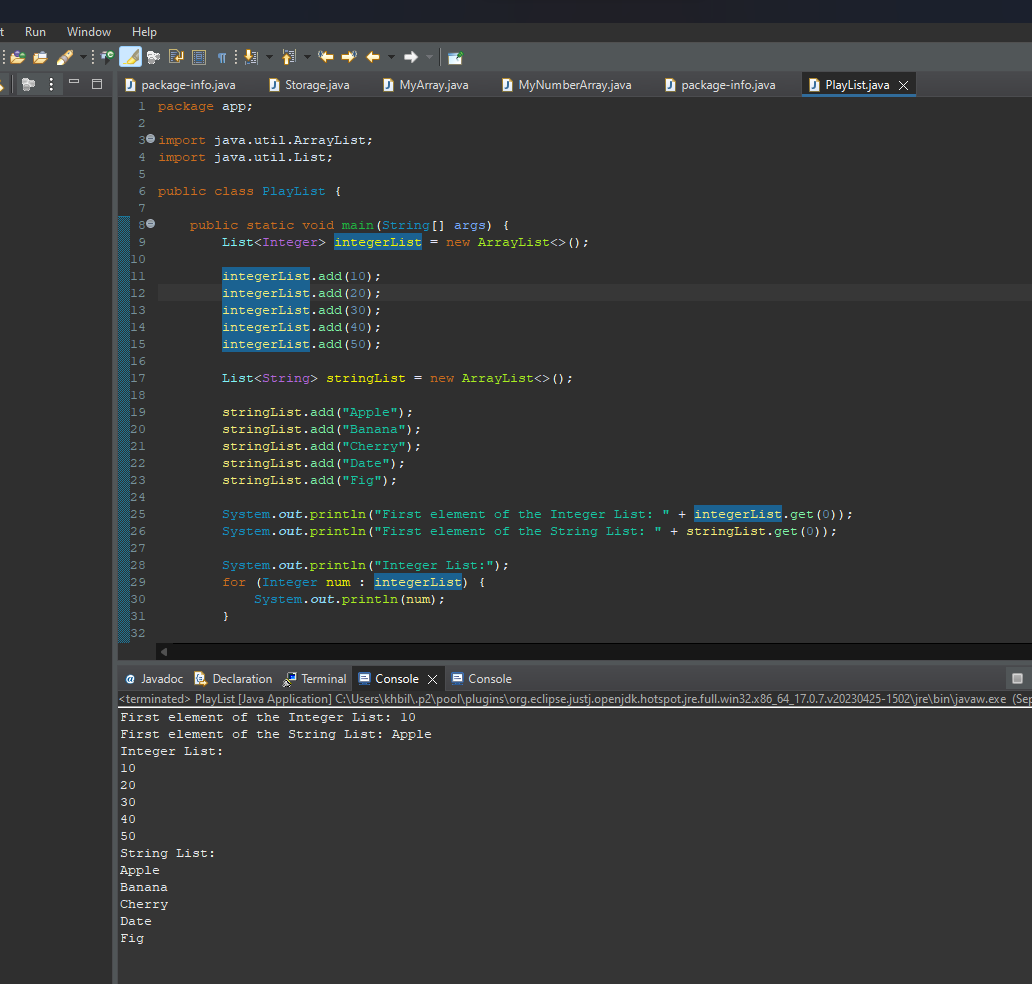


#3b

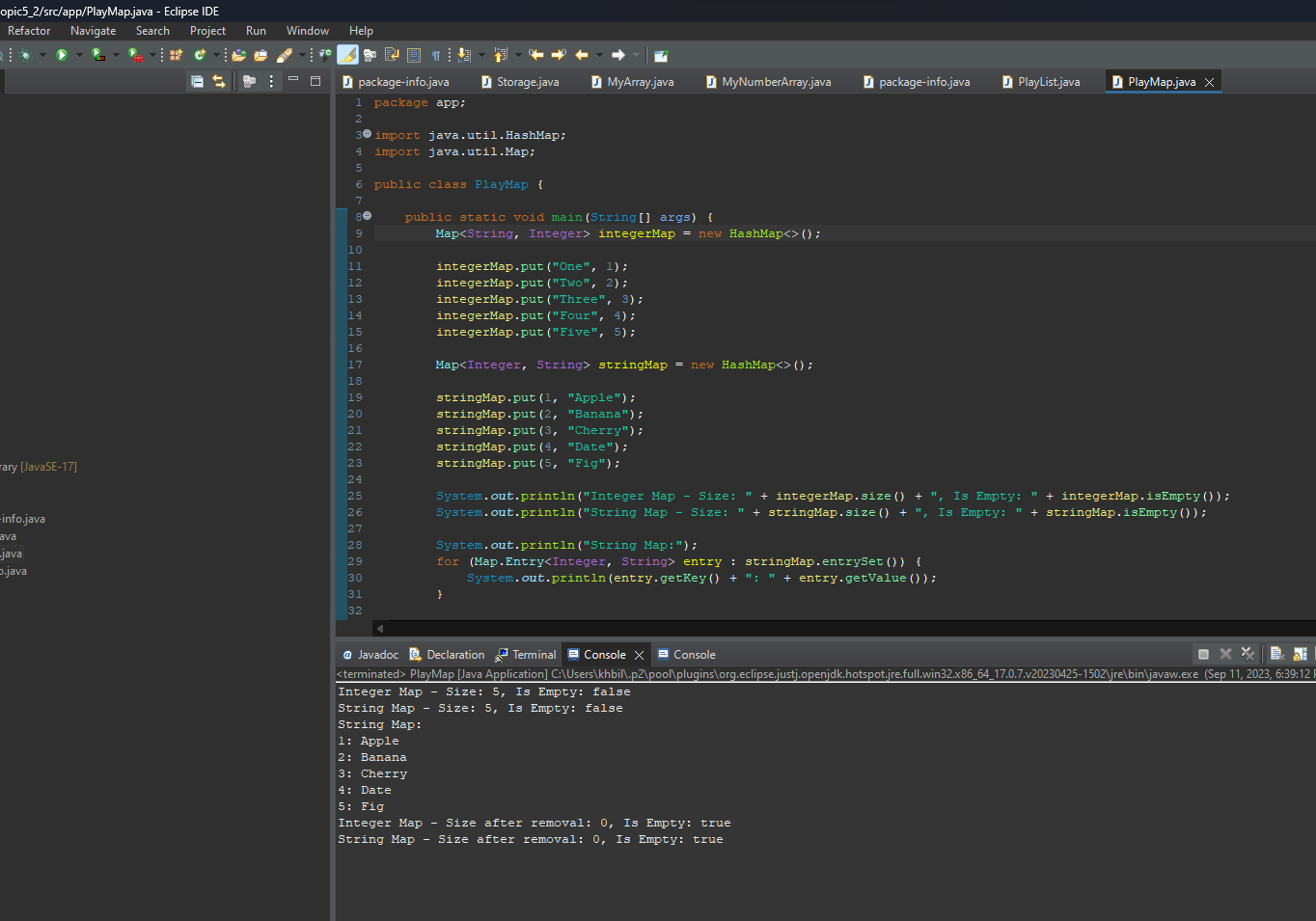


**Part 2: Java Collections Framework**

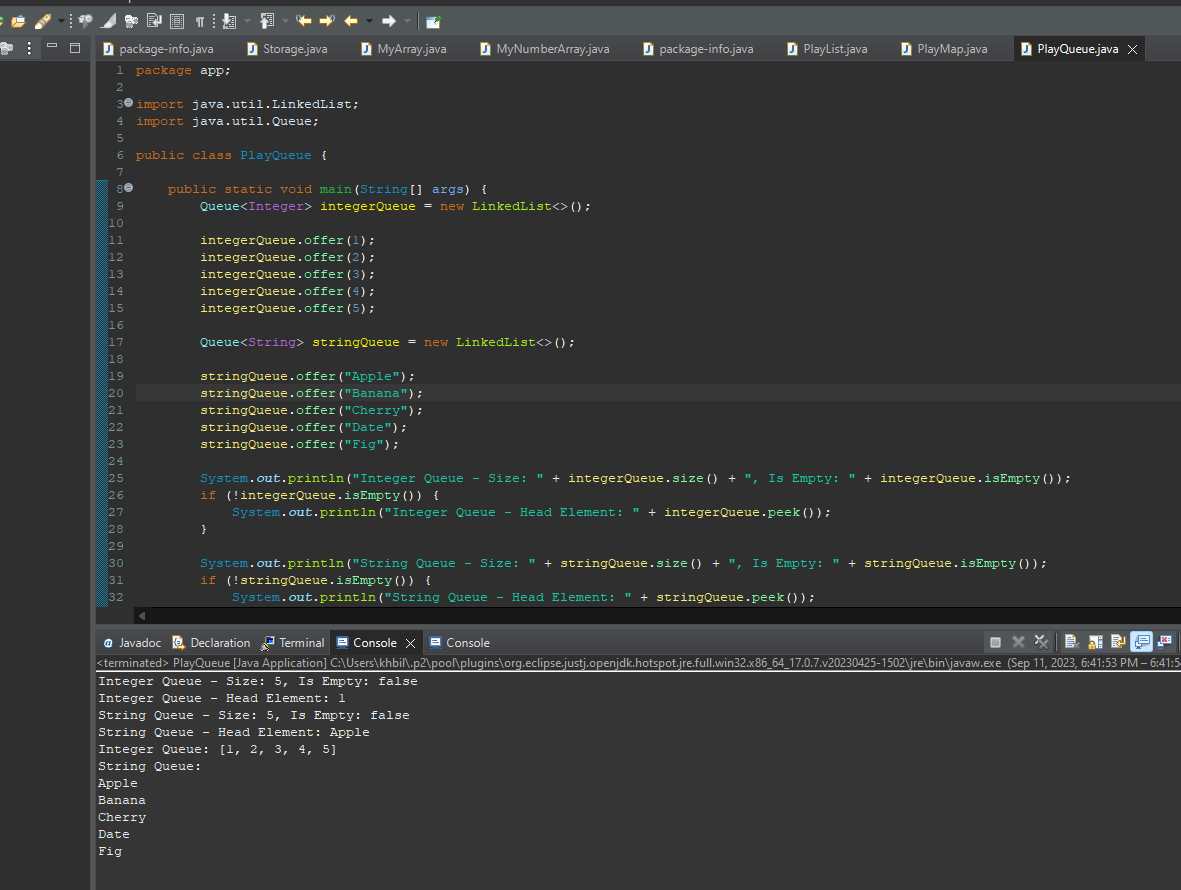
#2



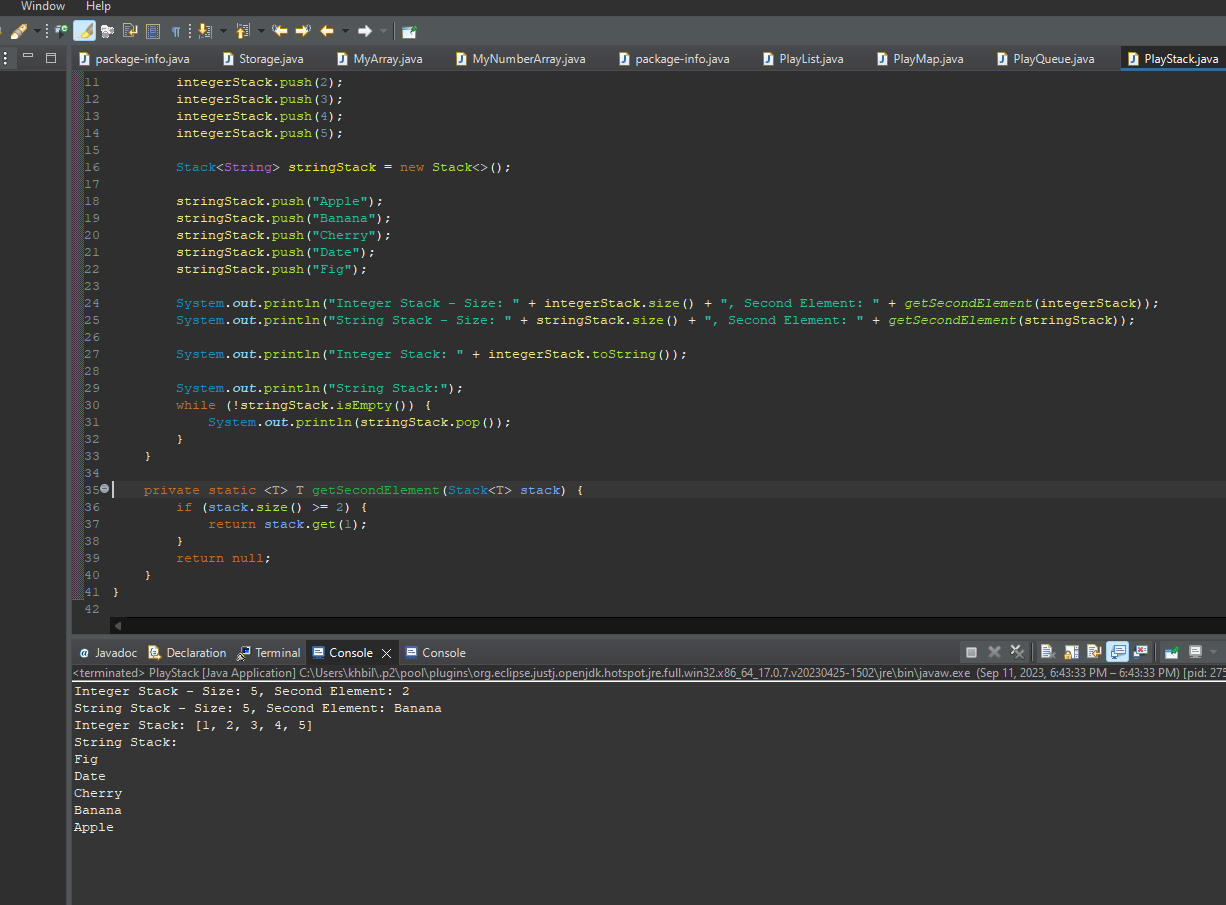
#3a



#4



#5



**Research Questions**

#1)

In Java, both ArrayList and LinkedList are commonly used to store and manage collections of data. However, they have different characteristics that make them suitable for different use cases, particularly when it comes to inserting and deleting elements at the beginning of the list.

ArrayList is implemented as a dynamic array. It provides fast random access to elements, making it efficient for retrieving elements by index. However, inserting or deleting elements at the beginning of an ArrayList can be inefficient because it requires shifting all subsequent elements to accommodate the change.   
Imagine you have a collection of coffee blends, and you want to add a new blend at the beginning and remove one from the start of the list. ArrayList is like a coffee rack, where adding or removing a coffee blend at the beginning requires moving other blends to make space.

import java.util.ArrayList;

public class CoffeeArrayListExample {  
 public static void main(String[] args) {  
 ArrayList<String> coffeeCollection = new ArrayList<>();

// Adding coffee blends to the end  
 coffeeCollection.add("Espresso");  
 coffeeCollection.add("Cappuccino");  
 coffeeCollection.add("Latte");  
 coffeeCollection.add("Mocha");

// Inserting a new blend at the beginning  
 coffeeCollection.add(0, "Americano");

// Removing the first blend  
 coffeeCollection.remove(0);

// Printing the coffee collection  
 System.out.println(coffeeCollection);  
 }  
}  
  
Now, let's consider a coffee cart, where you frequently update the menu by adding new blends at the beginning and occasionally remove the first blend. LinkedList is like this cart; it's efficient for such operations.

import java.util.LinkedList;

public class CoffeeLinkedListExample {  
 public static void main(String[] args) {  
 LinkedList<String> coffeeCart = new LinkedList<>();

// Adding coffee blends to the end  
 coffeeCart.add("Espresso");  
 coffeeCart.add("Cappuccino");  
 coffeeCart.add("Latte");  
 coffeeCart.add("Mocha");

// Inserting a new blend at the beginning  
 coffeeCart.addFirst("Americano");

// Removing the first blend  
 coffeeCart.removeFirst();

// Printing the coffee cart  
 System.out.println(coffeeCart);  
 }  
}  
  
In this example, LinkedList performs the insertions and deletions at the beginning more efficiently, just like updating a coffee cart menu. ArrayList, like a coffee rack, is suitable when you need quick access to blends by index. Understanding the differences between these data structures helps you make the right choice for your Java program. Enjoy your virtual coffee!

#2)

Using generics in Java provides several benefits, including type safety, code reusability, and better maintainability. Let's explore these benefits with an example based on the provided code:

In the given code, we have an ArrayList named coffeeCollection that stores coffee blends as strings. To demonstrate the advantages of generics, we can enhance the code by using a generic type to make it more flexible.

import java.util.ArrayList;

public class CoffeeArrayListExample {  
 public static void main(String[] args) {  
 // Use a generic ArrayList to store coffee blends  
 ArrayList<CoffeeBlend> coffeeCollection = new ArrayList<>();

// Adding coffee blends to the end  
 coffeeCollection.add(new CoffeeBlend("Espresso"));  
 coffeeCollection.add(new CoffeeBlend("Cappuccino"));  
 coffeeCollection.add(new CoffeeBlend("Latte"));  
 coffeeCollection.add(new CoffeeBlend("Mocha"));

// Inserting a new blend at the beginning  
 coffeeCollection.add(0, new CoffeeBlend("Americano"));

// Removing the first blend  
 coffeeCollection.remove(0);

// Printing the coffee collection  
 for (CoffeeBlend blend : coffeeCollection) {  
 System.out.println(blend.getName());  
 }  
 }  
}

// Define a CoffeeBlend class with a generic type for flexibility  
class CoffeeBlend {  
 private String name;

public CoffeeBlend(String name) {  
 this.name = name;  
 }

public String getName() {  
 return name;  
 }  
}

Now, let's summarize the benefits of using generics in this example:

Type Safety: With generics, we've created a CoffeeBlend class that encapsulates the name of the coffee blend. This ensures that only valid coffee blend objects can be added to the coffeeCollection. It prevents adding arbitrary objects like non-coffee-related strings or integers, enhancing type safety.

Code Reusability: By defining a CoffeeBlend class, we can reuse it throughout our code. If we need to store coffee blends in other parts of our application, we can simply instantiate CoffeeBlend objects.

Better Maintainability: Using generics and encapsulating the coffee blend's properties in a class (like CoffeeBlend) makes the code more maintainable. If we ever need to add more attributes or methods to coffee blends, we can do so within the CoffeeBlend class without affecting other parts of the code.

Clarity: Generics improve code readability by providing meaningful type information. In this example, it's clear that coffeeCollection stores objects of type CoffeeBlend.  
  
As you can see, using generics in Java allows us to write more robust, reusable, and maintainable code while ensuring type safety and clarity in our data structures!