Recitation 4

Adi Iyer

Feb 16’ 2024

Today:

- Building an abstract Linked List class, and building a pop operation on it
- Building our own class, which shall be our nodes
- Add our own class to the linked list

Problem Statement : Built Google Calendar

- From Scratch
- Using a linked list
- Define a Day in google calendar, a day should point to tomorrow
- Have at max 3 classes in a day

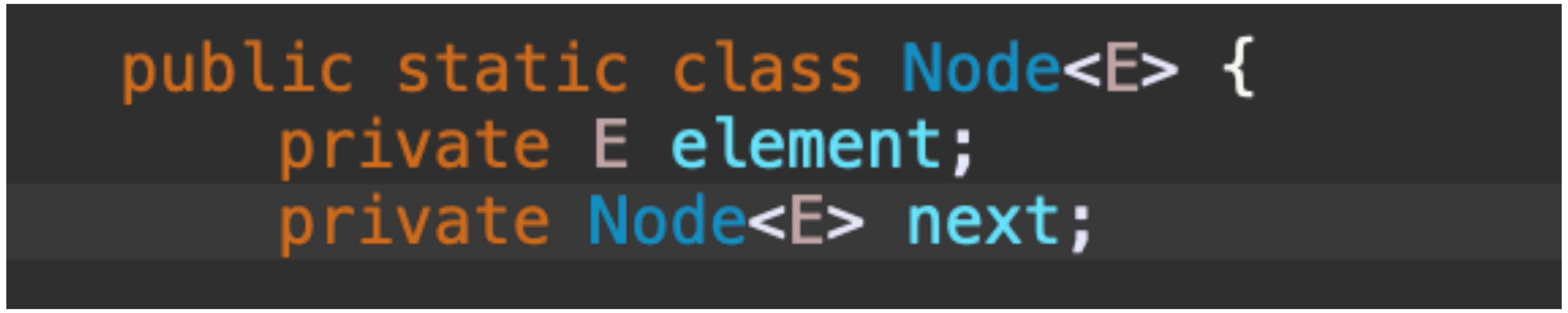
Building a generic class linked List

For any linked list, we want to define a few things

- What a node looks like, and contains?
- Get the element from a node
- Get the next node if given a node
- If doubly linked : previous node?
- Set the next node?
- Add an element at the top of the Linked list

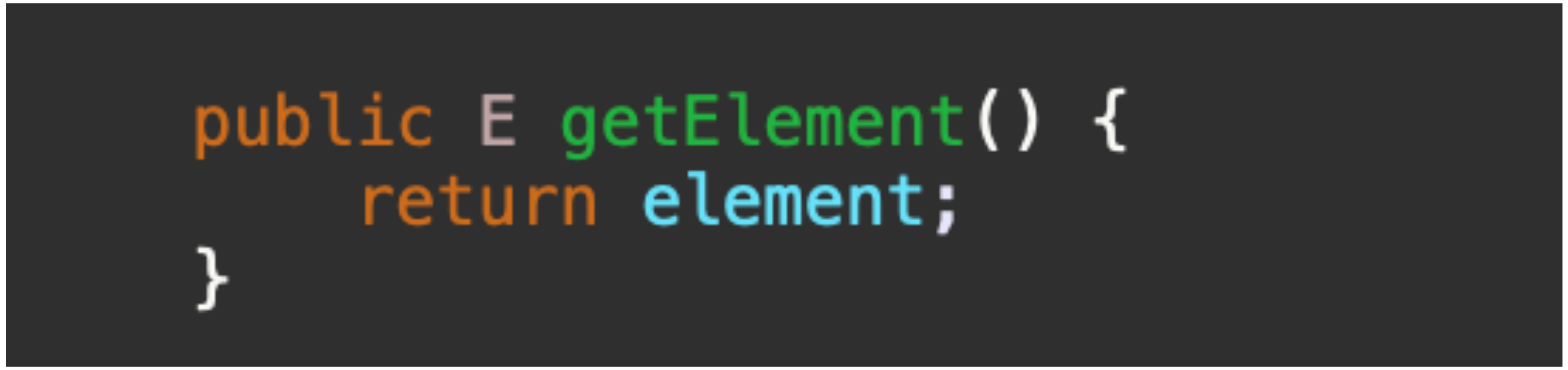
What a node looks like, and contains?

- Node should contain
    - Element
    - Next



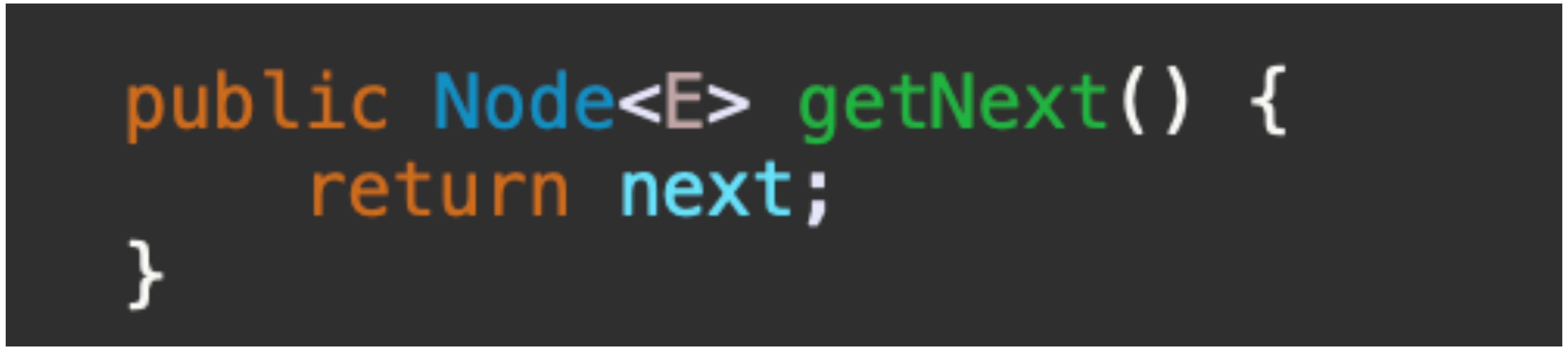
```
public static class Node<E> {
    private E element;
    private Node<E> next;
```

Get the element from a node



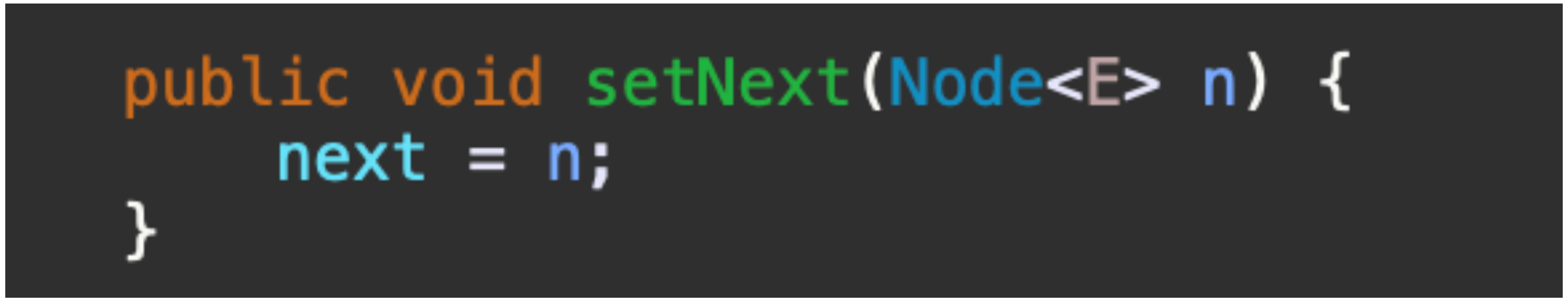
```java
public E getElement() {
    return element;
}
```

Get the next node if given a node



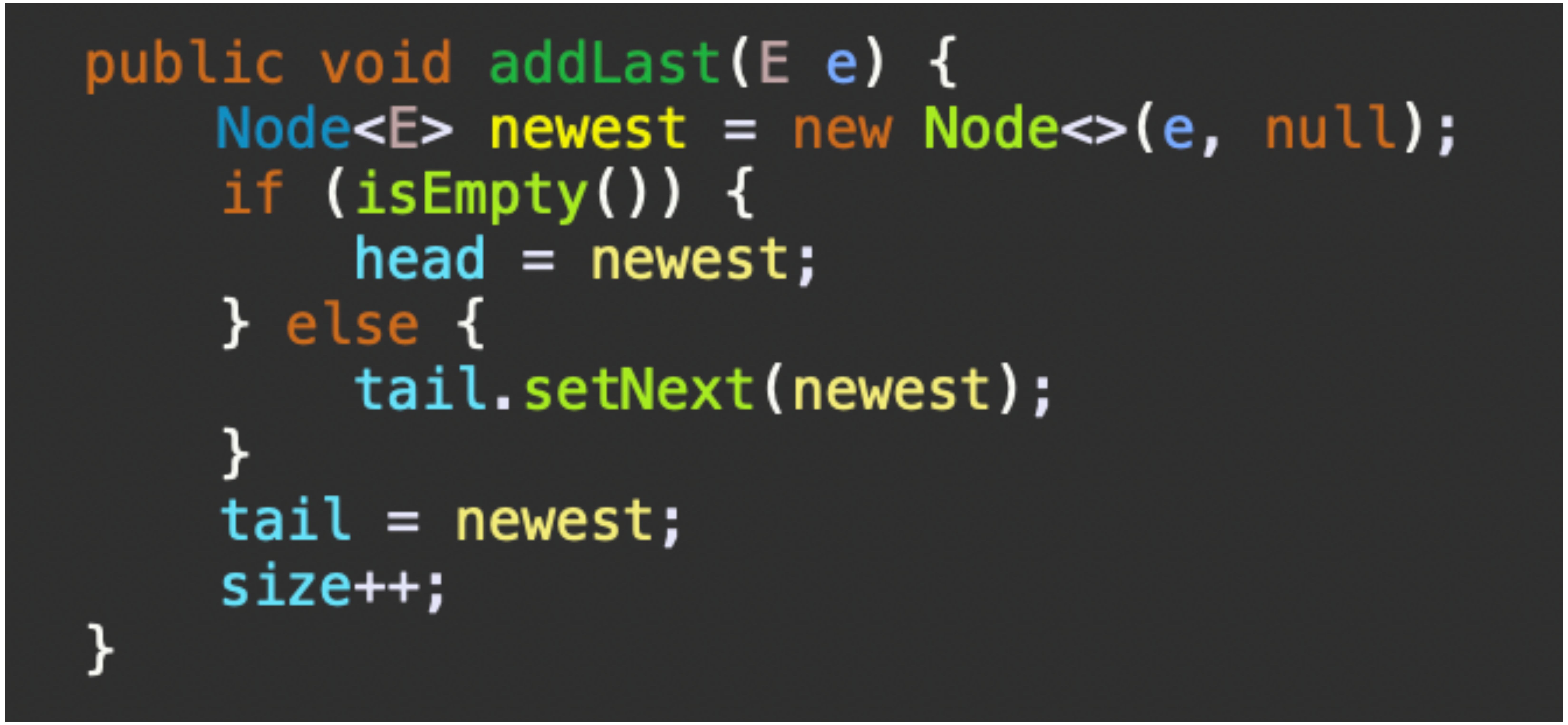
```
public Node<E> getNext() {
    return next;
}
```

Set the next node



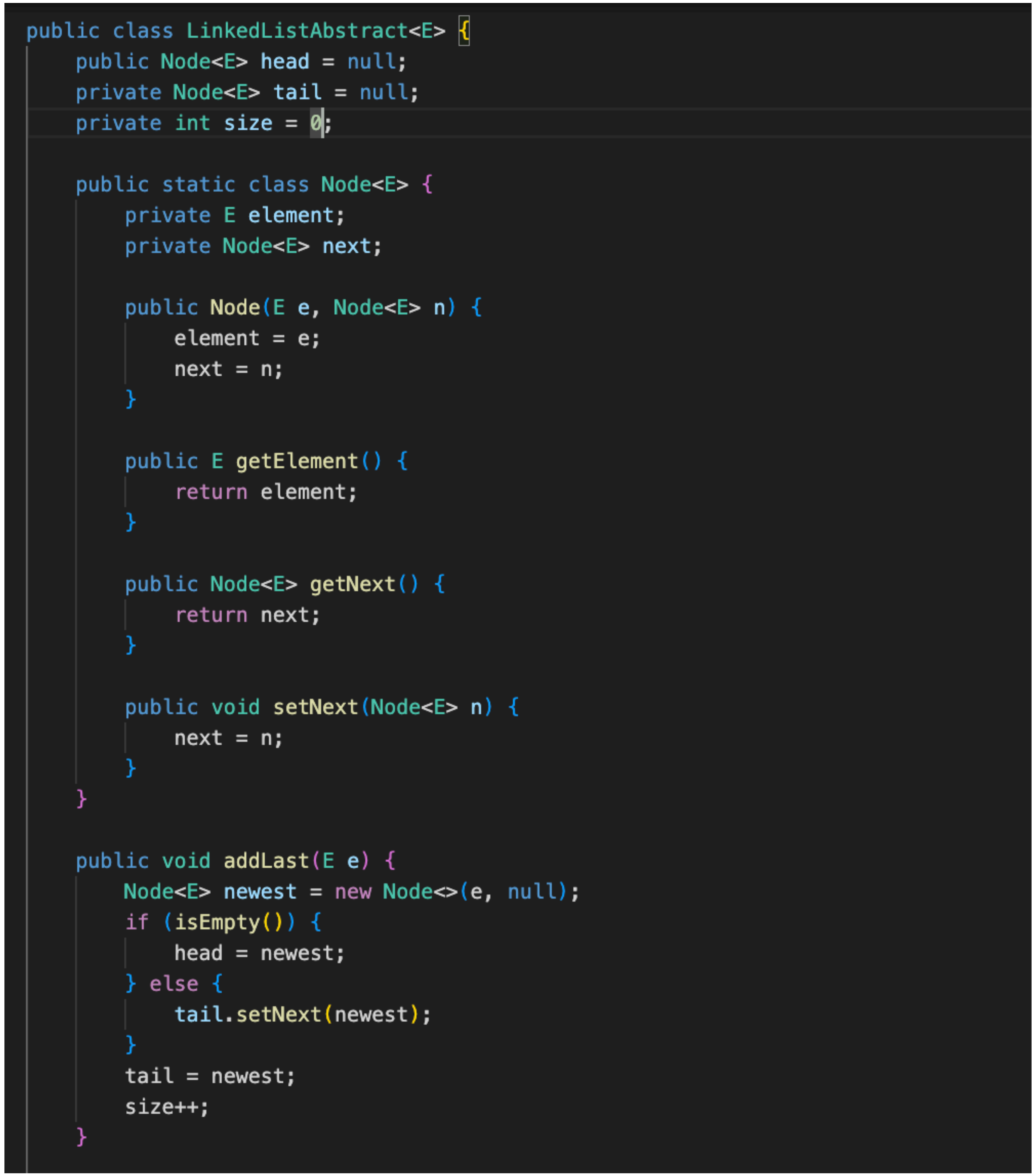
```
public void setNext(Node<E> n) {
    next = n;
}
```

Add a node to the linked list



```java
public void addLast(E e) {
    Node<E> newest = new Node<>(e, null);
    if (isEmpty()) {
        head = newest;
    } else {
        tail.setNext(newest);
    }
    tail = newest;
    size++;
}
```

Our final Linked List representation



```java
public class LinkedListAbstract<E> {
    public Node<E> head = null;
    private Node<E> tail = null;
    private int size = 0;

    public static class Node<E> {
        private E element;
        private Node<E> next;

        public Node(E e, Node<E> n) {
            element = e;
            next = n;
        }

        public E getElement() {
            return element;
        }

        public Node<E> getNext() {
            return next;
        }

        public void setNext(Node<E> n) {
            next = n;
        }
    }

    public void addLast(E e) {
        Node<E> newest = new Node<>(e, null);
        if (isEmpty()) {
            head = newest;
        } else {
            tail.setNext(newest);
        }
        tail = newest;
        size++;
    }
```

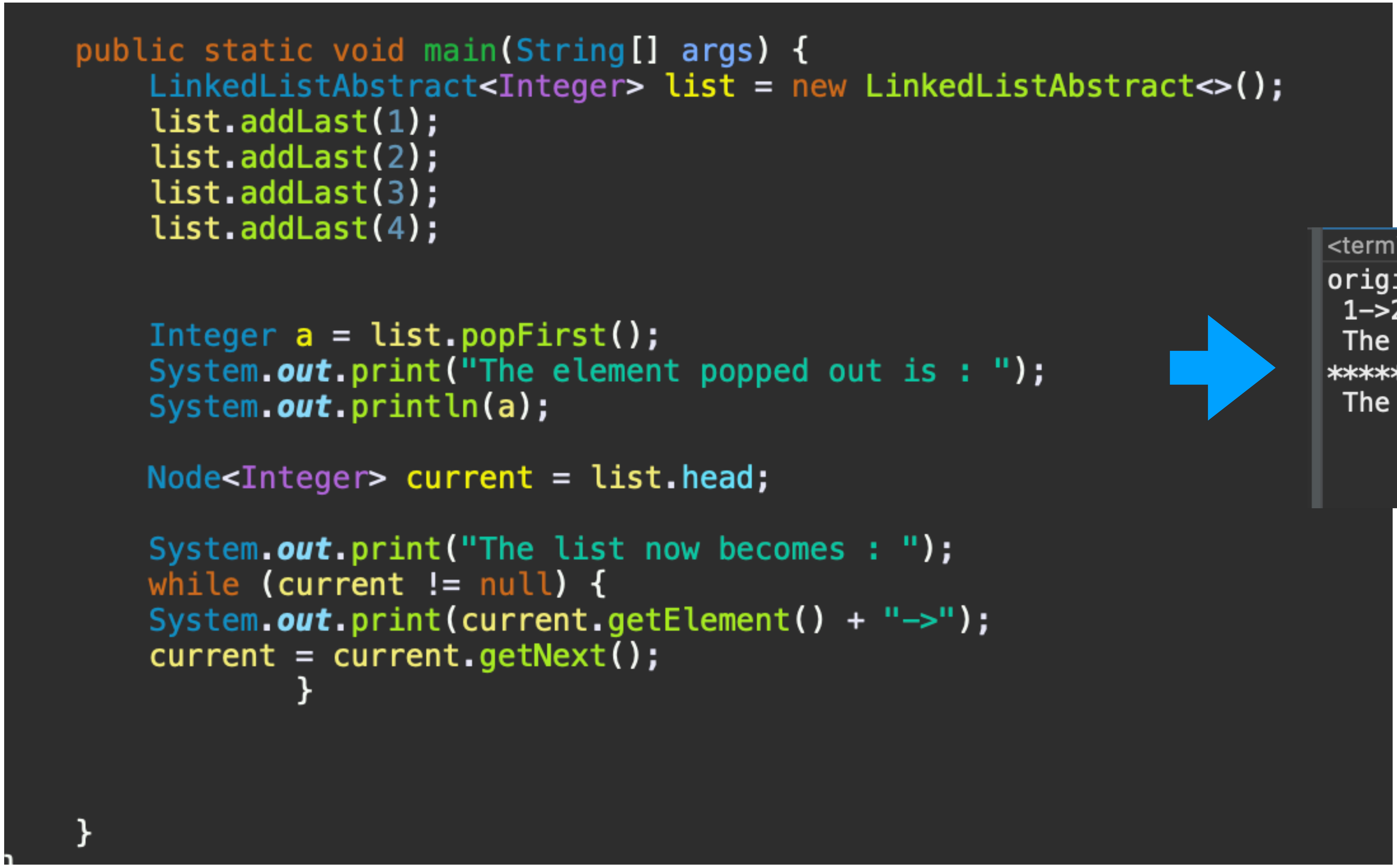
Why such an abstract linked list?

Abstraction is power

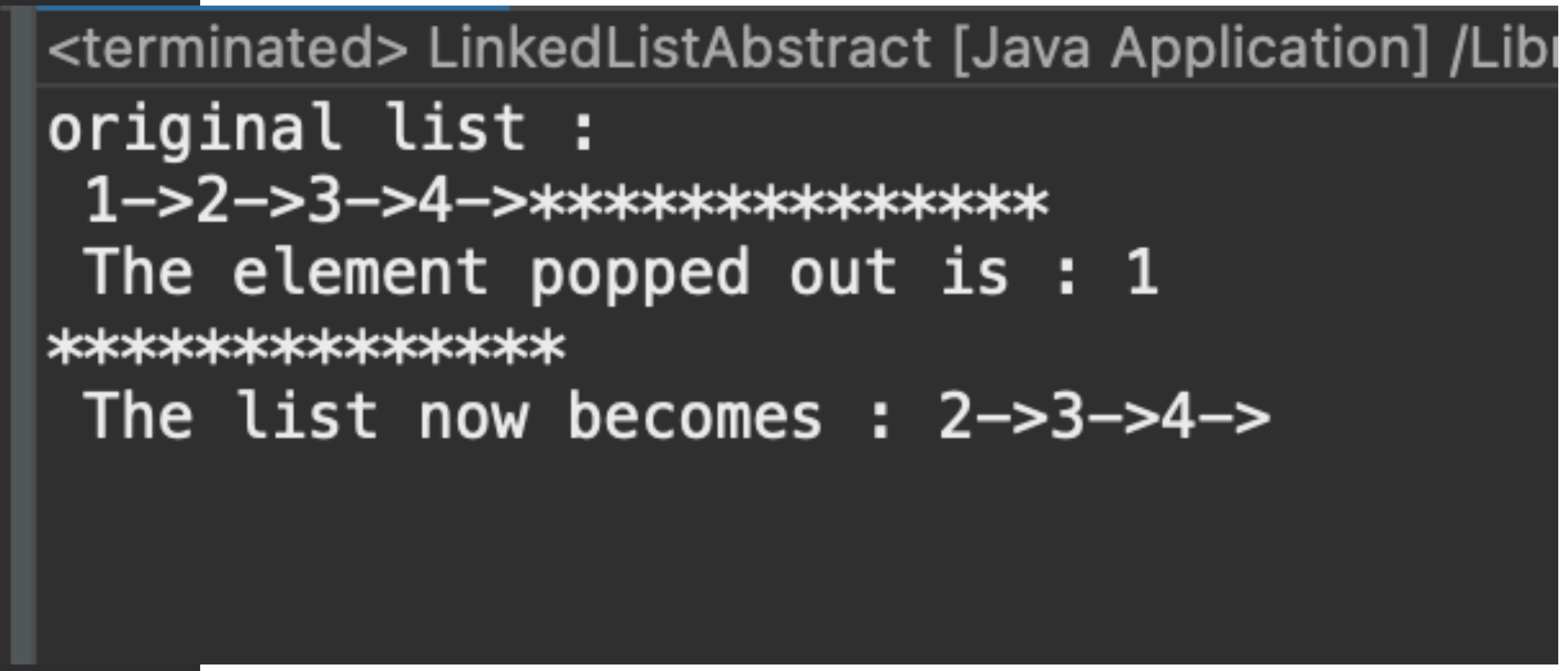
- Can hold anything : Integers, Strings etc

- ANYTHING : Days, Contacts, Emails, Google Maps routes

Testing this :



```
public static void main(String[] args) {
    LinkedListAbstract<Integer> list = new LinkedListAbstract<>();
    list.addLast(1);
    list.addLast(2);
    list.addLast(3);
    list.addLast(4);


    Integer a = list.popFirst();
    System.out.print("The element popped out is : ");
    System.out.println(a);

    Node<Integer> current = list.head;

    System.out.print("The list now becomes : ");
    while (current != null) {
    System.out.print(current.getElement() + "->");
    current = current.getNext();
            }



}
```



```
<terminated> LinkedListAbstract [Java Application] /Lib
original list :
 1->2->3->4->***************
 The element popped out is : 1
**************
 The list now becomes : 2->3->4->
```

Our list now works, lets make Google calendar

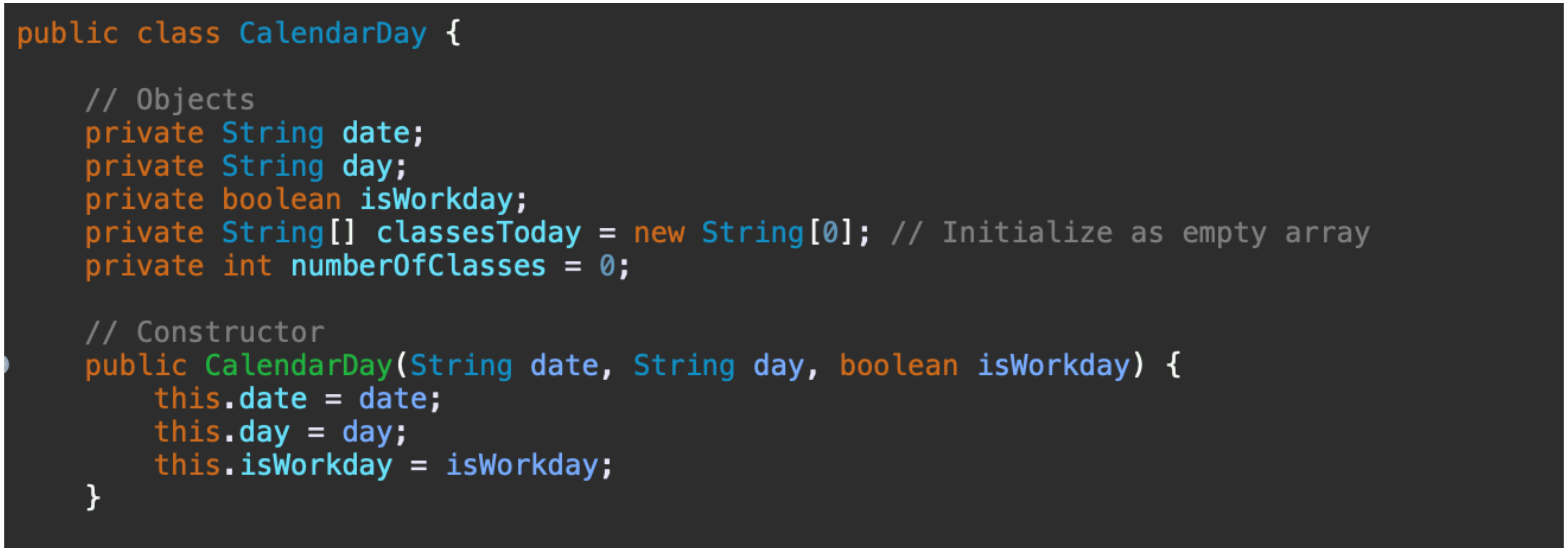
- Create a Day Class, with the following fields:
    - Date : String
    - Day : String
    - Number of Classes : Int
    - List of Classes : string[]
    - IsWeekend? : Bool

What else?

Getter and setter methods (so we can add classes, view schedule)

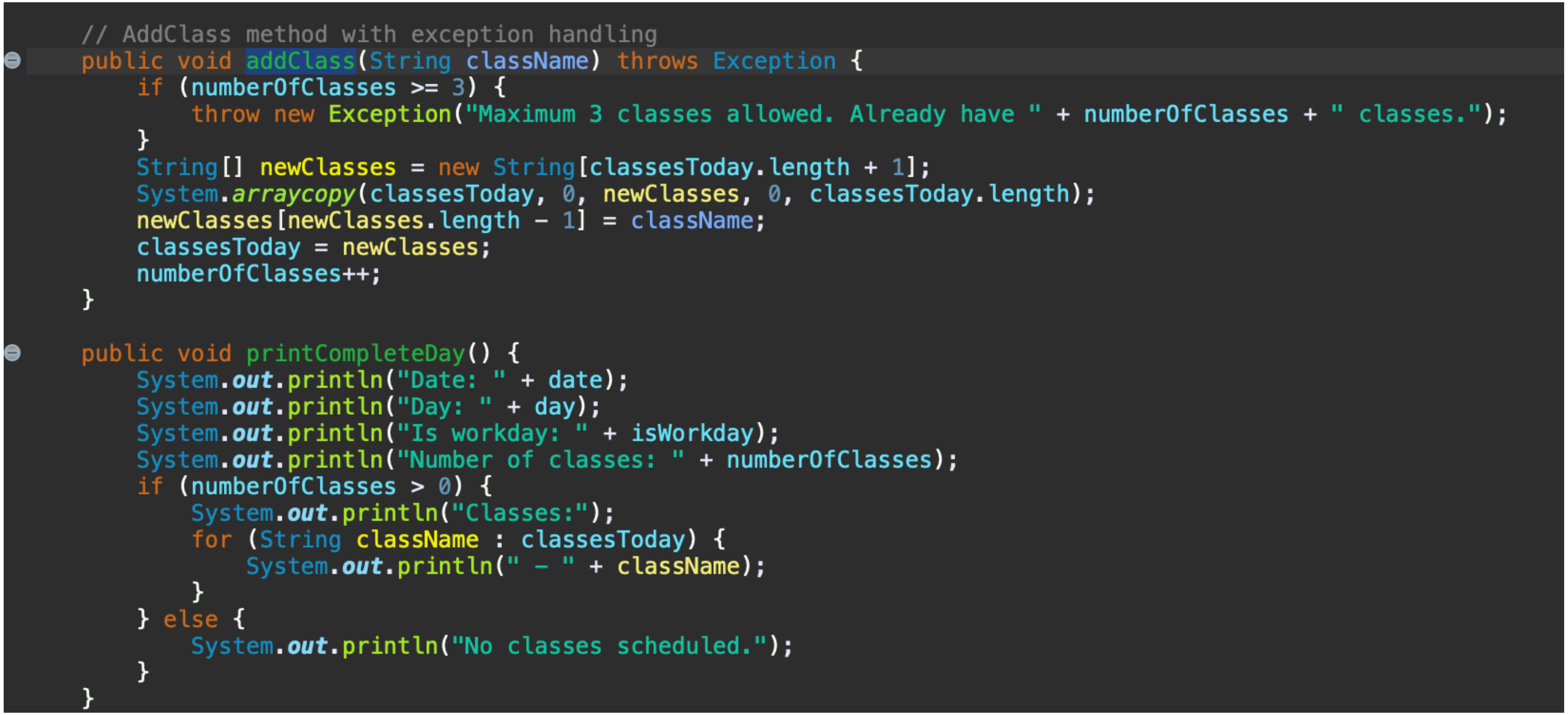
- AddClass Method?
- View whole day?
- Constructors?

What all this class looks like?



```java
public class CalendarDay {

    // Objects
    private String date;
    private String day;
    private boolean isWorkday;
    private String[] classesToday = new String[0]; // Initialize as empty array
    private int numberOfClasses = 0;

    // Constructor
    public CalendarDay(String date, String day, boolean isWorkday) {
        this.date = date;
        this.day = day;
        this.isWorkday = isWorkday;
    }
```

Adding classes and printing a day

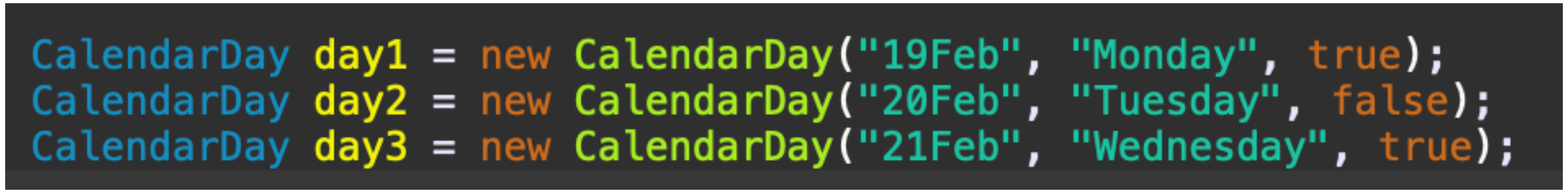


```
// AddClass method with exception handling
public void addClass(String className) throws Exception {
    if (numberOfClasses >= 3) {
        throw new Exception("Maximum 3 classes allowed. Already have " + numberOfClasses + " classes.");
    }
    String[] newClasses = new String[classesToday.length + 1];
    System.arraycopy(classesToday, 0, newClasses, 0, classesToday.length);
    newClasses[newClasses.length - 1] = className;
    classesToday = newClasses;
    numberOfClasses++;
}

public void printCompleteDay() {
    System.out.println("Date: " + date);
    System.out.println("Day: " + day);
    System.out.println("Is workday: " + isWorkday);
    System.out.println("Number of classes: " + numberOfClasses);
    if (numberOfClasses > 0) {
        System.out.println("Classes:");
        for (String className : classesToday) {
            System.out.println(" - " + className);
        }
    } else {
        System.out.println("No classes scheduled.");
    }
}
```

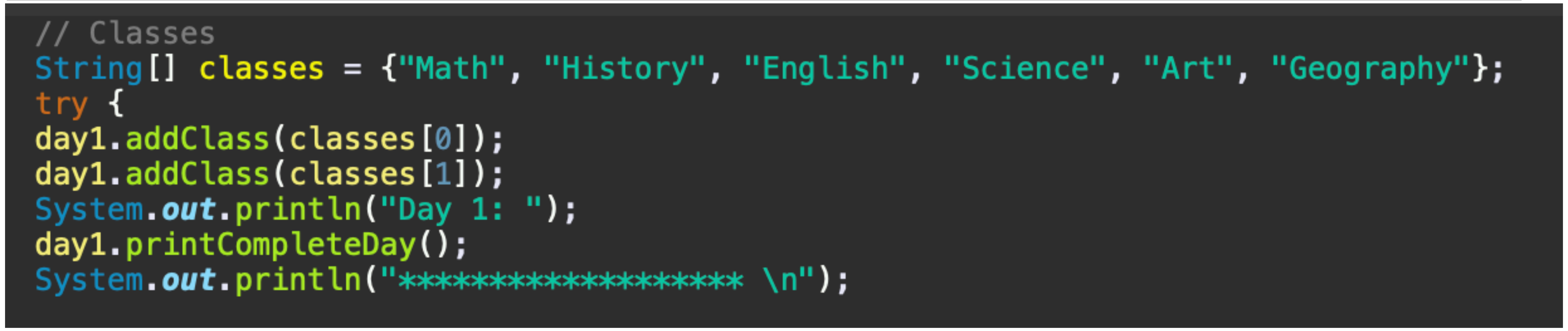
What we have now?

- An ABSTRACT Linked list class : waiting to be filled with objects
- A Calendar Day object Class
- To build Google calendar, we need to add Days into a LinkedList
- Lets run the app now?

Testing our app - testing Calendar Class

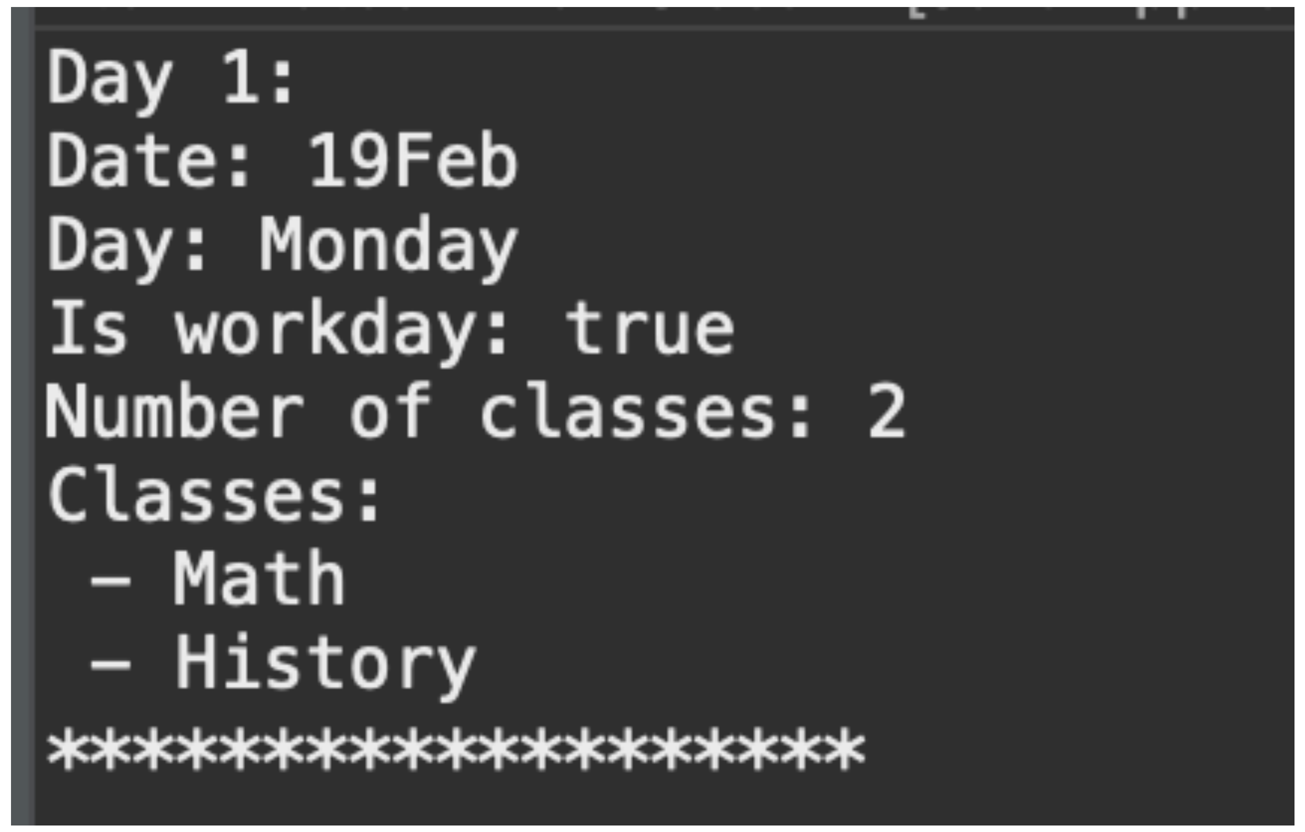


```java
CalendarDay day1 = new CalendarDay("19Feb", "Monday", true);
CalendarDay day2 = new CalendarDay("20Feb", "Tuesday", false);
CalendarDay day3 = new CalendarDay("21Feb", "Wednesday", true);
```



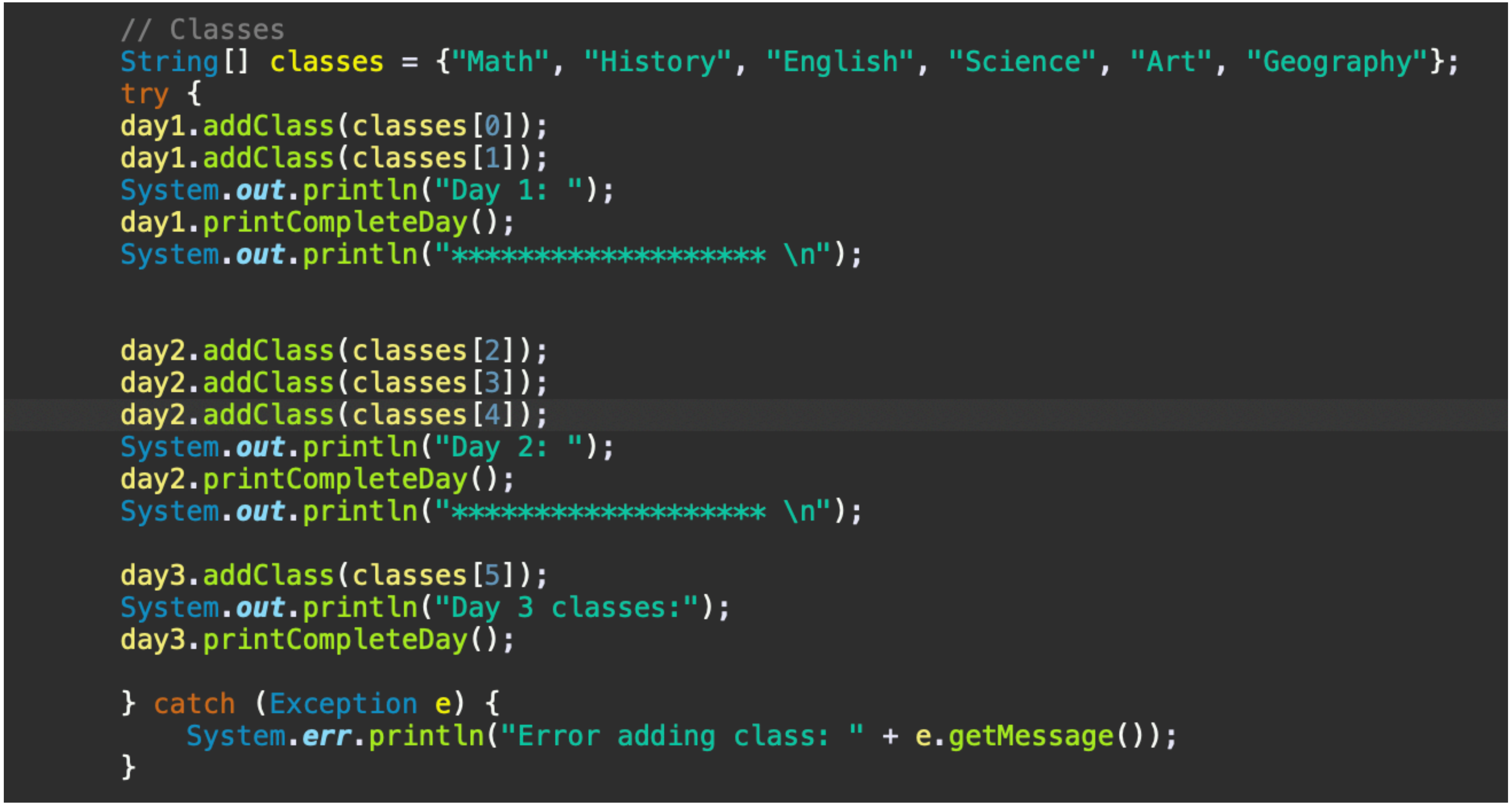
```java
// Classes
String[] classes = {"Math", "History", "English", "Science", "Art", "Geography"};
try {
day1.addClass(classes[0]);
day1.addClass(classes[1]);
System.out.println("Day 1: ");
day1.printCompleteDay();
System.out.println("******************* \n");
```

Expected Output :



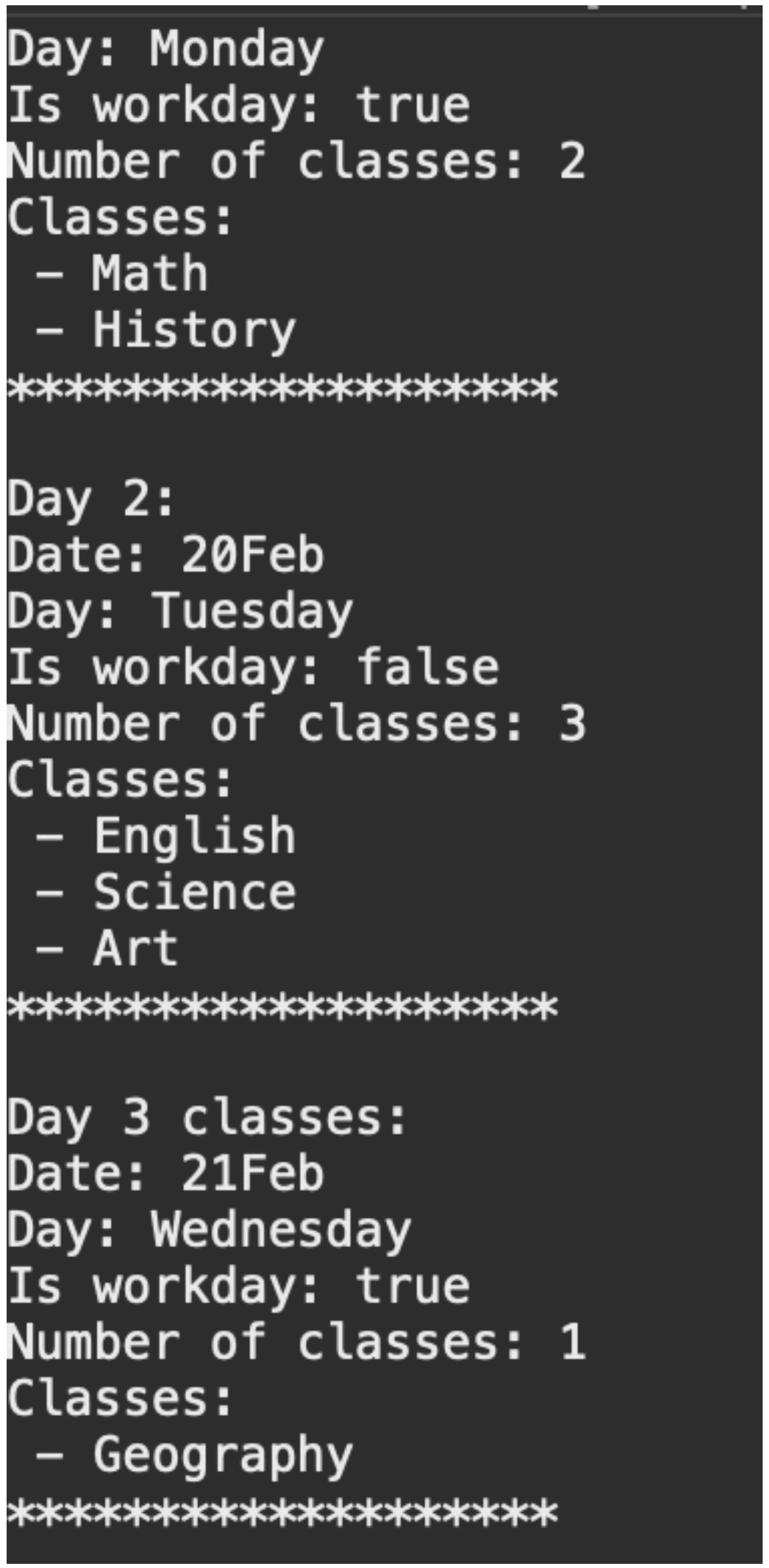
```
Day 1:
Date: 19Feb
Day: Monday
Is workday: true
Number of classes: 2
Classes:
 - Math
 - History
*******************
```

Creating all our days:



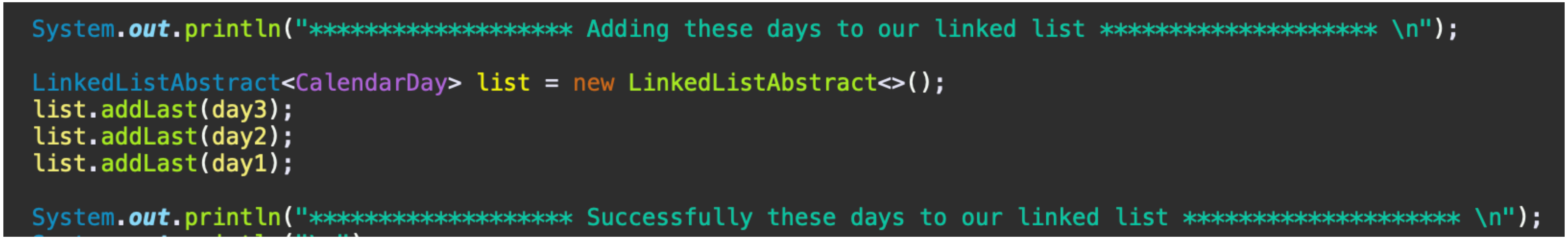
```
// Classes
String[] classes = {"Math", "History", "English", "Science", "Art", "Geography"};
try {
day1.addClass(classes[0]);
day1.addClass(classes[1]);
System.out.println("Day 1: ");
day1.printCompleteDay();
System.out.println("******************* \n");


day2.addClass(classes[2]);
day2.addClass(classes[3]);
day2.addClass(classes[4]);
System.out.println("Day 2: ");
day2.printCompleteDay();
System.out.println("******************* \n");

day3.addClass(classes[5]);
System.out.println("Day 3 classes:");
day3.printCompleteDay();

} catch (Exception e) {
    System.err.println("Error adding class: " + e.getMessage());
}
```

What it looks like now:



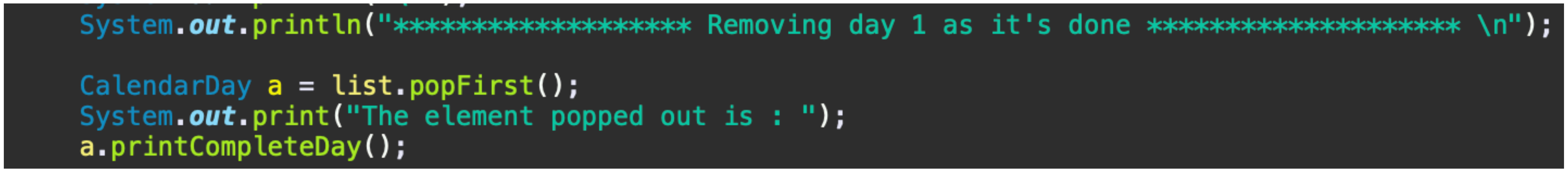
```
Day: Monday
Is workday: true
Number of classes: 2
Classes:
 - Math
 - History
*******************

Day 2:
Date: 20Feb
Day: Tuesday
Is workday: false
Number of classes: 3
Classes:
 - English
 - Science
 - Art
*******************

Day 3 classes:
Date: 21Feb
Day: Wednesday
Is workday: true
Number of classes: 1
Classes:
 - Geography
*******************
```

Add these “Days” to our linked list



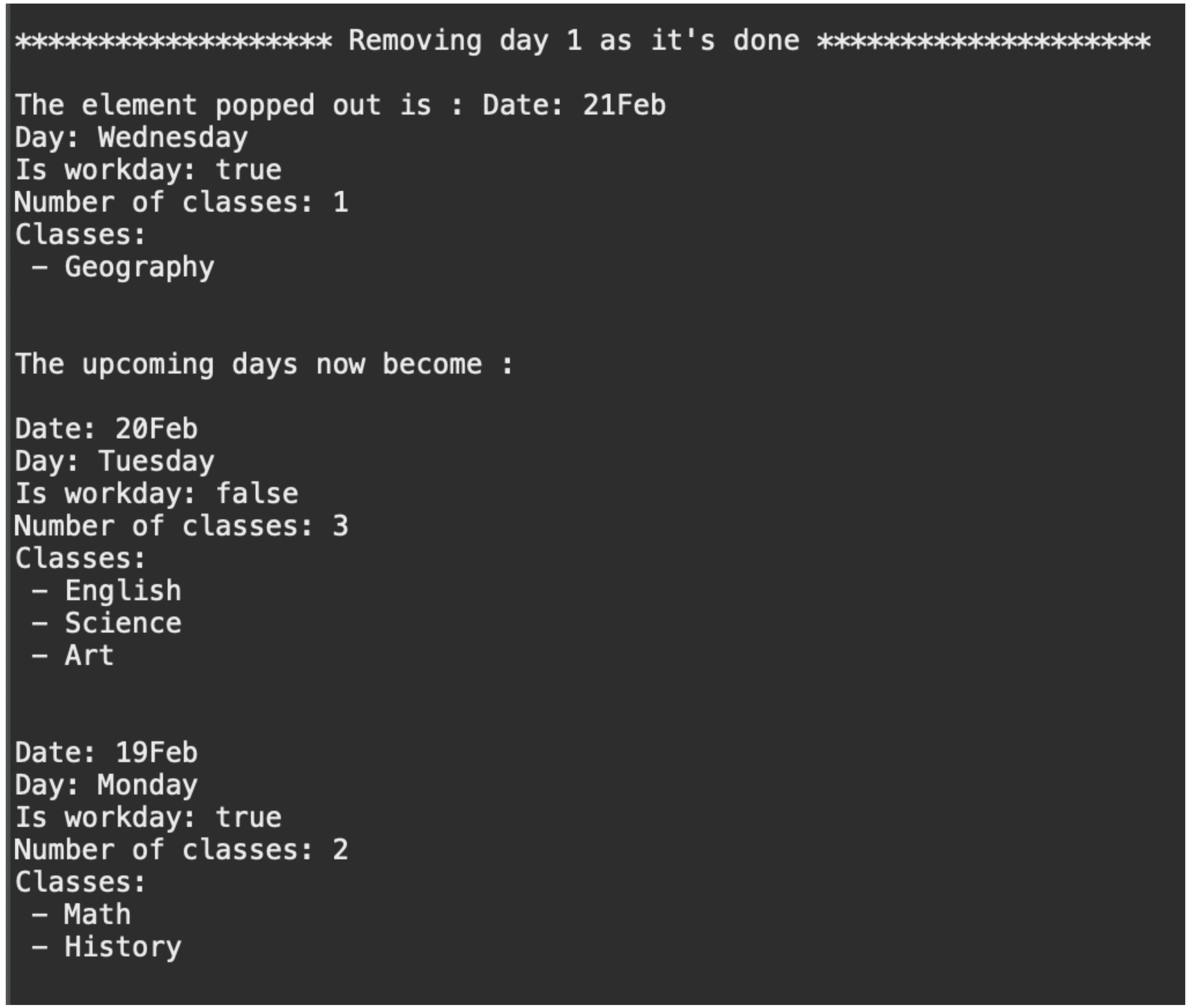
```java
System.out.println("******************* Adding these days to our linked list ******************** \n");

LinkedListAbstract<CalendarDay> list = new LinkedListAbstract<>();
list.addLast(day3);
list.addLast(day2);
list.addLast(day1);

System.out.println("******************* Successfully these days to our linked list ******************** \n");
```

Say Monday happened : pop it



```
    System.out.println("******************* Removing day 1 as it's done ******************** \n");

    CalendarDay a = list.popFirst();
    System.out.print("The element popped out is : ");
    a.printCompleteDay();
```

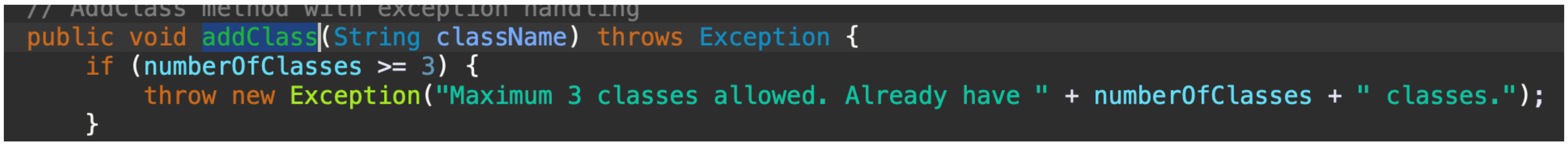
Output :



```
******************* Removing day 1 as it's done ********************

The element popped out is : Date: 21Feb
Day: Wednesday
Is workday: true
Number of classes: 1
Classes:
 - Geography


The upcoming days now become :

Date: 20Feb
Day: Tuesday
Is workday: false
Number of classes: 3
Classes:
 - English
 - Science
 - Art


Date: 19Feb
Day: Monday
Is workday: true
Number of classes: 2
Classes:
 - Math
 - History
```

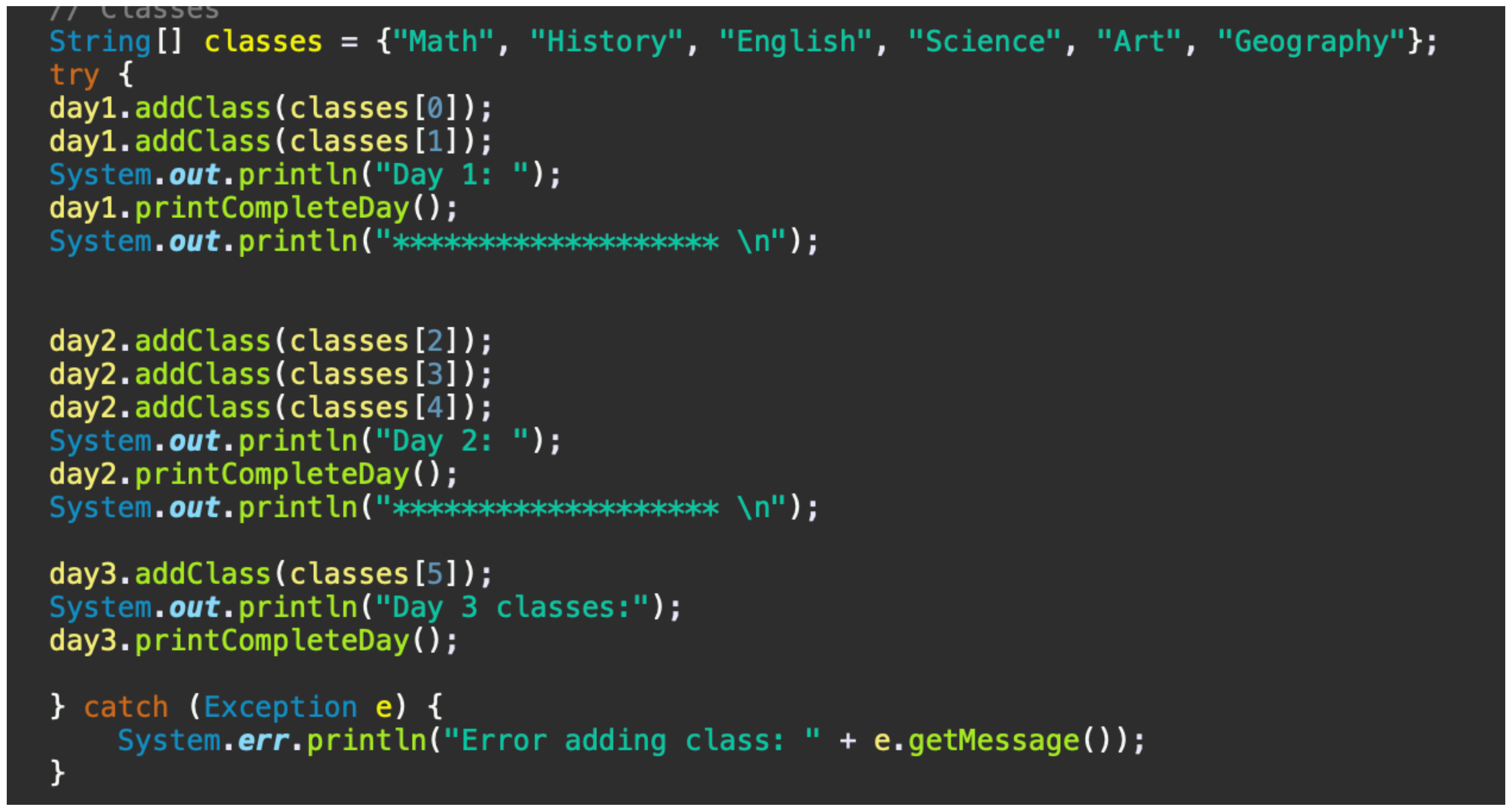
Resolving Try Catch

Define The exception : STOP THE PROGRAM if this happens



```
// AddClass method with exception handling
public void addClass(String className) throws Exception {
    if (numberOfClasses >= 3) {
        throw new Exception("Maximum 3 classes allowed. Already have " + numberOfClasses + " classes.");
    }
```

Try to Catch the exception if it EVER happens - deal with it



```
// Classes
String[] classes = {"Math", "History", "English", "Science", "Art", "Geography"};
try {
day1.addClass(classes[0]);
day1.addClass(classes[1]);
System.out.println("Day 1: ");
day1.printCompleteDay();
System.out.println("****************** \n");


day2.addClass(classes[2]);
day2.addClass(classes[3]);
day2.addClass(classes[4]);
System.out.println("Day 2: ");
day2.printCompleteDay();
System.out.println("****************** \n");

day3.addClass(classes[5]);
System.out.println("Day 3 classes:");
day3.printCompleteDay();

} catch (Exception e) {
    System.err.println("Error adding class: " + e.getMessage());
}
```

Your Next Homework

- Design Spotify
- Use a Doubly Linked List : So you can play previous and next songs
- Add a song to your playlist
- Remove a song from your playlist

How it works?

- You submit your classes (Say a song class, a linked list class etc)
- We give you a sample script which we will test on : make your classes in a way to satisfy our test cases

- We have a hidden test cases file : we test on it as well
- More details on submission soon.