Unit 6 - Lesson 1
Project Planning

Computer Science A

Natural language processing (NLP) is the ability of a computer program to understand human language.

📝 **Unit 6 Guide**

How do different apps and programs use natural language processing?

Documentation refers to the written descriptions of the purpose and functionality of code.

Java Lab Documentation

- Java Lab Shortcuts
- Java Basics
- org.code.theater
- org.code.media
- org.code.neighborhood
- java.io
- java.util
- java.lang
- Control Structures
- Data Structures

Painter

Category: org.code.neighborhood

Fields

Type	Name	Description
int	xLocation	the x coordinate of the `Painter` object
int	yLocation	the y coordinate of the `Painter` object
String	direction	the direction the `Painter` object is facing (`"North"`, `"South"`, `"East"`, or `"West"`)
int	remainingPaint	the number of units of paint the `Painter` object has in their paint bucket

Method Details

Painter

`public Painter()`

Creates a `Painter` object at `(0, 0)` facing `"East"` with `0` units of paint

Examples

```
Painter myPainter = new Painter();
```

📝 **Unit 6 Guide**

Javadocs is a documentation tool for Java that is created by writing comments inside `/** */` and using `@` tags.

Description of the code segment

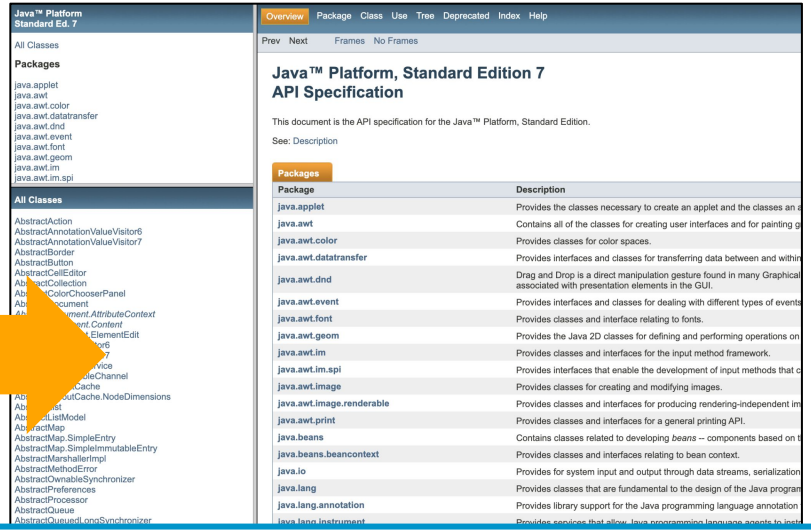
`@param` tags identify the parameters and explain what they represent

```
/**
 * Launch a URL from an intent.
 *
 * @param url            The url from the intent.
 * @param referer        Optional referer URL to be used.
 * @param headers        Optional headers to be sent when opening the URL.
 * @param externalAppId  External app id.
 * @param forceNewTab    Whether to force the URL to be launched in a new tab or to fall
 *                       back to the default behavior for making that determination.
 * @param isRendererInitiated Whether the intent is originally from browser renderer process.
 * @param initiatorOrigin Origin that initiates the intent.
 * @param intent         The original intent.
 */
private Tab launchIntent(
        LoadUrlParams loadUrlParams, String externalAppId, boolean forceNewTab, Intent i
```

📝 **Unit 6 Guide**

Javadocs generates an HTML document from comments written inside /** */ and formatted using @ tags.

```
/**
 * The AddNumbers program adds two integers
 * and prints the output to the console.
*/
public class AddNumbers {
    /**
    * Adds two integers
    *
    * @param numA The first integer to add
    * @param numB The second integer to add
    * @return the sum of the two integers
    */
    public int addNumbers(int numA, int numB) {
        return numA + numB;
    }
}
```



HTML stands for Hypertext Markup Language and is the standard system for tagging text files to be displayed on the World Wide Web.

📝 **Unit 6 Guide**

📝 Unit 6 Guide

Javadocs Tags

- **`@param parameterName description`**
  Adds a parameter with the specified **`parameterName`** followed by the **`description`**
- **`@return description`**
  Adds a returns section with the **`description`**

These aren't the only tags available! There are also tags for documenting things like author or version number.

💼 Key Vocabulary

- **natural language processing:** the ability of a computer program to understand human language
- **documentation:** written descriptions of the purpose and functionality of code
- **Javadocs:** the documentation tool for Java that generates an HTML document from comments written inside `/** */` and formatted using `@` tags
- **HTML:** stands for Hypertext Markup Language; the standard system for tagging text files to be displayed on the World Wide Web

Unit 6 - Lesson 2
Substrings

Computer Science A

🚀 Question of the Day

How can I analyze parts of a **`String`** object?

The **`String substring(int beginIndex)`** method returns a new **`String`** that is a substring of ***this*** **`String`** starting at **`beginIndex`** to the end of the **`String`**.

```
String message = "Hello World!";
System.out.println(message.substring(5));
```

```
World!
```

The **`String substring(int beginIndex, int endIndex)`** method returns a new **`String`** that is a substring of ***this*** **`String`** starting at **`beginIndex`** up to **but not including** the character at **`endIndex`**.

```
String message = "Hello World!";
System.out.println(message.substring(0, 5));
System.out.println(message.substring(2, 7));
```

```
Hello
llo W
```

The **`substring()`** method can also be used to obtain a **single character** in a **`String`**.

```
String message = "Hello World!";
System.out.println(message.substring(1, 2));
System.out.println(message.substring(6, 7));
```

```
e
W
```

String objects are **immutable**, meaning that the contents of the **String** cannot be changed after it is created.

```
String message = "Hello World!";
message.substring(0, 5);
System.out.println(message);
```

```
Hello World!
```

String methods do not change the original **String** object.

Unit 6 - Lesson 3
Integer and Double Objects

Computer Science A

💡 Discuss:

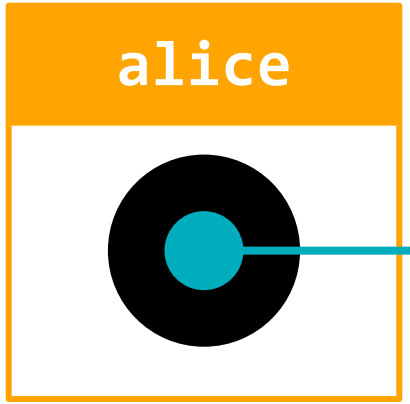
What are some **differences** between **primitive** and **reference types**?

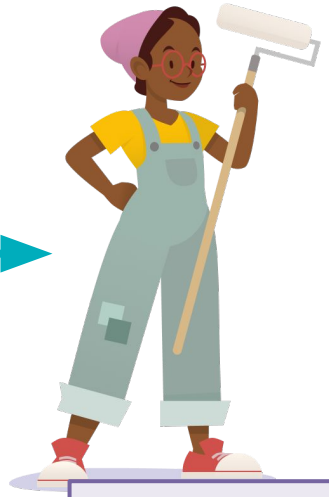
`int`

whole numbers, like 7 or 6784

`double`

decimal numbers, like `2.5` or `92.81`





📝 **Unit 6 Guide**

Primitive Types

- Predefined by Java. Consists of **`boolean`** and numeric types, like **`int`** and **`double`**.
- Have a default value of zero (**`0`** or **`0.0`**) or **`false`**.
- Single value only and does not have any attributes or behaviors.
- Initialized with a literal value.
- Memory location stores the actual data held by the primitive type.

Reference Types

- Defined by the user. Unlimited number of reference types, including **`String`**, **`Painter`**, and **`Scanner`**.
- Have a default value of **`null`**.
- Consists of local, instance, and static variables and methods.
- Initialized using the **`new`** keyword and a constructor call.
- Memory location stores a reference to the reference type.

🚀 Question of the Day

Why would I want to represent primitive values as objects?

How would storing integer and decimal values as objects be useful?

The **`Integer`** class is part of the **`java.lang`** package and has constants for the maximum and minimum values for an **`int`**.

It also has a method to return its value as an **`int`** and to convert a **`String`** to an **`int`**.

Integer
`MAX_VALUE : int` `MIN_VALUE : int`
`int intValue()` `static int parseInt(String s)`

The **`Double`** class is part of the **`java.lang`** package and has several constants, including to hold the negative infinity and positive infinity of a **`double`**.

It also has a method to return its value as a **`double`** and to convert a **`String`** to a **`double`**.

Double
`NEGATIVE_INFINITY : double` `POSITIVE_INFINITY : double`
`double doubleValue()` `static double parseDouble(String s)`

```
Integer(int value)
```

constructs a new **Integer** object that represents the specified **int** value

```
Integer someInteger = new Integer(10);
```

```
Double(double value)
```

constructs a new **Double** object that represents the specified **double** value

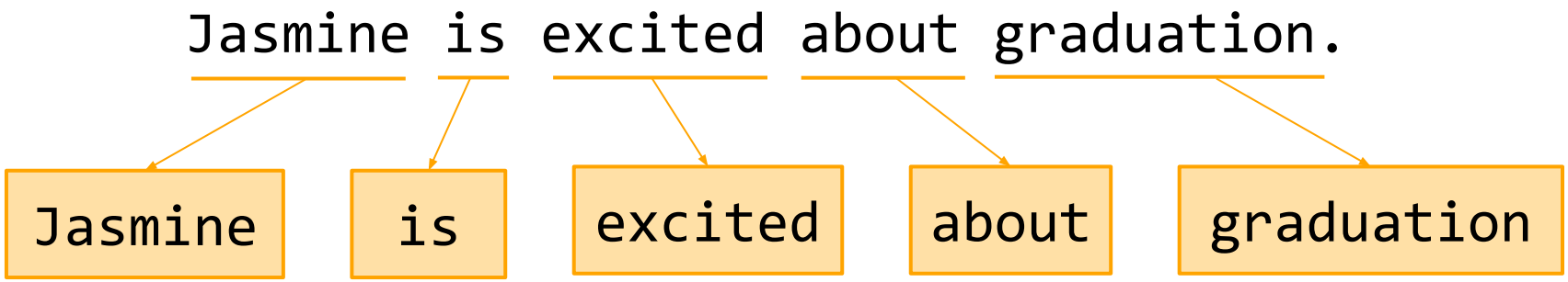
```
Double someDouble = new Double(4.5);
```

🎥 Wrapper Classes: Part 2

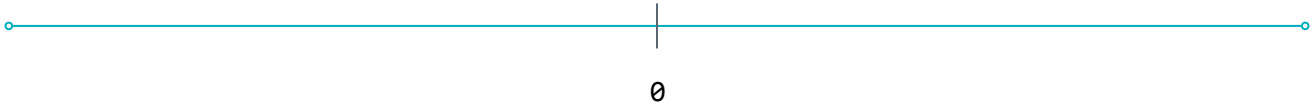
What might you use these methods for?

Complete the guided notes on the 📝 **Unit 6 Guide**.

Parsing is the process of **dividing text** into **parts** for analysis or conversion.



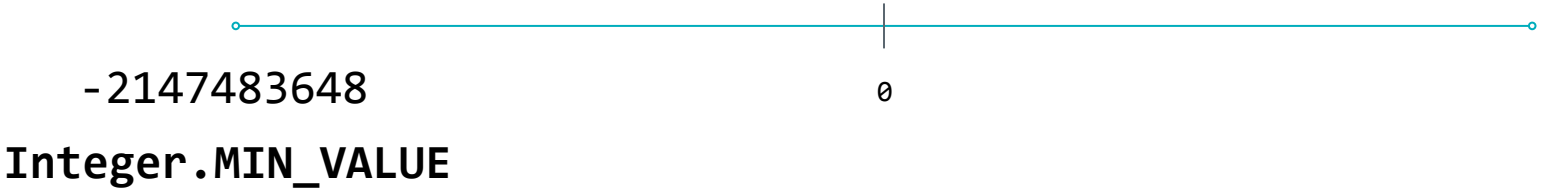
Integer Ranges in Java



Each integer in Java has **32 bits** of space, which represents whole numbers from **-2147483648** to **2147483647**.

📝 Unit 6 Guide

Integer Ranges in Java



`Integer.MIN_VALUE` is a constant in the `Integer` class that stores the **minimum** possible `int`, `-2,147,483,648`.

📝 **Unit 6 Guide**

Integer Ranges in Java

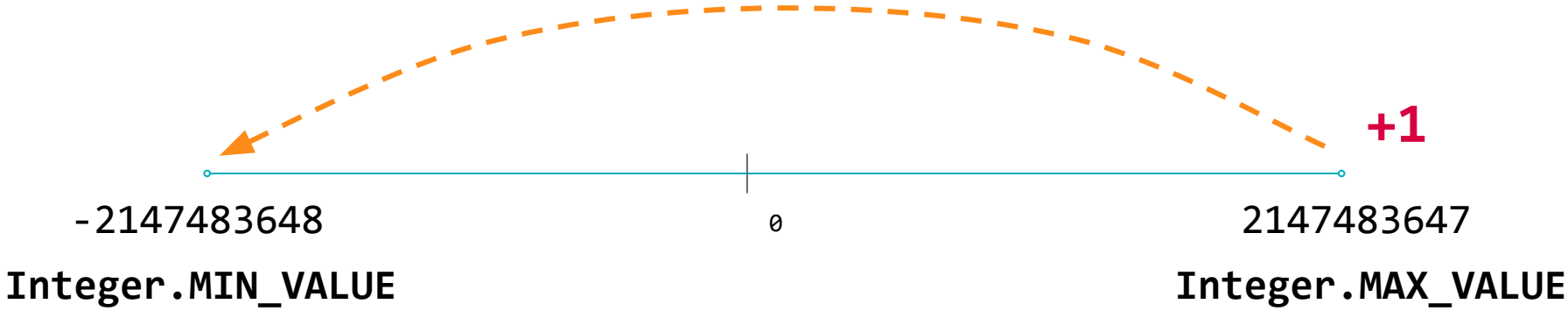
-2147483648 | 0 | 2147483647

`Integer.MIN_VALUE` | `Integer.MAX_VALUE`

`Integer.MAX_VALUE` is a constant in the `Integer` class that stores the **maximum** possible `int`, `2,147,483,647`.

📝 **Unit 6 Guide**

Integer Ranges in Java

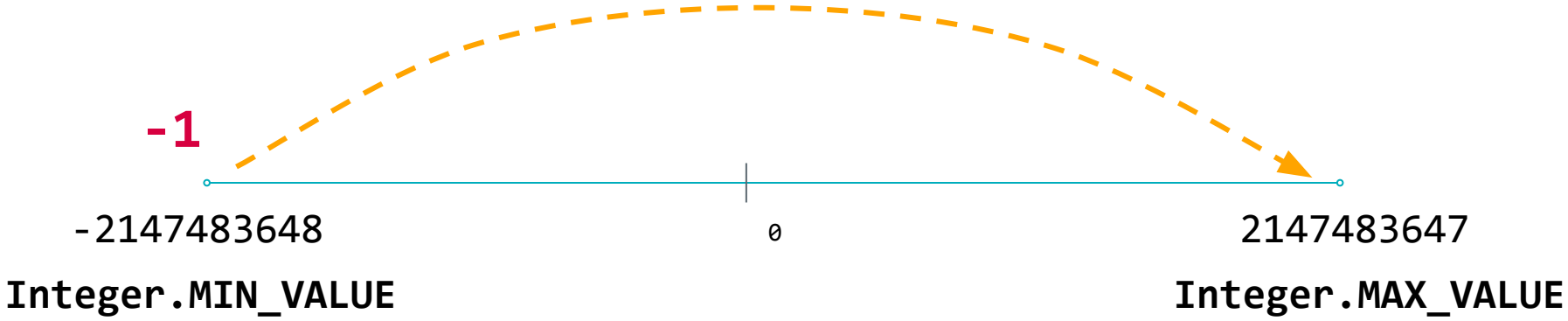


An **overflow error** is an error that occurs when an operation makes an integer value **greater** than its **maximum**.

Adding **1** to **MAX_VALUE** sets it to the **minimum**.

📝 **Unit 6 Guide**

Integer Ranges in Java



An **underflow error** is an error that occurs when an operation makes an integer value **less than** its **minimum**.

Subtracting **1** from `MIN_VALUE` sets it to the **maximum**.

Autoboxing automatically converts a **primitive type** value into an object of the corresponding **wrapper class**.

```
int x = 64;
System.out.println(x);
Integer y = x;
System.out.println(y.intValue());
```

primitive (int)

object (Integer)

```
> 64
  64
```

📝 Unit 6 Guide

Unboxing automatically converts an **object of a wrapper class** to its corresponding **primitive type**.

```
Integer x = new Integer(64);
int convertedInt = x;
Double y = new Double(2.99);
double convertedDbl = y;
```

unboxes into an `int`

unboxes into a `double`

💼 Key Vocabulary

- **wrapper class:** a class used to convert primitive data types into objects
- **parsing:** the process of dividing text into parts for analysis or conversion
- **overflow error:** an error that occurs when an operation makes an integer value greater than its maximum
- **underflow error:** an error that occurs when an operation makes an integer value less than its minimum
- **autoboxing:** the process of automatically converting a primitive type value into an object of the corresponding wrapper class
- **unboxing:** automatically converting an object of a wrapper class to its corresponding primitive type

Unit 6 - Lesson 4
ArrayLists

Computer Science A

🚀 Question of the Day

Why would I use an **`ArrayList`** instead of an array?

🎥 The ArrayList Class

How is an **`ArrayList`** different from a 1D array?

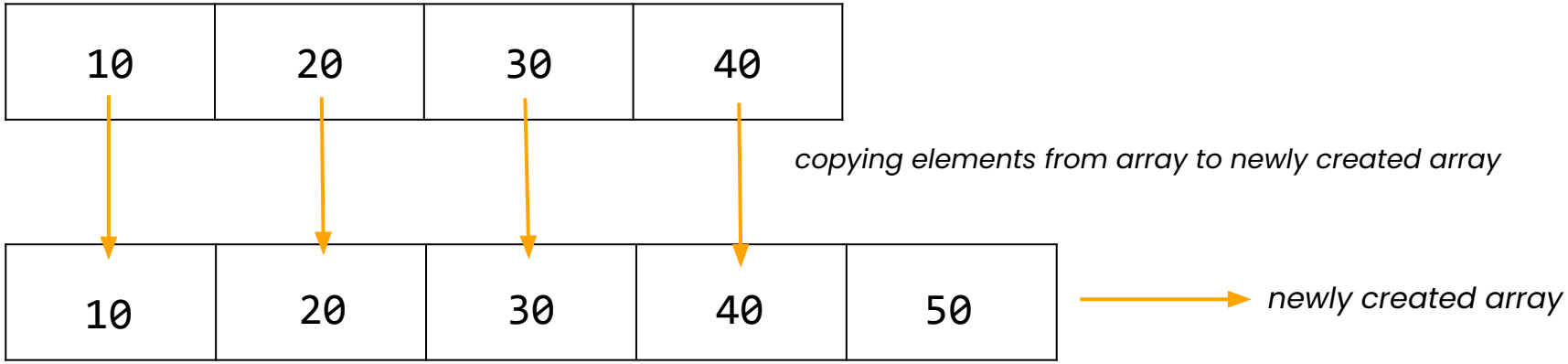
Complete the guided notes on the 📝 **Unit 6 Guide**.

A **static data structure** is a data structure that is **fixed in size**. For example, a 1D or 2D array.

10	20	30	40	50

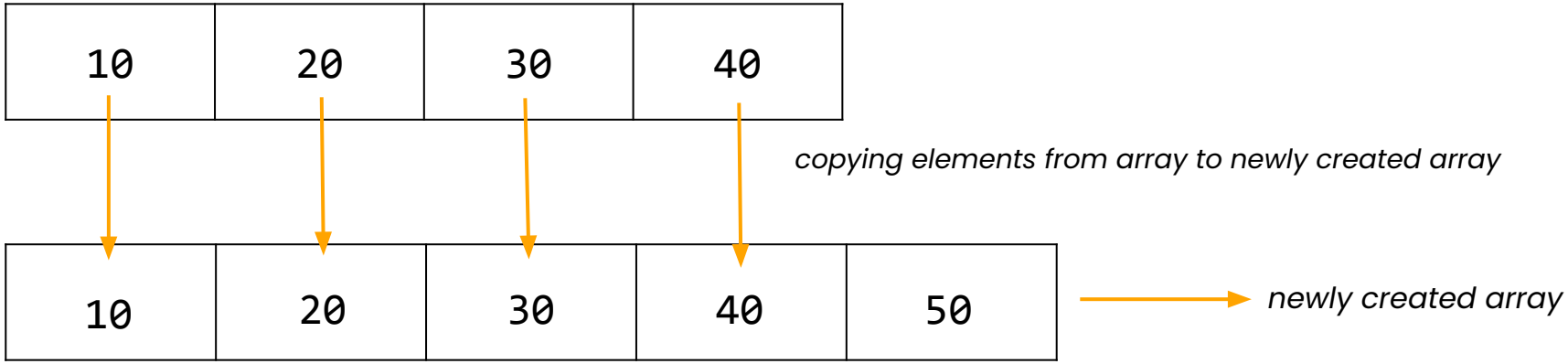
|---------- *fixed size* ----------|

A **dynamic data structure** is a data structure that **grows** and **shrinks** as needed. For example, an **`ArrayList`**.



📝 **Unit 6 Guide**

An **`ArrayList`** is a class that represents a **resizable list**.



`ArrayLists` are **mutable** . This means that the number of items they store **can change** after the list has been initialized.

The **`ArrayList`** class is part of the **`java.util`** package.

To use an **`ArrayList`** in our programs, we need to import **`java.util.ArrayList`**.

Java Lab Documentation

- Java Lab Shortcuts
- Java Basics
- org.code.theater
- org.code.media
- org.code.neighborhood
- java.io
- java.util
- java.lang
- Control Structures
- Data Structures

ArrayList

Category: java.util

The `ArrayList` class is part of the `java.util` package.

Syntax

```
ArrayList<DataType> name = new ArrayList<DataType>();
```

Method Details

ArrayList

```
public ArrayList()
```

The `ArrayList` constructor allows you to create a new `ArrayList` object.

Parameters

Name	Type	Description
capacity	int	the specified initial capacity of an `ArrayList`

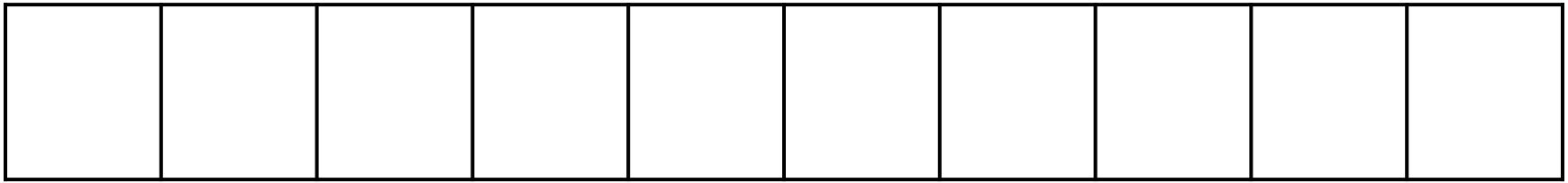
Examples

```
ArrayList<Integer> myNumbers = new ArrayList<Integer>();
ArrayList<FoodTruck> myFoodTrucks = new ArrayList<FoodTruck>();
```

To create an **ArrayList**, we call its constructor.

```
ArrayList<Integer> numbers = new ArrayList<Integer>();
```

This constructor creates a new empty **ArrayList** that can store **Integer** objects.



The **`ArrayList`** class has two versions of the **`add()`** method.

`numbers.add(30);` adds the element to the end of the **`ArrayList`**

10	20	30
0	1	2

`numbers.add(1, 30);` adds the element at index **`1`** of the **`ArrayList`** and shifts the rest of the elements to the right by one position

10	30	20
0	1	2

The **`ArrayList`** class also has a **`size()`** method that returns the **number of elements** in the list.

💼 Key Vocabulary

- **static data structure:** a data structure that is fixed in size
- **dynamic data structure:** a data structure that grows and shrinks as needed
- **mutable:** the ability to change after initialization

Unit 6 - Lesson 5
Manipulating Elements

Computer Science A

🚀 Question of the Day

How am I able to work with the data stored in an **`ArrayList`**?

🎥 Working with ArrayList Data

How is working with elements in an **`ArrayList`** similar and different from a 1D array?

Complete the guided notes on the 📝 **Unit 6 Guide**.

The **`E get(int index)`** method returns the element at position **`index`** in the list.

```
ArrayList<String> teamList = new ArrayList<String>();
teamList.add("Falcons");
teamList.add("Bears");
teamList.add("Titans");
System.out.println(teamList);
System.out.println(teamList.get(0));
System.out.println(teamList.get(2));
```

```
[Falcons, Bears, Titans]
Falcons
Titans
```

The **`E set(int index, E obj)`** method replaces the element at **`index`** with **`obj`** and returns the element that was replaced.

```
ArrayList<String> teamList = new ArrayList<String>();
teamList.add("Falcons");
teamList.add("Bears");
teamList.add("Titans");
System.out.println(teamList);
System.out.println(teamList.get(0));
System.out.println(teamList.get(2));
teamList.set(1, "Ravens");
System.out.println(teamList);
```

```
[Falcons, Bears, Titans]
Falcons
Titans
[Falcons, Ravens, Titans]
```

Since the indices for an **`ArrayList`** start at **`0`** and end at the number of elements - **`1`**, accessing an index value outside of this range will result in an **`IndexOutOfBoundsException`** being thrown.

```
for (int index = 0; index < myList.size(); index++) {
  System.out.println(myList.get(index));

  int currentValue = myList.get(index);
  System.out.println(currentValue);
}
```

We use the **`get()`** method to obtain the current value.

We can also store the current value in a variable. This is useful when we need to perform multiple tasks on the element.