

# Unit 6 - Lesson 1

## Project Planning

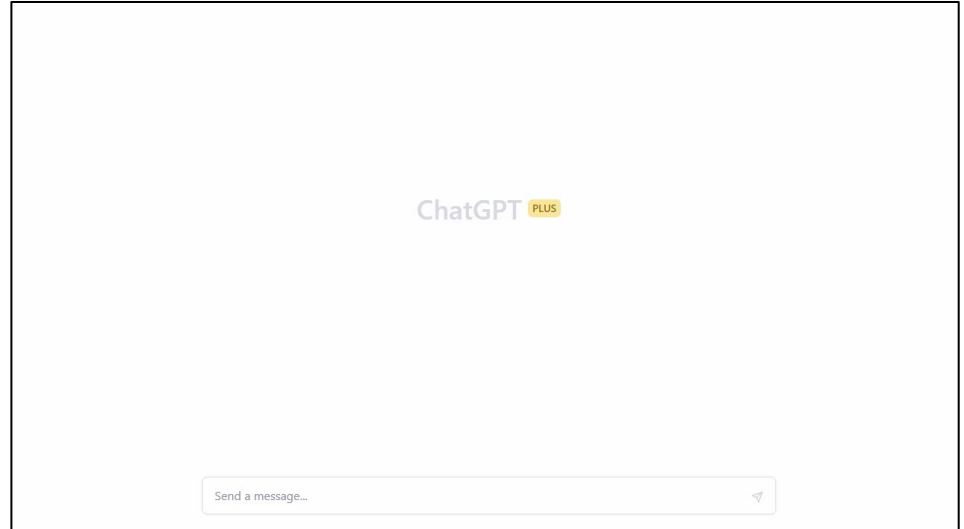


Computer Science A

# Warm Up



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



## Software Engineering: Natural Language Processing

How do different apps and programs use natural language processing?

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



## Retrieve

your knowledge and ideas and write it down silently



## Pair

up with a neighbor and talk about your reflections

## Share

your thoughts in a class discussion



## Discuss:

- ▶ What are some other examples of where you have seen or experienced NLP?
- ▶ What are some examples of where you think NLP could be useful?



# Activity



# Lesson Objectives

By the end of this lesson, you will be able to . . .

- Identify project requirements
- Write Javadocs comments to create program documentation



## Question of the Day

How can I identify the requirements and things I need to know to complete a project?

# Project Requirements

- **Create at least two ArrayLists** – Create at least two `ArrayLists` to store the data used in your program, such as data from text files or entered by the user
- **Implement or more algorithms** – Implement one or more algorithms that use loops and conditionals to find or manipulate elements in an `ArrayList` or `String` object
- **Use methods in the String class** – Use or more methods in the `String` class in your program, such as to divide text into sentences or phrases
- **Use at least one natural language processing technique** – Use a natural language processing technique to process, analyze, and/or generate text
- **Document your code** – Use comments to explain the purpose of the methods and code segments and note preconditions and postconditions

## Retrieve

your knowledge  
and ideas and write  
it down silently



## Pair

up with a neighbor  
and talk about your  
reflections

## Share

your thoughts in a  
class discussion



## Discuss:

- ▶ What are things in the project requirements that we **already know**?
- ▶ What are some things we **need to know**?






# Project Characteristics

 **You and your partner should have:**

- Project Characteristics activity guide
- pen / pencil



Name(s) \_\_\_\_\_ Period \_\_\_\_\_ Date \_\_\_\_\_

**Activity Guide - Project Characteristics** 

Review and analyze each of the project examples on Code Studio. As you review, use the box below to generate a list of some common characteristics and key features that you see. **This is simply an investigation to see what you notice.** We will cover some of the main project requirements throughout the lessons of this unit.

**Common Characteristics and Features**

1

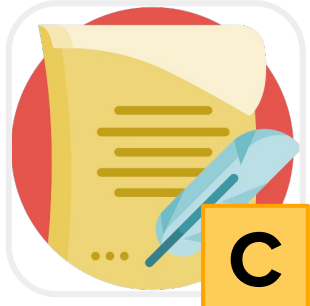
# Investigate



**A**



**B**



**C**



**D**



Navigate to Lesson 1, Level 1

## Do This:

1. Explore each project example
2. Identify common **characteristics** and key **features** of the projects
3. Complete the **Project Characteristics** activity guide



## Discuss:

What **words** and **phrases** are in the **requirements** that align with your list of **characteristics** and **features**?



## Retrieve

your knowledge and ideas and write it down silently



## Pair

up with a neighbor and talk about your reflections

## Share

your thoughts in a class discussion



## Discuss:

Based on our experiences with **writing** and **reading** each other's code, what are some **best practices** we should follow when **writing and documenting code**?



**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 <code>Painter</code> object
int	yLocation	the y coordinate of the <code>Painter</code> object
String	direction	the direction the <code>Painter</code> object is facing ( "North", "South", "East", or "West" )
int	remainingPaint	the number of units of paint the <code>Painter</code> 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();
```

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

```

/**
 * 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

```

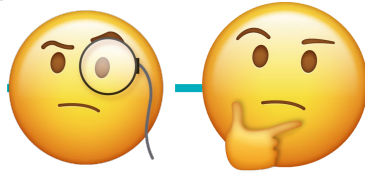
**Description of the code segment**

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





## Discuss: What do you notice? What do you wonder?



```
/**
 * 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 intent) {
```

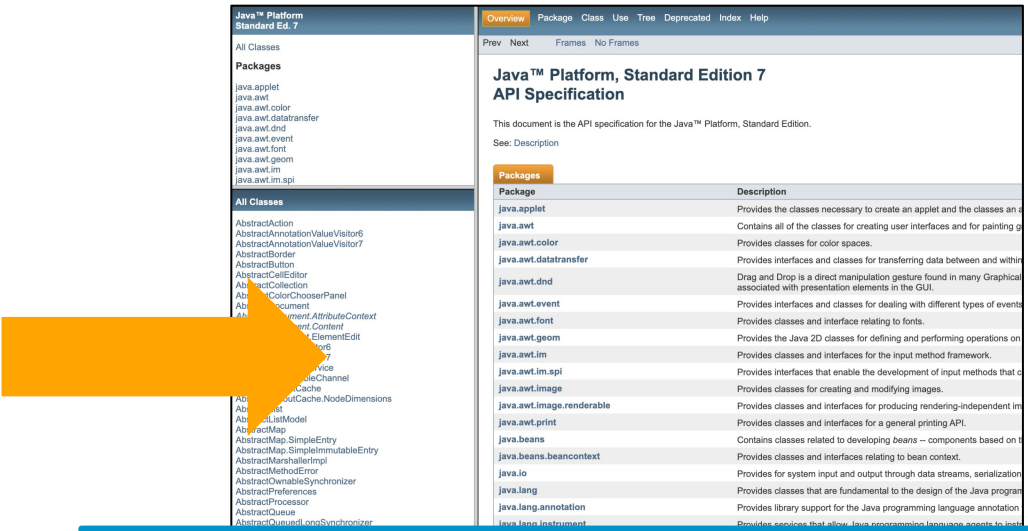


# Javadocs generates an HTML document from comments written inside `/** */` and formatted using `@` tags.

```

/**
 * 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.





## 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.





# Practice

## Javadocs Tags

```
@param parameterName description  
@return description
```



Navigate to Lesson 1, Level 2



## Do This:

1. Add documentation to your **Dessert** class
2. Have your neighbor look over your comments for feedback

# Wrap Up



# Three W's

1. What did we learn today?
2. So what?
3. Now what?





## **Today, you learned about . . .**

- How to identify project requirements
- Writing Javadocs comments to create program documentation



## Question of the Day

How can I identify the requirements and things I need to know to complete a project?



# 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