Day-Four

Notes

Git (How to create Branch or Pull)

1. How to create Branch

```
git checkout -b branch_name  # This command both creates and switches to the new branch.

git branch branch_name  # Just to create the branch
git checkout branch_name  # Switch to it manually
```

2. How to Pull Repository

```

cd /path/to/your/repo # **Navigate to your repository:*


git checkout branch_name # **Ensure you're on the correct branch (e.g., `main` or `branch_name`):**

git pull origin branch_name # **Pull the latest changes from the remote repository:**
```

Markdown File

1. Grammar : https://www.markdownguide.org/
   Markdown is a lightweight markup language commonly used to format text in a simple, readable way. It’s widely used for documentation, README files, and content on platforms like GitHub and GitLab.

Markdown is used because it's:

1. **Simple and Readable**: Easy to write and understand in plain text, unlike HTML.

2. **Lightweight**: No need for heavy formatting tools—works in any text editor.
3. **Widely Supported**: Platforms like GitHub, GitLab, and Slack use it for documentation and communication.
4. **Cross-platform**: Consistent across devices and editors.
5. **Version Control Friendly**: Works well with Git, making it ideal for project documentation.

It exists to simplify text formatting while maintaining readability in its raw form.

1. Headings

```
Use `#` for headings. More `#` symbols indicate smaller heading levels.
```

```
#### H1 Heading
#### H2 Heading
#### H3 Heading
#### H4 Heading
```

2. Bold and Italic Text

- **Bold**: Wrap the text with two asterisks (`**`) or underscores (`__`).
- *Italic*: Wrap the text with one asterisk (`*`) or underscore (`_`).
- ***Bold and Italic***: Wrap the text with three asterisks (`***`) or underscores (`___`).

```
**Bold text**
*Italic text*
***Bold and Italic text***
```

3. Lists

- **Unordered lists**: Use `*`, `-`, or `+` followed by a space.
- **Ordered lists**: Use numbers followed by a period (`1.`).

```
* Item 1
- Item 2
+ Item 3

1. First item
```

```
2. Second item
3. Third item
```

4. Links

To create a hyperlink, use the following syntax:

```
[Link text](URL)
```

Example:

```
[GitHub](https://github.com)
```

5. Images

Images are similar to links but start with an exclamation mark (`!`):

```
![Alt text](image_url)
```

Example:

```
![Markdown logo](https://markdown-here.com/img/icon256.png)
```

6. Code Blocks and Inline Code

- **Inline code**: Wrap the code with a single backtick (`` ` ``).
- **Code blocks**: Wrap the code with triple backticks ( ).

```
Inline code: `print("Hello, World!")`

Code block:
```

def hello():
print("Hello, World!")

```

```

7. Blockquotes

Use the `>` symbol to create blockquotes:

```
> This is a blockquote.
```

8. Horizontal Rule

Use three or more hyphens (`---`), asterisks (`***`), or underscores (`___`) to create a horizontal rule:

```
---
```

9. Tables

Create tables using pipes (`|`) and dashes (`-`) for the header row:

```
| Header 1 | Header 2 |
|----------|----------|
| Cell 1   | Cell 2   |
| Cell 3   | Cell 4   |
```

10. Task Lists

Use `- [ ]` for an unchecked task and `- [x]` for a checked task.

```
- [ ] Incomplete task
- [x] Completed task
```

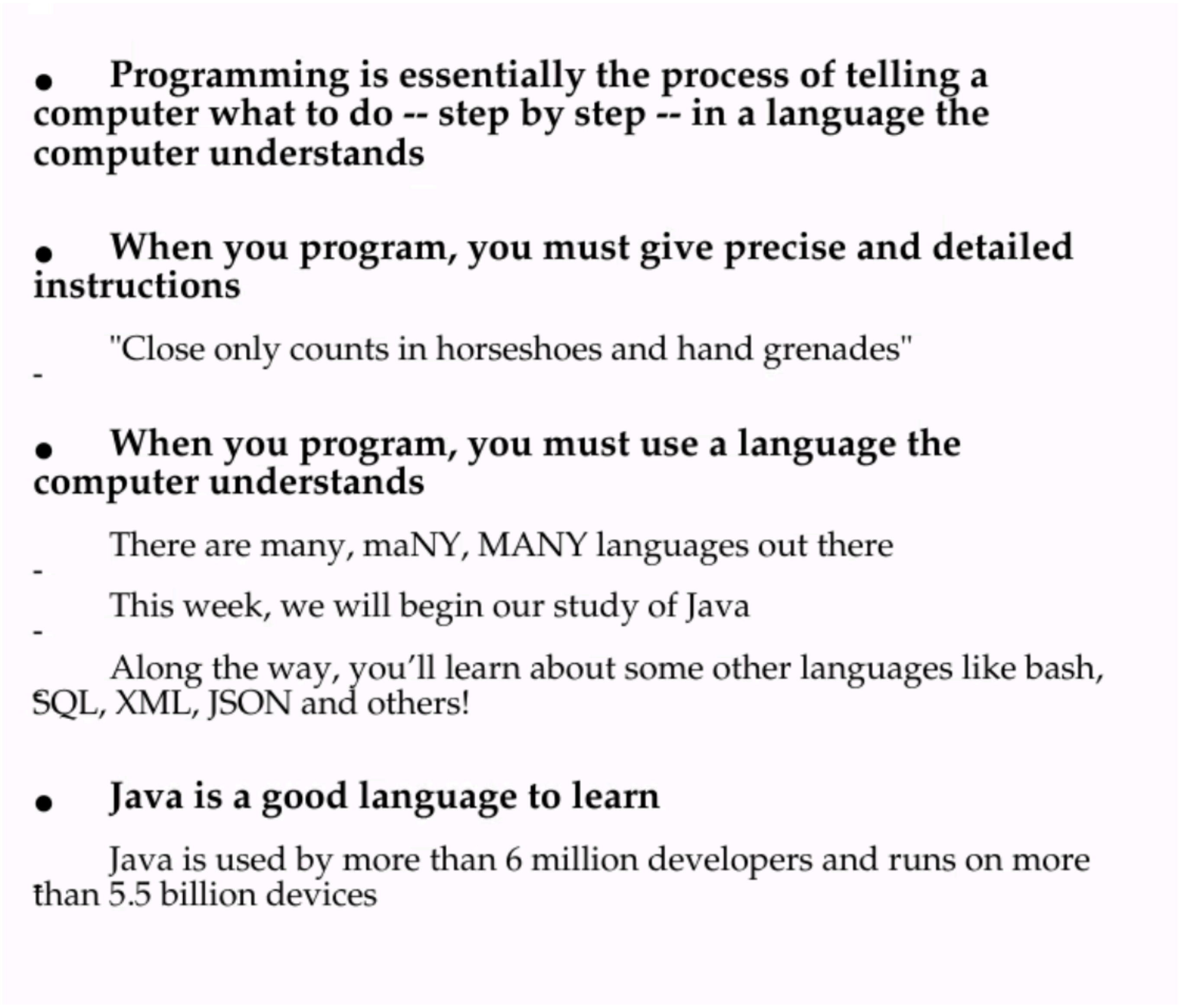
11. Escaping Characters

To display special Markdown characters (like `*`, `_`, `#`), use a backslash (`\`):

```
\*Escaped asterisk\*
```

These are the most common Markdown syntax elements. You can mix and match them to format your content effectively!

Java Fundamental SE / Shortcut IntelIJ IDEA



- **Programming is essentially the process of telling a computer what to do -- step by step -- in a language the computer understands**

- **When you program, you must give precise and detailed instructions**
  - "Close only counts in horseshoes and hand grenades"

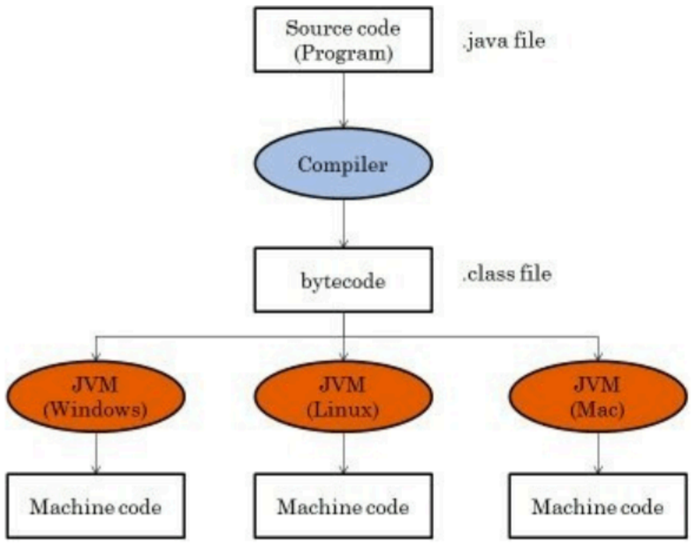
- **When you program, you must use a language the computer understands**
  - There are many, maNY, MANY languages out there
  - This week, we will begin our study of Java
  - Along the way, you'll learn about some other languages like bash, SQL, XML, JSON and others!

- **Java is a good language to learn**
  - Java is used by more than 6 million developers and runs on more than 5.5 billion devices

![[Screenshot 2024-09-26 at 10.23.42.png]

Java Bytecode and the Java Virtual Machine (JVM)

- **When a Java program is compiled, it is not translated into the machine code native to the developer's computer**
  - This would mean it could only run on similar computers
- **Instead, Java programs are translated into something called Java *bytecode***
  - Bytecode is stored in a `.class` file and is eventually executed on a computer by using a JVM
- **The Java Virtual Machine (JVM) is the program responsible for loading and executing a Java application**
  - It executes the Java bytecode instructions
  - There are several JVMs available depending on the type of computer you use



Java Runtime Environment (JRE)

- **You must install the Java Runtime Environment (JRE) in order to *run* Java applications on your computer**
- **The JRE is a software layer that sits on top of a computer's native operating system**
- **It provides the class libraries and other resources (including the JVM) needed to run Java applications**
- **However, the JRE does not contain the resources a developer needs for creating Java applications**

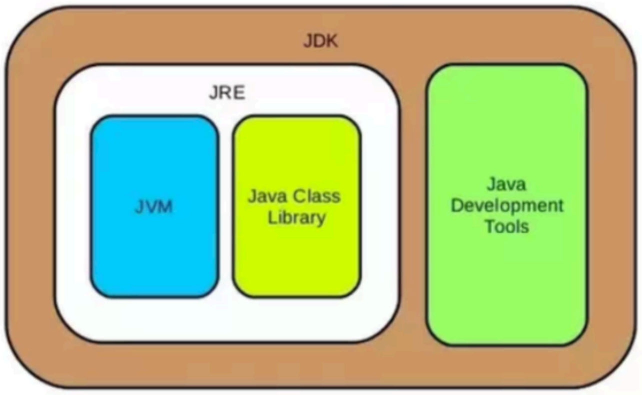
Java Developer Kit (JDK)

- **The Java Developer Kit (JDK) is set of tools for developing Java applications**
  - It includes a CLI, the Java compiler, debuggers, etc.

- **If you want to create Java applications, you must install a JDK**

- **Developers choose JDKs by Java version and by package or edition—**
  - Java Standard Edition (Java SE) <-- We want this one!
  - Java Enterprise Edition (Java EE)
  - Java Mobile Edition (Java ME)

- **The JDK also includes a compatible JRE because so that the Java application can be tested and run on the developer's machine**



Compiling Java

- **To compile a Java application into bytecode, you use** `javac`

- `javac` **is a Java compiler**
  - It confirms the grammar you coded is correct
  - It then translates the code into bytecode and places it in a `.class` file

- **If you have syntax errors, the compiler will not generate the bytecode**

- **This is the difference between a compiled language like Java and an interpreted language like JavaScript**
  - You can't even try to run the code if there are syntax errors

Configuring your Development Machine

- **If you don’t have a JDK, you can download the version you need here**

  https://www.oracle.com/java/technologies/downloads/

- **Once you have installed a JDK on your development machine, you will need to do some (hopefully) simple configurations**

- **Step 1: Configure the JAVA_HOME environment variable**

  On a Windows machine, this will be something like:
  `C:\Program Files\Java\jdk-17`

- **Step 2: Add the `bin` folder containing Java developer tools to your PATH environment variable**

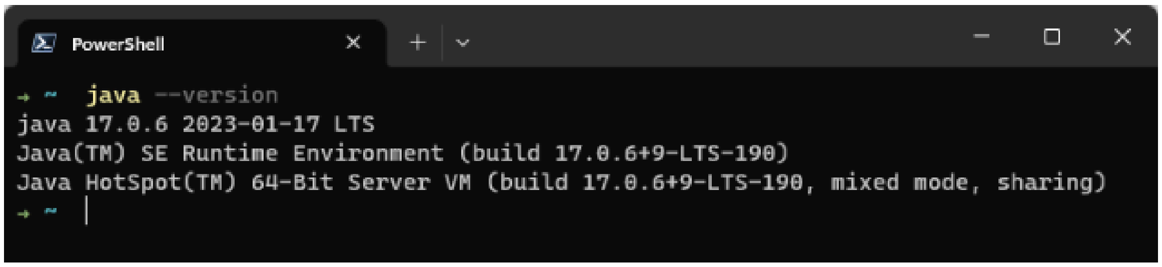
  On a Windows machine, this will be something like:
  `C:\Program Files\Java\jdk-17\bin`

- **Step 3 (possible): If your path variable includes an Oracle Java reference, either move it to the bottom of the PATH list or remove it**

Verify Java Configuration

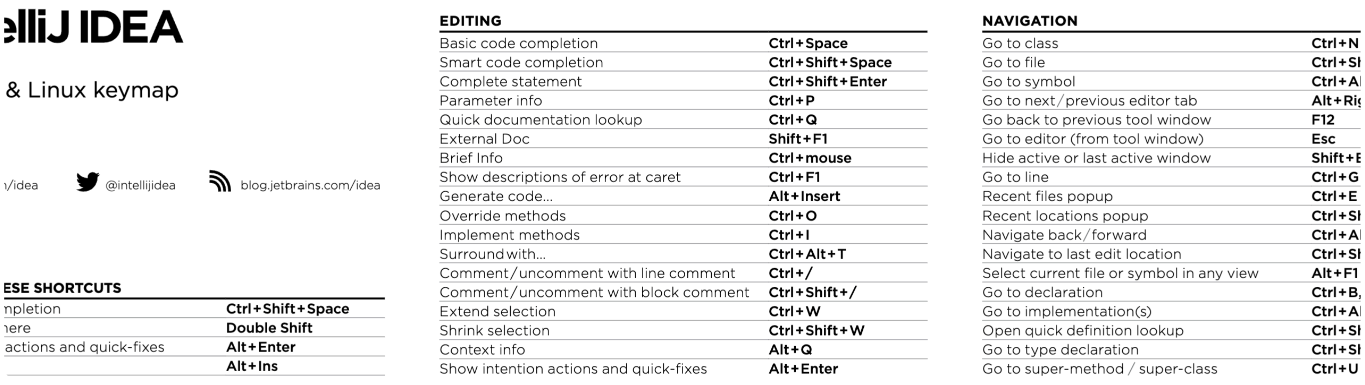
- Open Windows Terminal and execute the following command

```
java --version
```



```
→ ~  java --version
java 17.0.6 2023-01-17 LTS
Java(TM) SE Runtime Environment (build 17.0.6+9-LTS-190)
Java HotSpot(TM) 64-Bit Server VM (build 17.0.6+9-LTS-190, mixed mode, sharing)
→ ~
```

How to use the Shortout Of IntelIJ IDEA



elliJ IDEA

& Linux keymap

/idea @intellijidea blog.jetbrains.com/idea

ESE SHORTCUTS

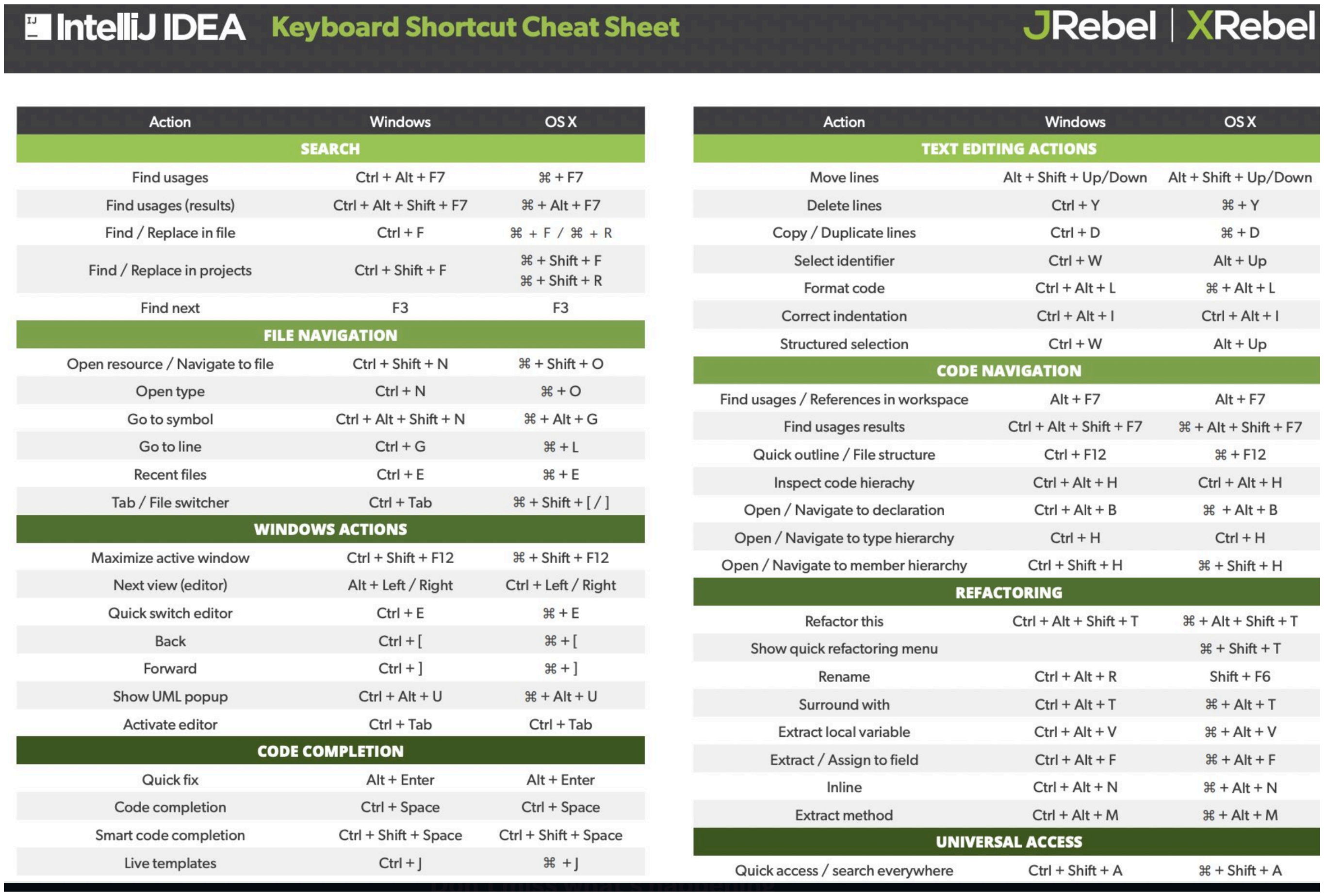
npletion	Ctrl+Shift+Space
nere	Double Shift
actions and quick-fixes	Alt+Enter
	Alt+Ins

EDITING

Basic code completion	Ctrl+Space
Smart code completion	Ctrl+Shift+Space
Complete statement	Ctrl+Shift+Enter
Parameter info	Ctrl+P
Quick documentation lookup	Ctrl+Q
External Doc	Shift+F1
Brief Info	Ctrl+mouse
Show descriptions of error at caret	Ctrl+F1
Generate code...	Alt+Insert
Override methods	Ctrl+O
Implement methods	Ctrl+I
Surround with...	Ctrl+Alt+T
Comment/uncomment with line comment	Ctrl+/
Comment/uncomment with block comment	Ctrl+Shift+/
Extend selection	Ctrl+W
Shrink selection	Ctrl+Shift+W
Context info	Alt+Q
Show intention actions and quick-fixes	Alt+Enter

NAVIGATION

Go to class	Ctrl+N
Go to file	Ctrl+Sl
Go to symbol	Ctrl+Al
Go to next/previous editor tab	Alt+Rig
Go back to previous tool window	F12
Go to editor (from tool window)	Esc
Hide active or last active window	Shift+E
Go to line	Ctrl+G
Recent files popup	Ctrl+E
Recent locations popup	Ctrl+Sl
Navigate back/forward	Ctrl+Al
Navigate to last edit location	Ctrl+Sl
Select current file or symbol in any view	Alt+F1
Go to declaration	Ctrl+B,
Go to implementation(s)	Ctrl+Al
Open quick definition lookup	Ctrl+Sl
Go to type declaration	Ctrl+Sl
Go to super-method / super-class	Ctrl+U



IntelliJ IDEA **Keyboard Shortcut Cheat Sheet** JRebel | XRebel

Action	Windows	OS X
SEARCH		
Find usages	Ctrl + Alt + F7	⌘ + F7
Find usages (results)	Ctrl + Alt + Shift + F7	⌘ + Alt + F7
Find / Replace in file	Ctrl + F	⌘ + F / ⌘ + R
Find / Replace in projects	Ctrl + Shift + F	⌘ + Shift + F ⌘ + Shift + R
Find next	F3	F3
FILE NAVIGATION		
Open resource / Navigate to file	Ctrl + Shift + N	⌘ + Shift + O
Open type	Ctrl + N	⌘ + O
Go to symbol	Ctrl + Alt + Shift + N	⌘ + Alt + G
Go to line	Ctrl + G	⌘ + L
Recent files	Ctrl + E	⌘ + E
Tab / File switcher	Ctrl + Tab	⌘ + Shift + [/]
WINDOWS ACTIONS		
Maximize active window	Ctrl + Shift + F12	⌘ + Shift + F12
Next view (editor)	Alt + Left / Right	Ctrl + Left / Right
Quick switch editor	Ctrl + E	⌘ + E
Back	Ctrl + [	⌘ + [
Forward	Ctrl +]	⌘ +]
Show UML popup	Ctrl + Alt + U	⌘ + Alt + U
Activate editor	Ctrl + Tab	Ctrl + Tab
CODE COMPLETION		
Quick fix	Alt + Enter	Alt + Enter
Code completion	Ctrl + Space	Ctrl + Space
Smart code completion	Ctrl + Shift + Space	Ctrl + Shift + Space
Live templates	Ctrl + J	⌘ + J

Action	Windows	OS X
TEXT EDITING ACTIONS		
Move lines	Alt + Shift + Up/Down	Alt + Shift + Up/Down
Delete lines	Ctrl + Y	⌘ + Y
Copy / Duplicate lines	Ctrl + D	⌘ + D
Select identifier	Ctrl + W	Alt + Up
Format code	Ctrl + Alt + L	⌘ + Alt + L
Correct indentation	Ctrl + Alt + I	Ctrl + Alt + I
Structured selection	Ctrl + W	Alt + Up
CODE NAVIGATION		
Find usages / References in workspace	Alt + F7	Alt + F7
Find usages results	Ctrl + Alt + Shift + F7	⌘ + Alt + Shift + F7
Quick outline / File structure	Ctrl + F12	⌘ + F12
Inspect code hierachy	Ctrl + Alt + H	Ctrl + Alt + H
Open / Navigate to declaration	Ctrl + Alt + B	⌘ + Alt + B
Open / Navigate to type hierarchy	Ctrl + H	Ctrl + H
Open / Navigate to member hierarchy	Ctrl + Shift + H	⌘ + Shift + H
REFACTORING		
Refactor this	Ctrl + Alt + Shift + T	⌘ + Alt + Shift + T
Show quick refactoring menu		⌘ + Shift + T
Rename	Ctrl + Alt + R	Shift + F6
Surround with	Ctrl + Alt + T	⌘ + Alt + T
Extract local variable	Ctrl + Alt + V	⌘ + Alt + V
Extract / Assign to field	Ctrl + Alt + F	⌘ + Alt + F
Inline	Ctrl + Alt + N	⌘ + Alt + N
Extract method	Ctrl + Alt + M	⌘ + Alt + M
UNIVERSAL ACCESS		
Quick access / search everywhere	Ctrl + Shift + A	⌘ + Shift + A

Java basic

Comments or block comments

```
1.`//                    Mac: Command  + /   |  windows control + /`
2. `Block comments are enclosed in slash-asterisk (/*) and asterisk-slash
(*/).       To add a block comment in IntelliJ IDEA, press ⌥ ⌘ / macOS
or Ctrl + Shift + / on Windows and Linux.`
```

Data Type

In Java, data types specify the size and type of data that can be stored in a variable. Java has two main categories of data types:

1. **Primitive Data Types**: These are the basic data types that are built into the language. There are eight primitive data types in Java:
   - **byte**: An 8-bit signed integer.
   - **short**: A 16-bit signed integer.
   - **int**: A 32-bit signed integer.
   - **long**: A 64-bit signed integer.
   - **float**: A single-precision 32-bit IEEE 754 floating point.
   - **double**: A double-precision 64-bit IEEE 754 floating point.
   - **char**: A single 16-bit Unicode character.
   - **boolean**: A type that can hold one of two values: `true` or `false`.
2. **Reference Data Types**: These types refer to objects and are created using classes. They include:
   - **Strings**: Used to store a sequence of characters.
   - **Arrays**: Used to store multiple values of the same type.
   - **Classes**: User-defined data types that can contain fields and methods.

Examples and Explanations

1. Primitive Data Types

Example:

```java
public class PrimitiveDataTypes {
    public static void main(String[] args) {
        // Integer Types
        byte b = 100; // 8-bit
        short s = 10000; // 16-bit
        int i = 100000; // 32-bit
        long l = 100000L; // 64-bit, 'L' denotes a long literal

        // Floating Point Types
        float f = 10.5f; // 32-bit, 'f' denotes a float literal
        double d = 20.5; // 64-bit

        // Character Type
        char c = 'A'; // 16-bit character
        // Boolean Type
        boolean bool = true; // can be true or false

        // Output
        System.out.println("byte: " + b);
        System.out.println("short: " + s);
        System.out.println("int: " + i);
        System.out.println("long: " + l);
        System.out.println("float: " + f);
        System.out.println("double: " + d);
        System.out.println("char: " + c);
        System.out.println("boolean: " + bool);
    }
}
```

Explanation:

- `byte`, `short`, `int`, and `long` are used to store integer values. The difference lies in the size and range of values they can hold.
- `float` and `double` are used for decimal values; `double` has a larger range and is more precise.
- `char` is used to store a single character, while `boolean` can only hold `true` or `false`.

2. Reference Data Types

Example:

```
public class ReferenceDataTypes {
    public static void main(String[] args) {
        // String
        String str = "Hello, World!";

        // Array
        int[] arr = {1, 2, 3, 4, 5}; // Array of integers

        // Output
        System.out.println("String: " + str);
        System.out.println("Array: ");
        for (int num : arr) {
            System.out.println(num);
        }
    }
}
```

Explanation:

- `String` is a class in Java that represents a sequence of characters. It is immutable, meaning that once created, it cannot be changed.
- Arrays are used to store multiple values of the same type in a single variable. They have a fixed size and can hold elements of a specified data type.

Summary

- **Primitive Data Types** are the basic building blocks of data in Java, each serving a different purpose and size.
- **Reference Data Types** allow for more complex data structures, capable of storing multiple values or objects.

Code Exercise

```java
import java.util.Random;
import java.math.BigDecimal;

public class DataType extends Object{


    private int identificationNumber;
    private String make;
    private String model;
    private String color;
    private boolean towing;
    private int odommeter;
    private BigDecimal price;        //   double
    private char qualityRating;
    private String phoneNumber;
    private String SSN;
    private String zipCode;

    public void setDataType(int identificationNumber, String make, String
model, String color, boolean towing, float odommeter, int price, char
qualityRating, String phoneNumber, String SSN, short zipCode) {
        this.identificationNumber = identificationNumber;
        this.make = make;
        this.model = model;
        this.color = color;
        this.towing = towing;
        this.odommeter = odommeter;
        this.price = price;
        this.qualityRating = qualityRating;
        this.phoneNumber = phoneNumber;
        this.SSN = SSN;
        this.zipCode = zipCode;
    }

    @Override
    public String toString() {
        return "DataType{" +
                "identificationNumber=" + identificationNumber +
                ", make='" + make + '\'' +
                ", model='" + model + '\'' +
                ", color='" + color + '\'' +
                ", towing=" + towing +
                ", odommeter=" + odommeter +
                ", price=" + price +
                ", qualityRating=" + qualityRating +
                ", phoneNumber='" + phoneNumber + '\'' +
                ", SSN='" + SSN + '\'' +
                ", zipCode=" + zipCode +
                '}';
    }

    public static void main(String[] args) {


        /*
        *
        * Declare each variable with the correct data types:
                ● a vehicle identification number in the range 1000000 -
 9999999                ● a vehicle make /model (i.e. Ford Explorer)
                ● a vehicle color
                ● whether the vehicle has a towing package
                ● an odometer reading
                ● a price
                ● a quality rating (A, B, or C)
                ● a phone number
                ● a social security number
                ● a zip code        * */

//        creating datatype obejct
```

```java
        DataType dataType1 = new DataType();
        dataType1.setDataType(
                new Random().nextInt(9999998), "Chevrolet Camero", "CMA",
"Purple", true, 58672F,  129999 ,'A', "917-862-8817", "088-928-9928",
(short) 11207);

//          print the object's data
        System.out.println(dataType1.toString());

        DataType dataType2 = new DataType();

        dataType2.setDataType(
                new Random().nextInt(9999998), "Toyota", "Corolla",
"Purple", true, 58672F,  129999 ,'A', "917-862-8817", "088-928-9928",
(short) 11207);

        System.out.println(dataType2.toString());


        DataType dataType = new DataType();
DataTypeDemo dataTypeDemo = new DataTypeDemo(dataType);
dataTypeDemo.setUpData(new Random().nextInt(9999998), "Toyota", "Corolla",
"Purple", true, 58672F,  129999 ,'A', "917-862-8817", "088-928-9928",
(short) 11207);
System.out.println(dataTypeDemo.toString());
    }
}

class DataTypeDemo {

    private DataType dataType;

    public DataTypeDemo(DataType dataType) {
        this.dataType = new DataType();
    }

    public void setUpData(int identificationNumber, String make, String
model, String color, boolean towing, float odommeter, short price, char
qualityRating, String phoneNumber, String SSN, short zipCode) {
        this.dataType.setDataType(identificationNumber, make, model, color,
towing, odommeter, price, qualityRating, phoneNumber, SSN, zipCode);
    }

}
```

[***Awesome-FALSEhood***](#)
[***Awesome-List***](#)

Demo1

```
package com.pluralsight;

public class Demo1 {

    public static void main(String[] args) {


        char status = 'm'; // declaration and initialization            DATA
TYPE: char
        int identifier = 1; // declaration and initialization           DATA
TYPE: int
        String name = "Yiming";   // declaration and initialization     DATA
TYPE: String

        String greeting = "Hello, " + name + "!";  // Expression DATA TYPE:
String
        int numberOfDaysPerWeekWatchTV = 5; // declaration and
initialization DATA TYPE: int
        int minutesPerDayOnDayWatchedOnAverage = 70; // declaration and
initialization DATA TYPE: int
        int minutesPerWeek = numberOfDaysPerWeekWatchTV *
minutesPerDayOnDayWatchedOnAverage; // Expression DATA TYPE: int


        int minutesPerYear = minutesPerWeek * 52; // Expression DATA TYPE:
int
        int hoursPerYear = minutesPerYear / 60; // Expression DATA TYPE: int

        int hoursInWorkWeek = 40; // declaration and initialization DATA
TYPE: int
        int workWeeksPerYearWatchingTV = hoursPerYear / hoursInWorkWeek; //
Expression DATA TYPE: int
                int workWeeksPerYearWatchingTV2 = (5 * 70 * 52 / 62) / 35

        System.out.println("If " + name + " did not watch TV during the work
week, they would have worked " + workWeeksPerYearWatchingTV + " weeks per
year."); // Statement  DATA TYPE: void
        System.out.println("Hours watched per year: " + hoursPerYear); //
Statement  DATA TYPE: void
```

```
        System.out.println("Total minutes watched: " + minutesPerWeek); // 
Statement  DATA TYPE: void
        System.out.println(greeting); // Statement  DATA TYPE: void

    }
}
```

Question/Inspired

1. ***Inspired*** : I am inspired that markdown file is so good to take notes.
2. ***Question***: why String is not primitive data type but rest of them are primitive?