

# KTP Coding Workshop

---

Coding in C++: Basics

# Chapters

**01**

Introduction to  
C++

**02**

Basic Syntax &  
Structure

**03**

Data Types &  
Variables

**04**

Operators

**05**

Control  
Structures

**06**

Functions

# Chapters (continued)

**07**

**Arrays**

**08**

**Pointers &  
References**

**09**

**Introduction to  
Object-Oriented  
Programming**

**10**

**Conclusion &  
Q&A**

# 01

# Introduction

# to C++

---



# **Importance of C++**

# C++ Environment Options

Options for C++ Environments:

- Visual Studio Code  
(recommended)
- Jupyter Notebooks
- Many many more

# Visual Studio Code Setup

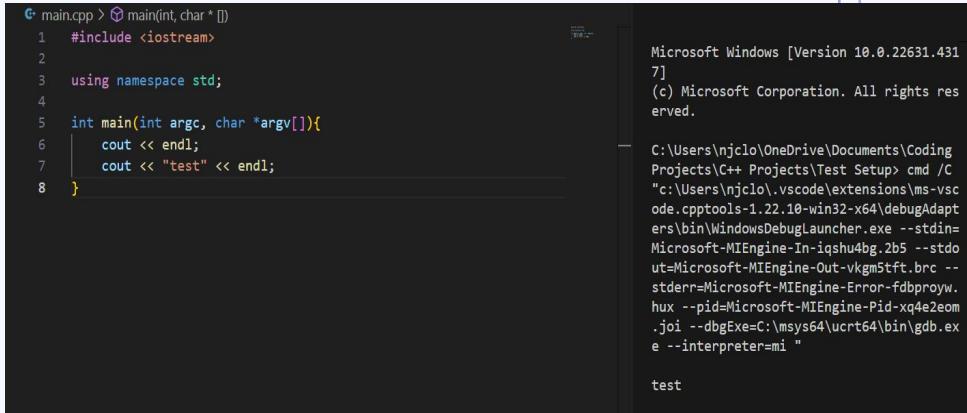
- Download Visual Studio Code
- Install both C/C++ and C/C++ Extension Pack from the Extension Menu (C/C++ Themes is also heavily recommended)

# Setup First Project

- Create a new project
- Within the project file create a file named **main.cpp**
- From here type in the following function:

```
int main(int argc, char* argv[]){  
}
```

- Press the run button and select g++ as the debugger



```
C:\main.cpp > main(int, char *[])
1 #include <iostream>
2
3 using namespace std;
4
5 int main(int argc, char *argv[]){
6     cout << endl;
7     cout << "test" << endl;
8 }
```

```
Microsoft Windows [Version 10.0.22631.431
7]
(c) Microsoft Corporation. All rights res
erved.

C:\Users\njclo\OneDrive\Documents\Coding
Projects\C++ Projects\Test Setup> cmd /C
"c:\Users\njclo\.vscode\extensions\ms-vsc
ode.cpptools-1.22.10-win32-x64\debugAdapt
ers\bin\WindowsDebugLauncher.exe --stdin=
Microsoft-MIEngine-In-1qshu4bg.2b5 --stdo
ut=Microsoft-MIEngine-Out-vkgm5tft.brc --
stderr=Microsoft-MIEngine-Error-fdbproyw.
hux --pid=Microsoft-MIEngine-Pid-xq4e2ezom
.joi --dbgExe=C:\msys64\ucrt64\bin\gdb.ex
e --interpreter=mi"

test
```

# 02

# Basic Syntax &

# Structure

---



# Whitespace

What is whitespace?:

- An empty space between or around objects
- Examples:
  - Space
  - Tab

Why is it important?:

- Very important for organization
- Some languages use whitespace as a part of the code itself

**Overall, whitespace a great aspect to have in any project whether for organization or syntax purposes**

```
int main(int argc, char *argv[]){
    cout << endl;
    cout << "test" << endl;
}
```

Whitespace

# Understanding the MAIN Function

The **main** function is where the program will begin executing it's code

Segments:

- int main - refers to how the main function must return an integer (explained further in chapter 5), this integer refers to the error code that will be displayed upon program completion, the integer will be 0 if there is no error
- int argc - the argument count, number of command-line arguments in the
- char \*argv[] - the argument vector, an array of strings representing each argument

```
int main(int argc, char *argv[]){  
}
```

# Library Inclusion

Library Inclusion:

- Represented by **#include** (desired file)
- Includes the contents of a file into your file before compilation
- Allows the programmer to use declarations from other files within the current file
- Examples: iostream, fstream, stringstream, string, etc.

**iosteam** is a library dedicated to handling input and output

```
#include <iostream>
#include <fstream>
#include <string>
#include <sstream>
```

# Namespaces

Namespaces:

- Represented by **using namespace** (desired namespace)
- Used to prevent naming issues between libraries
- Makes code more readable

```
using namespace std;
```

**std** is the standard namespace of C++ containing standardized names for input and output, strings, data structures, etc.

# Basic Output with <iostream>

In order to print information to the terminal  
the **iostream** term **cout** is used followed by **<<**  
and then what is to be printed

- If printing text use quotation marks with your text inside
  - Ex. cout << "Hello World";
- If printing a variable, type the variables name
  - Ex. cout << testVar;
- To print each line of text on a new line  
use **endl** at the end of **cout**
  - Ex. cout << "Hello World" << endl;
- **cout** can print multiple things at once,  
just use **<<** between each (**<<** does not add a space between objects)
  - Ex. cout << "Hello World" << testvar  
<< endl;

```
int testVar = 1;  
cout << "Hello World " << testVar << endl;
```

Hello World 1

# Basic Input with <iostream>

**iostream** can also be used to gain input from the user. I ask for input type **cin** followed by **>>** and then the variable that the input will go into

Ex. `cin >> testVar;`

```
int testVar = 0;  
cout << "Please type an integer: ";  
cin >> testVar;  
cout << "Is this your value?: " << testVar << endl;
```

Please type an integer: |

Please type an integer: 5  
Is this your value?: 5

# 03

# Data Types &

# Variables

---



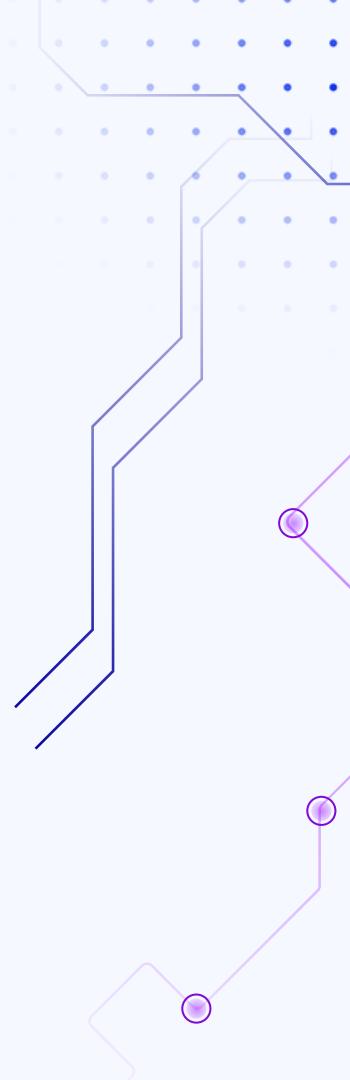
# Primitive Data Types

# Boolean

A boolean is

**Int**

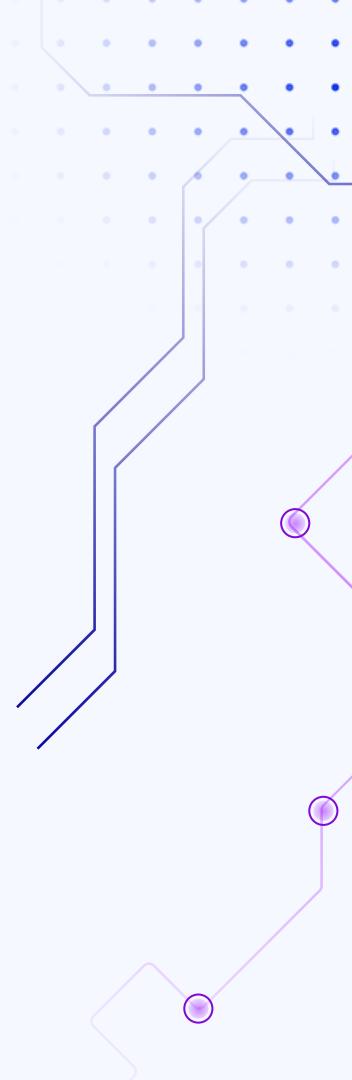
A



# Float



# Char



# <string>

# Constants

# 04

# Operators

---



# 05 Control Structures

---



# 06

# Functions

---



# 07

# Arrays

---



# 08 Pointers & References

---



# 09

## Introduction to Object-Oriented Programming

---



# 10 Conclusion & Q&A

---



# Questions?

# **Sign-Up for Future Classes and Tutoring on our Website**



# Structure of a Java Program

- **Import Statements:** Allow the use of classes from other packages

Example: `import java.util.Scanner;`

- **Class Declaration:** defines a class; every Java program must have at least one

Example: `public class MyClass {`

- **Main Method:** entry point of any Java application

Syntax: `public static void main(String[] args) {`

# Main Method Breakdown

## Components:

- **public**: Access modifier, allows the method to be called from anywhere.
- **static**: Indicates that the method can be called without creating an instance of the class.
- **void**: The return type, meaning the method does not return any value.
- **String[] args**: Parameter that accepts command-line arguments as an array of Strings
- **System.out.println()**: Prints out whatever is in parenthesis to the console, printed statement *must be in quotations unless it is a variable*

```
public class HelloWorld {  
  
    public static void main(String[] args) {  
  
        System.out.println("Hello, World!");  
  
    }  
  
}
```

# Data Types

## **int**

Represents integer values  
(whole numbers)

Ex: `int age = 25`

## **char**

Represents a single 16-bit  
character

Ex: `char initial = 'A'`

## **double**

Represents floating point  
numbers (decimals)

Ex: `double price = 20.99`

## **boolean**

Represents a true or false  
value

Ex: `boolean check = true`

# Declaring and Initializing Variables

## → Variable Declaration:

- ◆ Syntax: <data type> <variable name>;
- ◆ Ex:  
`int count;`  
`double price;`

## → Variable Initialization:

- ◆ Assign a value to the variable at the time of declaration or after
- ◆ Ex:

```
int count = 10; // Declaration and initialization
double price;
price = 99.99; // Initialization after declaration
```

## → Combined

- ◆ You can declare and initialize multiple variables in one line.
- ◆ Ex:  
`int x = 5, y = 10;`

# Arithmetic Operators

## Addition (+)

Adds two operands.

```
int sum = a + b;
```

## Subtraction (-)

Subtracts the second operand from the first.

```
int difference = a - b;
```

## Multiplication (\*)

Multiplies two operands.

```
int product = a * b;
```

## Division (/)

Divides the first operand by the second.

```
int quotient = a / b;
```

## Modulus (%)

Returns the remainder of the division.

```
int remainder = a % b;
```

# Comparison Operators

## Equal to (==)

Checks if two values are equal.

Example: if ( $a == b$ )

## Not Equal to(==)

Checks if two values are not equal.

Example: if ( $a != b$ )

## Greater than (>)

Checks if the left operand is greater than the right.

Example: if ( $a > b$ )

## Less than (<)

Checks if the left operand is less than the right.

Example: if ( $a < b$ )

## Greater than/Equal to (>=)

Checks if the left operand is greater than or equal to the right.

Example: if ( $a >= b$ )

## Less than/Equal to (<=)

Checks if the left operand is less than or equal to the right.

Example: if ( $a <= b$ )

# Logical Operators

## Logical AND (**&&**)

Returns true if *both* operands are true.

Example:  
`if (a > 0 && b > 0)`

## Logical OR (**||**)

Returns true if at least *one* operand is true.

Example:  
`if (a > 0 || b > 0)`

## Logical NOT (**!**)

Reverses the logical state of its operand.

Example:  
`if (!isTrue)`

02

# Conditionals and Loops

---



# Conditionals

## → If Statement:

- ◆ Executes a block of code if the condition is true

### Syntax:

```
if (condition) {  
    // code to be executed  
}
```

### Example:

```
if (score >= 50) {  
    System.out.println("Pass");  
}
```

**Output:** prints “Pass” in console if score is greater than or equal to 50.

# Conditionals

## → Else If Statement:

- ◆ Provides additional conditions if the previous if condition is false.

### Syntax:

```
if (condition1) {  
    // code for condition1  
}  
  
else if (condition2) {  
    // code for condition 2  
}
```

### Example:

```
if (score >= 50) {  
    System.out.println("Pass");  
}  
  
else if (score >= 30) {  
    System.out.println("Try Again");  
}
```

**Output:** prints “Try Again” if score is greater than or equal to 30, but not greater than or equal to 50.

# Conditionals

## → Else Statement:

- ◆ Executes a block of code if none of the preceding conditions are true.

### Syntax:

```
else {  
    // code to be executed if all  
    // conditions are false  
}
```

### Example:

```
else {  
    System.out.println("Fail");  
}
```

**Output:** Prints “Fail” in console if all preceding else/else if conditions are false.

# Conditionals - Switches

- Allows a variable to be tested for equality against a list of values (cases).
- Each case should end with a *break* statement to prevent fall-through.
- The *default* case is optional and executes if no cases match.

## Syntax:

```
switch (expression) {  
    case value1:  
        // code for value1  
        break;  
    case value2:  
        // code for value2  
        break;  
    default:  
        // code if no case matches  
}
```

## Example:

```
switch (day) {  
    case 1:  
        System.out.println("Monday");  
        break;  
    case 2:  
        System.out.println("Tuesday");  
        break;  
    default:  
        System.out.println("Other Day");  
}
```

# For Loops

- A control flow statement that repeats a section of code until a condition is met
- Syntax:

```
for (initialization; condition; update) {  
    // code to be executed  
}
```

- The *initialization* runs once before the loop starts.
- The *condition* is checked before each iteration; if true, the loop continues.
- The *update* statement is executed after each iteration.
- Example:

```
for (int i = 0; i < 5; i++) {  
    System.out.println("Iteration: " + i);  
}
```

## Output:

Iteration: 0  
Iteration: 1  
Iteration: 2  
Iteration: 3  
Iteration: 4

# While Loops

- Repeats a block of code as long as a condition is met.
- Syntax:

```
while (condition) {  
    // code to be executed  
}
```

- The condition is evaluated *before* the loop body executes, so it may not run at all if the condition is false initially.
- Example:

```
int count = 0;  
  
while (count < 5) {  
    System.out.println("Count: " + count);  
    count++;  
}
```

**Output:**  
Count: 0  
Count: 1  
Count: 2  
Count: 3  
Count: 4

# Do-While Loops

- Similar to the while loop, but guarantees at least *one* execution of the loop body.
- Syntax:

```
do {  
    // code to be executed  
} while (condition);
```

- Example:

```
int count = 0;  
  
do {  
    System.out.println("Count: " + count);  
    count++;  
} while (count < 5);
```

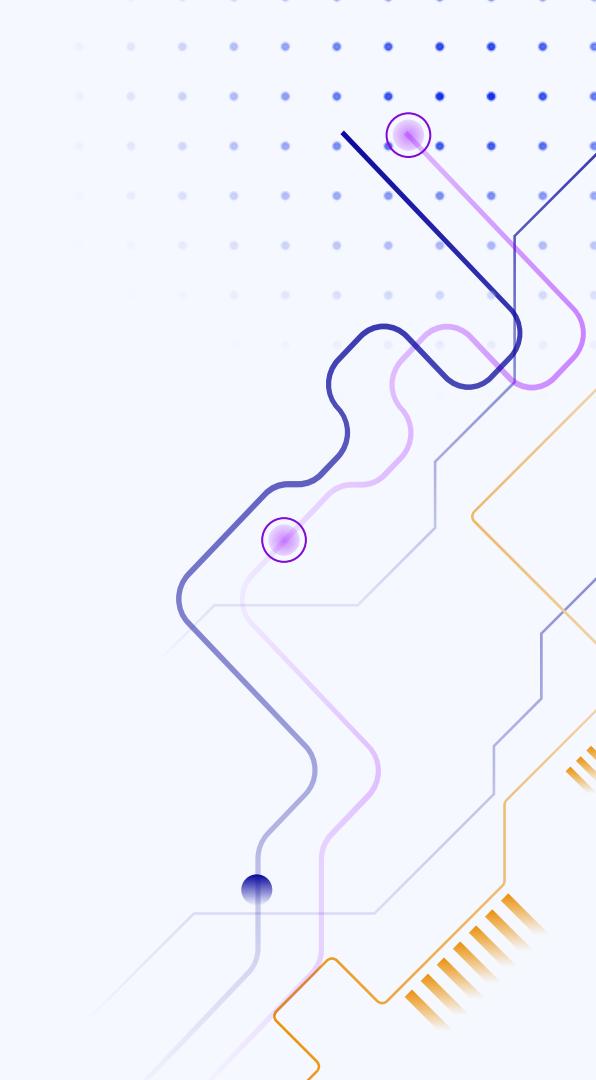
## Output:

```
Count: 0  
Count: 1  
Count: 2  
Count: 3  
Count: 4
```

03

# Introduction to Methods

---



# Defining Methods

- A method is a block of code designed to perform a specific task.

- Syntax:

```
returnType methodName(parameters) {  
    // code to be executed  
}
```

- **Return Type:** Type of value the method returns (ex: int, void).

- **Method Name:** Identifier for the method (ex: add).

- Parameters: Input values the method accepts (ex: int a, int b).

- Example:

```
public int add(int a, int b) {  
    return a + b;  
}
```

# Calling Methods

- *Calling a method with parameters:* Use the method name followed by parentheses containing any required arguments.
- Example:

```
int result = add(5, 10);
System.out.println("Sum: " + result);
```

**Output:**  
Sum: 15
- *Calling methods with no parameters:* Methods can also be define without parameters.
- Example:

```
public void greet() {
    System.out.println("Hello!");
}
```
- Call:  
`greet();`

**Output:**  
Hello!

# Method Overloading Basics

- Method overloading allows multiple methods in the same class to have the *same name* but *different parameter lists* (type, number, or order of parameters).
- Return type alone is not sufficient for overloading; the *parameters* must differ.
- Useful for creating intuitive APIs where the same method name can handle different data types or quantities.

# Method Overloading Example

```
public class Calculator {  
    // Method to add two integers  
    public int add(int a, int b) {  
        return a + b;  
    }  
  
    // Method to add three integers  
    public int add(int a, int b, int c) {  
        return a + b + c;  
    }  
  
    // Method to add two doubles  
    public double add(double a, double b) {  
        return a + b;  
    }  
  
    // Method to add two strings  
    public String add(String a, String b) {  
        return a + b;  
    }  
}
```

# 04

# Object-Oriented Programming Basics

---



# Intro to Classes

- A **class** is a blueprint or template for creating objects. It defines properties (attributes) and behaviors (methods) that the objects created from the class will have.
- **Syntax:**

```
public class ClassName {  
    // Attributes  
    dataType attributeName;  
  
    // Methods  
    public returnType methodName(parameters) {  
        // code  
    }  
}
```

## Example:

```
public class Car {  
    // Attributes  
    String color;  
    String model;  
    int year;  
  
    // Method  
    public void displayInfo() {  
        System.out.println("Model: " + model + ", Color: " + color  
        + ", Year: " + year);  
    }  
}
```

# Intro to Objects

- An **object** is an *instance* of a class. It represents a specific realization of the class, containing actual values for the attributes defined in the class.
- It bundles attributes and methods that operate on the data into a single class.
- Creating an object:
  - ◆ `Car myCar = new Car();`
- Setting attributes:

```
myCar.color = "Red";  
myCar.model = "Toyota";  
myCar.year = 2021;
```

- Calling Methods:
  - ◆ `myCar.displayInfo();`



# Constructors

- A **constructor** is a special method used to initialize objects when they are created. It has the same name as the class and does not have a return type.
- Types:
  - Default constructor:** No parameters, initializes objects with default values

```
public class Car {  
    String color;  
    String model;  
  
    // Default constructor  
    public Car() {  
        color = "Unknown";  
        model = "Unknown";  
    }  
}
```

# Constructors cont.

**-Parameterized Constructor:** Accepts parameters to initialize attributes with specific values.

```
public Car(String color, String model) {  
    this.color = color;  
    this.model = model;  
}
```

- Constructors are called automatically when an object is created.
- They can be overloaded (multiple constructors with different parameter lists).
- If no constructor is defined, Java provides a default constructor.

# “This” Keyword

- The **this** keyword is a reference variable that refers to the current object within an instance method or constructor.
- It differentiates between class attributes and parameters when they have the same name.
- *To Access Instance Variables:* When parameter names are the same as instance variables, this is used to clarify that we are referring to the instance variable.

```
public Car(String color, String model) {  
    this.color = color;      // 'this.color' refers to the instance variable  
    this.model = model;     // 'this.model' refers to the instance variable  
}
```

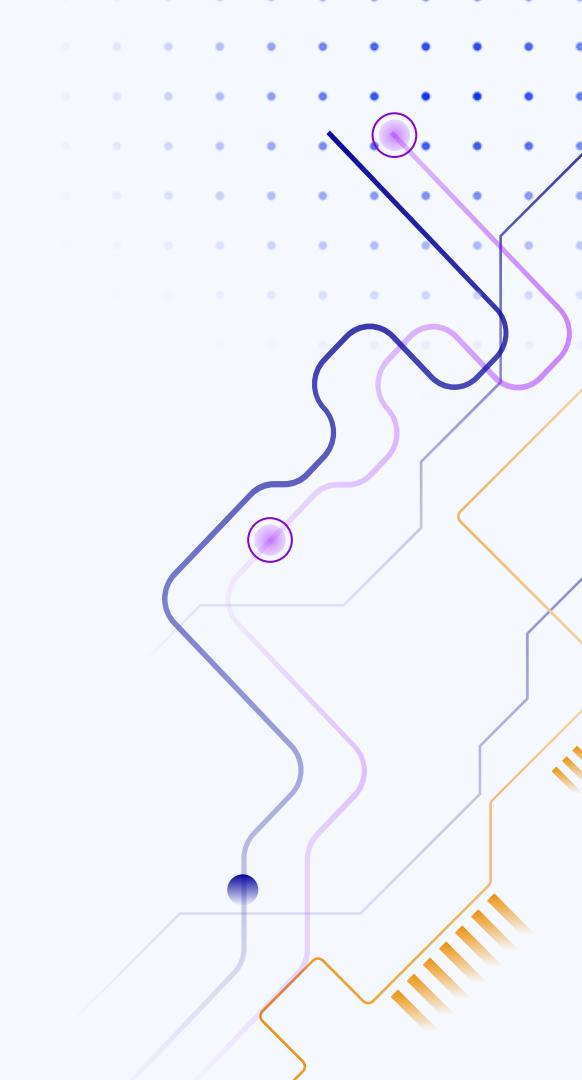
- *To Call Other Constructors:* You can use this() to call another constructor in the same class (constructor chaining).

```
public Car() {  
    this("Unknown", "Unknown"); // Calls the parameterized constructor  
}
```

# 05

# Scanners

---



# Introduction to Scanners

- Scanners are used to read input from various sources, including keyboard input, files, and strings.
- The Scanner class in Java is part of the `java.util` package.
- Common uses:
  - ◆ Reading user input from the console.
  - ◆ Parsing data from files (e.g., text files).
  - ◆ Extracting tokens from strings
- Import the Scanner class:
  - ◆ `import java.util.Scanner;`
- Create a Scanner object:
  - ◆ `Scanner scanner = new Scanner(System.in);`

# Introduction to Scanners cont.

- How to read different types of input:
  - ◆ String: `String name = scanner.nextLine();`
  - ◆ Integer: `int age = scanner.nextInt();`
  - ◆ Double: `double height = scanner.nextDouble();`
- Close the scanner:
  - ◆ `scanner.close();`

# Scanner Example

```
import java.util.Scanner;

public class UserInput {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter your name: ");
        String name = scanner.nextLine();
        System.out.print("Enter your age: ");
        int age = scanner.nextInt();
        System.out.println("Hello " + name + ", you are " + age + " years old.");
        scanner.close();
    }
}
```

# Exercise 1:

Write a Java program (call class “Calculator”) that takes two integers as input and performs addition, subtraction, multiplication, and division.

**Hints:** Use a scanner to ask for user inputs, cannot divide by 0 (use if/else statements)

# Solution

```
import java.util.Scanner;

public class Calculator {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter first integer:");
        int num1 = scanner.nextInt();

        System.out.print("Enter second integer:");
        int num2 = scanner.nextInt();

        System.out.println("Addition: " + (num1 + num2));
        System.out.println("Subtraction: " + (num1 - num2));
        System.out.println("Multiplication: " + (num1 * num2));
        if (num2 != 0) {
            System.out.println("Division: " + ((double) num1 / num2));
        } else {
            System.out.println("Division: Cannot divide by zero.");
        }
    }
}
```

## Exercise 2:

Create a program (call class TemperatureConverter) that converts temperatures from Celsius to Fahrenheit and vice versa.

**Hints:** Ask for user input using a scanner, create methods that convert  
 $C \rightarrow F$  and  $F \rightarrow C$

### Formulas:

$$\begin{aligned} ^\circ F &= (9/5) ^\circ C + 32. \\ ^\circ C &= (^\circ F - 32) \times 5/9 \end{aligned}$$

# Solution

```
import java.util.Scanner;

public class TemperatureConverter {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter temperature in Celsius: ");
        double celsius = scanner.nextDouble();
        double fahrenheit = celsiusToFahrenheit(celsius);
        System.out.println(celsius + "°C is " + fahrenheit + "°F");

        System.out.print("Enter temperature in Fahrenheit: ");
        fahrenheit = scanner.nextDouble();
        celsius = fahrenheitToCelsius(fahrenheit);
        System.out.println(fahrenheit + "°F is " + celsius + "°C");
    }

    public static double celsiusToFahrenheit(double celsius) {
        return (celsius * 9/5) + 32;
    }

    public static double fahrenheitToCelsius(double fahrenheit) {
        return (fahrenheit - 32) * 5/9;
    }
}
```

# Exercise 3:

Write a program (call class OddEvenChecker) with a method that checks if a given integer is odd or even.

**Hints:** Use modulus and if statements!

# Solution

```
import java.util.Scanner;

public class OddEvenChecker {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter an integer: ");
        int number = scanner.nextInt();

        if (isEven(number)) {
            System.out.println(number + " is even.");
        } else {
            System.out.println(number + " is odd.");
        }
    }

    public static boolean isEven(int num) {
        return num % 2 == 0;
    }
}
```

# Exercise 4:

Create a class called Book with attributes for title, author, and price.  
Include a method to display the book's details.

**Hints:**

# Solution

```
class Book {  
    String title;  
    String author;  
    double price;  
  
    // Constructor  
    public Book(String title, String author, double price) {  
        this.title = title;  
        this.author = author;  
        this.price = price;  
    }  
  
    // Method to display book details  
    public void displayInfo() {  
        System.out.println("Title: " + title + ", Author: " + author + ", Price: $" + price);  
    }  
}  
  
public class BookTest {  
    public static void main(String[] args) {  
        Book myBook = new Book("1984", "George Orwell", 9.99);  
        myBook.displayInfo(); // Output: Title: 1984, Author: George Orwell, Price: $9.99  
    }  
}
```