Starting

- Setup
- Basic Output

TABLE OF CONTENTS

Starting

Page Number

Setup ---- 1
Basic Output ---- 2

Key Basics

Number Data Types ---- 3
Integer ---- 3
Double ---- 3
Math Operations ---- 3
Printing Out Variables ---- 5
Using Different Data Types ---- 5
Division Using Integers ---- 5
Text Data Types ---- 6
String ---- 6
Char ---- 7
User Input ---- 7
Math Methods ---- 9
Math.random() ---- 9
Making Methods ---- 9

Java Logic

Boolean ---- 11
Conditionals ---- 11
If Statements ---- 11
Logical Operators ---- 13
Else If and Else ---- 15
Switch Statements ---- 16
Ternary Operators ---- 17
Loops ---- 20
For Loops ---- 20
While Loops ---- 23
Do While Loops ---- 24

Arrays

Single Dimensional Arrays ---- 34
Application of Loops in Arrays ---- 35
Multidimensional Arrays ---- 38
Searching Algorithms ---- 42
Sorting Algorithms ---- 44

Closure

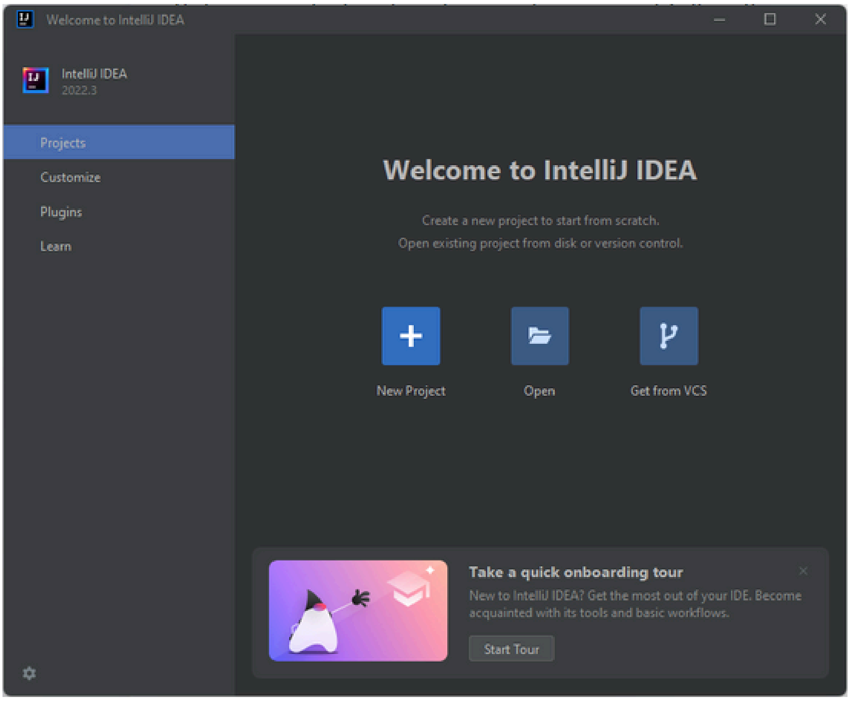
Final Projects ---- 48

Setup

Before you start writing Java code, follow the directions below. These instructions are for 2024 and may change over time.

1. Download and install the Java Development Kit (JDK) on your computer. You can download the JDK from the official Oracle website.
2. Choose an Integrated Development Environment (IDE) for Java. Some popular options include Eclipse, IntelliJ IDEA, and NetBeans. Install the software from their websites.
3. We will be using IntelliJ Community Version for this book.

Once you have followed those steps, select "New Project" on this screen.



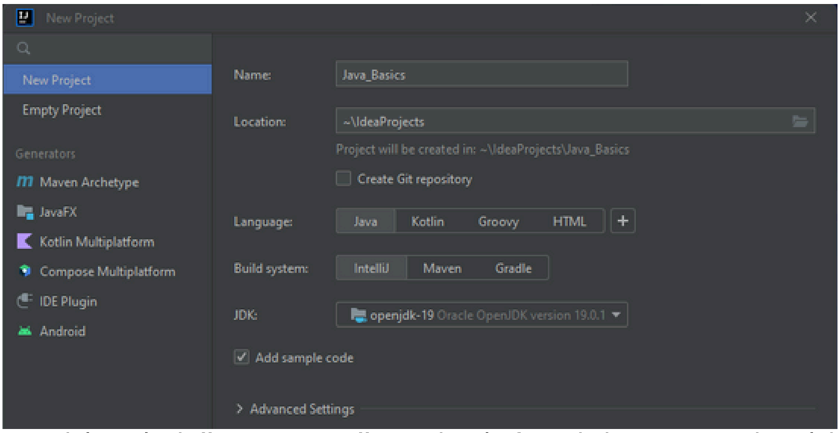
Welcome to IntelliJ IDEA

Create a new project to start from scratch.
Open existing project from disk or version control.

Take a quick onboarding tour

New to IntelliJ IDEA? Get the most out of your IDE. Become acquainted with its tools and basic workflows.

Once selected, copy the configurations below and click "Create."



Name: Java_Basics

Location: ~\IdeaProjects

Project will be created in: ~\IdeaProjects\Java_Basics

Create Git repository

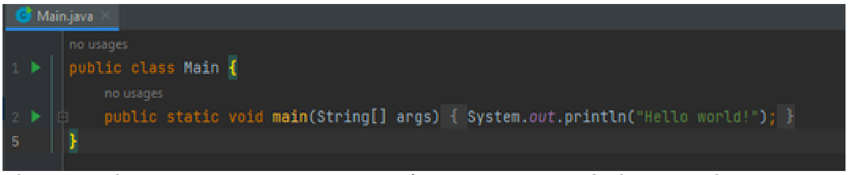
Language: Java | Kotlin | Groovy | HTML

Build system: IntelliJ | Maven | Gradle

JDK: openjdk-19 Oracle OpenJDK version 19.0.1

✓ Add sample code

Double-click "Main.java" on the left sidebar. You should see the following code:



```
no usages
public class Main {
    no usages
    public static void main(String[] args) { System.out.println("Hello world!"); }
}
```

This code is a starting template; most of the code you will write using this book will be between the two grey curly braces.

Basic Output

Each program you write in this book may have some kind of output you can see. For example, when you program a calculator later on in the bok, you will return the solution. The code to print text in Java is:

```
System.out.println("INTENDED OUTPUT");
```

The preset code will output "Hello World," which is a common coding phrase used for test outputs. To see outputs, click “Run” in the top bar of the screen, then select the first option in the drop-down. You will see the output, at the bottom of your screen. You can print numbers using the format shown above without using any quotes.

Key Basics

- Number Data Types
    - Integer
    - Double
- Math Operations
- Printing out Variables
- Using Different Data Types
- Division Using Integers
- Text Data Types
    - Char
    - String
- User Input
- Math Methods
    - Math.random()
- Making Methods

Definition of a Data Type: A data type defines the kind of value a variable can hold. For example, it can be a number with or without decimals, a single letter, or a word.

Number Data Type #1: Integers

In Java, an integer is a data type that can store a number without decimals. It has a range of values from -2^31 to 2^31 -1. Integers are one of the most important data types.
The following code is used to declare an integer named "a":

Variable Name Value

```
int a = -1289634;
```

Data Type

Note: “a” can be replaced with any name as long as it doesn't start with a number, have any spaces, and have any special characters other than “_” and “$”. Try to make variable names descriptive of the value being stored in it.

Most variables are declared in the same format as this.

Number Data Type #2: Double

In Java, a double data type is a number with decimal points. It has a range of values from approximately 4.9 x 10^-324 to 1.8 x 10^308. The following syntax is used to declare a double named "a":

```
double a = 3.14;
```

Math Operations

In Java, variables that store numbers can be subjected to +, -, /, *. For example, c = a + b. Any one of those four operations could be used there.

In addition to these operations, some other ones may be useful. Here are all of them with an example:

= Equals to: Used to set or change the value of a variable.

```
int b = 5;
b = 4;
b = 3;
```

Value of b: 3

Note: int/double needs to be mentioned only in the first declaration, after that simply use the name of the variable. Also, the value of a variable can be changed as many times using the = sign. The value must match the initial data type, however.

% Modulus: returns the remainder of 2 numbers when divided.

```
int b =  5%3;
```

Value of b: 2

++ Increment operator: adds 1 to the current value of a variable.

```
int b = 2;
b++;
```

Value of b: 3

-- Decrement operator: subtracts one from the current value of a variable.

```
int b = 2;
b--;
```

Value of b: 1

+= Compound addition operator: adds a number or variable to the current value of a variable.

```
int b = 2;
b+=2;
```

Value of b: 4

-= Compound subtraction operator: subtracts a number or variable from the current value of the variable.

```
int b = 2;
b-=2;
```

Value of b: 0

***=** Compound multiplication operator: Multiplies a number or variable with the current value of a variable.

```
int b = 2;
b*=2;
```

Value of b: 4

/= Compound division operator: Divides the current value of a variable by a number or variable.

```
int b = 2;
b/=2;
```

Value of b: 1

Printing Out Variables

```
System.out.println(nameOfVariable);
```

```
System.out.println("Value: " + nameOfVariable);
```

Simply put the name of the variable in the brackets to print it. If you are printing text with it, separate them by putting a + sign in between.

Using Different Data Types

Although not mentioned in this book, other number data types are less often used. All the math operators mentioned before can be used between 2 or more different data types. Note that they must be numbers. For example, dividing a double by an int does work perfectly fine. However, when doing such operations, the data type of the output may be promoted (changed). For example, a double/int returns a double, never an int. So the following code will result in an error:

```
int a = 5.0 / 1;
```

In Java, 5.0/1 = 5.0, not 5. So the result of that operation must be stored in a double. For our purposes, when using both integer and double in an operation, the output will be a double.

Division Using Integers

Since integers cannot have decimals, they deal with division operations that should have decimals non-intuitively. For example, in Java, 9/2 = 4, -7/2 = -3, 3/4 = 0, 99/100 = 0. This is because if a division operation leads to a decimal, Java only counts the integer before the decimal point and disregards anything after it. This is called truncation. So 4.5 -> 4, -3.5 -> -3, 0.75 -> 0, 0.99 -> 0. So, if you ever need the decimal part of a number, you need to use a double, not an integer.

Practice

1. Write code to declare two integers num1 and num2, and assign them the values 5 and 7 respectively. Add them together and store the result in an integer variable called sum.
2. Write code to declare a double variable radius and assign it a value of 2.5. Calculate the area of a circle using the formula area = radius * radius * 3.14, and store the result in a double variable called circleArea.
3. Write code to declare an integer variable quantity and assign it a value of 10. Decrease its value by 3 and store the result in an integer variable updatedQuantity.
4. Write code to declare two integers, x, and y, and assign them the values 12 and 3 respectively. Divide x by y and store the result in an integer variable divisionResult.

Answers

```
1.
int num1 = 5;
int num2 = 7;
int sum = num1 + num2;

2.
double radius = 2.5;
double circleArea = radius * radius * 3.14;

3.
int quantity = 10;
int updatedQuantity = quantity - 3;

4.
int x = 12;
int y = 3;
int divisionResult = x / y;
```

Text Data Type #1: String

In Java, a String is a text data type that is used to store a single letter or several letters or numbers. In order to declare a string, double quotes are required. Here is an example:

```
String a = "My name is Thomas Jansen";
```

To use some special characters or perform specific tasks inside a string, escape keys are required.

For example, you cannot simply put a double quote inside a string; hence you will have to do \". The same applies to a backslash. To put a backslash inside a string you must do \\. If this is not easy to understand, here are some examples.

\"	Adds a double quote	\\	Adds a backslash
\t	Adds a tab	\n	Skips to a new line

Double Quote:	**System.out.println(a):**
`String a = "This is so \"cool\"";`	This is so "cool"
Backslash:	
`String a ="Backslash: \\";`	Backslash: \
Tab:	
`String a = "Hot\tsauce";`	Hot sauce
New Line:	
`String a = "Hot\nsauce";`	Hot sauce

Text Data Type #2: Char

In Java, a char is a text data type that is used to store a single character. In most cases, it is used to store one letter. It uses single quotes instead of double.

```
char a = 'g';
```

User Input

Suppose you wanted to ask the user (you in this case) what their age is, you would have to use some new syntax to receive input. Above "public class Main {", make a new line and type in:

```
import java.util.Scanner;
```

Definition of “syntax”: code writing rules/language.

You will still need to initialize the scanner by writing the following code under "public static void main(String[] args) {"

```
Scanner reader = new Scanner(System.in);
```

“reader” is the name of the scanner and can be set to anything following the variable name rules.
Now write a prompt for the user to give us an entry.

```
System.out.print("Enter your name:");
```

After the user enters their response, we can make the scanner "read" it by using the following code. But you must note that the data type you expect the user to enter affects the code you write. This is the code for reading a String:

Data Type, Scanner syntax

```
String a = reader.nextLine();
```

Variable name

*When you plan on reading a string after reading an int or double, you must add reader.nextLine() before reading the string to reset the scanner. It only needs to be done once. Here is an example:

```
int a = reader.nextInt();
reader.nextLine();
String c = reader.nextLine();
```

Reading a int

```
int a = reader.nextInt();
```

Reading a double

```
double a = reader.nextDouble();
```

Reading a char

```
char inputChar = reader.next().charAt(0);
```

Example:
This program will ask the user for their age and then their name, and it will store them as variables and print them out.

```
import java.util.Scanner;
...
  Scanner reader = new Scanner(System.in);
  System.out.println("Enter your age here :");
  int age = reader .nextInt();
  reader.nextLine();// reset needed
  System.out.println("Enter your name here: ");
  String name = reader.nextLine();
  System.out.println("Your age is "+age +” and your
  name is: “ + name);
```

Sample output of this program is:

```
Enter your age here :
32
Enter your name here:
Tuco Salamanca
Your age is 32 and your name is: Tuco Salamanca
```

Math Methods

Math methods are inbuilt math operators that perform more complex tasks than +,-,*, and /. Only one method is mentioned in this book but feel free to learn more.

Math.random()

In itself, Math.random() is a method that generates a random double value from 0 (inclusive) and 1 (exclusive). This method can be used in several different use cases where a random value is needed. The range of values the method outputs can easily be altered by the following syntax. The random value generated will be stored in a double since it has decimal points (the generated value can be between two integers).

```
int randomInt =(int) (Math.random() * ((max - min) + 1))+min;
```

If min = 5 and max = 10, this method would have a range from 5 to 10. In order to include the max, the +1 is necessary. This is because the maximum value of the random() function is 0.999 repeating, and because of integer truncation, the number gets rounded down.

Making Methods

Suppose we would like to make our own method to do something. Here is how it would work.

```
public static void first (){
    System.out.println("This is the first method");
}
```

If this format looks familiar, it is because the code we have written so far was also written in a method, the main method.

Method Syntax Return Type Name Arguments

```
public static void first() {
    System.out.println("This is the first method");
}
```

*Ignore "static", just remember it.

The return type refers to what data type will be returned from the method. For example, Math.random() returns a double. Void is not a data type; it means that the method will not return anything, it represents the lack of returning. System.out.println() does not count as returning anything; it simply prints something. The arguments are the variables that will be used in the method. Since this method does not use any values, there is nothing in the parentheses. To call (use) the method, simply use the name followed by parentheses containing the parameters (don't include data types). If the method does not print anything (returns a value), it needs to be in a print statement. **Note**: any methods other than the main method must be put outside the brackets of the main method but inside the brackets of the class (public class...). Look at the example below. This code shows how a method that calculates an average is used in a Java program:

```
import java.util.Scanner;
public class avgFinder{
      public static void main(String[] args){
          Scanner reader = new Scanner (System.in);
          System.out.println("Enter the first number");
          double a = reader.nextDouble();
          System.out.println("Enter the second number");
          double b= reader.nextDouble();
          System.out.println(avg(a,b));
      }
public static double avg (double a, double b){
      return (a+b)/2;
}
}
```

The "avg" method's code is after the code of the main method(red braces) and before the end of the class code(green braces). In short, remember to place your method's code between the two ending braces.

Java Logic

- Boolean
- Conditionals
  - If Statements
  - Logical Operators
  - Else If and Else
  - Switch Statements
  - Ternary Operators
- Loops
  - For Loops
  - While Loops
  - Do While Loops

Boolean

Booleans are one of the most important data types when it comes to logic in computer programs. It can only have two possible values: true or false. This is a very useful tool in computer logic because it can be used to determine if something is... true or false. If not Boolean itself, the idea of being true or false is key to programming and will be one of the most important things you will use. Here is an example.

```
boolean isRaining = true;
```

Conditionals

Conditionals in programming are the idea of executing code only if they meet a certain condition. For example, you want to print "It is raining" only if it is raining.

If Statements

The scenario described is the fundamental use of the if statement. It executes code if a condition is true. Here is a basic example.

```
if(2>1){
      System.out.println("Hello!");
}
```

The output is *Hello*! because 2 is greater than 1.

The curly braces denote that the code in between them should run if the condition is proven true. You need to close the brackets when you are done with the if statement, or your code will not work. **Note**: if you do not include any curly braces at all, only the line of code right after the if statement will be considered to be part of it. Any code after the first line will run no matter if the condition was satisfied or not. So using { and } is best.

Important, if you are using the equal sign for variables that have the data type of integer, double, and boolean in an if statement, you need to use 2 equal signs like this: ==. You cannot use = to check values.

Note: Variables CANNOT be declared inside an if statement.

Type 1: integer, double, and char

```
int a = 5;
if(a==5){
        System.out.println("the variable a = 5");
}
```

Output: the variable a = 5
int can be replaced with double or char, altering the value accordingly, obviously. Strings use a different syntax.

Example with !

```
int a = 5;
if(a!=5){
        System.out.println("the variable a = 5");
}
```

“!” means not. If ! is used, the code in braces will only run if the condition is not met. So this code won't print anything, since a=5. It would only run if a did not equal 5.

Type 2: Boolean

```
boolean isRaining = true;
if(isRaining== true){
        System.out.println("it is raining");
}
```

Output: it is raining
Since the boolean declared is true and the condition states that it must be true, the output is printed.

Alternate syntax

```
boolean isRaining = true;
if(isRaining){
        System.out.println("it is raining");
}
```

Output: it is raining
Just putting the name of the boolean inside the parenthesis checks if the value of the boolean is true. It is the same thing as saying *variableName* == true.

Example with !

```
boolean isRaining = true;
if(!isRaining){
        System.out.println("it is raining");
}
```

Nothing will be printed because "is raining" is true. The “!” here implies that the boolean's value must be false, which it is not. That is why the code inside the code in the if statement will not execute.

Example with String

Unfortunately, in Java, you cannot simply use the equal signs for Strings. Instead, you need to use .equals(StringToBeCompared). Here is an example.

```
String cheese = "cheddar";
if(cheese.equals("cheddar")){
        System.out.println("it’s cheddar");
}
```

You can use the name of a variable in the brackets or an actual string in double quotes. If you want to see that the string is not equal to another string just put a ! at the front of the condition like this.

```
String cheese = "mozzarrella";
if(!cheese.equals(cheeseName)){
        System.out.println("it’s not mozzarrella");
}
```

Logical Operators

Suppose you wanted to have a more complex situation, like calling a rescue team if it is raining and the inches of rain on the ground are over 40 inches.

Technically, you could put one if statement inside the other, but there is an easier way to do this. To do this, we need to use logical operators. They are essentially special characters used to create compound conditions like the one stated above.

Checking for multiple conditions in 1 if statement:
In order to check more than one condition in one if statement we use “&&” this basically means "and".

```
boolean isRaining = true;
int inches = 45;
if(isRaining&&inches>40){
        System.out.println("Call rescue");
}
```

There can be as many conditions as needed.

```
boolean isRaining = true;
int inches = 45;
int population =235;
if(isRaining&&inches>40&&population>50){
        System.out.println("Call rescue");
}
```

Checking for either condition in 1 if statement:
Suppose we wanted to check if at least one statement is true. Suppose call for rescue if the population is over 50 or the number of inches of rain on the ground is greater than 40. If either one of these is true then rescue will come. If both are true they will still come. In that case, use “||” which basically means “or”.

```
int inches = 45;
int population =235;
if(inches>40||population>50){
        System.out.println("Call rescue");
}
```

Combining both operators:
New scenario: for the rescue to come:
it must be raining and inches of rain>40

or

population>50

```
boolean isRaining = true;
int inches = 45;
int population =235;
if((isRaining&&inches>40)||population>50){
        System.out.println("Call rescue");
}
```

The inner brackets are needed; otherwise, the situation would be to call rescue if isRaining is true and either inches > 40 or population > 50. This means that isRaining and inches wouldn't be paired together. The placement of the inner brackets is common sense; apply them whenever you need to group conditions in a compound condition.
Note: If any part of the condition makes the condition true in an if statement, Java will not look at anything after. For example, if the condition was (booleanA || booleanB), if booleanA is true, booleanB will not be checked at all.
>= and <=
Suppose the condition was if inches was 40 or more (40 is in the scope of the condition being true).

```
int inches = 40;
if(inches>=40){
        System.out.println("Call rescue");
}
```

Else If and Else

Suppose you want an alternative, suppose the first condition is false and you want to execute something in that case, this is where if-else and else statements come in. The syntax for both is very simple. Take a look at the example.

```
int inches = 40;

if (inches >= 40) {
   System.out.println("Call rescue");
} else if (inches > 5 && inches < 40) {
   System.out.println("Not severe, but take precaution");
} else {
   System.out.println("Normal");
}
```

Else if statements are the alternative to the original if statement but still imposes a condition. For else statements, it is different. If the initial if statement's condition was not met and all the other else if statement's condition was not met, it will execute no matter what. Note there can only be one else statement but an unlimited number of else if statements.

Switch Statements

Switch statements are a more simplified version of if statements but with much less functionality. There are very few cases where using a switch statement is needed. But it is still important to learn. Here is how it works.

```
switch (expression) {
    case value1:
        // code to be executed when expression = value1
        break;
    case value2:
        // code to be executed when expression = value2
        break;
    // more cases can be added as needed
    default:
        // code to be executed when none of the cases
match the expression
         break;
}
```

Take a look at this example using the days of the week:

```
int day= 4;

switch (day) {
    case 1:
        System.out.println("It's Monday.");
        break;
    case 2:
        System.out.println("It's Tuesday.");
        break;
    case 3:
        System.out.println("It's Wednesday.");
        break;
```

Note: “break” breaks out the code that is being executed when a condition is met. It needs to be used at the end of all cases, including the default case.

Continued on next page.

```
case 4:
 System.out.println("It's Thursday");
 break;

 default:
 System.out.println("Im not sure!");
 break;
}
```

It works the same for strings, except you need to use double quotes. There is not much more to switch statements, as they have limited functionality, but they are still good to know about.

Ternary Operators

Unlike switch statements, ternary operators are very useful and are used all the time. It does not have the same level of functionality as an if statement but is very easy to use and only a line long most of the time. They are an easy way to define a variable that involves conditionals. Here is the format:

```
variableName = booleanExpression ? expression1 : expression2
```

The variable that the value is to be assigned to.	It is a condition framed as a question. For example 2>1? Is 2 greater than 1?	Code to be executed if the answer to the question is true.	Code to be executed if the answer to the question is false.

Here is an example of it being used:

```
int number = 5;
int guess =6;
int value = (guess>number)? guess : number;
```

If guess>number, value = guess, if not value = number. If we used if statements coding this would have taken more time and lines.

Project #1: Cash Register

You have been assigned to develop a cashier calculation system for a retail store. The system should allow a cashier to enter the details of items purchased by a customer and calculate the total cost, including taxes and discounts. The tax rate is 10%, and the discount rates are as follows:

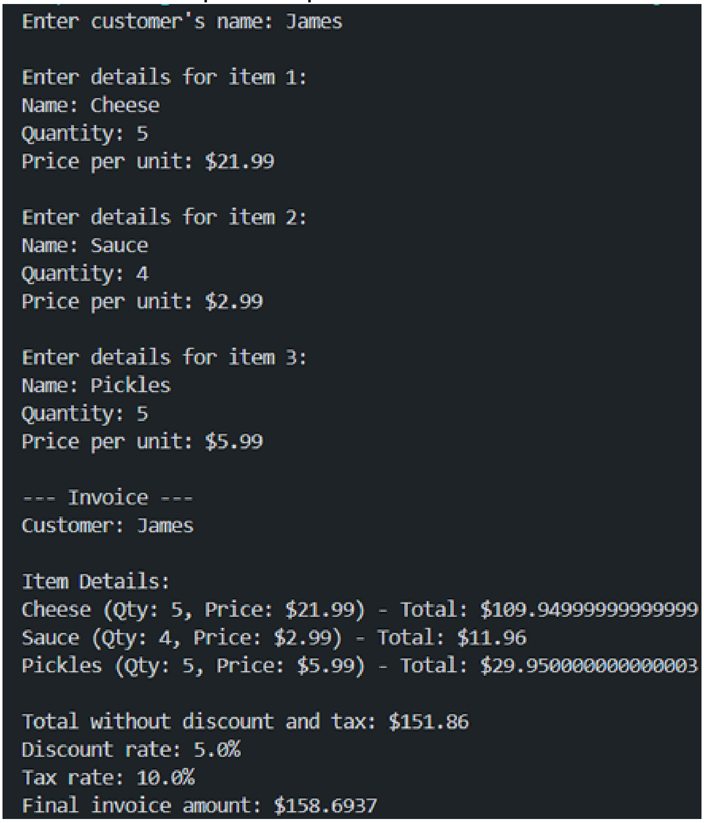
- 5% discount for a total amount (without tax) exceeding $100
- 10% discount for a total amount (without tax) exceeding $500

Write a Java program to implement this invoice calculation system. Your program should:

1. Prompt the cashier to enter the customer's name.
2. Prompt the cashier to enter the details of three items purchased (name, quantity, and price per unit).
3. Calculate the final invoice amount: using this formula : (total without discount)*(1-discount rate/100)*(1+tax rate/100)
4. Print the customer's name, and item details (name, quantity, price, total cost for each item).
5. Print the total cost for all 3 items without discount and tax.
6. Print the final invoice amount.

Your program should use if statements to apply discounts based on the total amount.

Here is a sample output


```
Enter customer's name: James

Enter details for item 1:
Name: Cheese
Quantity: 5
Price per unit: $21.99

Enter details for item 2:
Name: Sauce
Quantity: 4
Price per unit: $2.99

Enter details for item 3:
Name: Pickles
Quantity: 5
Price per unit: $5.99

--- Invoice ---
Customer: James

Item Details:
Cheese (Qty: 5, Price: $21.99) - Total: $109.94999999999999
Sauce (Qty: 4, Price: $2.99) - Total: $11.96
Pickles (Qty: 5, Price: $5.99) - Total: $29.950000000000003

Total without discount and tax: $151.86
Discount rate: 5.0%
Tax rate: 10.0%
Final invoice amount: $158.6937
```

Solution:

```
import java.util.Scanner;
public class CashierCalculationSystem {
  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);

    System.out.print("Enter customer's name: ");
    String customerName = scanner.nextLine();

    System.out.println("\nEnter details for item 1:");
    System.out.print("Name: ");
    String item1Name = scanner.nextLine();
    System.out.print("Quantity: ");
    int item1Quantity = scanner.nextInt();
    System.out.print("Price per unit: $");
    double item1Price = scanner.nextDouble();
    scanner.nextLine(); // Consume newline
    //repeat item 1 code item 2 and 3, change variable names

    // Calculate totals
    double item1Total = item1Quantity * item1Price;
    double item2Total = item2Quantity * item2Price;
    double item3Total = item3Quantity * item3Price;
    double totalWithoutDiscount = item1Total + item2Total +
item3Total;

    // Calculate discount
    double discountRate = 0;
    if (totalWithoutDiscount > 500) {
      discountRate = 10;
    } else if (totalWithoutDiscount > 100) {
      discountRate = 5;
    }

    // Calculate final amount
    double taxRate = 10;
    double finalAmount = totalWithoutDiscount * (1 -
discountRate / 100) * (1 + taxRate / 100);

    // Print invoice
    System.out.println("\n--- Invoice ---");
    System.out.println("Customer: " + customerName);
    System.out.println("\nItem Details:");
    System.out.println(item1Name + " (Qty: " + item1Quantity + ",
Price: $" + item1Price + ") - Total: $" + item1Total);
    System.out.println(item2Name + " (Qty: " + item2Quantity + ",
Price: $" + item2Price + ") - Total: $" + item2Total);
    System.out.println(item3Name + " (Qty: " + item3Quantity + ",
Price: $" + item3Price + ") - Total: $" + item3Total);
    System.out.println("\nTotal without discount and tax: $" +
totalWithoutDiscount);
    System.out.println("Discount rate: " + discountRate + "%");
    System.out.println("Tax rate: " + taxRate + "%");
    System.out.println("Final invoice amount: $" + finalAmount);

  }
}
```

Loops

Suppose you wanted to build a program that did something that was very repetitive, such as printing the same sentence 50 times. This is where loops come in. You can replicate an action many times. That is why it is so useful. There are a few main types of loops: for loop, while loop, and do while loop

For Loops

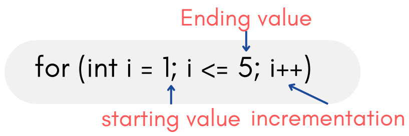
For loops are the more used type of loops, they are used to repeat something an exact number of times. They consist of an initialization, a condition, and an update statement, allowing precise control over iterations.

```
for (initialization; condition; update) {
// Code block to be executed repeatedly }
```

Here is an example.

```
 // Print numbers from 1 to 5 using a for loop
for (int i = 1; i <= 5; i++) {
 System.out.println(i);
}
```

Basically, the variable "i" acts like a counter. The counter will continue to increase by increments until it reaches the target number. Here is a breakdown.



```
for (int i = 1; i <= 5; i++)
```

In this example, the counter, i starts at 1 and will continue to execute the code inside the loop until it reaches 5. Each time the code gets executed the counter increases its value by 1. Like this:
i=1, i<=5 so execute the code, and i++
i=2, i<=5 so execute the code, and i++
i=3, i<=5, so execute the code, and i++
i=4, i<=5, so execute the code, and i++
i=5, i<=5, so execute the code, and i++

i=6,i>5, so end the loop. Here is another example of a loop

```
 // Prints every other number from 10 to 0
for (int i = 10; i >= 0; i-=2) {
 System.out.println(i);
}
```

The sign flips depending if you are going ascending or descending. If ascending, <. If descending, >.

If it is ascending, the loop will continue while i is less than the final value, and vice versa. Keep in mind that if you do not put the = after the < or >, the loop will not execute at the final value; it will execute up to one less than it.

```
 // Prints every other number from 10 to 1
for (int i = 10; i > 0; i--) {
 System.out.print(i);
}
```

This will only print 10987654321, if the = was added it would be 109876543210.

All the code inside the loop is exactly the same as everything before. **Note:** the counter initialized in a loop, cannot be accessed outside of the loop. In the example above, i can only be used inside the curly braces.

Practice:

Write a Java program that prints the multiplication table for a given number. The program should take an integer input from the user and then print the multiplication table up to a specific limit, such as 10.

For example, if the user enters 5, the program should print the multiplication table for 5 up to 10.

Solution:

```
System.out.print("Enter a number: ");
int number = scanner.nextInt();

System.out.println("Multiplication Table for " + number + ":");
for (int i = 1; i <= 10; i++) {
int result = number * i;
System.out.println(number + " x " + i + " = " + result);
}
```

Nested Loops:

Nested Loops are when one or more loops are embedded inside another loop. This concept may be a little hard to understand at first, so here is an example. To generate the times table of every number from 1-10, we could use a nested loop. Here is the code:

```
for (int i = 1; i <= 10; i++) {
        for (int j = 1; j <= 10; j++) {
              System.out.print("i * j : " + i*j + " ");
   }
         System.out.println();
}
```

- The outer loop starts with a counter **i** at 1.
- It then enters into the inner loop and begins iterating.
- For each value of **i** (from 1 to 10(inclusive)), the inner loop will execute 10 times.
- The inner loop's counter **j** starts from 1 and goes up to 10 (inclusive) for each iteration of the outer loop.
- Inside the inner loop, we calculate the product by multiplying the values of **i** (outer loop's counter) and **j** (inner loop's counter).
- We print the product of **i** and **j** on the screen for each iteration of the inner loop.
- After the inner loop finishes its 10 iterations, the outer loop proceeds to the next value of **i**.
- This process continues until the outer loop completes all its iterations from 1 to 10.
- Note }} is the same as separating them in 2 lines.

Here is another example. If we wanted to print out

```
*
**
***
****
*****
```

Here is how we can do it using nested loops.

```
for (int i = 1; i <= 5; i++) {
        // Inner loop for printing stars in each row
        for (int j = 1; j <= i; j++) {
           System.out.print("*");
        }
        // Move to the next row after
        // each inner loop completes
        System.out.println();
}
```

- The outer loop is used to iterate over the rows. It starts with **i = 1** and goes up to **i = 5**. For each value of **i**, the inner loop will execute **i** times.
- The inner loop prints the stars for each row. It starts with **j = 1** and goes up to **j = i**. So, in the first row (**i = 1**), the inner loop will execute once, and it will print a single star. In the second row (**i = 2**), the inner loop will execute twice and print two stars, and so on.
- **System.out.println()** is used to move to the next line after printing the stars for each row. This ensures that the stars are printed in a right-angled triangular pattern.

While Loops

For loops are used when code must be executed a specific number of times. While loops execute code based on a condition. It functions as an if statement but loops. The premise of the concept is that while something is true, code will run. Here is an example.

```
int x =0;
while(x<5){
     System.out.println("X is still not greater than 5");
     x++;
}
```

Note: You must have a statement inside the while loop that increments/decrements the variable being checked; otherwise, the loop will never end, and you will get an error.

Here is another example. Suppose you had to find the sum of all numbers between two numbers. This would be very time-consuming without loops. However, using while loops makes it very easy. Here is the code:

```
System.out.println("Enter the lower number");
int lower = scanner.nextInt() + 1;
//+1 is needed, since we don't want to add the lower number
System.out.println("Enter the upper number");
int upper = scanner.nextInt();
int sum =0;
while(lower<upper){
     sum+=lower;
     lower++;
}
System.out.println("Sum: "+sum);
```

The sum starts out at 0, then lower is added to the sum each iteration of the loop. At the end of each iteration of the loop, lower increases by 1. So in the next iteration, the next number can be added to the sum. The reason the loop condition is < instead of <= is because we don't want to include upper in the sum.

Do While Loops

Do While loops are essentially the same as a while loop; however, unlike while loops, they will always execute the code on their first iteration, even if the condition is not met. This is because they only check the condition after the first iteration of the loop. Here is an example.

```
int num =8;
do{
     System.out.println("Correct!");
}while(num<6);
```

Output: Correct!

This type of loop is not used very commonly. The while and for loop are the most important. Remember to include the semicolon at the end.

Note: Both While and Do While loops can be nested as well.

Practice

1.
Print this using nested for loops:
```
*****
****
***
**
*
```

2.
Print this using nested loops

```
****
****
****
```

3. Create a program that will find the product of all numbers from it to 1. For example 5: 1*2*3*4*5 =120;

4. Hard: Print this.
```
    1
   232
  34543
 4567654
567898765
```

5. Extremely Hard: Print this.

```
   *
  ***
 *****
*******
 *****
  ***
   *
```

Solution:

```
1.
for (int i = 5; i >=1; i--) {
    for (int j = 1; j <= i; j++) {
        System.out.print("*");
    }
    System.out.println();
}

2.
for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 4; j++) {
        System.out.print("*");
    }
    System.out.println();
}

3.
int num = 5; // example number
int product = 1;
for (int i = 1; i <= num; i++) {
    product *= i;
}
System.out.println("Product: " + product);

4.
int n = 5;
for (int i = 1; i <= n; i++) {
 for (int j = i; j < n; j++) {
 System.out.print(" ");
 }
 int numStart = i;
 for (int j = 1; j <= i; j++) {
 System.out.print(numStart++);
 }
 numStart -= 2;
 for (int j = 1; j < i; j++) {
 System.out.print(numStart--);
 }
 System.out.println();
}


5.
int n = 4; // height of upper triangle (excluding middle row)
for (int i = 1; i <= 2 * n - 1; i++) {
    int stars = i <= n ? 2 * i - 1 : 2 * (2 * n - i) - 1;
    int spaces = n - (i <= n ? i : 2 * n - i);
    for (int j = 0; j < spaces; j++) {
        System.out.print(" ");
    }
    for (int j = 0; j < stars; j++) {
        System.out.print("*");
    }
    System.out.println();
}
```

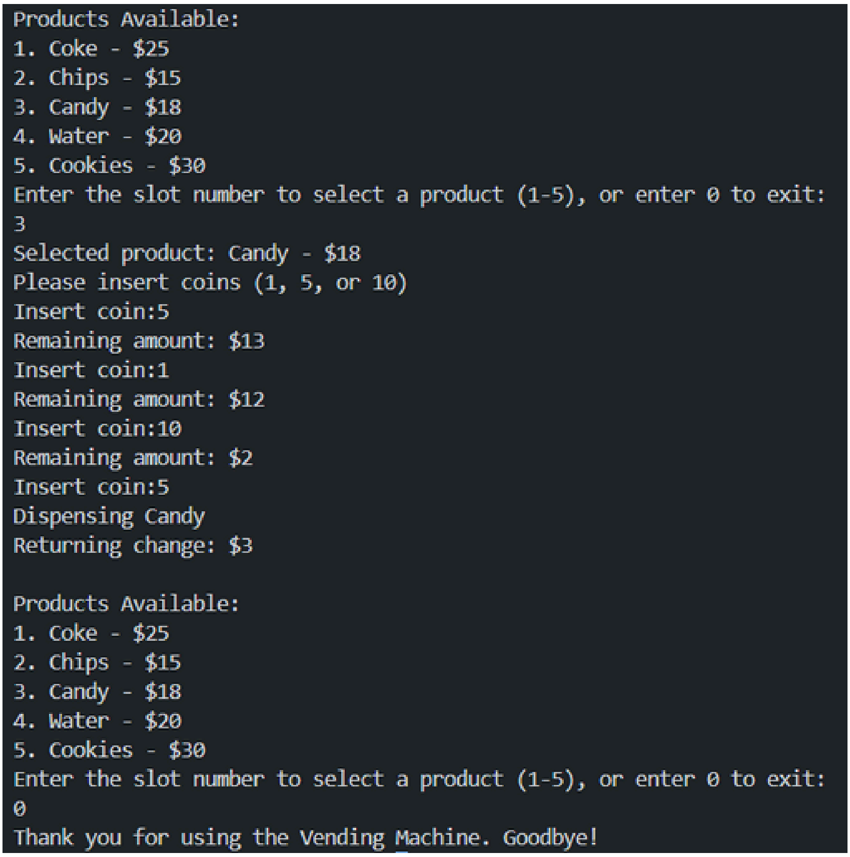
Project #2: Vending Machine

You are tasked with creating a Java program to simulate a vending machine. The vending machine has five slots, each containing different products. The machine only accepts coins of denominations 1, 5, and 10. The user can select a product by entering the slot number, and the machine will display the price of the selected product. The user needs to enter the required amount, and the machine will dispense the product and return the change (if any) in coins. Your code should do:

1. Create a Java program named VendingMachineSimulator.java.
2. Declare five products that are offered by the vending machine. You have to provide each product with its name and price. Product information shall be stored separately.
3. Print the product choices to the user.
4. The program should keep asking the user to choose a product until he/she wants to quit.
5. In case of invalid input of a slot number (less than 1 or greater than 5), show an error message and ask for input again.
6. If the user wants to exit the vending machine – entering 0, show some farewell message and exit from the program.
7. When a product is selected by the user, print the product name and price. Then prompt the user to insert coins (1, 5, or 10 denomination) until the required amount for the product is paid.
8. Print out how much they still owe after each payment. If a user enters an invalid coin (not 1, 5, or 10), then print an error message and ask for input again.

9. If the user inserts more money than the product's price, calculate the change and display it along with the dispensed product.
10. If the user inserts an exact change, dispense the product without returning any change.
11. Use appropriate data types, if statements, loops, and scanners for user input.

Here is a sample output:



```
Products Available:
1. Coke - $25
2. Chips - $15
3. Candy - $18
4. Water - $20
5. Cookies - $30
Enter the slot number to select a product (1-5), or enter 0 to exit:
3
Selected product: Candy - $18
Please insert coins (1, 5, or 10)
Insert coin:5
Remaining amount: $13
Insert coin:1
Remaining amount: $12
Insert coin:10
Remaining amount: $2
Insert coin:5
Dispensing Candy
Returning change: $3

Products Available:
1. Coke - $25
2. Chips - $15
3. Candy - $18
4. Water - $20
5. Cookies - $30
Enter the slot number to select a product (1-5), or enter 0 to exit:
0
Thank you for using the Vending Machine. Goodbye!
```

Solution:

```
import java.util.Scanner;
public class VendingMachineSimulator {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        // Define five products available in the vending machine.
        String product1 = "Coke";
        int price1 = 25;
        String product2 = "Chips";
        int price2 = 15;
        String product3 = "Candy";
        int price3 = 18;
        String product4 = "Water";
        int price4 = 20;
        String product5 = "Cookies";
        int price5 = 30;

        while (true) {
            System.out.println("Products Available:");
            System.out.println("1. " + product1 + " - $" + price1);
            System.out.println("2. " + product2 + " - $" + price2);
            System.out.println("3. " + product3 + " - $" + price3);
            System.out.println("4. " + product4 + " - $" + price4);
            System.out.println("5. " + product5 + " - $" + price5);
            System.out.println("Enter the slot number to select a
product (1-5), or enter 0 to exit:");

            int selection = scanner.nextInt();
            if (selection == 0) {
                System.out.println("Thank you for using the Vending
Machine. Goodbye!");
                break;
            }
            if (selection < 1 || selection > 5) {
                System.out.println("Invalid selection. Please try
again.");
                continue;
            }

            // Process the selection (Add functionality here as
needed)
```

```
// Get product details
String selectedProduct = "";
int price = 0;
switch (selection) {
    case 1:
        selectedProduct = product1;
        price = price1;
        break;
    case 2:
        selectedProduct = product2;
        price = price2;
        break;
    case 3:
        selectedProduct = product3;
        price = price3;
        break;
    case 4:
        selectedProduct = product4;
        price = price4;
        break;
    case 5:
        selectedProduct = product5;
        price = price5;
        break;
}
System.out.println("Selected product: " + selectedProduct + " -
$" + price);
System.out.print("Please insert coins (1, 5, or 10)\n");

int amountInserted = 0;
while (amountInserted < price) {
    System.out.print("Insert coin:");
    int coin = scanner.nextInt();

    if (coin != 1 && coin != 5 && coin != 10) {
        System.out.println("Invalid coin. Please insert coins (1, 5, or
10):");
        continue;
    }
    amountInserted += coin;
    if (amountInserted < price) {
        System.out.println("Remaining amount: $" + (price -
amountInserted));
    }
}
```

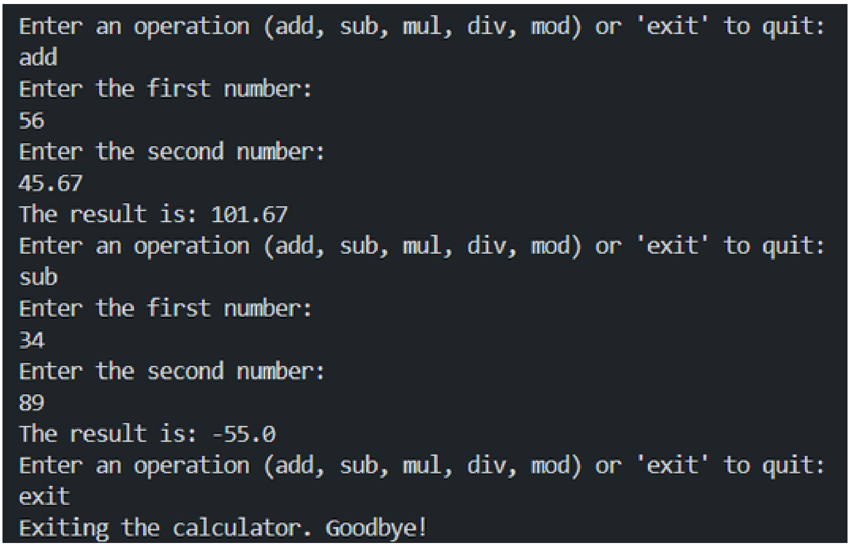
```
    // Dispense product and calculate change
      System.out.println("Dispensing " + selectedProduct);
      if (amountInserted > price) {
        int change = amountInserted - price;
        System.out.println("Returning change: $" + change);
      }

      System.out.println();
    }
  }
}
```

Project #3: Calculator

Design a basic application of a calculator in Java that performs one operation at a time, addition, subtraction, multiplication, division, and modulus by utilizing loops, conditionals, and math functions.

Requirements:

1. Your calculator is only supposed to perform one operation (addition, subtraction, multiplication, division, modulus).
2. Ask the user for the operation first and then ask for the first and second number.
3. In the case of a division by 0 print an error message.
4. Use loops to allow the user to make multiple calculations until "exit" is typed to end the program. Use conditionals to decide which operation to do based on the user input. Make sure the program exits when the user types "exit". Here's a sample output:



```
Enter an operation (add, sub, mul, div, mod) or 'exit' to quit:
add
Enter the first number:
56
Enter the second number:
45.67
The result is: 101.67
Enter an operation (add, sub, mul, div, mod) or 'exit' to quit:
sub
Enter the first number:
34
Enter the second number:
89
The result is: -55.0
Enter an operation (add, sub, mul, div, mod) or 'exit' to quit:
exit
Exiting the calculator. Goodbye!
```

Solution:

```
import java.util.Scanner;
public class calculator {
  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    String operation;

    while (true) {
      System.out.println("Enter an operation (add, sub, mul, div,
mod) or 'exit' to quit:");
      operation = scanner.nextLine();

      if (operation.equalsIgnoreCase("exit")) {
        System.out.println("Exiting the calculator. Goodbye!");
        break;
      }

      System.out.println("Enter the first number:");
      double num1 = scanner.nextDouble();
      System.out.println("Enter the second number:");
      double num2 = scanner.nextDouble();
scanner.nextLine(); // Consume the newline left-over
```

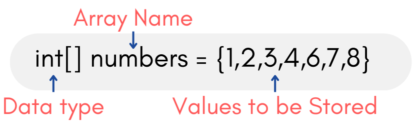
```
      switch (operation.toLowerCase()) {
        case "add":
          System.out.println("The result is: " + (num1 + num2));
          break;
        case "sub":
          System.out.println("The result is: " + (num1 - num2));
          break;
        case "mul":
          System.out.println("The result is: " + (num1 * num2));
          break;
        case "div":
          if (num2 == 0) {
            System.out.println("Error: Division by zero is not
allowed.");
          } else {
            System.out.println("The result is: " + (num1 / num2));
          }
          break;
        case "mod":
          if (num2 == 0) {
            System.out.println("Error: Division by zero is not
allowed.");
          } else {
            System.out.println("The result is: " + (num1 % num2));
          }
          break;
        default:
          System.out.println("Invalid operation. Please try
again.");
      }
    }

  }
}
```

Arrays

- Single Dimensional Arrays
- Application of Loops in Arrays
- Multidimensional Arrays
- Searching Algorithms
- Sorting Algorithms

Single Dimensional Arrays

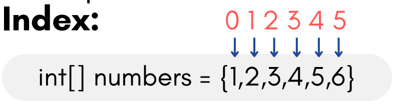
While this may sound very complex, the concept is very simple. Suppose you wanted to store 10 numbers in variables; normally, you would need 10 different variables. But, this is time-consuming to create and harder to use.
Arrays are a data structure that allows you to store multiple values in one place, making them easier to access and more functionally efficient. A data structure is a way of organizing data, and an array is one example. There are many more data structures. Feel free to search them up. Here is how you declare an array.



```
int[] numbers = {1,2,3,4,6,7,8}
```

In order to create the array, the [] is necessary. All values will be stored inside the curly braces. In order to access any of these values, the concept of index has to be thoroughly understood.
Index: An index is a value used to define the position of a value in an array. Each value in an array is assigned an index that can be used to access it. Take a look at this example.

Index:



```
int[] numbers = {1,2,3,4,5,6}
```

The first value of an array will have an index of 0, the second will have an index of 1, the third will have an index of 2, and the fourth will have an index of 3, and so on. It is basically the position minus 1.
Accessing a value in an array: To access a value in an array, simply put the index of the value in square brackets following the name of the array. Like this.

```
int first = numbers[0]
```

first = 1.
There is an alternate way of declaring an array. When doing it like this, an array of any type of number will have all values set to 0. A string value will preset all values to null. We have not covered this; null means empty. Here is the code:

```
int[] array = new int[5];
```

Where 5 is the length of the array (the total number of values that will be stored in the array), the array equals {0,0,0,0,0}.

Note: once an array has been declared either way the length of the array can **NEVER** change.

Important methods for arrays: **Result:**

.length = returns the length of the array
- Returned as an integer

Example:
System.out.println(array.length); **5**

Arrays.toString(*nameOfArray***);** = Printouts the array.
- Import java.util.Arrays to use it;

Example:
System.out.println(Arrays.toString(array)); **[0,0,0,0,0]**

Arrays.sort(nameOfArray); = Sorts the array.
- Import java.util.Arrays to use it;

Example:
int array = {5,4,3,2,1};
System.out.println(Arrays.sort(array)); **[1,2,3,4,5]**

Application of Loops in Arrays

The use of loops in arrays is key to its functionality. Here is an example. Suppose every number in an array had to be doubled, a for loop must be used. Here is the code:

```
int[] array = {1,2,3,4,5,6};
for(int i =0; i<array.length;i++){
array[i]*=2;
}
System.out.println(Arrays.toString(array));
```

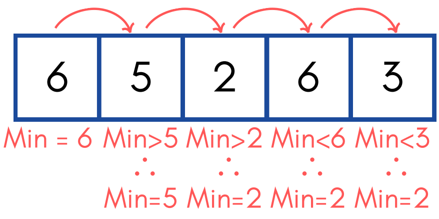
i is used to represent the index in an array. It increments, and goes through the array by doing so.

Output: [2,4,6,8,10,12]

Finding Minimum Value in an array:
The following code is a find to find the minimum value in an array using just a loop.

```
int min = arr[0];
for(int i =0;i<arr.length;i++){
    if(min>arr[i]){
        min = arr[i];
    }
}
```

Assume an array = {6,5,2,6,3}. Here is a visual of what is happening:



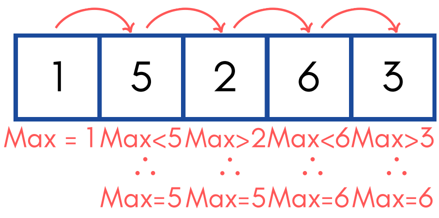
At the start, assume that the minimum value is the first value in the array. Then, iterate (loop) through the whole array to see if there is any value smaller than the current minimum. If there is, swap it to that and continue until the end of the array.

Finding Maximum Value in an array:
The following code is a way to find the maximum value in an array using just a loop.

```
int max= arr[0];
for(int i =0;i<arr.length;i++){
    if(max<arr[i]){
        max= arr[i];
    }
}
```

*Note: the logic is exactly the same as finding the minimum, except the sign is reversed in the if statement. Now, if the value in the array is greater than the current max, it becomes the new max. This is repeated till the end.

Assume an array = {1,5,2,6,3}. Here is a visual of what is happening:



Practice Problems

1. Given an array of numbers arr, write a program to find and print the sum of elements at even indices.
2. Write a program to reverse an array of integers. For example, {1,2,3,4,5} -> {5,4,3,2,1}
3. Given an array of integers arr, write a program to find and print the second largest element.
4. Given an array arr, write a program to rotate the array to the right by 2 positions. For example, {1,2,3,4} -> {3,4,1,2}

Solutions:

1.
```
int sum = 0;
for (int i = 0; i < arr.length; i += 2) {
 sum += arr[i];
}
System.out.println("Sum of even indexed elements: " + sum);
```

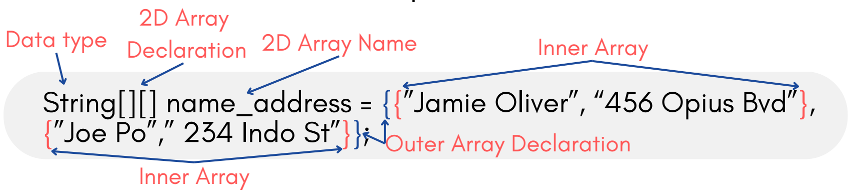
2.
```
int n = arr.length;
for (int i = 0; i < n / 2; i++) {
    int temp = arr[i];
    arr[i] = arr[n - 1 - i];
    arr[n - 1 - i] = temp;
}
System.out.println("Reversed array: " + Arrays.toString(arr));
```

3.
```
Arrays.sort(arr);
int secondLargest = arr[arr.length - 2];
System.out.println("Second largest value: " + secondLargest);
```

4.
```
int k = 2; // number of positions to rotate
int n = arr.length;
int[] rotated = new int[n];
for (int i = 0; i < n; i++) {
    rotated[(i + k) % n] = arr[i];
}
System.out.println("Rotated array: " + Arrays.toString(rotated));
```

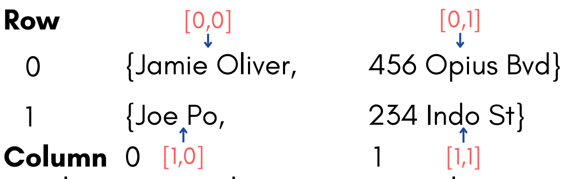
Multidimensional Arrays

In most cases, multidimensional arrays only extend to 2-D arrays, which will be covered here. If 2 values need to be stored together, for example, name and address, 2-D arrays are used. It is simply arrays inside an array. The outer array contains inner arrays, each of which carries information about different people's names and addresses. Here is an example of the declaration:



```
String[][] name_address = {{"Jamie Oliver", "456 Opius Bvd"},
{"Joe Po"," 234 Indo St"}};
```

Indices work differently for 2-D arrays. You need two indices: the first for the row (the inner array's position in the outer array) and the second for the column (the value's position in the inner array). Think of it as a table with rows and columns. The row index is for the inner array, and the column index is for the value within that array. You can access the value like this:
arrayName[row][column]
Using the name and address example:
Jamie Oliver = name_address[0][0]
456 Opius Blvd = name_address[0][1]
Joe Po = name_address[1][0]
234 Indo St = name_address[1][1]



Row	[0,0]	[0,1]
0	{Jamie Oliver,	456 Opius Bvd}
1	{Joe Po,	234 Indo St}
Column	0 [1,0]	1 [1,1]

Declaration without setting values:

```
String[][] table = new String[5][10];
```

There are 5 rows and 10 columns so 50 total elements. All of those elements are preset to 0, much like a 1-D array. Lets use all of this to fill a 2-D array with a multiplication table

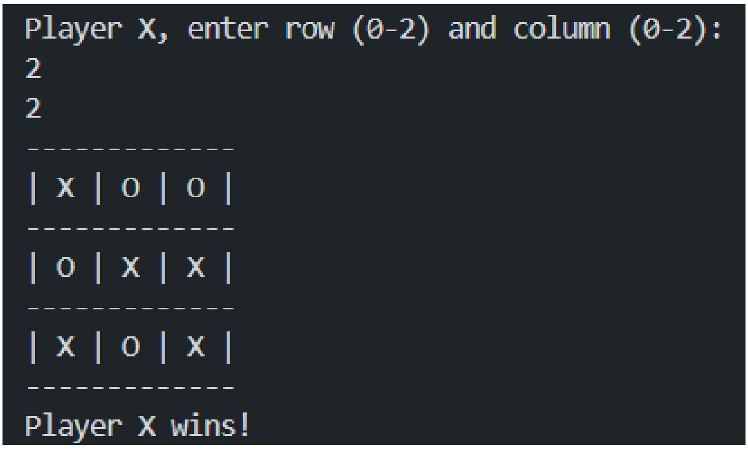
```
public class MultiplicationTable {
    public static void main(String[] args) {
        int[][] multiplicationTable = new int[10][10];// declaring
        //2-D array
        // Fill the 2D array with multiplication values
        for (int row = 0; row < 10; row++) {
            for (int col = 0; col < 10; col++) {
                multiplicationTable[row][col] = (row + 1) * (col + 1);
            }
        }
    }
}
```

Project #4 Tic-Tac-Toe Game

Create a Java Tic-Tac-Toe game. Your program should:

- Prompt the user to enter their move as row and column entries
- Make the computer play a random move after your move
- Create a method to check if a move is valid, and indicate if it is not
- Create a method to check if either computer has won, and indicate the winner
- Create a method to check if there is a tie(the board gets filled up)

Hint: Use a 2-D char array
Challenge: make the computer play optimally.
Here is an example of the ending of the program



```
Player X, enter row (0-2) and column (0-2):
2
2
-------------
| X | O | O |
-------------
| O | X | X |
-------------
| X | O | X |
-------------
Player X wins!
```

Solution

```
import java.util.Scanner;
public class TicTacToeWithComputer {
    public static void main(String[] args) {
        char[][] board = {
            {' ', ' ', ' '},
            {' ', ' ', ' '},
            {' ', ' ', ' '}
        };
        char currentPlayer = 'X';
        boolean gameWon = false;

        Scanner scanner = new Scanner(System.in);

        while (!gameWon && !isBoardFull(board)) {
            printBoard(board);

            if (currentPlayer == 'X') {
                System.out.println("Player X, enter row (0-2) and
column (0-2): ");
                int row = scanner.nextInt();
                int col = scanner.nextInt();

                if (isValidMove(board, row, col)) {
                    board[row][col] = currentPlayer;
                    gameWon = checkWin(board, currentPlayer);
                    currentPlayer = 'O';
                } else {
                    System.out.println("Invalid move. Try again.");
                }
            } else {
                System.out.println("Computer's Turn (Player O):");
                int compRow, compCol;
                do {
                    compRow = (int) (Math.random() * 3);
                    compCol = (int) (Math.random() * 3);
                } while (!isValidMove(board, compRow, compCol));

                board[compRow][compCol] = currentPlayer;
                gameWon = checkWin(board, currentPlayer);
                currentPlayer = 'X';
            }
        }
```

```
   printBoard(board);
        if (gameWon) {
            System.out.println("Player " + (currentPlayer == 'X' ? 'O'
: 'X') + " wins!");
        } else {
            System.out.println("It's a draw!");
        }


    }
    private static void printBoard(char[][] board) {
        System.out.println("-------------");
        for (int i = 0; i < 3; i++) {
            System.out.print("| ");
            for (int j = 0; j < 3; j++) {
                System.out.print(board[i][j] + " | ");
            }
            System.out.println("\n-------------");
        }
    }
    public static boolean isValidMove(char[][] board, int row,
int col) {
        return row >= 0 && row < 3 && col >= 0 && col < 3 &&
board[row][col] == ' ';
    }

    public static boolean checkWin(char[][] board, char player)
{
        for (int i = 0; i < 3; i++) {
            if (board[i][0] == player && board[i][1] == player &&
board[i][2] == player) {
                return true; // Check rows
            }
            if (board[0][i] == player && board[1][i] == player &&
board[2][i] == player) {
                return true; // Check columns
            }
        }
        if (board[0][0] == player && board[1][1] == player &&
board[2][2] == player) {
            return true; // Check diagonal
        }
```

```
 if (board[0][2] == player && board[1][1] == player &&
board[2][0] == player) {
            return true; // Check reverse diagonal
        }
        return false;
}
    private static boolean isBoardFull(char[][] board) {
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                if (board[i][j] == ' ') {
                    return false;
                }
            }
        }
        return true;
    }
}
```

Searching Algorithms

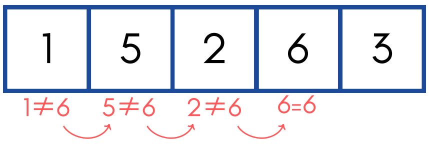
While programming, there are certain functions that are used very commonly, such as sorting an array and searching for a value in an array. There are many different ways to perform these functions.

Searching for a value in an array:

There are 2 main ways to search for a value in an array. First, **linear search**. Linear search involves iterating through the array element by element until the desired value is found. It checks all values in the array.

```
public static int linearSearch(int[] arr, int target) {
    for (int i = 0; i < arr.length; i++) {
        if (arr[i] == target) {
            return i; // Return index if found
        }
    }
    return -1; // Return -1 if not found
}
```

To show linear search visually, let arr={1,5,2,6,3} and the target =6.

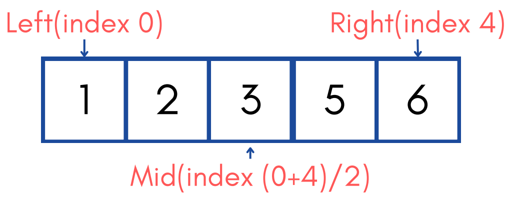


Starting with the first value, 1 is not equal to the target (6), so the algorithm proceeds to check the next value. 5 is not equal to the target, so the algorithm checks the next value. 2 is not equal to the target, so the algorithm checks the next value. 6 is equal to the target, so the loop ends as the index is returned. No values after 6 (index 3) are checked.

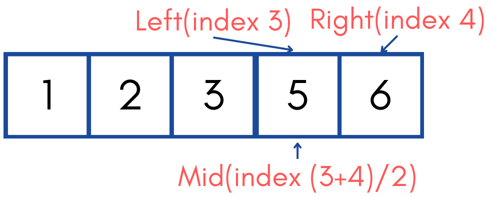
Binary search: a value is searched for by repeatedly dividing an array until it is found. The array **must** be sorted before using this algorithm. The algorithm has a left index(0) and a right index(last index in the array), it calculates the average of the 2(mid). If the value at index mid is greater than the target, the right index becomes mid -1. Since the array is sorted, all the values on the left of mid will be smaller than mid, which is where the target will be. If the target is greater than the value at index mid, left = mid+1. Here is the code:

```
public static int binarySearch(int[] arr, int target) {
    Arrays.sort(arr);
    int left = 0;
    int right = arr.length - 1;
    while (left <= right) {
        int mid = left + (right - left) / 2;
        if (arr[mid] == target) {
            return mid; // Return index if found
        }
        if (arr[mid] < target) {
            left = mid + 1;
        } else {
            right = mid - 1;
        }
}
    return -1; // Return -1 if not found
}
```

Lets use the same array and target as before. First the array must be sorted.



Since the value at the arr[mid] is less than the target, the target must be on the left of the middle value. Hence, we need to adjust the bounds of the next binary search to the right values of the middle value.



Now, the middle value (arr[mid]) is equal to the target; hence, the algorithm will return mid, which is the index of the target in the array. If the value did not exist, the while loop would end, and -1 would be returned.
Note: remember that indexes are integers, so mid is always going to truncate when being divided by 2. For example, (3+4)/2 = 4 instead of 4.5 or 5. This is because integers always loose the decimal.
When choosing which algorithm: if the array is sorted, use binary search as it is faster (there is a reason for that, which is not covered in this book). If the array is not sorted, use linear search.

Sorting Algorithms

There are many ways to sort an array. We will cover 2 of them in this book.

Selection sort: Imagine dividing the array into two parts:

1. Sorted part: Left side, grows as we sort.
2. Unsorted part: Right side, shrinks as we sort.

The algorithm works like this:

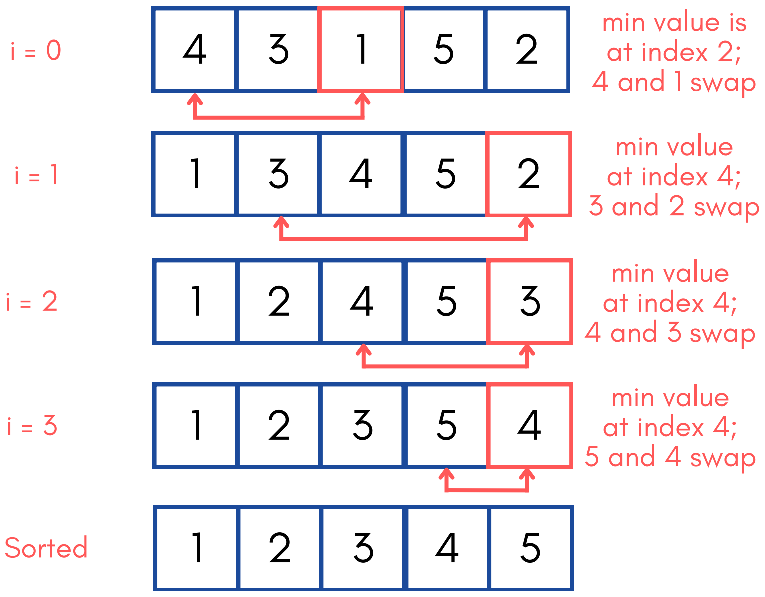
1. Find the smallest number in the unsorted part.
2. Swap it with the first number in the unsorted part.
3. This number now becomes part of the sorted part.
4. Repeat until everything is sorted.

See the visual on the next page for more clarity.

```
public static void selectionSort(int[] arr) {
    int n = arr.length;
    for (int i = 0; i < n - 1; i++) {
        int minIndex = i;
        for (int j = i + 1; j < n; j++) {
            if (arr[j] < arr[minIndex]) {
                minIndex = j;
            }
        }
        int temp = arr[minIndex];
        arr[minIndex] = arr[i];
        arr[i] = temp;
    }
}
```

Note: The temp variable is needed in order to swap the values. Store one value in temp, set one value = to the other value, then set the other value to temp.

Let's use an array with scrambled numbers 1-5.



i = 0
4 3 1 5 2
min value is at index 2; 4 and 1 swap
i = 1
1 3 4 5 2
min value at index 4; 3 and 2 swap
i = 2
1 2 4 5 3
min value at index 4; 4 and 3 swap
i = 3
1 2 3 5 4
min value at index 4; 5 and 4 swap
Sorted
1 2 3 4 5

Here are some more examples:

```
arr = {2,3,5,7,1}
i:
0.1,3,5,7,2
1.1,2,5,7,3
2.1,2,3,7,5
3.1,2,3,5,7
```

```
arr = {9,4,8,1,2}
i:
0.1,4,8,9,2
1.1,2,8,9,4
2.1,2,4,9,8
3.1,2,4,8,9
```

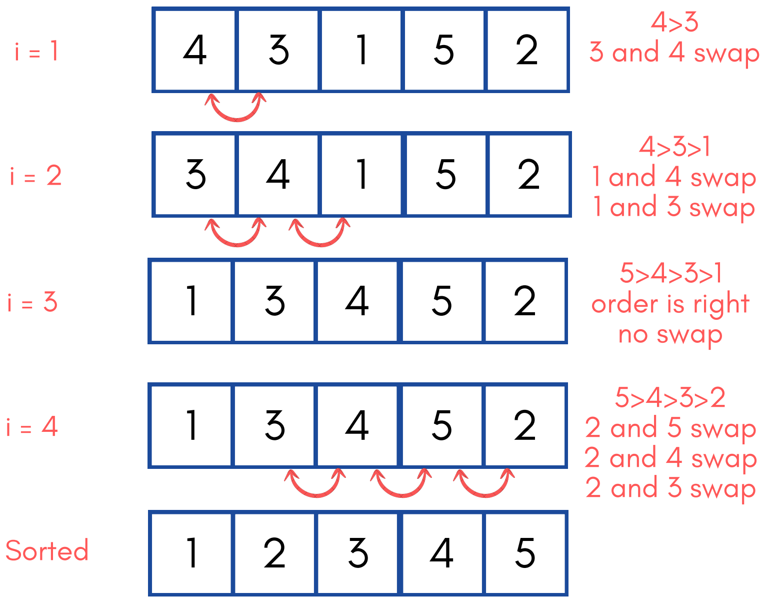
```
arr = {5,1,6,4,3}
i:
0.1,5,6,4,3
1.1,3,6,4,5
2.1,3,4,6,5
3.1,3,4,5,6
```

Insertion sort:
The algorithm works like this:

1. Start with the second element (index 1).
2. Compare it with the elements to its left.
3. If the element on the left is larger, swap values. Keep swapping until the start of the array.
4. Insert the element in its correct position.
5. Repeat for all remaining unsorted elements.

```
public static void insertionSort(int[] arr) {
    int n = arr.length;
    for (int i = 1; i < n; i++) {
        int key = arr[i];
        int j = i - 1;
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = key;
    }
}
```

Let's use the same array with scrambled numbers 1-5.



Closure

- Final Challenge#1: Integer to Roman Numeral
- Final Challenge#2:2D Array Rotation
- Final Challenge#3: Arithmetic Solver
- About the Authors

Final Challenge#1: Integer to Roman Numeral

Level: **Easy**

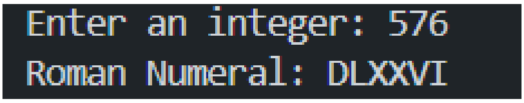
Create a Java Method that Converts an integer to a Roman number. Your program should:

- Input an Integer as a Parameter
- Convert and Return the Roman Numeral Form in String form

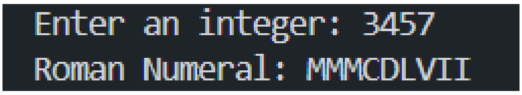
Here is the conversion table:

I = 1
V = 5
X = 10
L = 50
C = 100
D = 500
M = 1000

Examples:



```
Enter an integer: 576
Roman Numeral: DLXXVI
```



```
Enter an integer: 3457
Roman Numeral: MMMCDLVII
```

Solution:

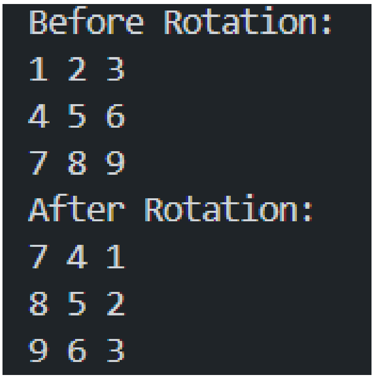
```
   public static String intToRoman(int num) {
      String result = "";
      int[] values = {1000, 900, 500, 400, 100, 90, 50, 40, 10, 9, 5,
  4, 1};
      String[] symbols = {"M", "CM", "D", "CD", "C", "XC", "L",
  "XL", "X", "IX", "V", "IV", "I"};

      for (int i = 0; i < values.length; i++) {
        while (num >= values[i]) {
          num -= values[i];
          result+=symbols[i];
        }
      }

      return result;
    }
```

Final Challenge#2: 2D Array Rotation

Level: Medium

Write a Java method that takes a 2D array as input and rotates it 90 degrees clockwise. Implement the method rotateArray and provide the transformed 2D array as output.

Example:



```
Before Rotation:
1 2 3
4 5 6
7 8 9
After Rotation:
7 4 1
8 5 2
9 6 3
```

Solution:

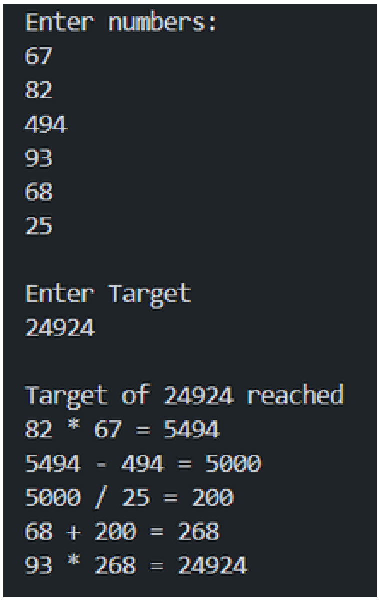
```
public static void rotateArray(int[][] arr) {
  int n = arr.length;

  for (int i = 0; i < n; i++) {
    for (int j = i; j < n; j++) {
      int temp = arr[i][j];
      arr[i][j] = arr[j][i];
      arr[j][i] = temp;
    }
  }

  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n / 2; j++) {
      int temp = arr[i][j];
      arr[i][j] = arr[i][n - j - 1];
      arr[i][n - j - 1] = temp;
    }
  }
}
```

Final Challenge#3: Arithmetic Solver

Level: Hard

Write a Java method that takes 6 numbers and a target value as parameters. It determines if using any combination of operations between those 6 numbers can compute the target value. Each number must be used and can only be used once.

Example:



```
Enter numbers:
67
82
494
93
68
25

Enter Target
24924

Target of 24924 reached
82 * 67 = 5494
5494 - 494 = 5000
5000 / 25 = 200
68 + 200 = 268
93 * 268 = 24924
```

Solution:

Due to the 50-page constraint, the code solution cannot be included in this book. But here is an overview:

- Compute Every Permutation of the Input Numbers (720 permutations)
- Compute Every Combination of 5 Operators (+, -, *, /) (1024 combinations)
- Compute Valid Operation Orders for Applying 5 Operators (Reverse Polish Notation)
- For Each Number Permutation, Try Every Combination of Operators with Every Valid Operation Order

About the Authors:

Rohan Sinha: As a freshman in high school, I took an introductory Java course that was slow-paced and repetitive, yet lacked crucial information. This experience inspired me to write a book that teaches beginners the most important Java topics in both a condensed and detailed manner. Now a rising junior, I've expanded my skills through various experiences. I'm currently a Computational Game Theory Research Intern at the University of Maryland, have interned as a software engineer at a startup, working on both frontend and backend development, and have taught 100+ students programming.
This book aimed to provide the engaging and efficient introduction to Java that I wish I'd had when starting out. I hope it did!

Samanth Jain: With similar experiences to Rohan as a freshman taking a Computer Science Course, I believe this book can serve as a tool to enrich one's computer science experience and help with preparing for APCSA. Now a rising junior, I have expanded my skills significantly to include libraries and languages such as React, React Native, Spring Boot, and MySQL, and achieved a score of 5 on the APCSA exam. Drawing from my own experiences and the skills I've developed, I hope this book serves as a valuable resource for you and inspires your passion for computer science. Happy coding!

About the Authors:

Rithik Paladugu: As a freshman going into high school without a real sense of direction, I took the same Computer Science class as my friends because I didn't want to feel left out. In the course itself, I developed a strong liking towards coding and machine learning, intrigued and wanting to learn more. Today, after getting a 5 on the APCSA exam, I'm trying to broaden my learning through studying quantum. After achieving the necessary certifications, I plan to try and build quantum algorithms to help people. I hope this book can serve as a guide for curious minds new to Computer Science, like how I was just a few years ago. Good luck coding!