

# **Java Fundamentals**

## **Student Workbook 2a – Strings Loops and Arrays**

Version 7.0 Y

# Table of Contents

<b>Module 1 Working with Strings</b>	<b>3</b>
Section 1-1 Strings	5
Strings	6
Comparing Strings	7
Examples: String Methods	9
Exercises – Full Names	13
Section 1-2 Converting Strings	16
Converting a String to a Number	17
Converting a String to a Date	18
Exercises – HighScoreWins	20
Using Artificial Intelligence	22
Section 1-3 Working with StringBuilder	25
StringBuilder	26
Example: Using a StringBuilder	27
Exercise – Address Builder	28
<b>Module 2 Breaking a Program into Classes</b>	<b>29</b>
Section 2-1 Classes	30
Object-Oriented Programming	31
Encapsulation	33
Creating the Class	34
Instantiating an Object	35
Exercises – Cell Phone Service	37
Section 2-2 Working with Objects	39
Working with Objects	40
Default Access Modifier	41
Other Methods	42
Adding Methods - Example	43
Exercises – Cell Phone	45
Section 2-3 Overloading Methods	47
Overloading Methods	48
Overloading Constructors	50
Example: Working with a Class	52
Exercises – Cell Phone Pt. 2	53
<b>Module 3 Loops and Arrays</b>	<b>54</b>
Section 3-1 Loops	55
Types of Loops	56
while Loop	57
do/while Loop	58
for Loop	59
break vs continue	60
Exercises - Looping	63
Section 3-2 Arrays	66
Arrays	67
for-each Loop	69
Passing an Array to a Method	70
Returning an Array from a Method	71
Sorting an Array	72
Copying an Array	73
Other Array Methods	75
Exercises – Test Statistics	76
Self-Check	79
<b>Module 4 Continued Learning</b>	<b>80</b>
Section 4-1 Practice Repository	81
Practice Problems	82
Section 4-2 Code Wars (Optional)	83
What is CodeWars	85
CodeWars Setup	86
Join a Clan	87
CodeWars Example	89
CodeWars Kata (Optional)	95
CodeWars Kata (Optional)	96
CodeWars Kata (Optional)	97

# **Module 1**

## **Working with Strings**



## Section 1–1

### Strings

# Strings

---

- The Java `String` class has many methods available to you to work with the data you encounter
  - See: [https://www.w3schools.com/java/java\\_ref\\_string.asp](https://www.w3schools.com/java/java_ref_string.asp)
- Java is very specific about using double quotes with strings
  - This is because single quotes are used with characters
- If your string needs to include quotes within the text, you must escape the quotes

## Example

```
String welcome = "Big Tex says \"Howdy\"!";
```

```
String welcome2 = "Let's get this started!";
```

– Other escape characters include:

- \* `\n` - newline

- \* `\t` - tab

- \* `\r` - return

# Comparing Strings

---

- **WARNING: Do not use the equality operator to compare strings**
  - In Java, the equality operator ( == ) only checks the referential equality of the string operands
  - This means they reference the same underlying object

## Example

```
// The compiler creates a string literal "ABC" and
// the s1 variable references it
String s1 = "ABC";

// s2 references the **same string literal**
String s2 = "ABC";

// But s3 does NOT reference that literal because its
// value was created by performing concatenations
String prefix = "A";
String s3 = prefix + "BC";

if (s1 == s2) {
    // this will be true
}

if (s1 == s3) {
    // this will be false even though they both
    // would display the string "ABC"
}
```

- **To compare strings, use the `equals()` method**
  - It compares two strings character by character

### **Example**

```
String s1 = "ABC";
String s2 = "ABC";
String s3 = "A" + "B" + "C";

if (s1.equals(s2)) {
    // this will be true
}

if (s1.equals(s3)) {
    // this will be true
}
```

- **You can also use the `equalsIgnoreCase()` if you want the capitalization of characters to be unimportant**

### **Example**

```
String s1 = "ABC";
String s2 = "abc";
String s3 = "A" + "b" + "C";

if (s1.equalsIgnoreCase(s2)) {
    // this will be true
}

if (s1.equalsIgnoreCase(s3)) {
    // this will be true
}
```



# Examples: String Methods

---

- The **String** class has many methods that allow you to manipulate strings
  - The following pages show some examples of commonly used String methods and coding tricks

## Example

You can get the length of a string.

```
String fullName = "Dana Wyatt";  
int length = fullName.length();  
  
// length contains 10
```

## Example

Trim leading and trailing spaces from a string.

```
String username = " danaw ";  
username = username.trim();  
  
// now it contains "danaw"
```

## Example

Converts a string to uppercase.

```
String username = "danaw";  
username = username.toUpperCase();  
  
// now it contains "DANAW"
```

## Example

You can chain the calls.

```
String username = " danaw ";  
username = username.trim().toUpperCase();  
  
// now it contains "DANAW"
```

## Example

You can chain the calls.

```
String state = "New Mexico";  
state = state.toLowerCase();  
  
// now it contains "new mexico"
```

## Example

You can determine if a string starts with a particular substring

```
String discountCode = "FAIR-150FF";  
  
if (discountCode.startsWith("FAIR")) {  
    // this is true  
}
```

## Example

You can determine if a string ends with a particular substring

```
String discountCode = "SAVE-50";  
  
if (discountCode.endsWith("-50")) {  
    // this is true  
}
```

## Example

You can return the character at a specific position

```
String trackingCode = "USA-12981-Y-22";  
char hasShipped = trackingCode.charAt(10);  
// hasShipped contains 'Y'
```

## Example

You can return the position where a substring begins. It returns -1 if it doesn't find it.

```
String productCode = "ACME-12213";  
int dashPosition = productCode.indexOf("-");  
// dashPosition contains 4
```

## Example

You can return a substring from a larger string if you know the starting and ending index. The ending index is exclusive (that is, it is not included in the substring. If you omit the ending index, it takes the remainder of the string.

```
String productCode = "ACME-12213";  
int dashPosition = productCode.indexOf("-");  
String vendor = productCode.substring(0, dashPosition);  
String productNum = productCode.substring(dashPosition + 1);  
// vendor contains ACME and productNum contains 12213
```

## Example

You can split a string based on a regular expression that describes the delimiter. The pipe character has special meaning in regular expressions, so we had to escape it in this example.

```
String input = "Dallas|Ft. Worth|Austin";  
String[] cities = input.split("\\|");  
// cities is an array containing 3 strings  
// [0] is Dallas, [1] is Ft. Worth, [2] is Auston
```

Alternatively we could have coded the following. The `Pattern.quote()` method returns a regular expression for the pipe.

```
String[] cities = input.split(Pattern.quote("|"));
```

# Exercises – Full Names

---

Begin by creating a new folder for this week's work. In your `C:/pluralsight` folder create a new subfolder named `workbook-2`. You will do all of your work this week in this folder

**Remember:** When creating a new Java project, create a new git repository and commit your changes often! Don't forget to push to `github.com`. Always create the GitHub repository before creating a new Java project.

## EXERCISE 1

### Step 1

Create a Java application called `FullNameGenerator`. Ensure that you set the `GroupId` to `com.pluralsight`. Create a new package named `com.pluralsight` and add a new class called `FullNameApplication` for the `main()` method.

You will prompt the user to enter the parts of their name. You will then create a new string that holds the user's full name.

The user will always have a first and last name, they may also have a middle name and a suffix (ex. Jr, PhD). If a user does not enter a value for the middle name or suffix, the full name should not include that part of the name. If the user has a suffix in their name, the full name should include a comma before the suffix.

```
Glen Williams
Glen C. Williams
Glen Williams, PhD
Glen C. Williams, PhD
```

Make sure to trim any leading or trailing spaces that the user may have entered.

**Note:** the `↵` represents the user hitting the **ENTER** key

```
Please enter your name
First name: Glen↵
Middle name: ↵
Last name: Williams↵
Suffix: ↵

Full name: Glen Williams
```

A different example would be a name with a middle initial:

```
Please enter your name
First name: Glen↵
Middle name: C. ↵
Last name: Williamson↵
Suffix: ↵

Full name: Glen C. Williams
```

## **Step 2**

1. Ensure all your changes are committed and pushed to GitHub
2. Send your repository URL to your Instructor

## EXERCISE 2

### Step 1

Create a Java application called `FullNameParser` that prompts the user to enter a name in one of the following two formats:

first last            or            first middle last

Make sure to trim the name before proceeding in case the user entered leading or trailing spaces.

Parse the name and process it so that you can display the individual pieces of the name.

A sample trace of your program output is shown below. Bolded code is what the user entered.

```
Please enter your name:   Dana L. Wyatt    ↵  
First name: Dana  
Middle name: L.  
Last name: Wyatt
```

A different trace would be:

```
Please enter your name: Dana Wyatt↵  
First name: Dana  
Middle name: (none)  
Last name: Wyatt
```

### Step 2

1. Ensure all your changes are committed and pushed to GitHub
2. Send your repository URL to your Instructor

## Section 1–2

# Converting Strings



# Converting a String to a Number

---

- **Java has a series of methods that convert a string to a number, including:**
  - `Integer.parseInt()`
  - `Float.parseFloat()`
  - `Double.parseDouble()`

## **Example**

```
// string contains "id|description|quantity|price"
String input = "111|Hot Chocolate (12-count)|21|4.99";
String[] tokens = input.split(Pattern.quote("|"));

int id = Integer.parseInt(tokens[0]);
String name = tokens[1];
int quantity = Integer.parseInt(tokens[2]);
double price = Double.parseDouble(tokens[3]);
```

# Converting a String to a Date

---

- **Java also has the ability to convert a String into a Date**

- `LocalDate.parse()`

## Example

```
String userInput = "2002-10-17";  
LocalDate birthDay = LocalDate.parse(userInput);
```

- **The `LocalDate.parse()` method requires the international date standard format**
  - This is the ISO 8601 standard
  - YYYY-MM-DD
- **However, it is possible to specify a different format when parsing a date by using the `DateTimeFormatter` class**
  - `DateTimeFormatter.ofPattern()`

## Example

```
String userInput;  
DateTimeFormatter formatter;  
  
userInput = "10/17/2022";  
formatter = DateTimeFormatter.ofPattern("MM/dd/yyyy")  
  
LocalDate birthDay = LocalDate.parse(userInput, formatter);
```

- The `DateTimeFormatter` can be used for many formats

### Example

```
String userInput;  
DateTimeFormatter formatter;  
  
userInput = "7 Oct 2002";  
formatter = DateTimeFormatter.ofPattern("d MMM yyyy")  
  
LocalDate birthDay = LocalDate.parse(userInput, formatter);
```

- **Formatter values**

Code	Description
d	Allows for single digit days (i.e. 2002-10-5)
dd	Requires 2 digit days (i.e. 2002-10-05)
M	Allows for single digit months (i.e. 2002-7-12)
MM	Required 2 digit months (i.e. 2002-07-12)
MMM	Abbreviated month name (i.e. 12 Aug 2002)
MMMM	Full month name (i.e. 12 August 2002)
yy	Last 2 digits of the year (defaults to the current century)
yyyy	Full 4 digit year

# Exercises – HighScoreWins

---

All projects should be created in the `workbook-2` folder.

## EXERCISE 3

Create a new Java application called `HighScoreWins` that prompts the user to enter team and score information of an event. The input will be a single string that contains both team names and the final score of the match.

**Remember:** When creating a new Java project, create a new git repository and commit your changes often! Don't forget to push to `github.com`. Always create the GitHub repository before creating a new Java project.

### Step 1

Prompt the user for a game score.

The user will input a score in the following format:

```
Home:Visitor|21:9
```

Based on the user input, you should determine which team had the higher score and display the name of the winning team.

**Hint:** You will have to split the user input multiple times to get all of the information

- one split will be on the pipe ( | )
- one split will be on the colon ( : )

A sample trace of your program output is shown below. Bolded code is what the user entered.

```
Please enter a game score: Home:Visitor|21:9
```

```
Winner: Home
```

A different trace would be:

```
Please enter a score: Slytherin:Gryffindor|23:59
```

```
Winner: Gryffindor
```

### Step 2

1. Ensure all your changes are committed and pushed to GitHub
2. Send your repository URL to your Instructor

## EXERCISE 4

Create a Java application called **TheaterReservations**. This application is used to allow customers to reserve tickets for a specific date.

Prompt the user for their full name, the date of the show, and the number of tickets they will need.

Once the reservation is made, display a confirmation message to the customer in this format:

```
# ticket(s) reserved for (date) under (LastName, FirstName)
```

If only one ticket is reserved, the confirmation should not include an "s"

A sample trace of your program output is shown below. Bolded code is what the user entered.

```
Please enter your name: Geri Johnson↵
What date will you be coming (MM/dd/yyyy): 5/13/2023↵
How many tickets would you like? 3↵

3 tickets reserved for 2023-05-13 under Johnson, Geri
```

A different trace would be:

```
Please enter your name: Geri Johnson↵
What date will you be coming (MM/dd/yyyy): 5/13/2023↵
How many tickets would you like? 1↵

1 ticket reserved for 2023-05-13 under Johnson, Geri
```

## Step 2

1. Ensure all your changes are committed and pushed to GitHub
2. Send your repository URL to your Instructor

# Using Artificial Intelligence

---

- **Let's talk about how you can effectively use AI tools like Claude and ChatGPT to support your learning in this Java class.**
- **What can AI tools do?**
  - AI tools can be valuable learning companions when used appropriately
  - These tools are designed to help you understand concepts, not to do your work for you. Using AI to generate solutions you submit as your own work is a form of academic dishonesty
- **How can we use AI effectively?**
  - Ask for concept explanations - not complete solutions
    - \* **Good:** "Can you explain how loops work in Java?"
    - \* **Avoid:** "Write me a Java program that uses loops to sum an array"
- **Request analogies that connect to your interests**
  - **Example:** "I play soccer. Can you explain Java arrays like positions on a soccer team?"
- **Get clarification on error messages**
  - When you've written code that doesn't work, you can ask about specific error messages
  - Always show what you've tried and what error you're getting

- **Deepen your understanding of class examples**
  - If we covered methods in class, you might ask "Why would I break my code into different methods instead of putting everything in main()?"
- **Using AI to Support Problem-Solving**
  - When you're working on assignments, AI tools should help you understand the problem, not solve it for you.
- **Break down complex problems**
  - "What steps would I need to think about when writing a program that averages test scores?"
- **Review your solutions**
  - After you've completed your own solution, you can ask: "What could go wrong with my approach to this for loop?"
  - This helps you review your own work critically
- **Understand course concepts in context**
  - "How do arrays and loops work together in Java programs?"

## • Example: Getting Help with a Concept

I'm confused about methods in Java. When should I create a method versus just putting code in the main method?

Reasoned about Java methods usage for 5 seconds >

In Java, the main method is just the entry point of your program. It tells the JVM where to start executing your code, but it doesn't mean you should put every single line of code there. Here are some guidelines to help you decide when to create your own methods versus placing code directly in the main method:

### 1. Separation of Concerns

- **Modularity:** Break your program into distinct parts that each handle a specific task. For example, if you have a piece of code that performs calculations, encapsulate that logic in its own method. This makes your code easier to read and maintain.
- **Single Responsibility:** Each method should ideally do one thing. This aligns with the Single Responsibility Principle, which makes debugging and updating your code much simpler.

### 2. Reusability

- If you find yourself using the same block of code multiple times, it's a strong indication that it should be turned into a method. This avoids code duplication and makes future modifications easier—you only need to change the code in one place.

### 3. Readability and Maintainability

- **Clarity:** A well-named method can act like a mini-document within your code, explaining what the block of code does without the need for extensive comments.
- **Organization:** Keeping the main method clean and concise allows you to see the overall flow of your program at a glance, which is particularly helpful as your project grows in size and complexity.

### 4. Testing and Debugging

- Isolated methods are easier to test. When your logic is divided into smaller methods, you can write unit tests for each method to ensure they work correctly in isolation.
- Debugging becomes simpler because you can pinpoint which part of the code might be causing an issue.

### 5. When to Use the Main Method Directly

- For very small programs or quick experiments, you might write everything in the main method. However, even in these cases, if you notice that a certain functionality could be reused or clarified, it's a good idea to extract it into its own method.

### Example

Imagine you have a program that needs to:

1. Greet the user.
2. Process some data.
3. Display results.

Instead of cramming all that into the main method, you could do something like this:

```
java
Copy

public class MyProgram {
    public static void main(String[] args) {
        greetUser();
        int result = processData(5, 10);
        displayResult(result);
    }

    public static void greetUser() {
        System.out.println("Welcome to MyProgram!");
    }

    public static int processData(int a, int b) {
        // Perform some calculations
        return a + b;
    }

    public static void displayResult(int result) {
        System.out.println("The result is: " + result);
    }
}
```

This structure makes each task clear, and if you ever need to change the way you greet the user or process data, you know exactly where to go.

### Conclusion

Creating separate methods helps manage complexity, improves readability, enhances maintainability, and allows for easier testing. Reserve the main method for orchestrating the flow of your program rather than handling all the logic directly. This approach is a best practice in Java programming and will serve you well as your projects grow.

Does this help clarify the difference between putting code in the main method and creating your own methods?

📄 🗨️ 🔄 🌐

## • Remember

- AI tools should supplement your learning, not replace it
- The goal is to understand Java basics deeply, not to collect code snippets · If you're unsure whether a particular use of AI is appropriate, ask your instructor
- The skills you build by solving problems yourself with loops, arrays, methods, and objects will be the foundation for everything else you learn in programming



## Section 1–3

### Working with StringBuilder

# StringBuilder

---

- **When the Java `String` class creates a string object, it is immutable**
  - This means it can't change
- **So if you make any change to a string variable (for example, by concatenating characters to it), it disposed of the previous string object and creates a new one**
  - This is very inefficient if you are going to make many changes to the string
- **In these situations, you should use the Java `StringBuilder` class**
- **It creates the underlying data structure as a mutable sequence of characters**
  - It will continue working on the same sequence of characters as you make changes to it
- **However, it isn't a `String` object so it has an entirely different collection of methods**
  - Once you use it to build the string you need, you call its `toString()` method and retrieve the result as a string

# Example: Using a StringBuilder

---

## Example

```
public class SkillsBuilding {  
    public static void main(String[] args) {  
        // create a StringBuillder object  
        StringBuilder skills = new StringBuilder();  
  
        // append strings to the StringBuilder object  
        skills.append("Git, ");  
        skills.append("HTML, ");  
        skills.append("CSS, ");  
        skills.append("and Bootstrap\n");  
        skills.append("JavaScript, ");  
        skills.append("ES6, ");  
        skills.append("jQuery, ");  
        skills.append("REST APIs, ");  
        skills.append("and Express\n");  
        skills.append("Angular\n");  
        skills.append("Java\n");  
  
        // retrieve the underlying characters from  
        // the StringBuilder  
        String mySkills = skills.toString();  
        System.out.println(mySkills);  
    }  
}
```

# Exercise – Address Builder

---

In this exercise you will create a new application that accepts user input for customer information such as billing and shipping addresses. The input information should be concatenated into a single variable.

**Remember:** When creating a new Java project, create a new git repository and commit your changes often! Don't forget to push to github.com. Always create the Github repository before creating a new Java project.

## EXERCISE 5

Create a Java application named **AddressBuilder**. You will prompt the user for their Billing and shipping addresses. Instead of changing the value of a string, you will use a **StringBuilder** to **dynamically build the string** that holds all the address information. Then print the information to the screen.

```
Please provide the following information:
```

```
Full name: Grover Smith
```

```
Billing Street: 123 Main Street
```

```
Billing City: Middleton
```

```
Billing State: TX
```

```
Billing Zip: 75111
```

```
Shipping Street: 456 Big Sky Blvd
```

```
Shipping City: Outer Rim
```

```
Shipping State: TX
```

```
Shipping Zip: 75333
```

As the user inputs the information you should use the `StringBuilder append()` method to append the new input to the string.

Display the customer information

```
Grover Smith
```

```
Billing Address:
```

```
123 Main Street
```

```
Middleton, TX 75111
```

```
Shipping Address:
```

```
456 Big Sky Blvd
```

```
Outer Rim, TX 75333
```

## Step 2

1. Ensure all your changes are committed and pushed to GitHub
2. Send your repository URL to your Instructor

## **Module 2**

# **Breaking a Program into Classes**

## Section 2–1

### Classes

# Object-Oriented Programming

---

- **Let's take a moment to glance at classes in Java**
- **Classes are a way of packaging data and the operations that act upon that data in a single unit**
  - In JavaScript, we mostly used classes to group data together
  - In TypeScript, we represented components, models and services using classes and defined data in them, as well as functionality like event handlers in components or methods that fetched API data in services
- **In Java, we can't have global functions so all executable code must belong to a class**
- **Thus far, we've been looking at our Java application class that has the `static main()` method**
  - This type of class isn't used to create objects -- but is used to bootstrap the application to start running
- **When we typically talk about objects, they are instances of classes that represent the nouns of a business**
  - A university would have classes for Student, Course, Department, Faculty, Semester, Transcript etc
  - An insurance company would have classes for Customer, Policy, Claim, Agent, etc.

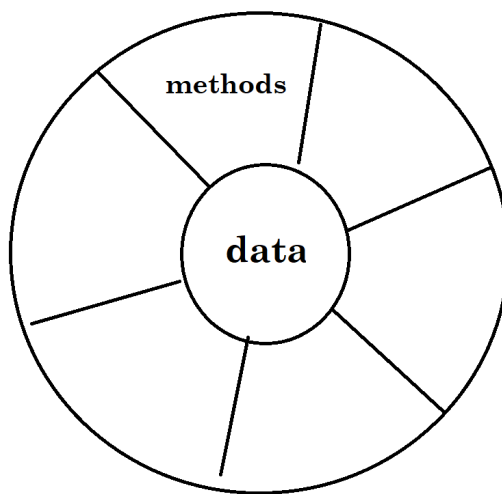
- **We also have classes that represent software nouns**
  - For example, `String`, `File`, `Connection`, `Exception`, etc
  - These are part of the thousands of Java classes shipped as part of the JRE
- **We will spend extensive time next week learning about and practicing object-oriented programming skills using Java**
- **Now, let's figure out how to create a basic class in Java so that we can start becoming comfortable with terms like:**
  - encapsulation
  - class
  - object
  - constructor
  - instantiate an object
  - call a method
  - overload a method



# Encapsulation

---

- When we create classes in Java, we talk about the term *encapsulation*
  - An object-oriented programmer wants to design a class that hides how it works *inside* the class
  - They provide methods someone can call to interact with the class' objects
- This "black box" approach allows the way a class works to be tweaked without affecting code that interacts with an object of that class
- Central to the concept of encapsulation are the access specifiers **public** and **private**
  - Public members are accessible outside of the class
  - Private members are only accessible within an object



you can use an object's public methods  
which in turn access the object's private data

# Creating the Class

---

- **Recall that in Java, a class has to be coded in a .java file of the same name**
  - Data is typically private
  - Data is usually defined as instance variables
  - A constructor (with the same name as the class) is used to create an instance of the class
  - Getter and setter methods provide access as needed to an object's internals

## Example

### Person.java

```
public class Person {
    private String name;
    private int age;

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public String getName() {
        return this.name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return this.age;
    }

    public void setAge(int age) {
        this.age = age;
    }
}
```

# Instantiating an Object

---

- **Once your class is defined, you can use it to create objects that hold your data**
  - Creating objects is also called instantiating an object
- **You must declare a variable of your class type**
  - If declared at the class level, it defaults to `null`
  - `null` means it doesn't refer to an object
- **You then use your constructor to instantiate the object by using the `new` keyword**
  - You pass values to your constructor's parameters

## Example

```
// create the variable
Person p1;

// instantiate the Person object
p1 = new Person("Dana", 63);
```

- The values you pass to the constructor might also come from variables

## Example

```
String name = "Dana";
int age = 63;

Person p1;
p1 = new Person(name, age);
```

- **Often people declare the variable and instantiate the object in one step**

### **Example**

```
String name = "Dana";  
int age = 63;  
  
Person p1 = new Person(name, age);
```

- **You can also instantiate multiple objects**

### **Example**

```
// create the variables  
Person p1, p2, p3;  
  
// instantiate the Person object  
p1 = new Person("Dana", 63);  
p2 = new Person("Natalie", 37);  
p3 = new Person("Zachary", 31);
```

- **Once we know about arrays in Java, we can even have arrays of class objects**

### **Example**

```
Person[] family = new Person[4]; // 0, 1, 2, 3 (4 - size)  
family[0] = new Person("Dana", 63);  
family[1] = new Person("Natalie", 37);  
...
```

# Exercises – Cell Phone Service

---

In this exercise you will create an application that manages cell phones. You will create a class that is responsible for managing all information related to a cell phone. Complete your work in the `workbook-2` folder.

## EXERCISE 1

Create a new project called `CellPhoneService`. Add a package `com.pluralsight`.

**Remember:** When creating a new Java project, create a new git repository and commit your changes often! Don't forget to push to `github.com`. Always create the Github repository before creating a new Java project.

### Step 1

Add a class named `CellPhoneApplication` and add the `main()` method.

Create a class named `CellPhone`. The class should have the following data members:

- `serialNumber` (ex: 1000000 - 9999999)
- `model` (ex: iPhone X)
- `carrier` (ex: AT&T)
- `phoneNumber` (ex: 800-555-5555)
- `owner` (ex: Dana Wyatt)

Add a parameterless constructor. Provide the following default values for all string data types in the constructor.

- `serialNumber` = 0
- `model` = ""
- `carrier` = ""
- `phoneNumber` = ""
- `owner` = ""

Provide getter and setter methods for all 5 data members.

## Step 2

Verify that your class is built correctly by creating a new instance of a `CellPhone` in your `CellPhoneApp.main()` method.

Create a new instance of a `CellPhone`, then prompt the user for the cell phone information:

```
What is the serial number? 2597153
What model is the phone? iPhone 15 Pro Max
Who is the carrier? Verizon
What is the phone number? 888-555-1234
Who is the owner of the phone? Sandra
```

Use the setters of your `CellPhone` object to add the values entered by the user.

Using the getters of the `CellPhone` print the properties of the phone to the screen.

## Step 3

1. Ensure all your changes are committed and pushed to GitHub
2. Send your repository URL to your Instructor

## Section 2–2

# Working with Objects

# Working with Objects

---

- **In Java, we traditionally declare our instance variables as private**
  - This enforces the principle of encapsulation which says that we only expose data that we want from an object
  - This is why we write getter and setter methods
- **If we tried to access the private data members of the object through the object name, the compiler would generate an error**

## Example

```
Person p1 = new Person("Dana", 63);  
System.out.println("Your name is " + p1.name); // error
```

- **We must use the getter method to retrieve a data member's value**

## Example

```
Person p1 = new Person("Dana", 63);  
System.out.println("Your name is " + p1.getName());
```



# Default Access Modifier

---

- If you don't use explicitly specify an access modifier, Java assumes a default called package-private
- Package-private which means the member is visible within the same package but isn't accessible from other packages

## Example

Person.java

```
public class Person {  
    String name;  
    int age;  
  
    Person(String name, int age) {  
        this.name = name;  
        this.age = age;  
    }  
  
    String getName() {  
        return this.name;  
    }  
  
    int getAge() {  
        return this.age;  
    }  
}
```

- Java programmers will argue whether the best practice is to explicitly specify your access modifier
  - I fall in the group that prefers explicit specification

# Other Methods

---

- **Up until now we have only included member variables and methods that give us access to those variables**
  - Getters are special methods that always begin with **get** – they allow us to read the value of a variable from a class
  - Setters are special methods that always begin with **set** – they allow us to change the value of a variable in a class
- **Classes can also contain other methods**
  - It is common to create methods that perform tasks so that we don't have to repeat that logic throughout our application
  - Encapsulation means that we can group variables and methods that are closely related to each other into the same class
- **Getters/Setters vs Methods**
  - Getters and setters describe an object and store the values that the object needs to know
  - Methods are actions that the object can do, or interfaces how other code can interact with the object

# Adding Methods - Example

---

- We will add a variable named `energy` for this example

## Example

### Person.java

```
public class Person {
    private String name;
    private int age;
    private int energy;

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
        this.energy = 100;
    }

    public String getName() {
        return this.name;
    }

    public int getAge() {
        return this.age;
    }

    public int getEnergy() {
        return this.energy;
    }

    public void work(int hours) {
        this.energy -= (hours * 10);
    }

    public void sleep(int hours) {
        this.energy += (hours * 10);
    }

    public void eat() {
        this.energy += 20;
    }
}
```

- **Note that initially a person has 100% energy**
- **A persons energy can only be modified by the actions (methods) that the person performs**
  - These methods are not getters or setters, they are simply actions that affect the persons energy
  - There is no **setter** for energy, which means that we cannot just add more energy to the person
  - In order for a person to gain more energy, that person must either eat or sleep

### **Example**

#### **MainApp.java**

```
public class MainApp {  
    public static void main(String[] args) {  
        Person person = new Person("Dana", 63);  
  
        System.out.println(person.getEnergy()); // 100  
        person.work(5); // this will deplete energy  
        System.out.println(person.getEnergy()); // 50  
        person.eat(); // this will add energy  
        System.out.println(person.getEnergy()); // 70  
    }  
}
```

# Exercises – Cell Phone

---

## EXERCISE 2

Open your `CellPhoneService` application. Modify the `CellPhone` class as follows.

### Step 1

Add a `dial()` method to the `CellPhone` class that accepts a phone number parameter and displays the message "*owner's phone is calling phone-parameter*"

#### Example

```
cellPhone1.dial("855-555-2222");

//This would display: Dana Wyatt's phone is calling 855-555-2222
```

Back in the `main()` function, create a second instance of a `CellPhone` object.

In your `Main` class, add a static method called `display()` that accepts a `CellPhone` object as a parameter. (see below)

```
public static void display(CellPhone phone) {
}
```

Have it display the properties of a cell phone in a meaningful way. Then call that method twice -- passing it your two `CellPhone` objects.

```
display(cellPhone1);
display(cellPhone2);
```

Finally, write code to have your first cell phone dial your second one using the code:

```
cellPhone1.dial( cellPhone2.getPhoneNumber() );
```

Then your second cell phone dial your first one using the code:

```
cellPhone2.dial( cellPhone1.getPhoneNumber() );
```

## **Step 2**

1. Ensure all your changes are committed and pushed to GitHub
2. Send your repository URL to your Instructor

## Section 2–3

# Overloading Methods

# Overloading Methods

---

- **Java allows us to overload methods in a class**
  - Overload means we can have more than one method with the same name as long as the *signature* is different
  - The signature is determined by taking the name of a method and adding to it the type of each parameter

## Example

### Thingy.java

```
public class Thingy {  
  
    public void foo() {  
        // signature  foo  
    }  
  
    public void foo(int x) {  
        // signature  foo_int  
    }  
  
    public void foo(int x, int y) {  
        // signature  foo_int_int  
    }  
  
    public void foo(int x, String s) {  
        // signature  foo_int_String  
    }  
  
    public void foo(String s, int x) {  
        // signature  foo_String_int  
    }  
  
    public void foo(String x) {  
        // signature  foo_String  
    }  
  
}
```



- The compiler can deduce which overloaded method you wanted to call by examining the arguments

### Example

```
Thingy obj = new Thingy();  
obj.foo(7);  
obj.foo(7, "Xyz");  
obj.foo("Xyz", 7);
```

- Overloads allow us to use methods in different ways
  - Sometimes an overload offers features that another overload doesn't
  - Sometimes it just gives us a different way of passing arguments

### Example

```
int answer = myCalculator.add(x, y);
```

-- or --

```
MathOperands ops = new MathOperands(x, y);  
int answer = myCalculator.add(ops);
```

# Overloading Constructors

---

- In this workbook, you may see situations where there is more than one way to create an object
  - That is because the constructor is overloaded
  - Overloading the constructor helps provide a robust class that can be used by many people in many different situations
- Let's overload the constructor

## Example

Person.java

```
public class Person {
    private String name;
    private int age;

    public Person() {

    }

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public String getName() {
        return this.name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return this.age;
    }

    public void setAge(int age) {
        this.age = age;
    }
}
```

- Now that we have two constructors, let's look at two different ways to create an object

### Example

```
// This uses the parameterized constructor
Person p1 = new Person("Dana", 63);

// This uses the constructor without parameters and
// then places data in the object using the setter methods
Person p2 = new Person();
p2.setName("Natalie");
p2.setAge(37);

System.out.println("P1's name is " + p1.getName()); System.out.println("P2's name is " + p2.getName());
```

# Example: Working with a Class

---

## Example

### Person.java

```
public class Person {
    private String name;
    private int age;

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public String getName() {
        return this.name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return this.age;
    }

    public void setAge(int age) {
        this.age = age;
    }
}
```

### MainApp.java

```
public class MainApp {
    public static void main(String[] args) {
        // create a Person object
        Person person = new Person("Dana", 63);

        // display the person's information
        System.out.printf("%s is %d years old\n",
                           person.getName(),
                           person.getAge());
    }
}
```

# Exercises – Cell Phone Pt. 2

---

## **EXERCISE 3**

Modify the `CellPhone` class from the previous exercise.

### **Step 1**

Add an overloaded constructors to the class. This one should take 5 input parameters, 1 for each variable.

Back in the `main()` function, create a new `CellPhone` object. Instantiate the new cell phone by calling the overloaded constructor that has 5 parameters. You will need to call setters to give the data members values.

Call the `display()` method of the new cell phone to verify that all of your data members display correctly. Also try dialing the new phone from one of the other phones.

### **(Optional)**

In your `CellPhone` class, add an overloaded version of the `dial` method. The second method should accept a `CellPhone` object as an input parameter. This overloaded method should get the phone number of the input parameter.

Your class should have 2 methods that look similar to this.

```
public void dial(String phoneNumber){ ... }  
public void dial(CellPhone phone) { ... }
```

### **Step 2**

1. Ensure all your changes are committed and pushed to GitHub

# **Module 3**

## **Loops and Arrays**

## Section 3–1

### Loops

# Types of Loops

---

- **Java supports several loop statements, including:**
  - `while`
  - `do / while`
  - `for`
- **You can exit any of these loops early using a `break` statement**
- **Java also supports a `continue` statement that skips the rest of the code in a specific iteration, but continues the loop**



# while Loop

---

- The **while** loop is used to execute a block of code repeatedly as long as a condition is true

## Syntax

```
while (condition) {  
    // code  
}
```

## Example

```
int i = 1;  
int sum = 0;  
  
while (i <= 10) {  
    sum += i;  
    i++;  
}  
  
System.out.println("Sum = " + sum);
```

- Often, the while loop is used to loop through a stream of data when the length of the stream is unknown
  - Streams might include data from a file or database

# do/while Loop

---

- The **do/while** loop is similar to **while** loop in that it executes a block of code while a condition is true
  - However, the **do/while** checks the condition at the bottom of the loop so it is guaranteed to execute at least once

## Syntax

```
do {  
    // code  
} while (condition);
```

## Example

```
int i = 1;  
int sum = 0;  
  
do {  
    sum += i;  
    i++;  
} while (i <= 10);  
  
System.out.println("Sum = " + sum);
```

# for Loop

---

- Like other C-based languages, the **for** loop is used to execute a loop a fixed number of times
  - The initialization is executed at the beginning of the loop.
  - The condition defines is evaluated during each iteration to determine whether the loop keeps executing
  - The other part is executed during each iteration after the code block executes
    - \* It is typically used to increment or decrement a loop counter

## Syntax

```
for (initialization; condition; other) {  
    // code  
}
```

## Example

```
for (int i = 0; i < 10; i++) {  
    System.out.println(i);  
}
```

- It is used frequently with arrays and we will learn about those soon

# break VS continue

---

- The **break** statement is used to exit a loop early

## Example

```
int sum = 0;
for(int i = 1; i <= 10; i++) {
    if (i % 3 == 0) {
        break;
    }
    sum += i;
}
System.out.println("Sum = " + sum);
```

## OUTPUT

Sum = 3

- More often than not, **break** is used to exits a search loop early once you find what you are looking for!
- The **continue** statement is used to exit a loop early

## Example

```
int sum = 0;
for(int i = 1; i <= 10; i++) {
    if (i % 3 == 0) {
        continue;
    }
    sum += i;
}
System.out.println("Sum = " + sum);
```

## OUTPUT

Sum = 37

- **This instructor doesn't like the `continue` statement**
  - Instead, I would write the `if` to be based on what I wanted to ***include*** rather than what I wanted to ***skip***

### **Example**

```
int sum = 0;
for(int i = 1; i <= 10; i++) {
    if (i % 3 != 0) {           // no need for continue
        sum += i;
    }
}
System.out.println("Sum = " + sum);
```

### **OUTPUT**

Sum = 37

# Exercises - Looping

---

In these exercises you will practice the various loops that are available in Java. I.e. `while`, `do/while`, and `for` loops. Remember to complete this work in your `workbook-2` folder.

## EXERCISE 1

Create a Java application called `LoopingExercises`.

**Remember:** When creating a new Java project, create a new git repository and commit your changes often! Don't forget to push to `github.com`. Always create the Github repository before creating a new Java project.

### Step 1

Create a class called `WhileLoop`. Add a `main()` method and create a `while` loop to display the text "I love Java" 5 times.

Create another class named `DoWhileLoop`. Add a new `main()` method and print "I love Java" again, 5 times, but this use a `do/while` loop.

### Step 2

Add another class to your application. Name it `ForLoop` and add a `main()` method. Use a `for` loop to display a countdown from 10 to 1 and then ends by displaying the word "Launch!"

**HINT:** Inside the loop add `Thread.sleep(1000) ;` to pause 1 second between each number to slow the countdown. You will also need to update the `main` method definition as follows:

```
public static void main(String[] args) throws InterruptedException
```

### Step 3

1. Ensure all your changes are committed and pushed to **GitHub**
2. Send your repository URL to your Instructor

## EXERCISE 2

Create a new Java application called RollTheDice.

### Step 1

You will program the application will roll a pair of dice 100 times. Display the value of the two dice each time they are rolled. Count how many times the numbers 2, 4, 6 and 7 are rolled and display the result.

To make this work, you need to create a class named `Dice` and add a method named `roll()` that generates a random number between 1 and 6. You can generate a random number within a range using the following algorithm:

```
int randomNumber = (int)(Math.random() * maxValue) + minValue;
```

Now switch back to your `main()` function:

1. Create an instance of `Dice` named `dice`. Also create integer variables for `roll1`, `roll2`, and four different counters for the number of times 2, 4, 6 and 7 are rolled
2. Create a loop that executes 100 times.
3. Within the loop, call your `dice`'s `roll()` method two times:

```
roll1 = dice.roll();  
roll2 = dice.roll();
```

4. Print the value of each roll of the dice formatted like this:

```
Roll 1:    4    -    6    Sum:   10
```

5. Determine if the sum of `roll1` and `roll2` is 2, and if so increment the `twoCounter`.
6. Determine if the sum of `roll1` and `roll2` is 4, and if so increment the `fourCounter`.
7. Determine if the sum of `roll1` and `roll2` is 6, and if so increment the `sixCounter`.
8. Determine if the sum of `roll1` and `roll2` is 7, and if so increment the `sevenCounter`.
9. When the loop terminates, display your counters!



Now, use this knowledge to go play Craps at your local casino! ☺

## **Step 2**

1. Ensure all your changes are committed and pushed to GitHub
2. Send your repository URL to your Instructor

## Section 3–2

# Arrays

# Arrays

---

- **Arrays are lists of elements**
- **Basic Java arrays are declared by specifying the data type of the array data followed by square brackets**
  - Values are accessed with subscripts surrounded by square brackets ( [ ] ) and can determine the length of an array using `.length`
    - \* Subscripts are 0-based
- **The size of the array, once set, *cannot* change!**
  - You implicitly set the size when specifying initial values
- **Use the final (read-only) Array variable `length` to get the length of an array**

## Example

```
int[] nums = {63, 65, 60, 53, 58, 37, 35, 31};  
for (int i = 0; i < nums.length; i++) {  
    System.out.print(nums[i] + " ");  
}  
System.out.println();
```

## OUTPUT

63 65 60 53 58 37 35 31

- You can also create an array with a fixed number of elements without explicitly setting its values

\* These array elements have default values appropriate for the type

### **Example**

```
int[] nums = new int[5];  
for (int i = 0; i < nums.length; i++) {  
    System.out.print(nums[i] + " ");  
}  
System.out.println();
```

#### **OUTPUT**

0 0 0 0 0

### **Example**

```
String[] words = new String[5];  
for (int i = 0; i < words.length; i++) {  
    System.out.print(words[i] + " ");  
}  
System.out.println();
```

#### **OUTPUT**

null null null null null

# for-each Loop

---

- Java has a for-each statement that is used to loop through the elements in an array

## Syntax

```
for (type elementname : arrayName) {  
    // code  
}
```

## Example

```
String[] schools = {"Calliet", "Cary", "Skyline",  
                   "SFASU", "TAMU"};  
  
for (String school : schools) {  
    System.out.println(school);  
}
```

## OUTPUT

```
Calliet  
Cary  
Skyline  
SFASU  
TAMU
```

- When you use the **for** statement to loop through an array, you have access to the index and can use that to access the element of an array
- When you use the **for-each** statement to loop through an array, you have the element in the array
  - You don't have access to its index

## Passing an Array to a Method

---

- In addition to data types like `int`, `double` and `String`, methods can have arrays as an element
  - NOTE: You don't specify the size of the array in the parameter declaration

### Example

```
public class Program
{
    public static void main(String args[])
    {
        int[] nums = {63, 65, 60, 53, 58, 37, 35, 31};
        displayNumbers(nums);
    }

    public static void displayNumbers(int[] nums)
    {
        for (int i = 0; i < nums.length; i++) {
            System.out.print(nums[i]+ " ");
        }
        System.out.println();
    }
}
```

# Returning an Array from a Method

---

- In addition to returning values like `int`, `double` and `String`, methods can return arrays
  - NOTE: You don't specify the size of the array in the method declaration

## Example

```
import java.util.Arrays;

public class Program
{
    public static void main(String args[])
    {
        int[] nums = getNumbers();

        for (int i = 0; i < nums.length; i++) {
            System.out.print(nums[i]+ " ");
        }
        System.out.println();
    }

    public static int[] getNumbers()
    {
        int[] nums = {63, 65, 60, 53, 58, 37, 35, 31};
        return nums;
    }
}
```

# Sorting an Array

---

- There are several ways to sort a list in Java but we will focus on this one for now
- The `Arrays.sort()` method can be passed an array and will sort it based upon its natural ordering of elements
  - This is good when your array contains strings or numbers

## Example

```
import java.util.Arrays;

public class Program
{
    public static void main(String[] args)
    {
        String[] nameList = {
            "Natalie", "Brittany", "Zachary", "Ezra", "Ian",
            "Siddalee", "Elisha", "Pursalane", "Zephaniah", "Alice"
        };

        Arrays.sort(nameList);

        for(String name : nameList) {
            System.out.println(name);
        }
    }
}
```

## OUTPUT

```
Alice
Brittany
Elisha
Ezra
... not all names shown ...
Zachary
Zephaniah
```



# Copying an Array

---

- If you assign one array to another, it makes both variables reference the same underlying array

## Example

```
String[] colors = {"red", "white", "blue"};
String[] copy = new String[3];

copy = colors; // they **reference** the same underlying array
```

- If you want to copy the elements of one array to another, either use a **for** loop and move them one at a time or use the **arrayCopy()** method
  - You specify the index to start copying at in each array and how many to copy

## Example

```
class ArrayCopyDemo {
    public static void main(String[] args) {
        String[] colors = {"red", "white", "blue"};
        String[] colorCopy = new String[3];

        // copy colors to colorCopy 1 at a time
        for (int i = 0; i < 3; i++) {
            colorCopy[i] = colors[i];
        }

        for (int i = 0; i < 3; i++) {
            System.out.println(colorCopy[i]);
        }
    }
}
```

## Example

```
class ArrayCopyDemo {  
    public static void main(String[] args) {  
        String[] colors = {"red", "white", "blue"};  
        String[] colorCopy = new String[3];  
  
        // copy from colors at subscript 0 to  
        // colorCopy at 0 -- move 3 elements  
        System.arraycopy(colors, 0, colorCopy, 0, 3);  
  
        for (int i = 0; i < 3; i++) {  
            System.out.println(colorCopy[i]);  
        }  
    }  
}
```

## Example

```
class ArrayCopyDemo {  
    public static void main(String[] args) {  
        String[] colors = {"red", "white", "blue"};  
        String[] colorCopy = new String[3];  
  
        // copy from colors at subscript 1 to  
        // colorCopy at 0 -- move 2 elements  
        System.arraycopy(colors, 1, colorCopy, 0, 2);  
  
        for (int i = 0; i < 2; i++) {  
            System.out.println(colorCopy[i]);  
        }  
    }  
}
```

## OUTPUT

white  
blue

## Other Array Methods

---

- **There are many other array methods that are interesting, including:**
  - `fill()` - Fills an array with a specific value
  - `equals()` - Compares two arrays to determine if they are equal
  - `binarySearch()` - Efficiently searches an ***ordered*** array for a specific value to find out where it is located

# Exercises – Test Statistics

---

**Remember:** the code for these exercises should be completed in the `workbook-2` folder.

When creating a new Java project, create a new git repository and commit your changes often! Don't forget to push to github.com. Always create the GitHub repository before creating a new Java project.

## EXERCISE 3

### Step 1

Create a Java application named `TestStatistics`. Create an array of 10 test scores. Print out the following statistics of your test scores:

- average
- high score
- low score

**BONUS:** Calculate and display the `median` value (what is the difference between the average and the median).

### Step 2

1. Ensure all your changes are committed and pushed to GitHub
2. Send your repository URL to your Instructor

## EXERCISE 4

### Step 1

Create a Java application called `VehicleInventory`. This application is intended to manage the inventory for a used car dealership. Users will use a menu to lookup vehicles or add vehicles to the list.

Create a class named `Vehicle`. Add the following data members to it:

```
vehicleId      - a long (ex: 101121)
makeModel      - a string (ex: Ford Explorer)
color          - a string (ex: Red)
odometerReading - an int (ex: 32775)
price          - a float (ex: 12250.00) (no Lamborghinis here!)
```

Add a constructor and get/set methods for each property of the car.

Back in `main()`, create an array capable of holding up to 20 vehicles and a counter variable that tells you how many vehicles are in the array right now.

Preload the array with 6 vehicles:

```
vehicleId,makeModel,color,odometerReading,price
101121,Ford Explorer,Red,45000,13500
101122,Toyota Camry,Blue,60000,11000
101123,Chevrolet Malibu,Black,50000,9700
101124,Honda Civic,White,70000,7500
101125,Subaru Outback,Green,55000,14500
101126,Jeep Wrangler,Yellow,30000,16000
```

Create a loop and prompt the user for a command within the loop. The code the user sees should be:

```
What do you want to do?
1 - List all vehicles
2 - Search by make/model
3 - Search by price range
4 - Search by color
5 - Add a vehicle
6 - Quit

Enter your command
```

You may not get all options in the command list done.

Begin with options 1, 2 and 5. Once you've completed those options, add logic to allow the user to search for a vehicle by price or color.

Use methods wisely. It seems like your loop could match a command to a number and then call a method to do the processing. For example:

```
System.out.println("What do you want to do?");
System.out.println(" 1 - List all vehicles");
System.out.println(" 2 - Search by make/model");
System.out.println(" 3 - Search by price range");
System.out.println(" 4 - Search by color");
System.out.println(" 5 - Add a vehicle");
System.out.println(" 6 - Quit");
System.out.println("Enter your command: ");

int command = scanner.nextInt();

switch(command) {
    case 1:
        listAllVehicles();
        break;
    case 2:
        findVehiclesByPrice();
        break;
    case 5:
        addAVehicle();
        break;
    case 6:
        return;
}
```

## **Step 2**

1. Ensure all your changes are committed and pushed to GitHub
2. Send your repository URL to your Instructor

## Self-Check

---

- **You can check your understanding of Java thus far at:**
  - `www.w3schools.com/java/java_exercises.asp`
  - `www.w3schools.com/java/java_quiz.asp`
- **Clearly we haven't completed our study of Java yet, but you may be surprised at how many you get correct!**

# **Module 4**

## **Continued Learning**



## Section 4–1

# Practice Repository

# Practice Problems

---

- We have a repository with additional practice problems and projects
- Your Instructor will inform you which exercises and projects you will work on within this repository.
- If the files contain -optional, you are more than welcome to work on these as you please.
- <https://github.com/Pluralsight-ILT/YUU-Java-Academy-Bonus>

## Section 4–2

### Code Wars (Optional)



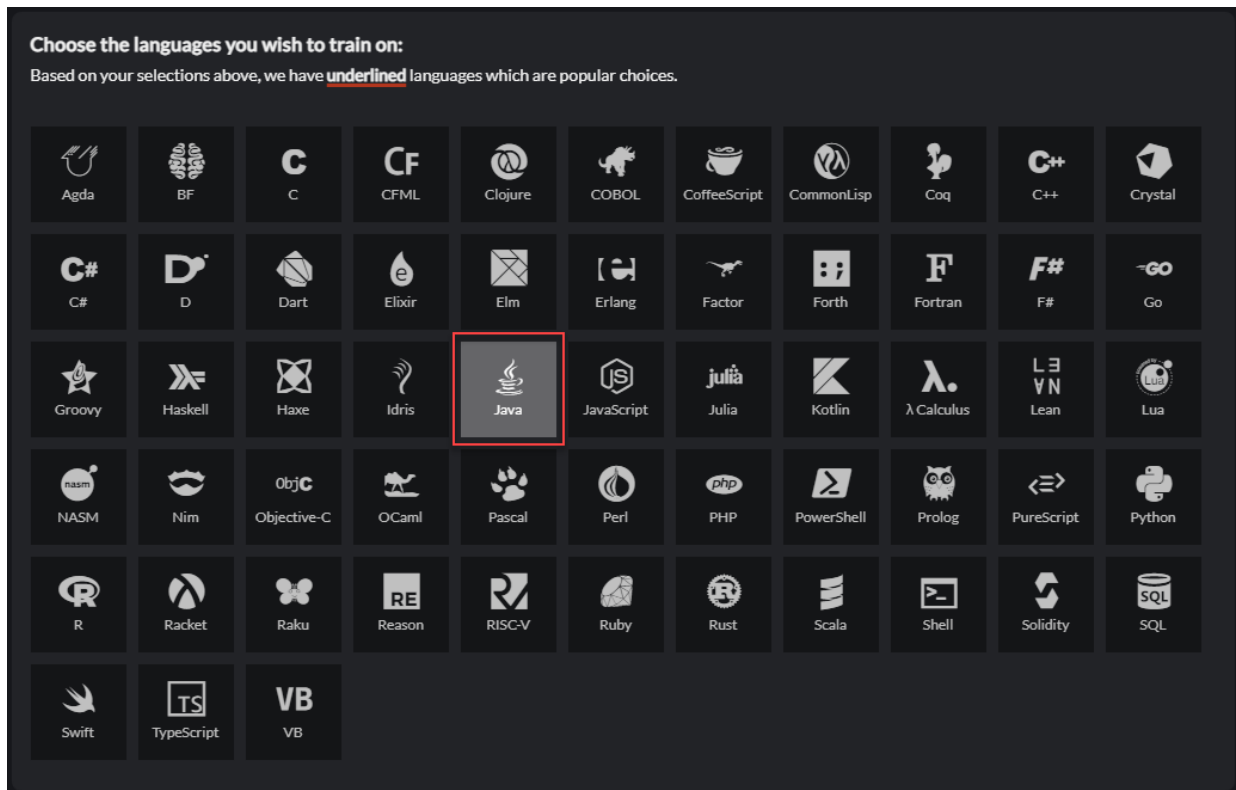
# What is CodeWars

---

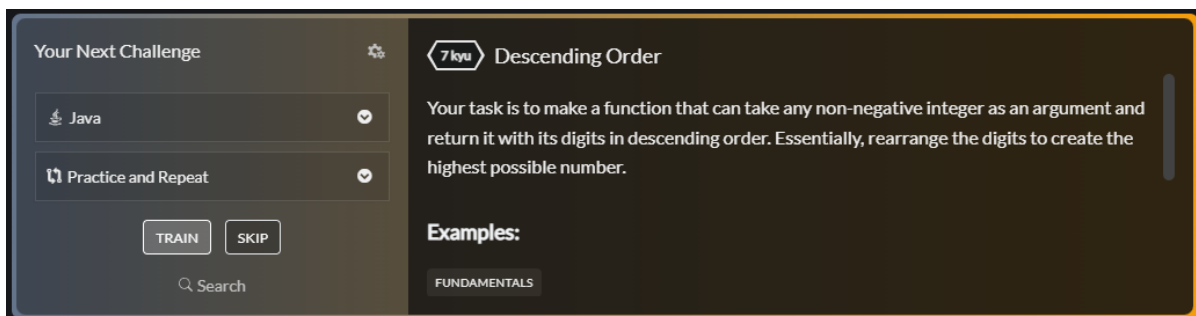
- **CodeWars is a website that is designed for developers to practice solving coding challenges**
  - Register at `www.codewars.com`
- **Katas**
  - Kata is taken from the world of Karate – it is a system of individual training exercises to perfect a skill
  - Each challenge in CodeWars is a Kata
- **Kyu Ranking System**
  - Kyus are ranking systems in Karate (and in Code Wars)
  - When you register you begin at level 8
  - The goal is to advance your skills and achieve lower levels
  - Each Kata has a kyu rating from 8 – 1
  - The more difficult the rating the more points you earn by solving it
- **We will assign CodeWars Katas to you throughout this academy**
  - You are welcome to do as many as you wish, but we will assign certain challenges that match the content you are learning

# CodeWars Setup

- When you register you select the programming language(s) that you want to practice
  - Choose Java and any other languages you would like



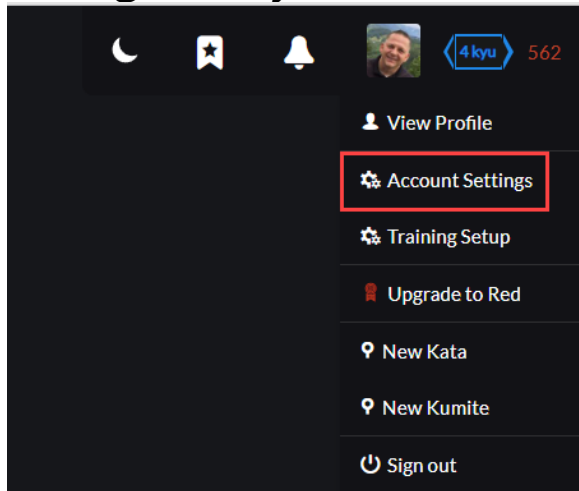
- On your home page you will be prompted to train on the next Kata/challenge at your current level



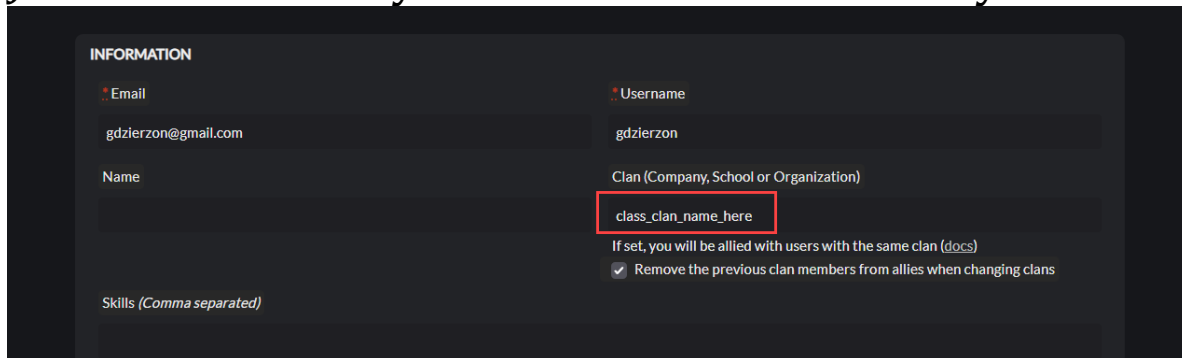
# Join a Clan

---

- CodeWars allows you to join a clan so that you can compete against other clan members
- Navigate to your account settings



- Anyone with a CodeWars account can create a clan
- Join the clan that your instructor created for your class











- Let your instructor know your username

- Navigate to the home page to view the leaderboard of all allies in your clan

**Allies**

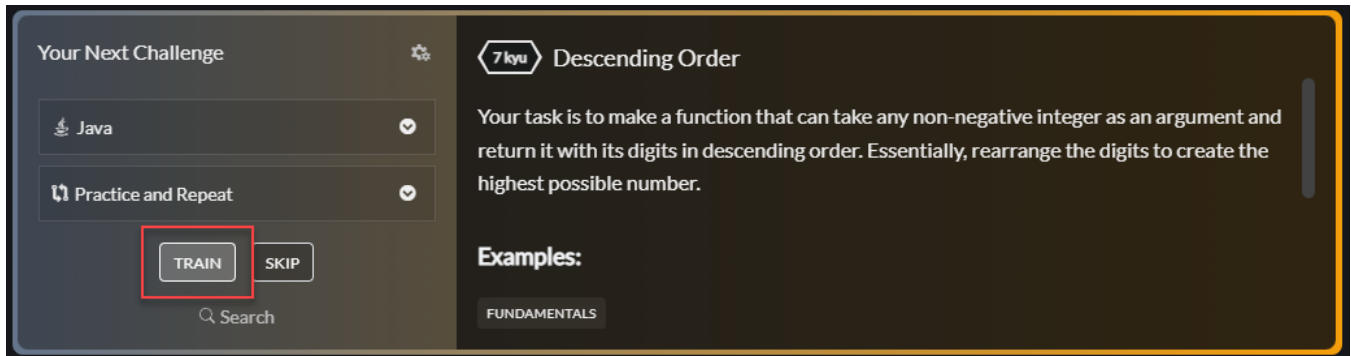
You are automatically given an allegiance with anyone who is in the same clan as you. You can also become allies with other warriors by following each other or inviting new warriors to join.

Position	User	Clan	Honor
1	  gdzierzon		562
2			450
3			226
4			16
5			10
6			3
7			3

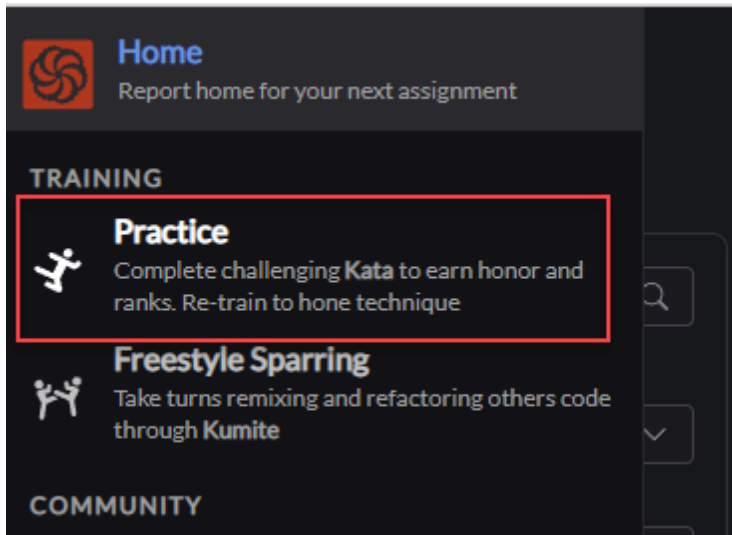


# CodeWars Example

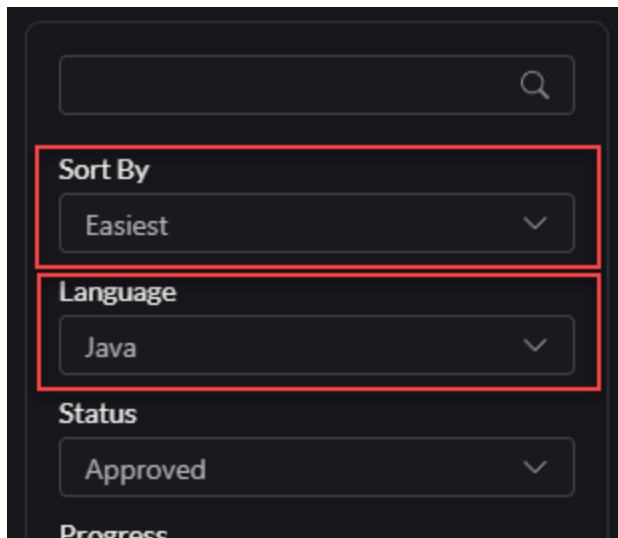
- You can read the description of the Kata and choose to "train"



- You can also search for other Katas by hovering over the left navigation and clicking **Practice**

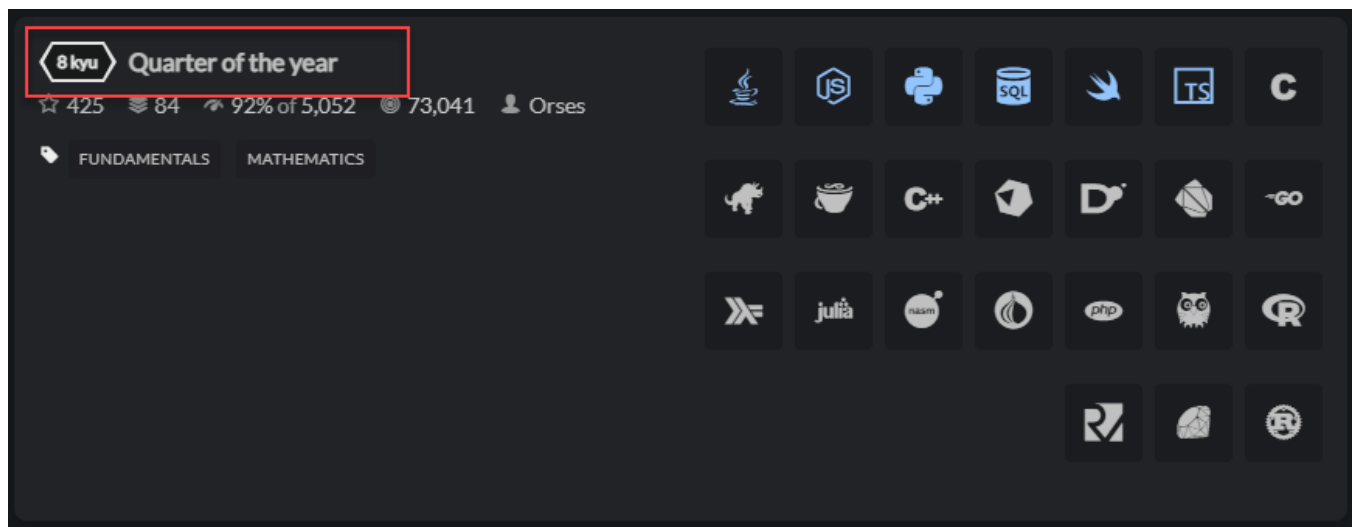


- This will take you to the practice page where you can sort and filter based on your preferences



- Find the Kata that you want to practice and click on the link

– <https://www.codewars.com/kata/5ce9c1000bab0b001134f5af>



- Read the description/instructions for the Kata

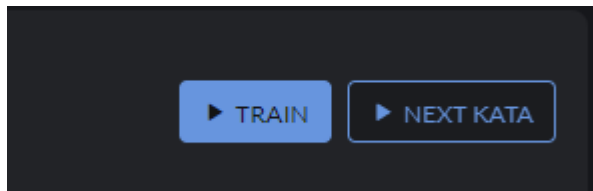
Given a month as an integer from 1 to 12, return to which quarter of the year it belongs as an integer number.

For example: month 2 (February), is part of the first quarter; month 6 (June), is part of the second quarter; and month 11 (November), is part of the fourth quarter.

Constraint:

- `1 <= month <= 12`

- After reading the instructions, click the "Train" button



- Determine how you would solve the Kata and write your code in the code editor window
  - You can try to solve it on your computer first in IntelliJ
  - There are MANY solutions to the same problems, so don't feel like you have to have the "perfect" solution

- Use a notepad or whiteboard to come up with a solution

### **Example**

Month	Quarter
-----	-----
1,2,3	1
4,5,6	2
7,8,9	3
10,11,12	4

### **Possible Option**

```
public class Kata {
    public static int quarterOf(int month){

        if(month == 1 || month == 2 || month == 3){
            return 1;
        } else if(month == 4 || month == 5 || month == 6){
            return 2;
        } else if(month == 7 || month == 8 || month == 9){
            return 3;
        } else {
            Return 4;
        }
    }
}
```

### **Another Possible Option**

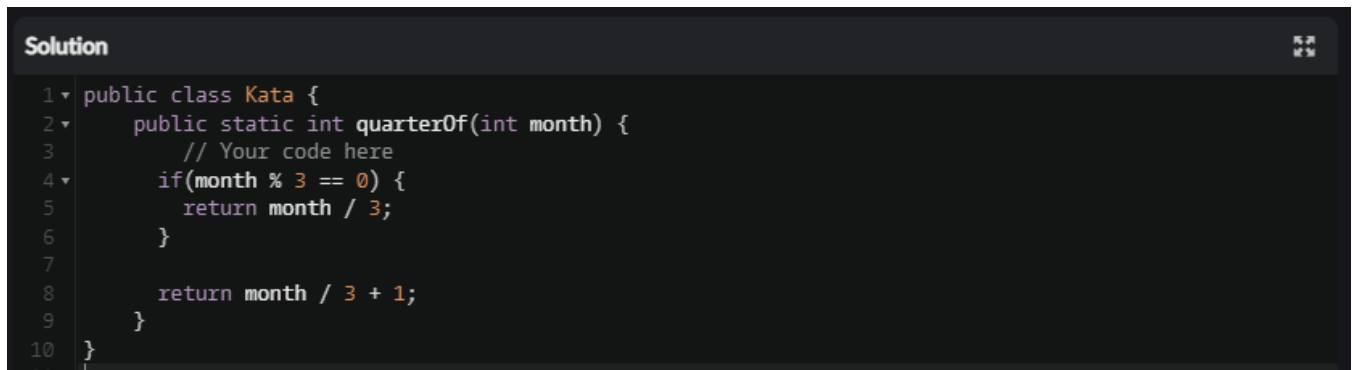
```
public class Kata {
    public static int quarterOf(int month){

        if(month % 3 == 0){
            return month / 3;
        }

        return month / 3 + 1;

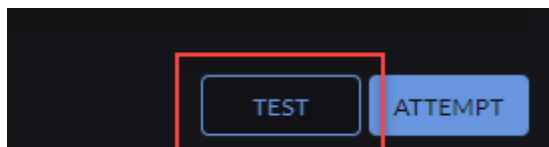
    }
}
```

- Enter your code into the CodeWars code editor

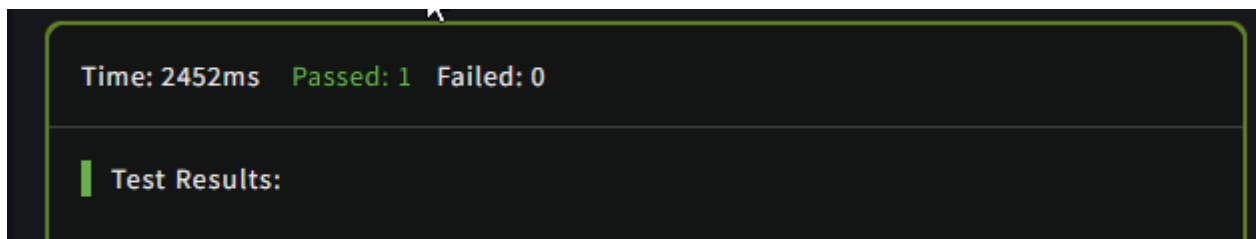


```
Solution
1 public class Kata {
2     public static int quarterOf(int month) {
3         // Your code here
4         if(month % 3 == 0) {
5             return month / 3;
6         }
7
8         return month / 3 + 1;
9     }
10 }
```

- Click the "Test" button at the bottom of the page

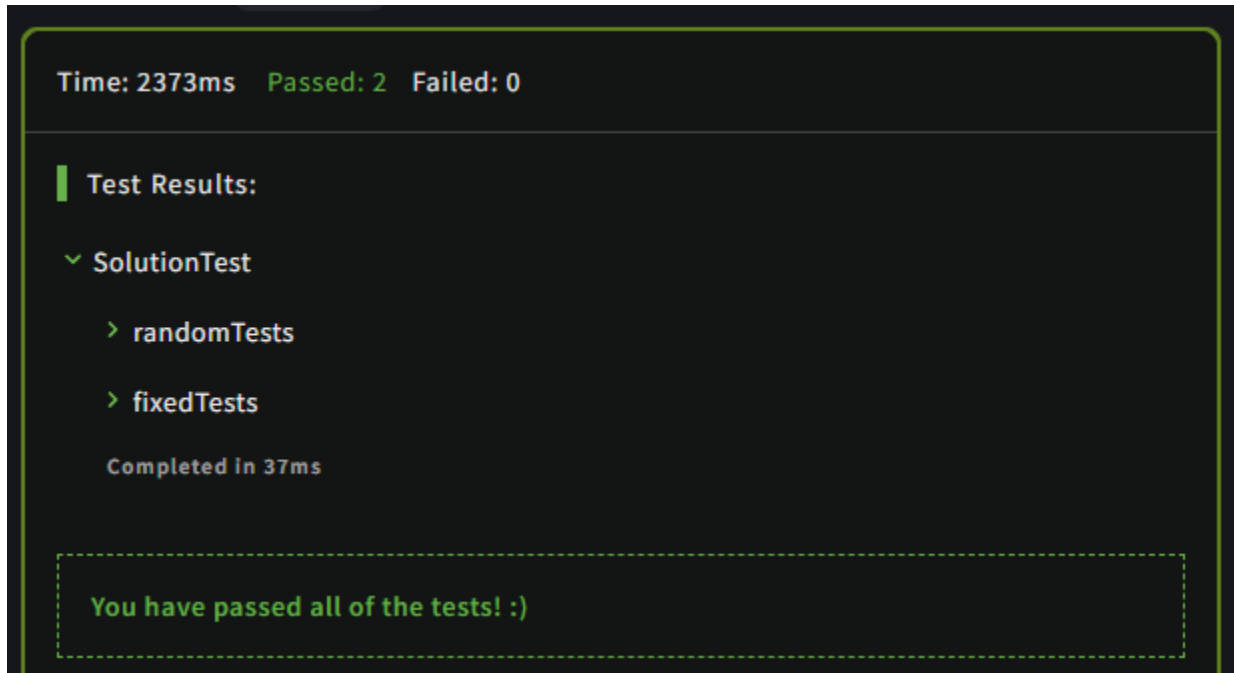


- This will run the sample test(s)



- If all of the sample tests pass, click the "Attempt" button to run additional tests and submit your answer
  - If your code does not pass the additional tests, you need to modify your code and submit again

- Your test results will then display the results of all tests that were run



- Now you can move on to your next Kata

# CodeWars Kata (Optional)

---

- **Basic Mathematical Operations**

- The Kata accepts 3 input parameters
  - \* The operation ( + - / \*)
  - \* 2 numbers
- Perform the specified arithmetic operation and return the answer

- **Complete this Kata for additional Java practice**

- <https://www.codewars.com/kata/57356c55867b9b7a60000bd7/java>

# CodeWars Kata (Optional)

---

- Here are 2 Katas that are similar

- Counting Sheep

- <https://www.codewars.com/kata/54edbc7200b811e956000556/java>

- If You Can't Sleep, Just Count Sheep

- <https://www.codewars.com/kata/5b077ebdaf15be5c7f000077/java>



# CodeWars Kata (Optional)

---

- **The Feast of Many Beasts**

- Beasts are coming for dinner, and they are each bringing a dish
- But there is a rule about what dish each beast can bring
- See if you can solve it

- **Complete this Kata for additional Java practice**

- <https://www.codewars.com/kata/5aa736a455f906981800360d/java>