Introduction to Computer Programming

#What You Should Know About Computer Programming

Computers have been around for a long time. You've probably heard stories about warehouses full of machinery that crunched the numbers to help send human beings to the moon, but there's more to the history of computers than the actual machines that got us to where we are today. It's interesting stuff, and the internet abounds with articles and documentaries tracing that storied history.

What's more important for you, however, is how humans interface with that technology: computer programming. Computer programming is the process of giving machines a set of instructions that describe what the computer should do. Everything that a computer does has to be defined by human beings, whether it's adding two numbers or transmitting your bank account information over the internet.

At its most basic, computers are run by a series of ones and zeroes–a programming language called binary–but practically nobody writes code in binary these days. We don't have to. Successive generations of developers have written layers of code on top of those binary basics and built computer programming into something that humans can more easily read and understand. They've layered technologies on top of each other, each of which gets compiled (interpreted) into the more complex code that the computer needs. This continues from the code we write, in Java or Javascript for example, all the way down to the basics of ones and zeroes. In the modern programming world, you barely need to know anything about how your computer does what it does, yet you can make complex programs that the founding fathers of computer science could never have dreamed of.

#Computer Programming In the Modern World

Computer programming is a fundamental skill for many different applications in the modern world. While it may seem like it's just for cutting-edge research in artificial intelligence, it's actually what underpins the very nature of modern society. Computer programming makes banking more accessible, smoothes out our supply lines, and creates the fantastic online experiences we access every day. Programming means your favourite jeans are one click away, and that governments can open services faster, more efficiently, and more equitably during crises.

Whether you're consciously aware of it or not, every aspect of your life involves coding, from your fitness tracker on your smartphone to your tax returns. As a result, programming skills can help you in your job search even if you aren’t interested in technology-specific fields and can build employer-desired 21st-century skill sets like problem-solving skills and critical thinking. Most of us engage with programming in nearly every part of the modern world, so a basic understanding of its principles will always be an essential skill.

A basic understanding of **programming languages**, **algorithms**, and **models** makes our lives easier. At Academy, you'll learn about all of these skills while getting the hands-on time you need to create the programs that make the world work.

#Modern Applications of Computer Programming

It’s challenging to think of a way computer programming doesn't prop up the modern world. Nearly everyone in the world is either connected or readily seeking ways to increase connections to everyone else. However, programming is used for more than basic consumer applications. Here are just a few paths you can take as a modern developer, outside of simply making mass-market products:

1. **Research relies on data**: Machines can help researchers aggregate, analyze, synthesize, and visualize data in ways human beings have not been capable of before. Programming skills allow people to build the vehicles that connect machines and humans.
2. **Government operations:** Coders are responsible for large portions of the government’s digital transformation. New online portals allow citizens, organizations, and businesses to access government services more efficiently.
3. **Web development and design:** Designers leverage computer programming skills to build online experiences in fields like eCommerce or entertainment. These sites provide access to the information and services held within the digital world and rely on user research to create experiences.
4. **Marketing and business operations:** Computer programmers can also help businesses manage operations by building apps and tools for internal use or customer-facing solutions. Data science and artificial intelligence: These exploding fields require software engineers to build environments suitable for processing and visualizing the data necessary to train models for data science and machine learning projects.
5. **Cybersecurity:** Developers also build the solutions we need to keep our online interactions safe from threat actors. Mastery of different programming languages can help students launch their careers in this in-demand information technology sector.

#Modern Programming Languages

We will learn more about programming languages–how they work, why they do what they do, and why they were invented–as we uncover them throughout the course. For now, understand that different languages serve different purposes in the modern world, and that they come with their own difficulty curves, varying levels of employer demand, and salaries.

If you'd like to know more about in-demand coding languages, feel free to check out the Extra Resources section at the bottom of this page. These Extra Resources sections will be a common occurrence throughout the bootcamp. If you have time to look through them, there's a good chance that they'll help you with the deeper topics and keep you on track as your learning progresses.

#Extra Resources

Here is an article about modern programming languages, demand for them, and what developers are using them for.

And here are two videos explaining some of the broad strokes of how computer programming and applications work:

https://youtu.be/FCMxA3m_Imc

https://youtu.be/3gMOYZoMtEs

The Java Programming Language

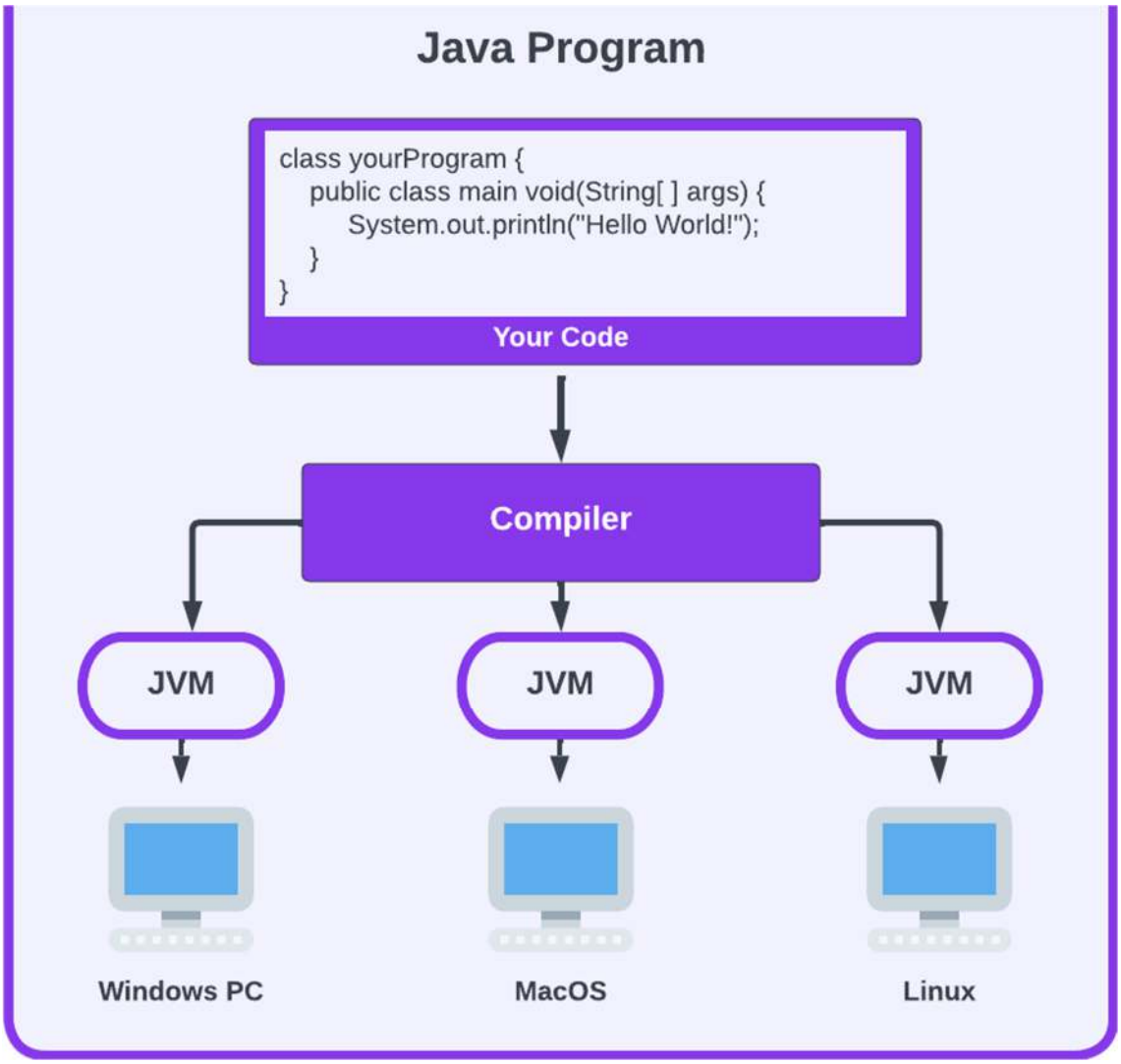
Early in the history of computer programming, resources were incredibly tight. Computers weren't capable of keeping track of much information at all, and it became necessary to efficiently use every bit of programming power that computers had. At the same time, we needed to make that power accessible for humans who write the programs that make it all work.

In 1995, Sun Microsystems released the Java programming language to tackle these issues. Since then, Java has been known for being simple, portable, secure, and robust. Though it was released over twenty years ago, Java remains one of the most popular programming languages today. Due to its innovative use of a built-in **compiler**, Java code can be run on different operating systems and platforms. Indeed, Sun Microsystems' slogan for Java was, and largely still is, "write once, run everywhere".

As stated earlier in the course, more than 10,000 companies use Java somewhere in their stack today, and that's why we're going to start by learning about it first.

#Welcome to the world of Java Programming!

As a developer, you will write Java programs. The code that you write will then be passed into a compiler, which will hand off that code to what's known as a Java Virtual Machine (JVM). The JVM's job is to act like a universal translator, making your code available to any **operating system** that tries to run it, such as Windows, MacOS, and Linux:



Java programs are passed into a compiler, which then uses a JVM to get that code running on modern operating systems.

This **compilation** is why Java is still so useful today. When programs are written to run on Windows, they generally won't be able to run MacOS or Linux. This is true of many high-level languages–the ones that we developers write–because they're usually designed for a specific operating system. When we run a program, the OS does the work of translating that higher-level language down into a lower-level language, which eventually leads to the ones and zeroes that the hardware needs to do its job. With Java, we can write our code wherever we need to because it will be compiled for us. This makes Java a "**Platform Independent Language**".

#What is the Java Virtual Machine?

The JVM (Java Virtual Machine) is an abstract machine that enables your computer to run a Java program. The computer you're using to view this page is a **physical machine** running software that has produced this page. You can think of **virtual machines** as being implementations of software designed to run like hardware, but in a way that allows one set of hardware to run many different machines for many different purposes.

When you run a Java program, the Java compiler first compiles your Java code into an intermediate step called bytecode. Bytecode is essentially a quick stopover–one that doesn't depend upon the OS, and can easily be translated out to whatever language it needs to be. From here, the JVM translates that bytecode into native **machine code**, a specific set of instructions that a computer's CPU can execute directly.

Java is a platform-independent language because when you write Java code, it's ultimately written for the JVM, not for your physical machine. Since the JVM executes this platform-independent Java bytecode, anything you write in Java can be run on any system that wants to run it.

How a Java Program works: from developer to user

#Zooming Out from the JVM

The JVM is just one piece of a larger puzzle when it comes to Java delivery pipeline. Let's keep zooming out for a moment and see what else is required to get a program up and running on a user's computer.

#The Java Runtime Environment

The Java Runtime Environment (JRE) is a software package that provides a bunch of useful tools that we'll be using to get our Java programs running. This includes Java

class libraries, the aforementioned Java Virtual Machine (JVM), and other components that are required to run Java applications.

You can think of the JRE as a container that holds the JVM.

The JRE is a runtime environment that contains our JVM and some helpful class libraries.

If you only need to **run** Java programs, but you don't need to develop them, the JRE is all you need.

#Java Development Kit

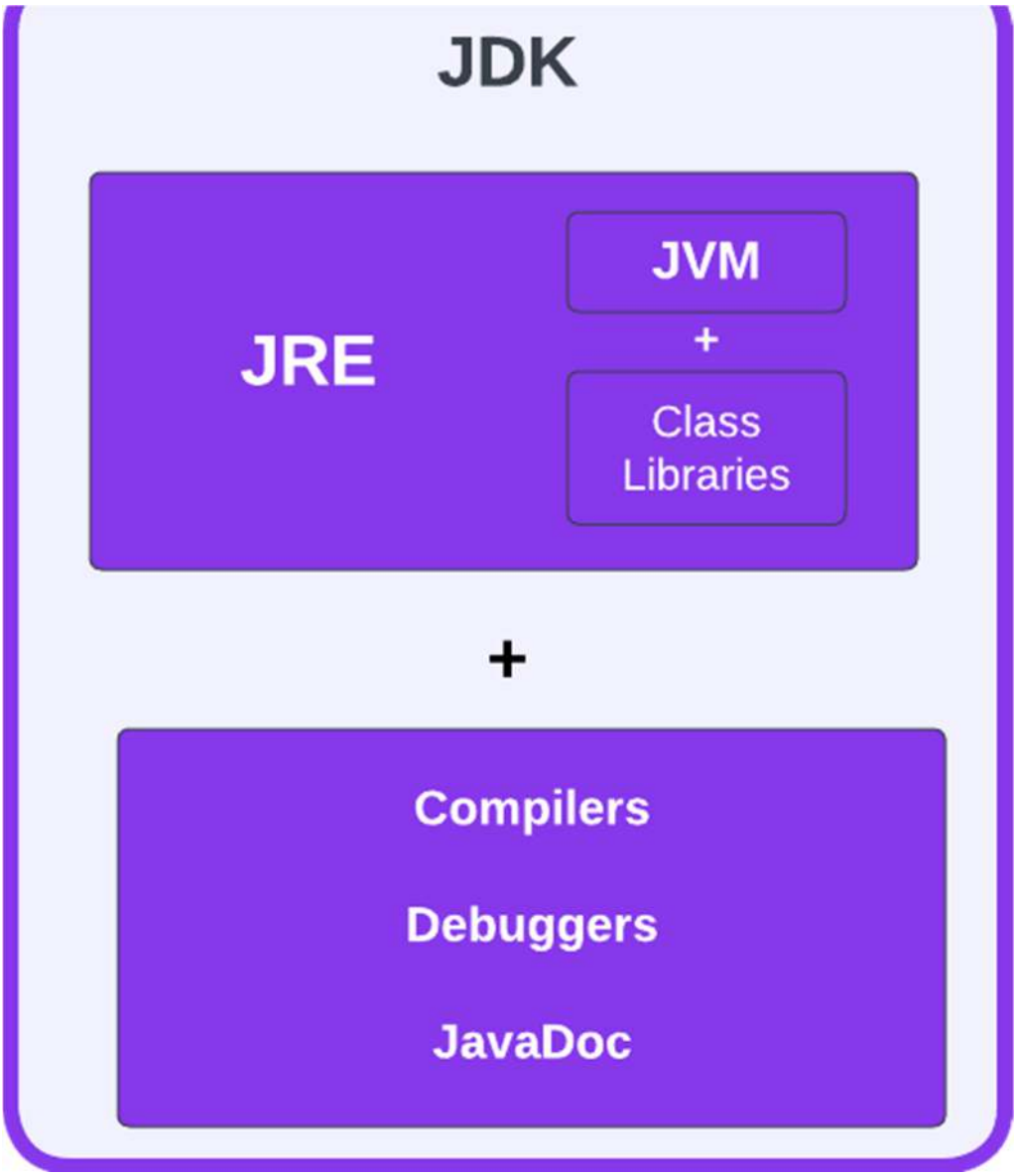
The Java Development Kit (JDK) is a software development kit required to develop applications in Java. When you download a JDK, the JRE described above will be included in that installation.

In addition to the JRE, the JDK also contains a number of development tools, like compilers, JavaDoc for documentation purposes, a Java Debugger, and a few other tools:

The JDK contains our JRE, plus some helpful tools for making sure that our application runs as planned.

#When You Put It All Together:

All together, this is what the JDK ends up looking like:



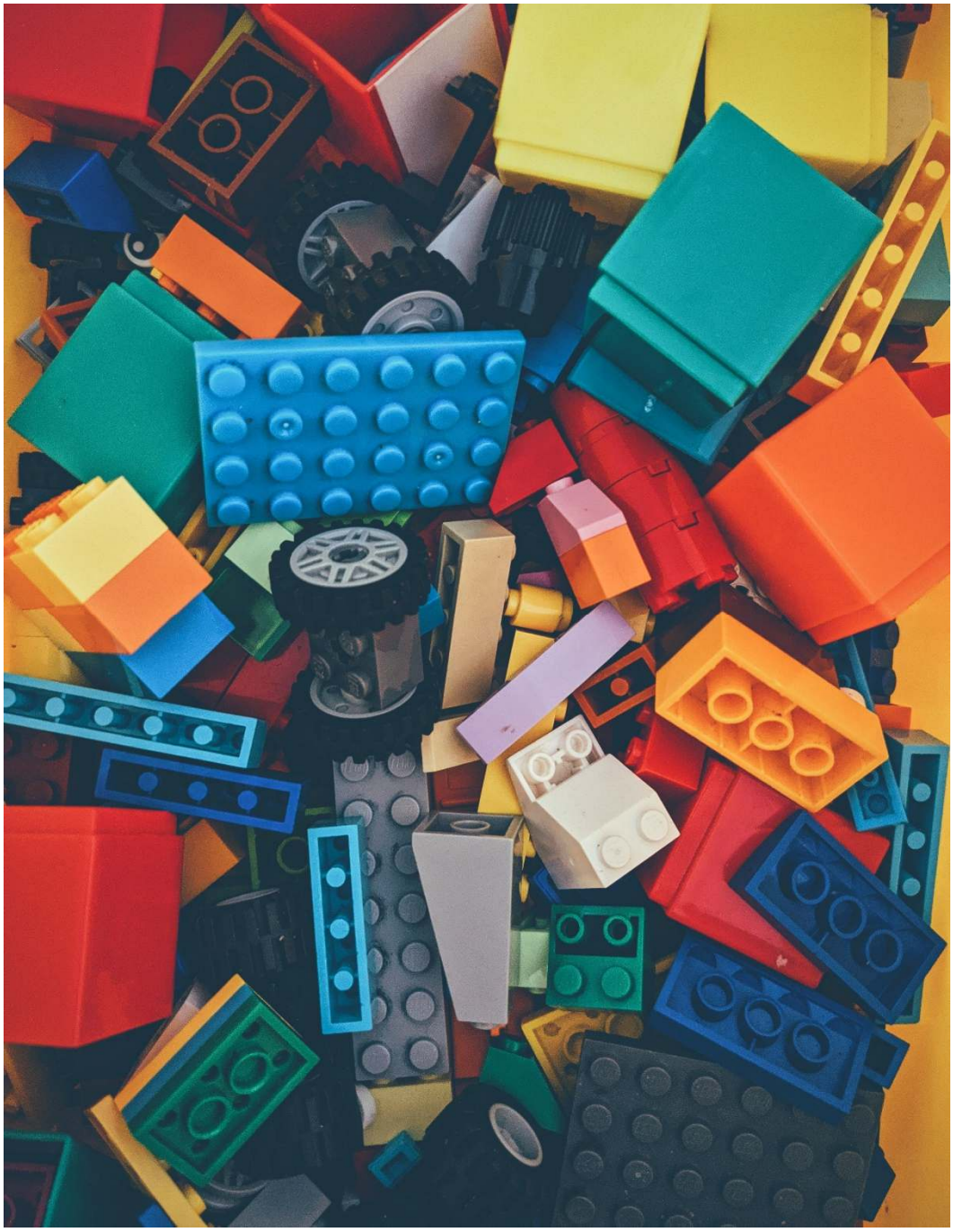
The Java Development Kit all in one image.

#The Basics: Java and OOP

Now that you understand how a Java program gets from the developer's keyboard through to the user's screen, we should talk briefly about how we write Java programs: **object-oriented programming** (OOP). There are many different ways of programming, but Java adheres to the OOP paradigm. In OOP, we write our code to

create objects which contain code including variables and methods that they use to interact with other objects.

For now, let's use a metaphor: think of objects as Lego bricks. Each brick will have its own specific size and shape, and it can be combined with other bricks to create larger structures. Some Legos will be more complex than simply interlocking with other bricks–like having axels, wheels, or hinges–but none of them is particularly impressive alone. It is only when they come together that they can create something truly spectacular.



A pile of Legos, care of Nick Nice on Unsplash

Every object in OOP has a purpose to fulfill, but each one is a separate entity that can be detached from the program as a whole. OOP, therefore, represents a **modular** design paradigm where individual entities come together to create complex programs capable of nearly anything you can imagine a computer doing.

#Summary

These are the basics that make Java work, but remember: this lesson is just a primer to launch us into writing a bit of code ourselves. We'll dive much deeper into each of these pieces as we progress through the curriculum, so don't worry about researching the above for hours on your own. We'll get there.

#Extra Resources

If you'd like more information about how a platform-independent language works, check out this quick two minute video:
https://youtu.be/wsAbhPQKHvo

Back
Unsubmit
Next

Writing Java

It's a tradition in the programming world for new devs to run a little app called "Hello World" first. No matter what language you're working in, the first step is always to simply say "hello".

Look at the left side of the file below–see the line of numbers going down? On line 5, write **Hello World** in the parentheses `""`, then click the green `Run` arrow above:

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

CONSOLESHELL

Give it a few seconds to compile, and it should say Hello World in the `CONSOLE` portion of the screen below. If it did, you've done it. Congratulations. You've run a program. Now let's talk about what just happened.

#The IDE

Coding Rooms, the site you're using right now, has built-in IDEs–Independent Development Environments. Soon, we'll move to an IDE outside this website, but

we're starting you off easy. The **code** portion of the IDE is where you typed `Hello World`, and the space underneath is called the **Console**. All IDEs will have these partitions, and when you're working with Java code, you'll also find some version of that green "Run" button to run your program. If you're using commands that print to the console, pushing that button will run the code and output that information.

The IDE above is yours to experiment with. Coding Rooms will save your work, and your instructors will be able to see it. We'll be using these IDEs to check your work a few times a day, but we'll make it explicit if that's what we're using them for. If we don't tell you to achieve a specific goal in them, consider these IDEs an experimental space. Do you want to try printing other things? Go for it. Do you want to see what happens when you add or remove some lines from the above? Go for it. You may get an error, but keep undoing your actions (`option + z` or `ctrl + z`, depending on your operating system) until you get it back to a working state. It's yours to tinker with.

Remember: the more you experiment, the more likely you'll start to understand what's happening. Before you do that this time, it might be helpful to know a bit about boilerplate code.

#The Bytecode - CAFEBABE: Java's Magic Word

In the previous lesson, we pointed out that Java programs need to be compiled into bytecode to execute on the JVM. Let's take a second and do that process manually. Using this process, you could compile your Java program even if you didn't have an IDE handy. Will you ever encounter this scenario? Probably not! Still, it's fun to explore what's going on under the hood. Follow along as we show you what the Java bytecode looks like:

1. Open a text editor such as Sublime or textEdit and copy & paste the following code.

```
public class Hello {

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

2. Save the file as Hello.java. Ensure there are no typos; otherwise, it won't work. Remember, Java is a bit picky, especially when you're tinkering under the hood.
3. Open a Windows Command Prompt or a Mac Terminal in the folder where you saved the Hello.java file.
4. Compile the Hello.java program using the `javac` utility (Java compiler utility). This little utility was installed when you installed JDK. It's used for occasions such as this.
    - Run `javac Hello.java` on your Windows command prompt or MacOS Terminal.
5. You will notice a new file called Hello.class in your folder has appeared. The javac utility compiled and generated this bytecode file for you. Note: a bytecode's file extension is always .class, just like the keyword we used in the code we wrote above.
6. Execute the bytecode using the `java` utility. This was installed when you installed JDK, too.
    - Run `java Hello` on your Windows command prompt or MacOS Terminal.
    - The program runs and prints out "Hello World!".
7. Finally, using a text editor, open the bytecode file Hello.class in hex mode. Notice the first four bytes (the file format code) are "CAFEBABE" (two hexadecimal is a byte).

```
cafe babe 0000 003d 001d 0a00 0200 0307
0004 0c00 0500 0601 0010 6a61 7661 2f6c
616e 672f 4f62 6a65 6374 0100 063c 696e
6974 3e01 0003 2829 5609 0008 0009 0700
```

If you want to know more about how Java works under the hood, stay tuned. We get into much more detail later in the course. If, on the other hand, you're not interested in bytecode and hexadecimals–don't worry. The majority of devs don't dig into processes such as the above unless they're working on hobby projects of their own, like Arduinos and Raspberry Pis.

#Boilerplate Code

Boilerplate code seems extraneous at first but it is necessary to run the code you write. For instance: to make a Java program run, you're going to need at least two lines of boilerplate:

```
public class Hello {
public static void main(String[] args) {
// Everything you write will live here.
}

}
```

These lines are important because they tell Java what to do with the code you're writing, but you'll only copy and paste them at first. We'll get into the details of what they mean and how to adapt them to different circumstances later. For now, know that all programming languages require a bit of boilerplate to run–some more than others, and Java more than most.

#Reviewing Your Hello World

Starting at line one, we're declaring a **class**. Every program in Java is written as a class. This is an essential building block in any object-oriented programming language, and we'll learn more about it soon. Inside that class, we have a **main method** on line two that starts the program running. When you ask the Java environment to *run* a class, the compiler will look for this **main method**. If you don't have one, you don't have a Java program.

In Java, every open **curly brace** `{` must have a matching closing curly brace `}`. These are used to start and end class and method definitions. When we say "inside of that class," we mean contained within these curly braces because that defines where the Main class begins and ends.

The special characters `//` as seen on line three, are used to mark the rest of the line as a **comment**. Comments are essential to writing code, as they allow you to explain what the code is doing. You can safely use the double forward slash whenever you like to leave yourself notes about your code. They're also helpful for leaving comments for other developers when you're working on a team, and good comment etiquette is an important part of being a developer. As you go through this course,

feel free to write yourself comments and make the code your own–especially if you figure out something difficult and don't want to forget it.

In your Hello World, you also made use of the command `System.out.println("");`. This command uses one of **Java's built-in classes** (**System**) to print a line to the console. Notice here, too, that the line ends in a `;` semicolon. In Java, all of our lines of code that include instructions (aka, "commands") that we want the program to execute will end in **semicolons**, like how written English ends each sentence with a period. If your code isn't running for whatever reason, check to see that you have your semicolons in place.

InfoWarningTip
If you're wondering why `System` is capitalized but `out` and `println` are not, it's because `System` is a class while `out` and `println` are just **commands**. In Java and many other languages, certain keywords will be capitalized for various reasons. It's a good idea to pay attention not only to punctuation, spacing, and opening/closing markers like curly braces, but also to **capitalization** as you learn.

#Another Exercise

Click the `Run` button below to have the computer execute the `main` method in the following class. `System.out.println("Hi there!");` prints out the characters between the first `"` and the second, `"` followed by a **new line**. The `"Hi there!"` is called a **string literal**, and it can have zero-to-many characters enclosed in between the starting and ending double quotes.

After you've run the program as is, change the code to print your name. Be sure to keep the starting `"` and ending `"`. Run the modified code to test your changes. If you revisit this page later while logged in, click the `Load History` button and move the bar above it to see your previous code changes.

If you mess up, hit the `Load History` button and use the slider to go back to a previously working version, or use `cmd + z` or `ctrl + z` until you make it back to a

working version. You can run the program as many times as you like, so have fun with it:

```
public class HelloExample {
   public static void main(String[] args) {
      System.out.println("Hi, ");
      System.out.println("How ");
      System.out.println("Are ");
      System.out.println("You?");

      // couple of ways to put it in one line
      //1:
      System.out.println("Hi, How Are You?");
      //2:
      System.out.print("Hi, ");
      System.out.println("How ");
      System.out.print("Are ");
```

CONSOLESHELL

#Print commands

Java has two different print commands to print output to the screen:

- **System.out.println(value)**: This command prints the value inside the parentheses followed by a new line (**ln**).
- **System.out.print(value)**: This command prints the value inside the parentheses without advancing to the next line.

Coding Exercises:

What follows are a few quick exercises and a question or two. See if you can achieve the goals outlined in each statement.

1. Run this code to see the output below it. How would you change it to print the `!` on the same line as "Hi there" *while* keeping all three separate print statements?

```
public class HelloExample2 {
   public static void main(String[] args) {
      System.out.print("Hi ");
      System.out.print("there");
```

```
        System.out.print("!");
    }
}
```

CONSOLESHELL

Quick Question!

Answer the following, then let's keep on going:

Consider the following code segment.

CopyCC#C++ClojureCSSDartGoHaskellHTMLJavaJavaScriptJSONJSXKotlinMarkdown PascalPerlPHPPlain TextPythonRRubyRustSchemeShellSQLSwiftTypescriptVB.NETVBScriptXMLYAML

```
System.out.print("Java is ");
System.out.println("fun ");
System.out.print("and cool!");
```

What is printed as a result of executing the code segment?

Java is fun and cool!

Java isfun and cool!

Java is fun and cool!

Java is fun and cool!

A print statement can also contain **numeric** values and **arithmetic expressions**, as seen in the code below. Make sure you don't use double quotes for expressions with a numeric value, as that will make Java interpret them as Strings instead. One of the best things about modern IDEs, including the ones built into Coding Rooms, is that values will generally change colour for you if their type has changed. Notice how, in the above examples, our text is a reddish-orange colour, but in the below, it's green for all of the numbers. Get used to using these helpful hints.

2. Run the code to see the output below it. Can you modify the final print statement to print the sum of the values from 1 to 10?

```
public class CalculationExample {
    public static void main(String[] args) {
        System.out.println(570 * 23);
        System.out.println(12.34 / 5);
        int sum = 0;
        for (int i = 1; i <= 10; i++) {
            sum += i;
        }
        System.out.println(sum);
    }
}
```

CONSOLESHELL

3. Run the code to see the output below. The output is not correct. The second `System.out.println` statement should print the value resulting from the computation, not a literal string for the computation. Get rid of the double quotes in the second `println` statement and see if you can fix the program. Click Run again when you've achieved the desired results:

4. Assume you have some bills to pay. The individual bill amounts are 89.50, 14.75, 45.12, and 92.50. Add another print statement to sum and print the total bill amount on a separate line. Don't just add the numbers in a calculator and print the result. Write the code to add the numbers and print the result.

```
public class CalculateBillTotal {

   public static void main(String[] args) {
      double bill1 = 89.50;
      double bill2 = 14.75;
      double bill3 = 45.12;
      double bill4 = 92.50;

      double totalBill = bill1 + bill2 + bill3 + bill4;

      System.out.println("Bill total:");
      System.out.println(totalBill);
   }
```

}
CONSOLESHELL

5. A bus starts with no passengers. Three people get on at the first stop. Five people get on at the second stop. One person gets off, and eight people get on at the third stop. Three people get off at the fourth stop. How many people are left on the bus? Add another print statement to calculate and print the passengers remaining on the bus.

```
public class PassengersOnBus {
   public static void main(String[] args) {
      int passengers = 0; // Initialize the count of passengers

      // First stop: 3 people get on
      passengers += 3;

      // Second stop: 5 people get on
      passengers += 5;

      // Second stop: 1 person gets off, 8 people get on
      passengers -= 1;
      passengers += 8;

      // Fourth stop: 3 people get off
```
CONSOLESHELL

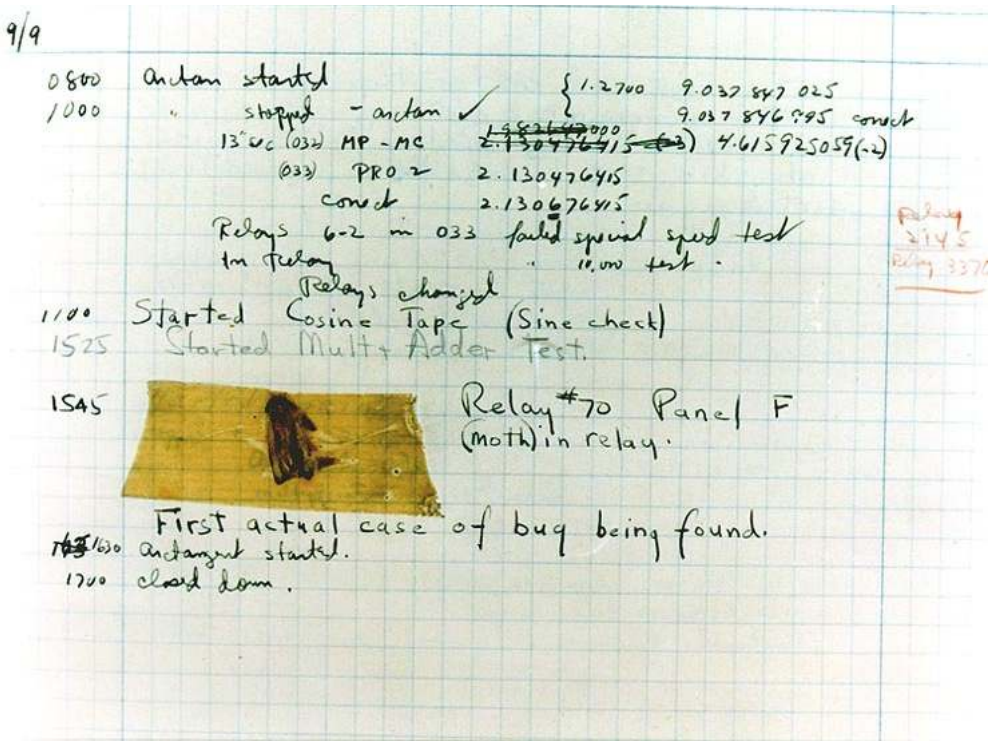
#Syntax errors

As described before, no computer *speaks* Java, so we must **compile** (translate) Java source files that we write into class files. These class files represent code that a computer can understand and run. In Coding Rooms, the Java code we write is sent to a Java server to compile and run, and the output is sent back to your browser to show your results on the page.

When something goes wrong, you're going to see a Syntax Error. **Syntax errors** are reported to you by the compiler if your Java code is written incorrectly. Examples of

syntax errors include missing semicolons `;` , or having an open curly brace `{` or open quote `"` without its matching closing brace or quote.

Informally, a syntax error is called a **bug**, and the process of removing errors is called **debugging**. This name comes from when actual bugs got into computers and disrupted their functionality. One early computer science pioneer, Grace Hopper, documented one such bug: a moth that flew into a computer she was working on in 1947.



One of the first documented "bugs" in computer programming.

To make a long story short, the compiler tries to make sense of your code, but if your code has **syntax errors**, you will see error messages displayed below the code. Compiler error messages will tell you the **line number** at which the compiler found the error and the type of error it found. These error messages are not always easy to understand, and sometimes the actual error may occur *before* the line that the compiler says is the problem.

The key is to take a deep breath and enjoy the experience of hunting down the bug. Computers don't simply work how we want them to just because we ask nicely. They'll always do what we've instructed them to do, so embrace debugging, error tracing, and figuring out how to write your code better and prevent that particular

bug from happening again. No matter how seasoned a programmer you are, you'll still find bugs, so you may as well get used to it. It can be a bit frustrating, but you will get better with practice!

Let's start now:

Check Your Understanding: Mixed-up programs

The following has all the correct code needed to print out “Hi my friend!” when the code is run. However, the code is mixed up.

Drag the blocks from left to right and put them in the correct order. You can look at the previous programs on this page if you're having trouble.

Click on the “Check” button to check your solution. You will be told if any blocks are in the wrong order or if you need to remove one or more blocks. Hint: this problem has no extra blocks.

After three incorrect attempts, you can use the “Help me” button to make the problem easier.

Options

Solution

```
public class HelloExample3 {

public static void main(String[] args) {
System.out.println("Hi my friend!");
}
}
```

The following has the correct code to print out “Hi there!” when the code is run, but the code is mixed up **and** contains some extra blocks with errors. Drag the needed blocks from left to right and put them in the correct order, then check your solution.

Options

```
public static void hello() {
```

Solution

```
public class HelloExample4 {

public static void main(String[] args) {

System.out.println("Hi there!");

}

}
```

Coding Exercise: Compile Time Error 1

Run the following code and look for the error message after it runs. This is called a **compile-time error** because the compiler detects it.

What's wrong? Can you fix it? The error message will tell you the line number that it thinks is causing the error (`Error1.java:5: error: unclosed string literal`). Check that line to make sure that everything looks correct.

If you need a hint, reveal it below. We recommend you try it first, though.

Reveal Content

Fix the code below

```
public class Error1 {
   public static void main(String[] args) {
      System.out.println("Hi there!");
   }
}
```

CONSOLESHELL

Coding Exercise: Compile Time Error 2

Try and run the following code. Look for an error message after the code. What's wrong this time? Can you fix it? The same hint from above applies here.

Fix the code below.

```
public class Error2 {
   public static void main(String[] args) {
      System.out.println("Hi there!");
   }
}
```

CONSOLESHELL

Coding Exercise: Compile Time Error 3

Try and run the following code. What is wrong this time? Can you fix it? After you fix the first error, you may encounter a 2nd error! Fix that one too!

Fix the code below.

```
public class Error3 {
   public static void main(String[] args) {
      System.out.println("Hi there!"); // "System" should start with an upperc
ase 'S'
   }
}
```

CONSOLESHELL

If you're stuck, reveal the hint below:

Reveal Content

#Debugging challenge

Debug the following code. Can you find all the bugs and get the code to run?

```
public class Challenge1 {
```

```
    public static void main(String[] args) {
        System.out.print("Good morning! ");
        System.out.print("Good afternoon!");
        System.out.print(" And good evening!");
    }
}
```

CONSOLESHELL

Take a moment and reflect: what clues did you have in the original code that hinted at where the errors were? What can you look out for, even before you click run on your code?

#Comments

We mentioned them briefly, but let's try to understand them now. Adding comments to your code helps to make it more readable and maintainable. In the commercial world, software development is usually a team effort where many programmers will use your code and maintain it for years. Commenting is essential in this environment and a good habit to develop. Comments will also help you to remember what you were doing when you look back at your code a month or even years from now.

There are three types of comments in Java:

1. `//` Single line comment
2. `/*` Multiline comment `*/`
3. `/**` Documentation comment `*/`

In Java and many other text-based coding languages, `//` it is used to mark the beginning of a comment. Everything on the line that follows the `//` is ignored by the compiler. For multi-line comments, use `/*` to start the comment and `*/` to end the comment. There is also a particular version of the multi-line comment, `/**` `*/`, called the documentation comment. Java has a cool tool called Javadoc that will pull all of these comments out of your files to make documentation for your class. It then displays these compiled comments as a web page.

The compiler will skip over comments. However, using comments to make notes to yourself and other programmers working with you is a good idea. Here is an example of commenting:

```
/** MyClass.java
Programmer: My Name
Date:
*/

int max = 10; // this keeps track of the max score
```

Check your understanding

Options

//

single-line comment

/* */

multi-line comment

/** */

Java documentation comment

InfoWarningTip

Remember: the compiler will skip over comments, and they won't affect how your program runs. They are for you and other programmers working with you. Use them wisely!

#Summary

- A basic Java program looks something like this:

```
public class MyClass {
public static void main(String[] args) {
System.out.println("Hi there!");
}
}
```

- A Java program starts with `public class NameOfClass { }`. If you are using your files for your code, each class should be in a separate file that matches the class name inside it, for example `NameOfClass.java`. We'll review this again when we work with a real IDE, but it's essential to know where the names need to match. Read more here if you like.

- Most Java classes have a main method that will be run automatically. It looks like this: **public static void main(String[] args) { }**.

- The `System.out.print()` and `System.out.println()` methods are used to display information given inside the parentheses in the console.

- `System.out.println` moves the cursor to a new line after the information has been displayed while `System.out.print` does not.

- A **string literal** is enclosed in double quotes `"your string here"`.

- Java command lines end in `;` (semicolons). `{ }` are used to enclose blocks of code.

- `//` and `/* */` are used for single-line and multiline comments, respectively.

- A **compiler** translates Java code into a class file that can be run on your computer. **Compiler or syntax errors** are reported by the compiler if the Java code is not correctly written. Some things to check for are `;` semicolons at the ends of command lines, matching `{ }`, `( )`, and `""`.

#Extra Resources

Here's an article about errors from Medium. It's a good place to look for blog-style information when you're learning something new: https://medium.com/javarevisited/10-deadly-mistakes-to-avoid-when-learning-java-aead894e64f4

Back
Unsubmit
Next