# What is Java?

Wednesday, November 7, 2018 9:42 AM

## What is Java?

- Created in 1995
- Based on C and C++ programming languages
- Open sourced

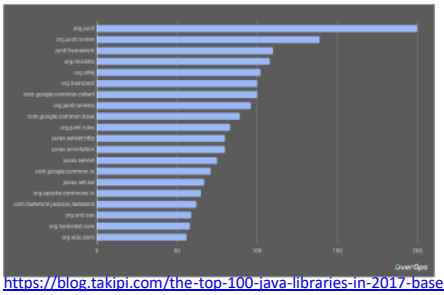
IT IS **NOT** JAVASCRIPT!
They are syntactically similar but that's about it!

### Java is MASSIVE and DENSE

*Way too big* for us to cover all aspects in one session. There are different ways to use Java, different versions of Java, and thousands of libraries and APIs to utilize.

**Find out how you're going to USE Java and use that knowledge to determine how your should LEARN Java.**

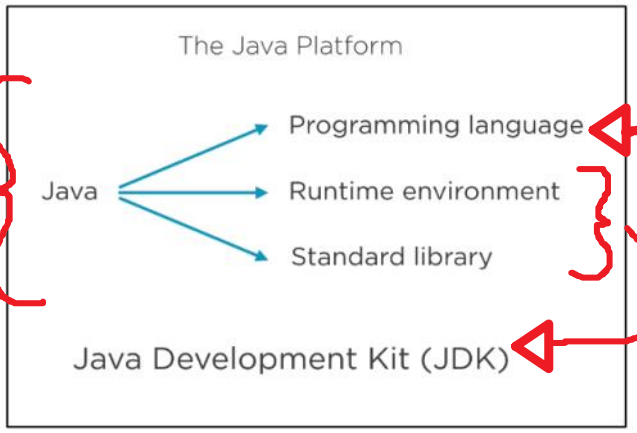

https://www.pluralsight.com/courses/modern-java-big-picture



https://blog.takipi.com/the-top-100-java-libraries-in-2017-based-on-259885-source-files/

In general, when people refer to Java, they are usually talking about the actual *programming language.*

But Java is actually made up of the programming language, the *runtime environment*, and the *standards library(ies).*




The *runtime environment* and the *standard library* are what make up the *Java Development Kit* and are critical to making Java *platform independent.*

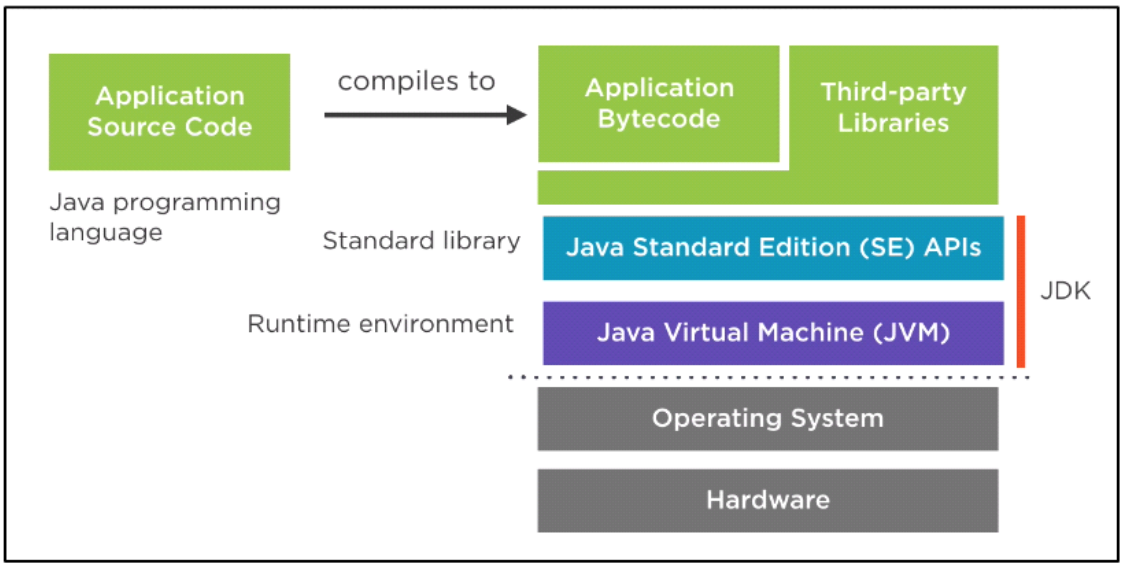
https://www.pluralsight.com/courses/modern-java-big-picture

## What does *platform independent* mean??

A programming language or package being *platform independent* means that you only need to write one set of code which can then be run on different platforms.

Java accomplishes this by compiling the *application source code* into *application bytecode*.

That then gets combined with any libraries and APIs, which the *Java Virtual Machine* then sends to the operating system which is able to convert it into code that is compatible with the OS and hardware of the device attempting to run the Java program.




## Ok, how do I USE Java?

Make sure you have a Java JDK installed on your machine!

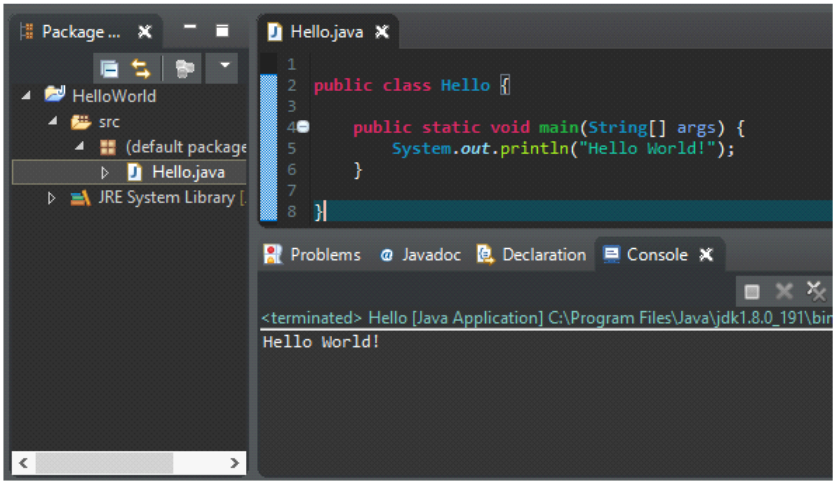
*Most HIG laptops at least have the JRE (runtime environment) installed but a simple MTE request can get you the JDK for whatever version of Java you'll need.*

Then get yourself an IDE that has the features you're looking for.

## What the heck is an IDE??

An IDE is an *Integrated Development Environment*. It's a program that allows you to write, debug, compile, and run Java code.

Common examples are Eclipse, MyEclipse, NetBeans, IntelliJ, and VSCode.




Example of source code compiled and run inside of an IDE (Eclipse).

# The Java Programming Language

Friday, November 9, 2018 7:59 AM

https://app.pluralsight.com/library/courses/java-fundamentals-language/

## Variables

In Java, VARIABLES, are STRONGLY TYPED, and can be modified.

Strongly typed

```
int dataValue;
```

Variable **dataValue** can only hold data type **int**

**Why??**

Catch bugs early

```
public class Hello {

   public static void main(String[] args) {
      int message = "Hello PluralSight!";
      System.out.println(message);
   }

}
```

```
Hello.java:4: error: incompatible types: String
cannot be converted to int
     int message = "Hello PluralSight!";
                   ^
```

https://www.pluralsight.com/courses/modern-java-big-picture

## Naming Variables

- Rules allow the use of LETTERS, NUMBERS, $ and _
  - Typically, you would only use letters and numbers
- The first character CANNOT be a NUMBER
- Written in camelCase

## Primitive Data Types

- Data types BUILT INTO the language
  - Foundation of ALL other types
- Four categories:
  - **Integer**
  - **Floating point**
  - **Character**
  - **Boolean**

## Integer Types

Four different types, but the only real difference is the size of storage they take up which does affect the range of values that can be stored in them

| Type | Size (bits) | Min Value | Max Value | Literal Format |
|---|---|---|---|---|
| byte | 8 | -128 | 127 | 0 |
| short | 16 | -32768 | 32767 | 0 |
| int | 32 | -2147483648 | 2147483647 | 0 |
| long | 64 | -9223372036854775808 | 9223372036854775807 | 0L |

```
byte numberOfEnglishLetters = 26;
short feetInAMile = 5283;
int milesToSun = 92960000;
long nationalDebt = 18100000000000L;
```

# Floating Point Types

Want more info on floating points?

http://bit.ly/psjavafp

A bit more complicated, but main points are:

- Can store values containing a fractional portion
- Supports positive, negative, and zero values

| Type | Size (bits) | Smallest Positive Value | Largest Positive Value | Literal Format |
|---|---|---|---|---|
| float | 32 | $1.4 \times 10^{-45}$ | $3.4 \times 10^{38}$ | 0.0f |
| double | 64 | $4.9 \times 10^{-324}$ | $1.7 \times 10^{308}$ | 0.0 or 0.0d |

Any number with a decimal and no additional notation will be assumed to be a double!

```
float milesInAMarathon = 26.2f;
double atomWidthInMeters= 0.0000000001d;
```

# Character and Boolean Types

The **char** type stores a single Unicode character

```
char regularU = 'U';
```

**Single quotes!**

Want to use a character that isn't on your keyboard?
Use Unicode code points!

- \u followed by the 4-digit hex value

```
char accentedO = '\u00D3'; Ó
```

https://en.wikipedia.org/wiki/List_of_Unicode_characters

The **boolean** type stores true/false values (who would have guessed??)

```
boolean iLoveJava = true;
```

# THINGS TO KNOW

Primitive data types are stored **by-value**

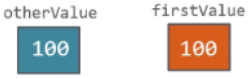
- This means that you can assign one value:

```
int firstValue = 100;
```

- Then assign another variable to equal that same value:

```
int otherValue = firstValue;
```

- And what you'll wind up with is two copies of the value 100




- If you then change the value of `firstValue`:

```
firstValue = 50;
```

- `otherValue` will retain the original value of 100 because that is the value stored in that variable


## Basic Math Operators

| | Operator |
|---|---|
| Add | + |
| Subtract | - |
| Multiply | * |
| Divide | / |
| Modulus | % |

## Prefix / Postfix Operators

++ increments value by 1

-- decrements value by 1

As prefix applies operation before returning value

```
int myVal = 5;
System.out.println(++myVal);   6
System.out.println(myVal);     6
```

As postfix applies operation after returning value

```
int myVal = 5;
System.out.println(myVal++);   5
System.out.println(myVal);     6
```

# What if I need to change my data type??

**Enter…. Type Conversion!**

**There are two types of type conversion:**

- **Implicit**
- **Explicit**

## Implicit Type Conversion

- Widening conversions are automatic
  - For example:
    - `byte` to `short`
    - `short` to `int`
    - `int` to `long`
- This will happen when there are mixed integer sizes
  - Uses the largest integer in the equation

- For example:
  - Adding a `short` to a `long`, the `short` will be implicitly cast into a `long`
- This will also happen when there are mixed floating point sizes
  - Uses `double`
  - For example:
    - Adding a `float` to a `double`, the `float` will be implicitly cast into a `double`
- Lastly, mixing integer and floating point
  - Will cast all integers into the largest floating point in the equation
  - For example:
    - Adding a `long` to a `float`, the `long` will be implicitly cast into a `float`

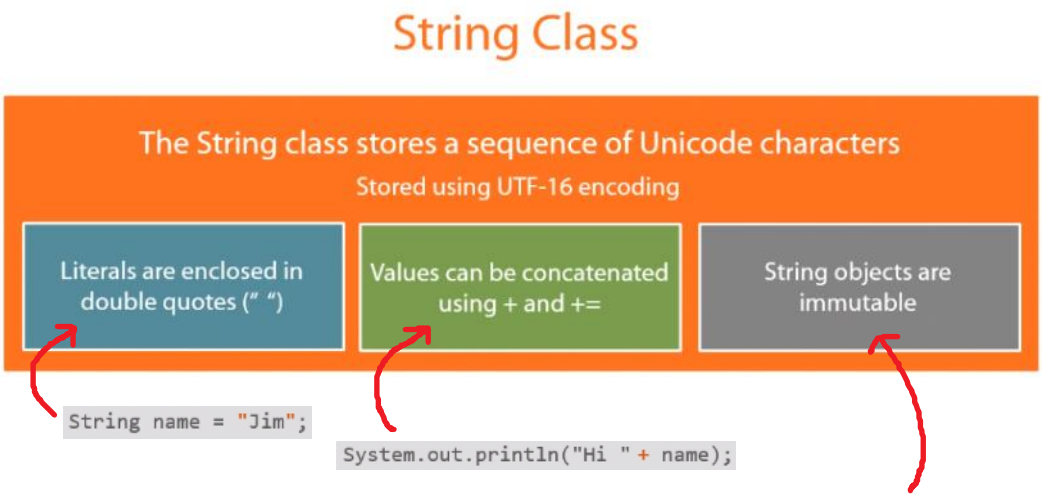
# Explicit Type Conversion

- These conversions are not automatic and only happen when we call them.
- This means they can both *widen* and *narrow*
- Be mindful when narrowing!
  - Casting from a floating point to an int will drop the fraction, etc.

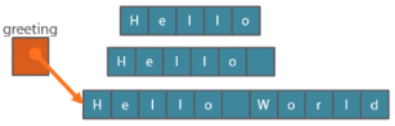
Want more info on type conversion?
http://bit.ly/pstypeconversion

## What about strings??

**In Java, strings aren't actually a data type, they are a *class*.**
**Since they are a class, any reference to String should be capitalized.**




```
String greeting = "Hello";
greeting += " ";
greeting += "World";
```




Once a String value has been assigned, it can never be changed without a whole new instance of the variable being created.
- This is SUPER INEFFICIENT!

Use the `StringBuilder` class for instances like this!

https://docs.oracle.com/javase/7/docs/api/java/lang/StringBuilder.html

# Object Oriented Programming Basics

Friday, November 9, 2018 12:56 PM

https://app.pluralsight.com/library/courses/what-is-programming

http://www.ntu.edu.sg/home/ehchua/programming/java/J3a_OOPBasics.html

Java, like a lot of other programming languages, is Object Oriented.

Object orientation boils down to *the way that we separate and group things*.

## Understanding Objects

**Begin with non-technical language**

Think about what your program needs to do, in plain English, and then look at the nouns

"A customer can begin a new order
by adding an item to the shopping cart"

Potential objects: customer, order, item, shopping cart...

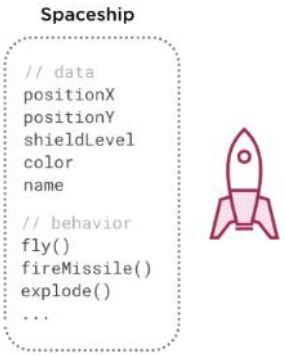
For a media player: album, track, playlist...

For a game: spaceship, enemy, asteroid, missile...

## Objects are NOT JUST DATA

They are data and functionality with behavior and code that perform tasks on that data

Each object almost becomes a self-contained miniature program, able to manage both its own data and its own behavior




## OO Terminology

| Class | Object |
|---|---|
| How we define an object | The object itself |
| "The Blueprint" | "The house made from the blueprint" |
| "The Recipe" | "The dish made from the recipe" |
| One class can be used to make multiple objects | |
| The class comes first! | |

## Defining a Class

```
class Lamp {
   boolean isOn;

   void turnOn() {
      isOn = true;
   }

   void turnOff() {
     isOn = false;
   }

   void displayLightStatus() {

       System.out.println("Light on? " + isOn);
   }
}
```

<- Properties

} Methods

There's no point in defining a class unless you plan to make an object, so…

# Creating an Object

or *Instantiation*

## Declare and Construct

```
// Declare 3 instances of the class Circle, c1, c2, and c3
Circle c1, c2, c3;  // They hold a special value called null
// Construct the instances via new operator
c1 = new Circle();
c2 = new Circle(2.0);
c3 = new Circle(3.0, "red");

// You can Declare and Construct in the same statement
Circle c4 = new Circle();
```

Q. How do I access a class's methods?

A. Using the dot operator!

1. First identify the instance you are interested in, and then,
2. Use the *dot operator* (`.`) to reference the desired member variable or method.

```
class ClassObjectsExample {
  public static void main(String[] args) {

    // create a new object from that class
    Lamp lamp1 = new Lamp();
    //call methods
    lamp1.turnOn();
    lamp1.displayLightStatus();

    // instantiate another object
    Lamp lamp2 = new Lamp();
    //call methods
    lamp2.turnOn();
    lamp2.displayLightStatus();

  }
}
```

The `ClassObjectsExample` class has a `main()` method and can be executed.

You would save this as "ClassObjectsExample.java", compile it, run it, and be able to study the output.

## Basic Idea…

Once a class has been defined, you can create, or instantiate, multiple objects (instances) based on that class.

Also, most of the objects you instantiate, you won't have to make the class.

- In most real world programming environments, there are a huge amount of classes already defined and available for use.
  - Dates
  - Times
  - Image
  - Sound Players
  - Graphics
  - Audio
  - Encryption
  - User Interfaces

Want to be productive as an OOP programmer?

Don't re-invent the wheel, find what is available and tap into it!

# The Four Main OOP Concepts in Java

Monday, November 12, 2018 12:36 PM

https://stackify.com/oops-concepts-in-java/
http://www.ntu.edu.sg/home/ehchua/programming/java/J3a_OOPBasics.html
http://www.ntu.edu.sg/home/ehchua/programming/java/J3b_OOPInheritancePolymorphism.html
https://www.dineshonjava.com/difference-between-abstraction-and-encapsulation-in-java/

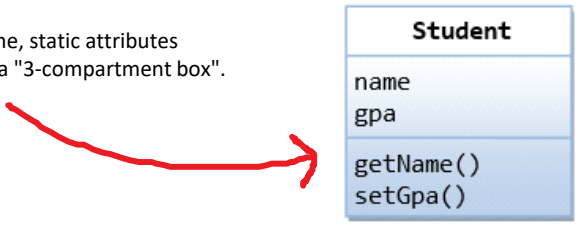
There are four main object oriented programming concepts in Java.

The first…

## Encapsulation

This is the practice of keeping fields within a class private, then providing access to them via public methods. It keeps the data and code safe within the class itself. This way, we can re-use objects like code components or variables without allowing open access to the data system-wide.

Think of it this way:
A class encapsulates the name, static attributes and dynamic behaviors into a "3-compartment box".




You can then seal up the "box" and allow others to use it over and over again without worrying about someone "breaking" a "working box".

You seal the box by using a `private` access control modifier.

Access to the member variables are provided via `public` assessor methods, e.g., `getName()` and `setGPA()`.

This follows the principle of *information hiding*. That is, objects communicate with each other by using well-defined interfaces (public methods). Objects are not allowed to know the implementation details of others. The implementation details are hidden or encapsulated within the class. Information hiding facilitates reuse of the class.

**A what?**

An *access control modifier* is used to *control the visibility* of a class, or a member variable or a member method within a class.

1. `public`: The class/variable/method is accessible and available to *all* the other objects in the system.
2. `private`: The class/variable/method is accessible and available *within this class only*.

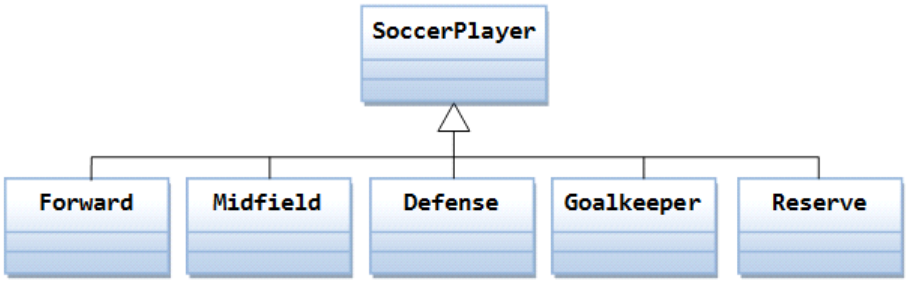
**Rule of Thumb:** Do not make any variable `public`, unless you have a good reason.

Next up…

## Inheritance

This concept allows us to avoid duplication and reduce redundancy.

Think of all the classes in a program belonging to a hierarchy:




The classes in the lower hierarchy inherit all the variables (static attributes) and methods (dynamic behaviors) from the higher hierarchies.

- A class in the lower hierarchy is called a *subclass* (or *derived*, *child*, *extended class*).
- A class in the upper hierarchy is called a *superclass* (or *base*, *parent class*).

Remember: a subclass is not a "subset" of a superclass. Subclasses are actually "supersets" of a superclass. Why? Because a subclass inherits all the variables and methods of the superclass AND it extends the superclass by providing more variables and methods.

In Java, a subclass is defined by using the keyword "`extends`", e.g.,

```
class Goalkeeper extends SoccerPlayer {......}
class MyApplet extends java.applet.Applet {.....}
class Cylinder extends Circle {......}
```

Moving on to…

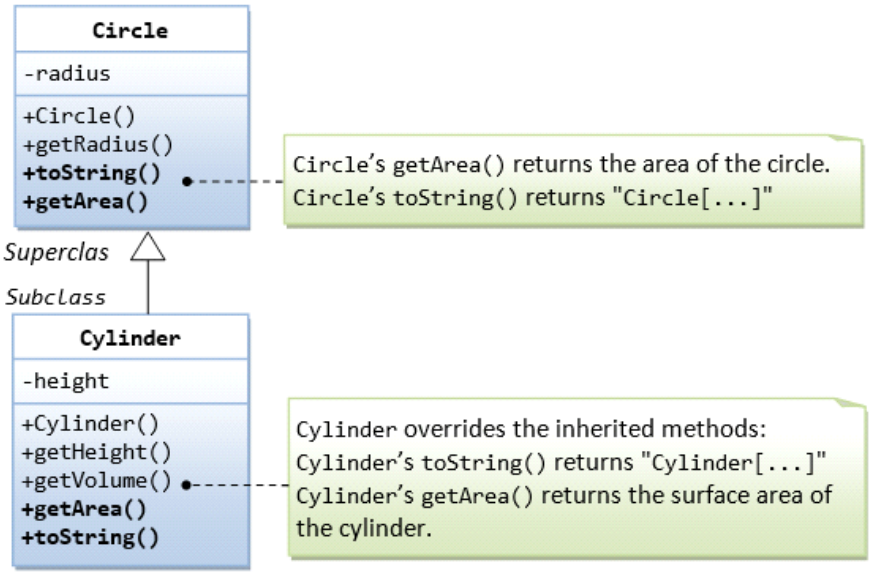
# Polymorphism

A subclass instance possesses all the attributes and operations of its superclass, right? This means that a subclass object can do whatever its superclass can do! Therefore, when a superclass instance is expected, it can be substituted by a subclass instance.

In other words, a reference to a class may hold an instance of that class or an instance of one of its subclasses - this is called *substitutability*.

Once substituted, we can invoke methods defined in the superclass; we cannot invoke methods defined in the subclass.

However, if the subclass overrides inherited methods from the superclass, the subclass's version will be executed, instead of the superclass's version.




And lastly…

# Abstraction

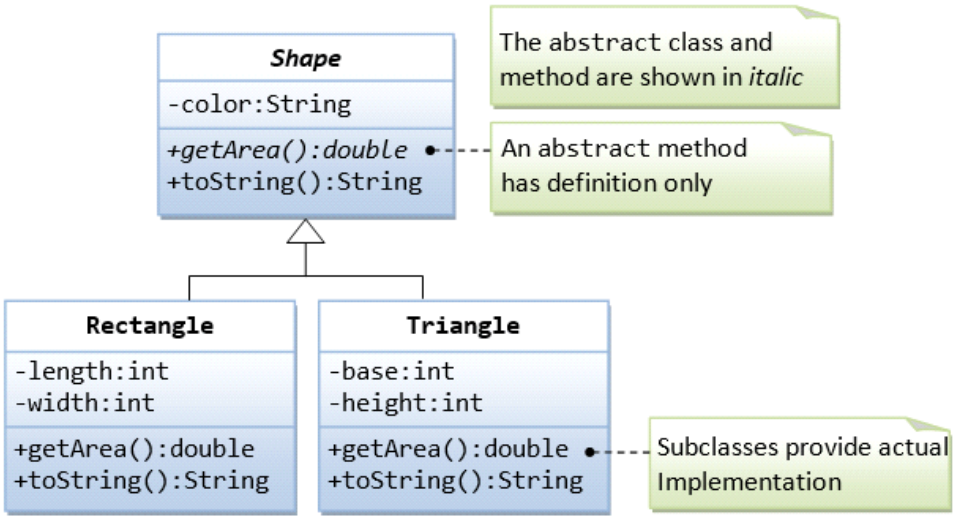
Abstraction is a general concept which you can find in the real world as well as in OOP languages.

Its main goal is to handle complexity by hiding unnecessary details from the user using `Abstract` and `Interface` classes.

This allows the user to implement more complex logic on top of the provided abstraction without understanding or even thinking about all the hidden complexity.

Any objects in the real world, like your coffee machine, or classes in your current software project, that hide internal details provide an abstraction.

These abstractions make it a lot easier to handle complexity by splitting them into smaller parts.



## Wait a minute… This sounds an awful lot like encapsulation!

Well, you're right, in a way. The easiest way to tell the two apart is with this simple explanation:
∴ Information hiding.
∴ Implementation hiding.

| Abstraction | Encapsulation |
|---|---|
| Abstraction is a general concept formed by extracting common features from specific examples or The act of withdrawing or removing something **unnecessary**. | Encapsulation is the mechanism that binds together code and the data it manipulates, and keeps both **safe from outside interference** and **misuse**. |
| You can use abstraction using **Interface** and **Abstract** Class | You can implement encapsulation using **Access Modifiers** (Public, Protected & Private) |
| Abstraction solves the problem in **Design** Level | Encapsulation solves the problem in **Implementation** Level |
| For simplicity, abstraction means hiding implementation using Abstract class and Interface | For simplicity, encapsulation means hiding data using getters and setters |