Java coding language

Java is known for being simple, portable, secure, and robust.

Java Virtual Machine, which ensures the same Java code can be run on different operating systems and platforms. Sun Microsystems' slogan for Java was "write once, run everywhere".

Java files have a .java extension.

displayed the text "Hello World" on the screen. This was accomplished using a print statement:

```
System.out.println("Hello World");
```

System is a built-in Java class that contains useful tools for our programs.

out is short for "output".

println is short for "print line".

System.out.print() prints all statements on one line there are no new line.

Short comments use the single-line syntax: //

Multi line comments use the multi-line syntax: /* and */.

Another type of commenting option is the Javadoc comment which is represented by /** and */. Javadoc comments are used to create documentation for APIs.

Compile the java file in the terminal by typing (javac filename.java)

Java is a case-sensitive language. Case sensitivity means that syntax, the words our computer understands, must match the case.

Data types

- Int: used for store whole number (positive numbers, negative numbers, and zero) (values between -2,147,483,648 and 2,147,483,647)
- Double: hold decimals as well as very large and very small numbers. (The maximum value is 1.797,693,134,862,315,7 E+308. The minimum value is 4.9 E-324,)
- Boolean: hold one of two values: true or false.
- Char: hold only one character. (It must be surrounded by single quotes, ')
- String: hold sequences of characters. enclosed in double-quotes ("").
  There are three escape sequences to be aware of for string:
  1. The \" escape sequence allows us to add quotation marks " to a String value.
  2. Using the \\ escape sequence allows us to place backslashes in our String text.
  3. place a \n escape sequence in a String, the compiler will output a new line of text.

Note: Variable names of only one word are spelled in all lowercase letters. Variable names of more than one word have the first letter lowercase while the beginning letter of each subsequent word is capitalized. This style of capitalization is called camelCase.

Note: A variable starts with a valid letter, or a $, or a _. No other symbols or numbers can begin with a variable name. 1stPlace and *Gazer are not valid variable names.

The order of operations: parentheses -> exponents -> multiplication, division, modulo -> addition, subtraction

Compound assignment operators perform an arithmetic operation on a variable and then reassign its value. Compound assignment operators for all of the arithmetic operators we've covered:

- Addition (+=)
- Subtraction (-=)
- Multiplication (*=)
- Division (/=)
- Modulo (%=)

equals() for comparing Strings and other objects

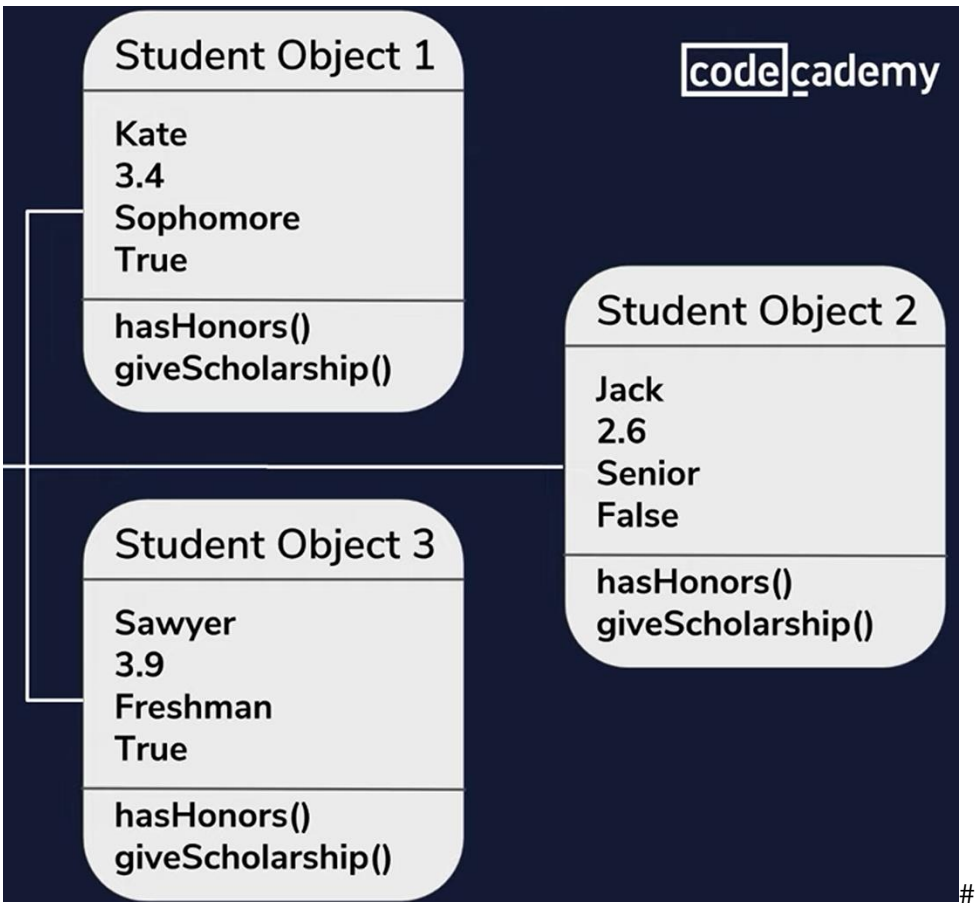
Note: To declare a variable with a value that cannot be manipulated, we need to use the final keyword.

```
final int yearBorn = 1968;
```

When we declare a variable using final, the value cannot be changed; any attempts at doing so will cause an error to occur

Create data type

To create a new custom data type we can use classes and identify the attribute and functions of the new data type.


Student Object 1
Kate
3.4
Sophomore
True
hasHonors()
giveScholarship()
codecademy
Student Object 2
Jack
2.6
Senior
False
hasHonors()
giveScholarship()
Student Object 3
Sawyer
3.9
Freshman
True
hasHonors()
giveScholarship()
#


Lightbulb
isOn (boolean)
watts (number)
turnOn() (func)
Microphone
color (string)
isAnalog (boolean)
record() (func)
Animal
color (string)
age (number)
speak() (func)
Date
month (number)
year (number)
toString() (func)

INTRODUCTION TO CLASSES

A class is a template for creating objects in Java. A class outlines the necessary components and how they interact with each other. Example:

```
public class Car {
  // Empty Java Class
}
```

constructor is a special type of method defined within the class, used to initialize fields when an instance of the class is created.

Note: The name of the constructor method must be the same as the class itself.

```
public class Car {

 // Constructor
 public Car() {

   // instructions for creating a Car instance
 }
}
```

Create an instances of the constructor

```
Car ferrari = new Car();
```

If we print the value of the variable ferrari we would see its memory address: Car@76ed5528 In the above example, our variable ferrari is declared as a reference data type rather than with a primitive data type like int or boolean. This means that the variable holds a reference to the memory address of an instance. During its declaration, we specify the class name as the variable's type, which in this case is Car.

If we use a special value, null, we can initialize a reference-type variable without giving it a reference. If we were to assign null to an object, it would have a void reference because null has no value.

```
Car thunderBird = new Car();

System.out.println(thunderBird); // Prints:
Car@76ed5528

thunderBird = null; // change value to null

System.out.println(thunderBird); // Prints: null
```

instance variables are often characterized by their "has-a" relationship with the object. each object created from the class will have its own copy of these variables.

```
public class Dog{

  public String name;
  public String breed;
  public int weight;
  public Dog(){
    //DO NOT WRITE ANYTHING HERE!!
  }
}
```

These fields can be set in the following three ways:

- If they are public, they can be set like this instanceName.fieldName = someValue;
- They can be set by class methods.
- They can be set by the constructor method

parameters are placeholders that we can use to pass information to a method.

```
public class Store{
  public String productType;
  public Store(String product){
    productType =  product;
  }
```

Note: A class can have multiple constructors. We can differentiate them based on their parameters. The signature helps the compiler to differentiate between different methods.

Note: An argument refers to the actual values passed during the method call while a parameter refers to the variables declared in the method signature.

Methods are repeatable, modular blocks of code used to accomplish specific tasks.

In order to call a non-static method, we must call the method on the object we created.

```
Object myObject = new Object("red");
Object.methodName();
```

Note: code generally runs in a top-down order where code execution starts at the top of a program and ends at the bottom of a program; however, methods are ignored by the compiler unless they are being called.

Note: We mark the domain of this task using curly braces: {, and }. Everything inside the curly braces is part of the task. This domain is called the scope of a method.

toString() method can return a String that will print when we print the object

```
String object;
public String toString(){
      return "This is a " + object + " car!";
  }
```

When printing an instance of the constructor the result will be memory position.

```
//instance of the constructor
Store cookieShop = new Store("Cookies", 5);
//printing the instance
System.out.println(cookieShop);
//result: Store@7ad041f3
```

To make it print a useful text we use toString() method. When we define a toString() method for a class, we can return a String that will print when we print the object(instance).

CONDITIONALS AND CONTROL FLOW

- If-Then-Else

```
if (True) {

  // Enroll in course

} else {

  // Enroll in prerequisite

}
```

- If-Then-Else-If

```
else if (course.equals("Theatre")) {

  // Enroll in Theatre course

}
```

Note: When we implement nested conditional statements, the outer statement is evaluated first. If the outer condition is true, then the inner, nested statement is evaluated.

- Switch Statement: check a given value against any number of conditions and run the code block where there is a match.

```
String course = "History";

switch (course) {
  case "Algebra":
    // Enroll in Algebra
    break;
  case "Biology":
    // Enroll in Biology
    break;
  case "History":
    // Enroll in History
    break;
  case "Theatre":
    // Enroll in Theatre
    break;
  default:
    System.out.println("Course not found");
}
```

Conditional Operators

- AND operator (&&): used when multiple conditions are true.
- OR operator (||): used when at least one of two conditions are true.
- produce the opposite value, where true becomes false and false becomes true, with the NOT operator: !

Combining Conditional Operators

```
boolean foo = true && !(false || !true)
```

The order of evaluation when it comes to conditional operators is as follows:

- Conditions placed in parentheses - ()
- NOT - !
- AND - &&
- OR - ||