Java Programming in VS Code

Neba Nfonsang, 2021

University of Denver

Setting up VS Code for Java development

- You can either install Coding Pack for Java which includes:
  - ☐ VS Code
  - ☐ Java Development Kit (JDK)
  - ☐ Essential Java extensions
- Windows: https://aka.ms/vscode-java-installer-win
- MacOS: https://aka.ms/vscode-java-installer-mac

- If you already have VS code, you can install the **Java Extension Pack** instead, which includes:
  - ☐ Language Support for Java(TM) by Red Hat
  - ☐ Debugger for Java
  - ☐ Java Test Runner
  - ☐ Maven for Java
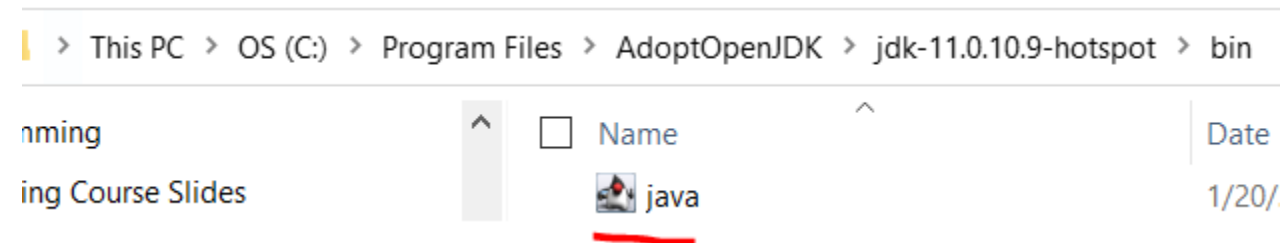 - ☐ Project Manager for Java

Java Extension Pack

- Go to the command Palette using CTRL + SHIFT + P

- Then search **Java Extension Guide.**

- You would see all the extensions installed that come with the Java Extension Pack.

- To run VS Code for Java, Java SE 11 or above version is required;
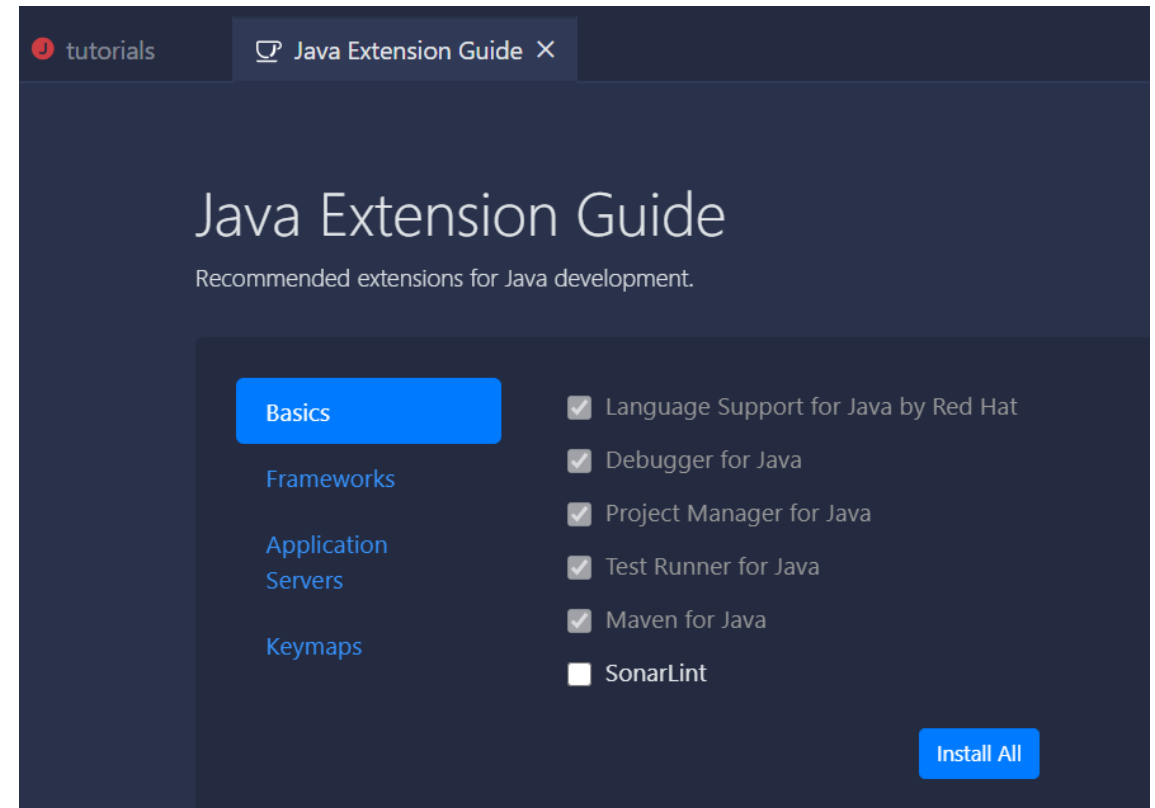
JRE and JDK

- Java run environment (JRE) is used to run java applications
- Java development kit (JDK) is used to build and develop java programs.
- You can go to the command palette and find **Java Run Time** and scroll down to download an open JDK which has the java.exe

> This PC > OS (C:) > Program Files > AdoptOpenJDK > jdk-11.0.10.9-hotspot > bin

nming

ing Course Slides
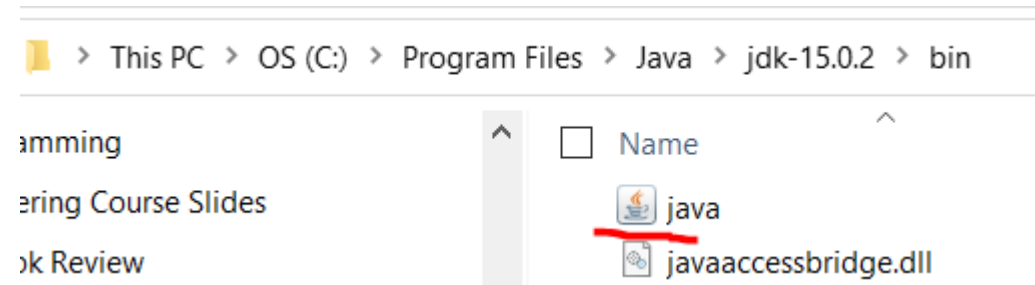
Name

☒ java

Date

1/20/

Download and Install JDK from VS Code

- Use the command palette, find the **Configure Java Runtime** and scroll down and download JDK and install it on your computer
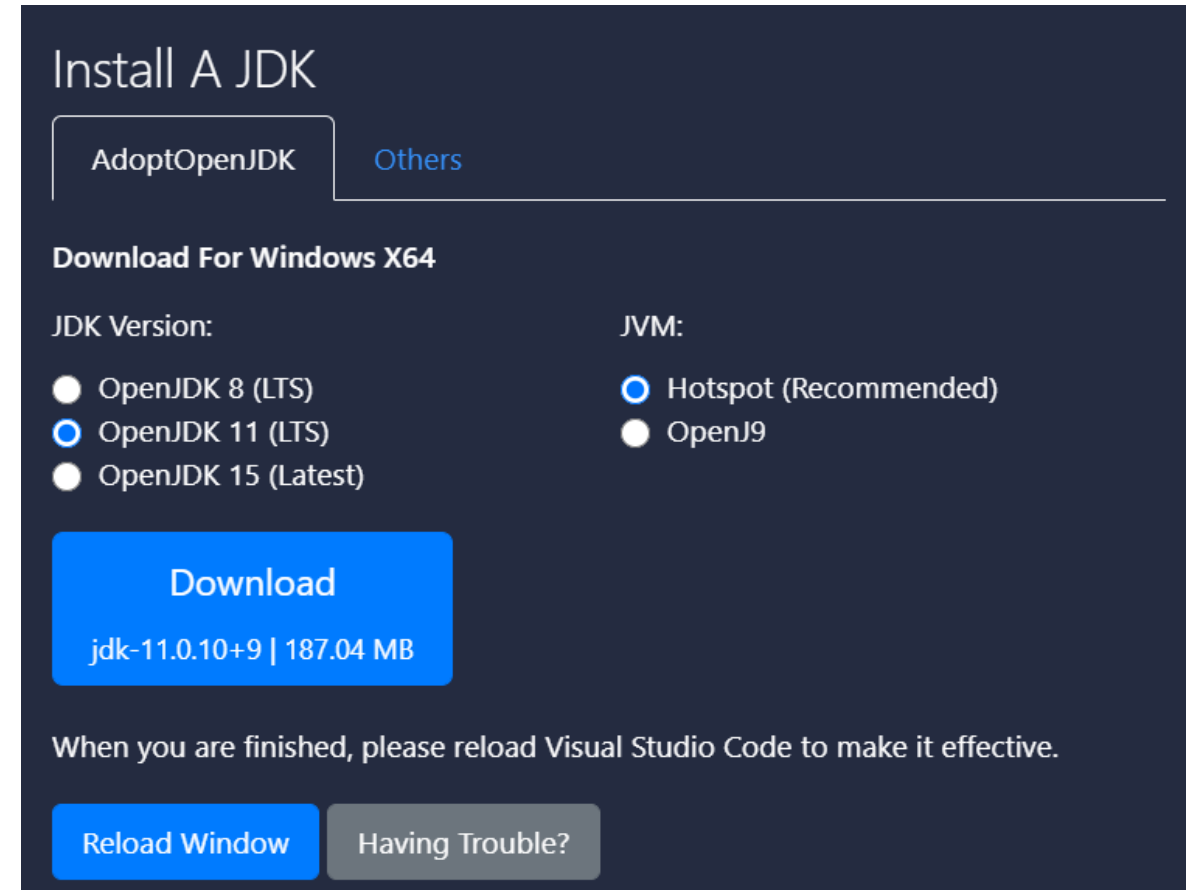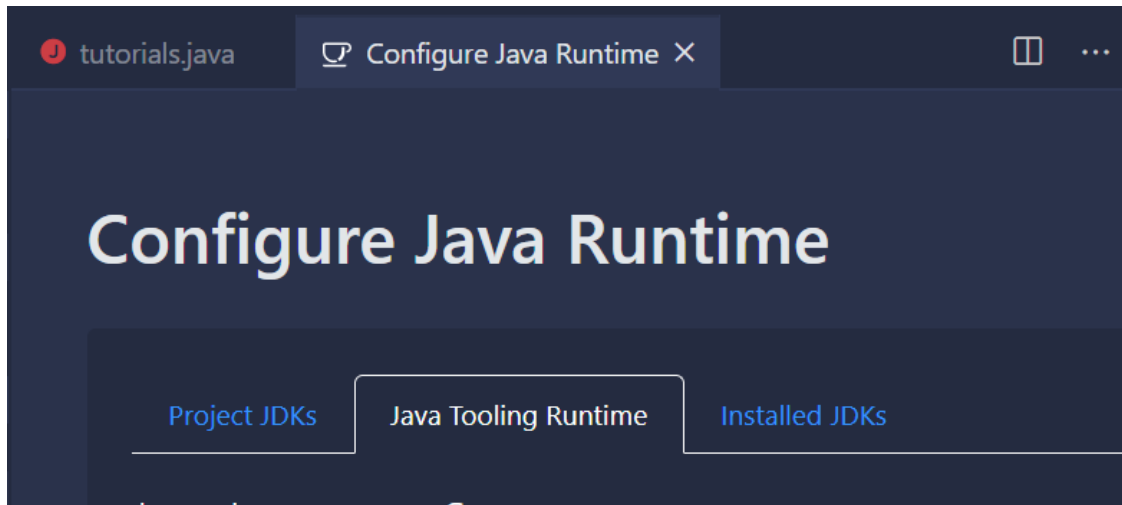
Download and Install Java SE

- JDK, JRE and other tools for creating Java programs.
- Java SE contains JDK + JRE
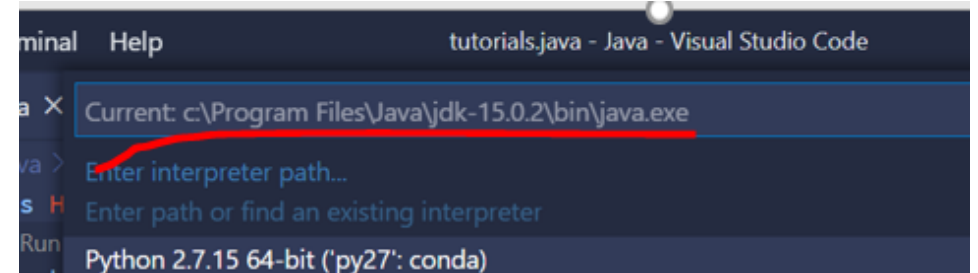- Alternative to the open JDK download and install the latest version of the Java SE to have access to the interpreter, java.exe https://www.oracle.com/java/technologies/javase-jdk15-downloads.html

> This PC > OS (C:) > Program Files > Java > jdk-15.0.2 > bin

amming

□ Name

ering Course Slides

🍵 java

ok Review

javaaccessbridge.dll

Before running a java file, make sure that the path to java interpreter is selected (you can use the status bar)

⚠ Select Python Interpreter   ⊗ 0 ⚠ 1

Create Java File

- Create a Java file that ends with a file extension, .java
- Make sure to select the java interpreter that you downloaded with the Java SE.
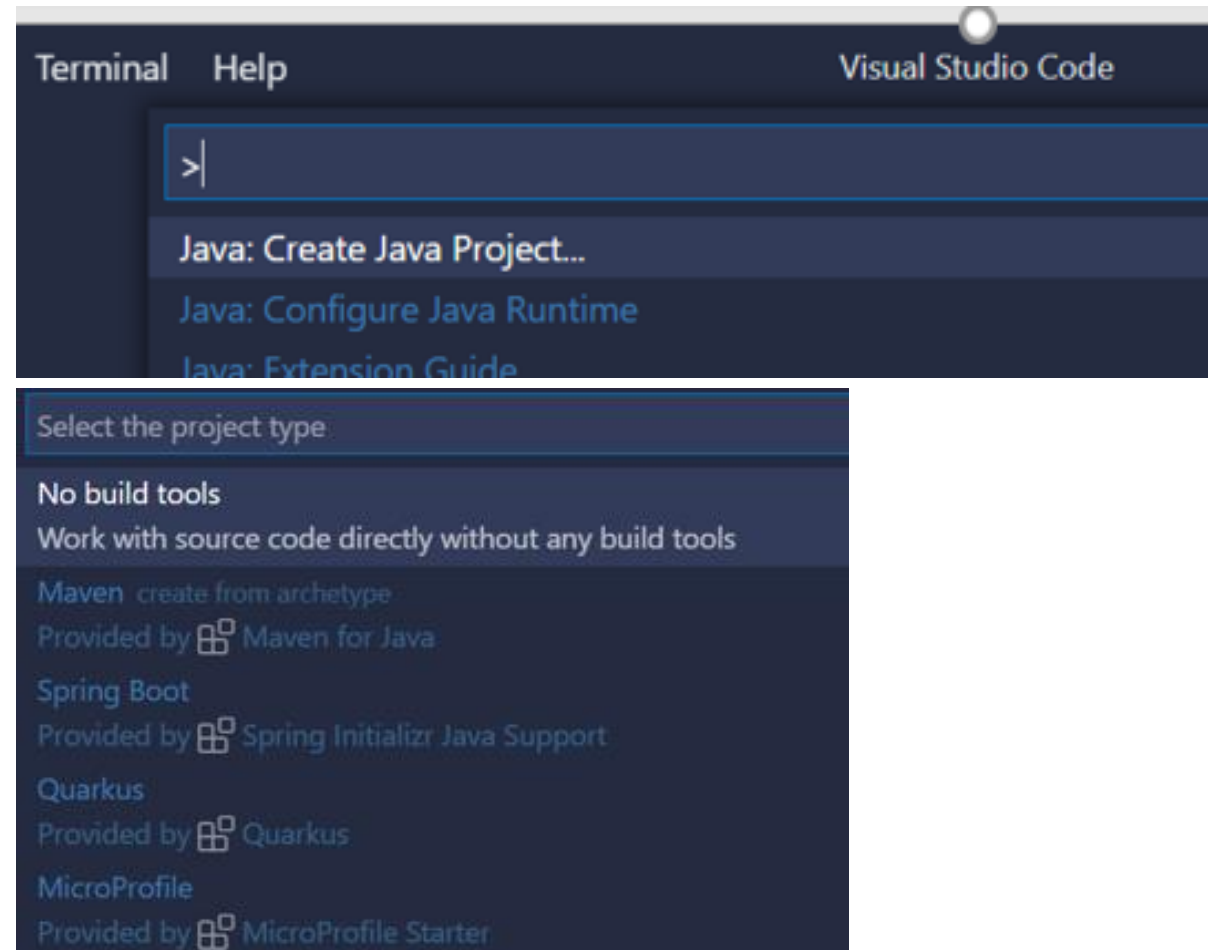


```java
public class App {
    Run | Debug
    public static void main(String[] args) {
        System.out.println("Hello World");
    }
}
```

Java Syntax

- In Java, every line of code to be run must be written inside a class

- The class name (App in this case) must match the file name.

- The main() method, `public static void main(String[] args)`, is required, it is used to execute any code written inside that method.

- That is, the main() method executes your Java program.

```
App.java   ×

src >  App.java > ...
  1    public class App {
         Run | Debug
  2        public static void main(String[] args) {
  3            System.out.println("Hello World");
  4        }
  5    }
  6
```

Java's print method `.println()` is used to print a value.
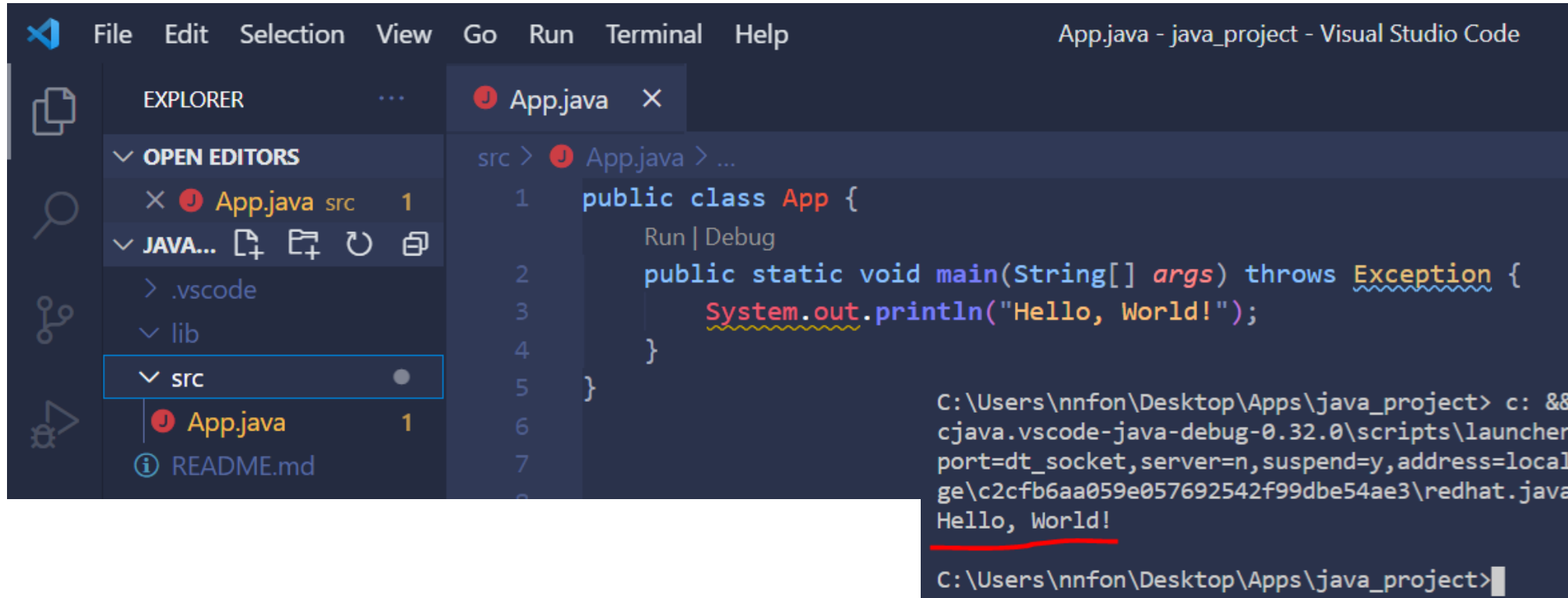`{ }` indicates the beginning and end of a code block

Create a Java Project

- To create a java project, use the command palette and select Create Java Project.

- Then select the build tools to use with the project.

- For now, we will select the **No build tools** option.

- You will be prompted to select project location and to provide the name of the project.

Java Project

The java project is created with some initial folders and files including the source code folder and App file

Data Types in Java

Data Type	Size	Description
byte	1 byte	Stores whole numbers from -128 to 127
short	2 bytes	Stores whole numbers from -32,768 to 32,767
int	4 bytes	Stores whole numbers from -2,147,483,648 to 2,147,483,647
long	8 bytes	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4 bytes	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits
double	8 bytes	Stores fractional numbers. Sufficient for storing 15 decimal digits
boolean	1 bit	Stores true or false values
char	2 bytes	Stores a single character/letter or ASCII values

https://www.w3schools.com/java/java_data_types.asp

Data Types

- The main method is required in Java as this is the entry point into the program.
- We need to start writing the program after this entry point.
- Note that a comment is added in Java using two forward slaphes //

```java
public class App {
    Run | Debug
    public static void main(String[] args) {
        byte b1 = 123;
        short sh1 = 7654;
        int num1 = 456789;
        long l1 = 3456789;
        float f1 = 12.5678f;
        double d1 = 234.456789;
        char ch1 = 'A';
        String str1 = "qwerty12345";
        boolean b2 = true;
        System.out.println(num1);
    }
}
```

Type casting

- Type casting is the conversion of one primitive type to another. This happens when we assign the value of one primitive type to another.

- **Widening Casting** (automatically) - converting a smaller type to a larger type size
byte -> short -> char -> int -> long -> float -> double

Type Casting

- **Narrowing Casting** (manually) - converting a larger type to a smaller size type double -> float -> long -> int -> char -> short -> byte

```java
public class App {

    public static void main(String[] args) {
        float length = 25f;
        int height = (int) length; // manually cast int to float
        System.out.println(length); //outputs 25
        System.out.println(height); //outputs 25.0
    }
}
```

Java Operators

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
++	Increment	++x
--	Decrement	--x

Arithmetic operators

Java Operators

Operator	Name	Example
==	Equal to	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Comparison operators

Operator	Name	Example
&&	Logical and	x < 5 && x < 10
||	Logical or	x < 5 || x < 4
!	Logical not	!(x < 5 && x < 10)

Logical operators

Examples of Operators

```java
public class App {
    Run | Debug
    public static void main(String[] args) {
        float length = 25f;
        ++ length; // increase value by 1
        System.out.println(length); //outputs 26

    }
}
```

```java
public class App {
    Run | Debug
    public static void main(String[] args) {
        float length = 25f;
        float width = 10;
        float area = length*width;
        System.out.println(area); //outputs 250

    }
}
```

```java
public class App {
    Run | Debug
    public static void main(String[] args) {
        float length = 25f;
        length += 10; // increase value by 10
        System.out.println(length); //outputs 35

    }
}
```

```java
public class App {
    Run | Debug
    public static void main(String[] args) {
        float length = 25f;
        float width = 10;
        System.out.println(length > width); //outputs true

    }
}
```

String Methods

- `.toUpperCase()` //converts to upper case
- `.toLowerCase()` // converts to lowercase
- `.length()` //prints the length of the sting
- `.indexOf()` // prints the position of the first occurrence

```java
public class App {
    Run | Debug
    public static void main(String[] args) {
        String mystring = "Hello, how are you today?";

        System.out.println(mystring.toUpperCase());
        //outputs HELLO, HOW ARE YOU TODAY?

    }
}
```

String Concatenation

- "john" + "is a good boy"
- String name = "john"
- String text = "is a good boy"
- String sentence = name + text

Or

- name.concat(text)

```java
public class App {
    Run | Debug
    public static void main(String[] args) {
        String mystring = "Hello, how are you today?";
        String answer = " I am doing good!";

        System.out.println(mystring + answer);
        //outputs HELLO, HOW ARE YOU TODAY?

    }
}
```

Escape Character

```java
public class App {
    Run | Debug
    public static void main(String[] args) {
        String text1 = "It\'s ok";
        String text2 = "It is called, \"encapsulation\"";

        System.out.println(text1);
        System.out.println(text2);
    }
}
```

```
It's ok
It is called, "encapsulation"
```

Escape character	Result	Description
\'	'	Single quote
\"	"	Double quote
\\	\	Backslash

Other escape characters
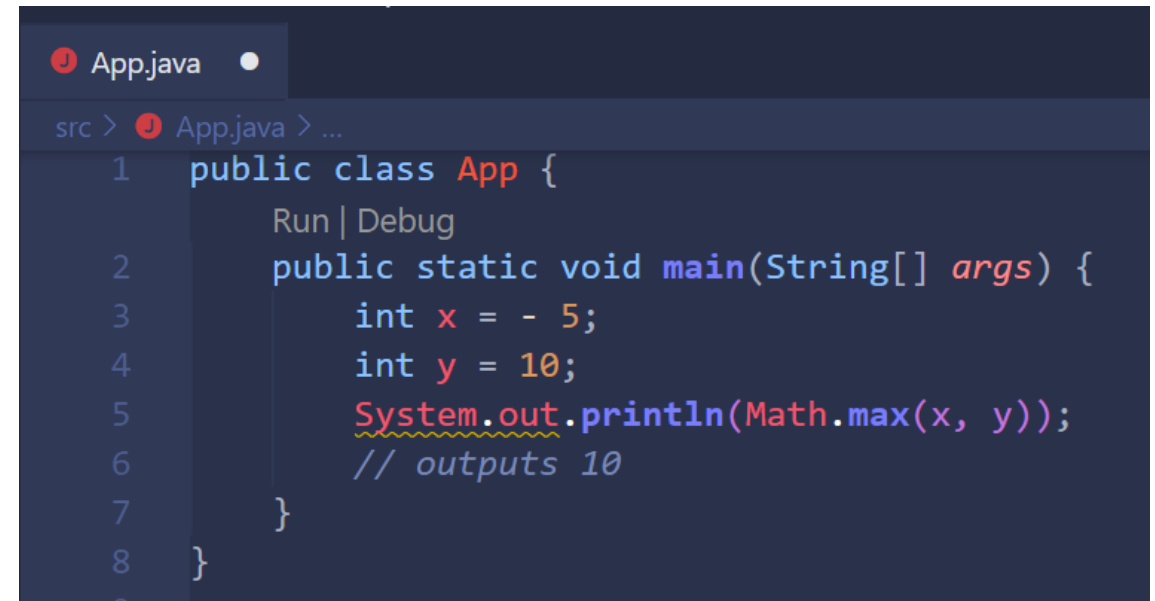
\n ➜ new line
\t ➜ tab
\b ➜ backspace

Math Methods

- `int x = -5;`
- `int y = 10;`
- `Math.max(x, y);`
- `Math.min(x, y);`
- `Math.abs(x);`
- `Math.sqrt(x);`
- `Math.random()` // generate a random number between 0 and 1

```java
public class App {
    Run | Debug
    public static void main(String[] args) {
        int x = - 5;
        int y = 10;
        System.out.println(Math.max(x, y));
        // outputs 10
    }
}
```

Conditions: if statements

- `if (`*`condition`*`) { // block of code to be executed if the condition is true }`

Condition: if/else statement

```
if (condition) { //
    block of code to be
    executed if the
condition is true
} else { //
    block of code
to be executed if the
condition is false
}
```

Condition: if/else if/else statement

```
if (condition) { //
        block of code to be
        executed when the if
    condition is true
} else if { //
        block of code
    to be executed when the if
    condition is false
} else { //
        block of code
    to be executed when the else if
    condition is false
}
```

Switch statement

- Switch statement is used to switch between case

```
int num = 1;
switch (num) {
    case 1:
        System.out.println("Male");
        break;
    case 2:
        System.out.println("Female");
        break;
}
```

Switch Statement Example

```java
public class App {
    Run | Debug
    public static void main(String[] args) {
        int num = 1;
        switch (num) {
            case 1:
                System.out.println("Male");
                break;
            case 2:
                System.out.println("Female");
                break;
        }

    }
}
    // outputs Male
```

```java
public class App {
    Run | Debug
    public static void main(String[] args) {
        int num = 5;
        switch (num) {
            case 1:
                System.out.println("Toyota");
                break;
            case 2:
                System.out.println("Honda");
                break;
            default:
                System.out.println("Other");
                break;
        }
    }
}
    // outputs Other
```

We can also include a default statement

While statement

```
while (condition) {
    // code block to be
    executed
}
```

For Loop

```
for (statement 1; statement 2;statement 3) { //
     code block to be executed }
```

- **Statement 1** is executed (one time) before the execution of the code block.

- **Statement 2** defines the condition for executing the code block.

- **Statement 3** is executed (every time) after the code block has been executed.

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

For-Each Loop

- If we have elements inside an array, we can use "for-each loop" to loop through each element and execute some code block.

```java
// general syntax for looping through an array
for (type variable : arrayname) { ... }
// looping through an array
String[] names= {"Jane", "James", "Jack", "Jill"};
for (String name : names) {       System.out.println(name);
}
```

Break Statement

```java
for (int i = 0; i < 10; i++) {
    if (i == 6) {
        break;
    }
    System.out.println(i);
}
```

Break out of the loop or stop executing the loop when I is 6

Break Statement Example

```java
public class App {
    Run | Debug
    public static void main(String[] args) {
        for (int i = 0; i < 10; i++) {
            if (i == 6) {
                break;
            }
            System.out.println(i);
        }
    }
}
// outputs 1 2 3 4 5
```

Continue Statement

```java
for (int i = 0; i < 10; i++) {
    if (i == 6) {
        continue;
    }
    System.out.println(i);
}
```

When i is 6, don't execute the code block below, instead run the for loop header line again and continue executing code body until condition is met.

Continue Statement Example

```java
public class App {
    Run | Debug
    public static void main(String[] args) {
        for (int i = 0; i < 10; i++) {
            if (i == 6) {
                continue;
            }
            System.out.println(i);
        }
    }
}
    // outputs 1 2 3 4 5 7 8 9 10
```

Java Arrays

```java
String[] names = {"Jane", "James", "Jack", "Jill"};

int[] nums = {100, 200, 300, 400 , 500}


names[0] = "Jacky" //change the name in the first position

System.out.println(names.length) //print the length of the array
```

Looping through Arrays

```java
// looping through an array
String[] names= {"Jane", "James", "Jack", "Jill"};
for (String name : names) {
    System.out.println(name);
}


// another example of how to loop through an array
String[] names= {"Jane", "James", "Jack", "Jill"};
for (int i = 0; I < names.length < 10; i++) {
    System.out.println(name);
}
```

Multidimensional Array

```java
int[] nums = {{100, 200}, {300, 400}, {500, 600}};
int[] ind10 = nums[1][0]
System.out.println(ind10); // outputs 300
```

Note that Java starts counting from zero, so indexinf starts from zero

Nested for loop

```java
int[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };
for (int i = 0; i < myNumbers.length; ++i) {
    for(int j = 0; j < myNumbers[i].length; ++j) {
        System.out.println(myNumbers[i][j]);
    }
}
```

Java Methods

- Methods are functions inside a class that take input arguments and perform a task or process some data.
- Java has builtin methods such as Math.min()
- You can create your own methods
- The name of the method here is `myMethod`

- A method is a block of code that starts and ends with curly brackets
- The methods are executed when called inside the main method.

```java
public class App{

    static void myMethod() {
        // code to be executed
    }

}
```

Java Methods

- **Static** means the method can be access at the class level [App.myMethod()] or belongs to the class.

- That means we do not need to create an instance before we access the method.

- Void means the method does not have to return a value.

```java
public class App{
    static void myMethod() {
        // code to be executed
    }
}
```

Note: An object could be created a follows and the method access but we don't need to do this when **static** is used.
App myObject = new App();
 myObject.myMethod();

Java Methods

```java
public class App{

    static void myMethod() {

        // code to be executed

    }

    public static void main(String[] args) {

        myMethod();

    }

}
```

To execute a Java method, the method needs to be called inside the main method.

Java Method Example

```java
public class App {
    static void sayHello(){
        System.out.println("Hello");
    }
    Run | Debug
    public static void main(String[] args) {
        sayHello();
    }
}
// Outputs "Hello"
```

The method here is called **sayHello()** and it outputs "Hello" when run

A Java Method with Parameters

```java
public class App {
    static void sayHello(String name){
        System.out.println("Hello " + name);
    }
    Run | Debug
    public static void main(String[] args) {
      sayHello("John");
      sayHello("Peter");
    }
  }
// Outputs "Hello John" "Hello Peter"
```

Parameter are variables that are assign values or arguments when the method is run.

So the parameter type has to declared when creating the method.

A Method with Multiple Parameters

```java
public class App {
    static void sayHello(String name, int age){
        System.out.println("Hello " + "my name is " +
        name + ", I am " + age);
    }
    Run | Debug
    public static void main(String[] args) {
      sayHello("John", 30);
      sayHello("Peter", 32);
    }
  }
// Outputs    Hello my name is John, I am 30
            Hello my name is Peter, I am 32
```

A Methods with a Return Statement

- A method with a return statement does not have the void.

- The type of value to be returned needs to be declared before the method name.

```java
App.java   ✕

src >  App.java > ...
1    public class App {
2        static int add(int num1, int num2){
3            int total = num1 + num2;
4            return total;
5        }
     Run | Debug
6        public static void main(String[] args) {
7            System.out.println(add(20, 30));
8            System.out.println(add(100, 100));
9        }
10
11   }
12   // Outputs
                        50
                        200
```

A Method with Conditions

```java
public class App {
    static void checkNumber(int num){
        if (num==0){
            System.out.println("This number is zero");
        } else if (num>0){
            System.out.println("This number is positive");

        } else {
            System.out.println("This number is negative");
        }
    }
    Run | Debug
    public static void main(String[] args) {
        checkNumber(0);
        checkNumber(-5);
        checkNumber(2);
    }
}
// Outputs
```

```
This number is zero
This number is negative
This number is positive
```

Method Overloading

- If a method does the same thing but takes different types as input arguments, we can overload the method by creating it with same name several times to return different types.

```java
App.java  ×

src > App.java > ...
1   public class App {
2       static int add(int num1, int num2){
3           return num1 + num2;
4       }
5       static float add(float num1, float num2){
6           return num1 + num2;
7       }
8
    Run | Debug
9       public static void main(String[] args) {
10          System.out.println(add(5, 5));
11          System.out.println(add(5.1f, 5.2f));
12      }
13  }
14  // Outputs
            10
            10.299999
```

Variable Scope

- Variables are only accessible within the scope or region where they declared.

- A value of a variable cannot be used before it is declared.

- The variable can be used block after it is declared not before declaration or outside the block.


```java
public class Main {
    Run | Debug
    public static void main(String[] args) {
        // Code here CANNOT use x
        { // This is a block
            // Code here CANNOT use x
            int x = 20;
            // Code here CAN use x
            System.out.println(x);
        } // The block ends here
        // Code here CANNOT use x
    }
}
```

Java Object Oriented Programming

OOP in Java

- Java is an object-oriented programming (OOP) language.
- OOP promotes code reuse.
- OOP involves the use of classes to create objects.
- You could have a class of fruits and the specific objects of could be fruits like a mangoes, orange, apple etc.

- A class is a template, blueprint for constructing an object.
- The keyword class is used to create a class.
- A classes and objects have associated attributes and methods.
- Class names should always start with upper case and file names should always match class name.

Create a Class

```java
public class Employee {
    String city = "Denver";
    Run | Debug
    public static void main(String[] args){

        Employee emp1 = new Employee(); // create an object, emp1
        Employee emp2 = new Employee(); // create an object, emp1
        System.out.println(emp1.city);
        System.out.println(emp2.city);
    }
}
// outputs     Denver
             Denver
```

Employee.java

src > Employee.java > ...

city is a class attribute, all employees will have that same attribute

Modified Attribute

```java
public class Employee {
    String city = "Denver";
    Run | Debug
    public static void main(String[] args){

        Employee emp1 = new Employee(); // create an object, emp1
        emp1.city = "Aurora";
        Employee emp2 = new Employee(); // create an object, emp1
        emp2.city = "Lakewood";
        System.out.println(emp1.city);
        System.out.println(emp2.city);
    }
}
// outputs
Aurora
Lakewood
```

We can override the class attribute and assign different values to specific objects after creating the object

To Avoid Overriding Class Attribute

```java
Employee.java

src > Employee.java > ...
1   public class Employee {
2       final String city = "Denver";
        Run | Debug
3       public static void main(String[] args){
4
5           Employee emp1 = new Employee(); // create an object, emp1
6           emp1.city = "Aurora";
7           Employee emp2 = new Employee(); // create an object, emp1
8           emp2.city = "Lakewood";
9           System.out.println(emp1.city);
10          System.out.println(emp2.city);
11      }
12  }
13  // outputs
            The final field Employee.city cannot be assigned
            The final field Employee.city cannot be assigned
```

Include the keyword **final** when creating the class attribute to avoid overriding it

Multiple Attributes

```java
Employee.java

src > Employee.java > ...
1   public class Employee {
2       String city;
3       String fname;
4       String lname;
        Run | Debug
5       public static void main(String[] args){
6
7           Employee emp1 = new Employee(); // create an object, emp1
8           emp1.city = "Aurora";
9           emp1.fname = "James";
10          emp1.lname = "Smith";
11          System.out.println(emp1.city);
12          System.out.println(emp1.fname);
13          System.out.println(emp1.lname);
14      }
15  }
16  // outputs    Aurora
                  James
                  Smith
```

Class Method (Static Method)

```java
Employee.java

src > Employee.java > ...
1   public class Employee {
2
3       // static method
4       static void getName(String fname, String Lname){
5           System.out.println(fname + " " + lname);
6       }
    Run | Debug
7       public static void main(String[] args){
8           getName("John", "Smith");
9
10      }
11  }
12  // outputs   John Smith
```

Static methods are methods that don't need an instantiated object to run. The do not belong to the object but to the class.

Public Method

```java
Employee.java ✕

src > Employee.java > ...
1    public class Employee {
2
3        // public method
4        public void companyName(){
5            System.out.println("University of Denver");
6        }
     Run | Debug
7        public static void main(String[] args){
8            Employee emp1 = new Employee();
9            emp1.companyName();
10
11       }
12   }
13   // outputs  University of Denver
```

Public methods can be accessed only through an object. These methods belong to the object not to the class.

The Constructor Method

The constructor method is used to initialize an object.

With the constructor method, it is easier to create ab object with several attributes.

```java
public class Employee {
    // attributes
    String fname;
    String lname;


    // create a constructor method
    public Employee(String firstName, String LastName){
        fname = firstName;
        lname = lastName;
    }


    // method
    public String getName(){
        return fname + " " + lname;

    }
Run | Debug
    public static void main(String[] args){
        Employee emp1 = new Employee("James", "Smith");
        System.out.println(emp1.getName());
    }
}
// outputs      James Smith
```

Encapsulation

- This is when we hide sensitive data from uses by making variables and attributes inside the class private.

- With encapsulation, we instead provide uses with:
  - ☐ **Set** methods to set or update attribute values
  - ☐ **Get** method to retrieve attribute values

this keyword is used to refer to the current object in a method or constructor. It is commonly used to distinguish between the class attribute and the parameter of the method especially when they have the same name

```java
Employee.java ✕

src  Employee.java  ...
1    public class Employee {
2        // attribute
3        String name;
4
5        // create a getter method
6        public String getName(){
7            return name;
8        }
9
10       // create a setter method
11       public void setName(String newName){
12           this.name = newName;
13
14       }
     Run | Debug
15       public static void main(String[] args){
16           Employee emp1 = new Employee();
17           emp1.setName("James Smith");
18           System.out.println(emp1.getName());
19       }
20   }
21   // outputs
```

Access Modifiers

- When a class is public, it can be accessed by other classes.

- When attributes/methods are public, they code can be accessed from other classes.

- When attributes/methods are made private, they can be accessed only within the class where they are declared.

```java
public class Employee {
    // attribute
    private String name;

    // create a getter method
    public String getName(){
        return name;
    }

    // create a setter method
    public void setName(String newName){
        this.name = newName;
    }

    public static void main(String[] args){
        Employee emp1 = new Employee();
        emp1.setName("James Smith");
        System.out.println(emp1.getName());

    }
}

// outputs
```

Public Attributes and Methods

```java
public class Employee {
    // attribute
    public String name;

    // create a getter method
    public String getName(){
        return name;
    }

    // create a setter method
    public void setName(String newName){
        this.name = newName;
    }
    Run | Debug
    public static void main(String[] args){
        Employee emp1 = new Employee();
        emp1.setName("James Smith");
        System.out.println(emp1.getName());

    }
}
```

```java
public class Main {


    Run | Debug
    public static void main(String[] args){
        Employee emp1 = new Employee();
        emp1.name = "Jack";
        // emp1.setName("James Smith");
        System.out.println(emp1.getName());
    }
}
```

PROBLEMS 2   DEBUG CONSOLE   OUTPUT   **TERMINAL**

Jack

C:\Users\nnfon\Desktop\Apps\java_project>

The **public** attribute in the employee class can be accessed directly in the Main class.

Public Attributes and Methods


```java
public class Employee {
    // attribute
    public String name;

    // create a getter method
    public String getName(){
        return name;
    }

    // create a setter method
    public void setName(String newName){
        this.name = newName;
    }

    Run | Debug
    public static void main(String[] args){
        Employee emp1 = new Employee();
        emp1.setName("James Smith");
        System.out.println(emp1.getName());
    }
}
```

```java
public class Main {

    Run | Debug
    public static void main(String[] args){
        Employee emp1 = new Employee();
        //emp1.name = "Jack";
        emp1.setName("James Smith");
        System.out.println(emp1.getName());
    }
}
```

```
PROBLEMS 2    DEBUG CONSOLE    OUTPUT    TERMINAL
ge\c2cfb6aa059e057692542f99dbe54ae3\redhat.java\jdt_ws\java_proje
James Smith

C:\Users\nnfon\Desktop\Apps\java_project>
```

The **public** method in the employee class can be accessed in the Main class

Private Attributes and Methods

```java
public class Employee {
    // attribute
    private String name;

    // create a getter method
    public String getName(){
        return name;
    }

    // create a setter method
    public void setName(String newName){
        this.name = newName;
    }
    Run | Debug
    public static void main(String[] args){
        Employee emp1 = new Employee();
        emp1.setName("James Smith");
        System.out.println(emp1.getName());

    }
}
```

```java
public class Main {


    Run | Debug
    public static void main(String[] args){
        Employee emp1 = new Employee();
        emp1.name = "Jack";
        //emp1.setName("James Smith");
        System.out.println(emp1.getName());
    }
}
```

PROBLEMS 2   DEBUG CONSOLE   OUTPUT   TERMINAL

The field Employee.name is not visible

at Main.main(Main.java:5)

C:\Users\nnfon\Desktop\Apps\java_project>

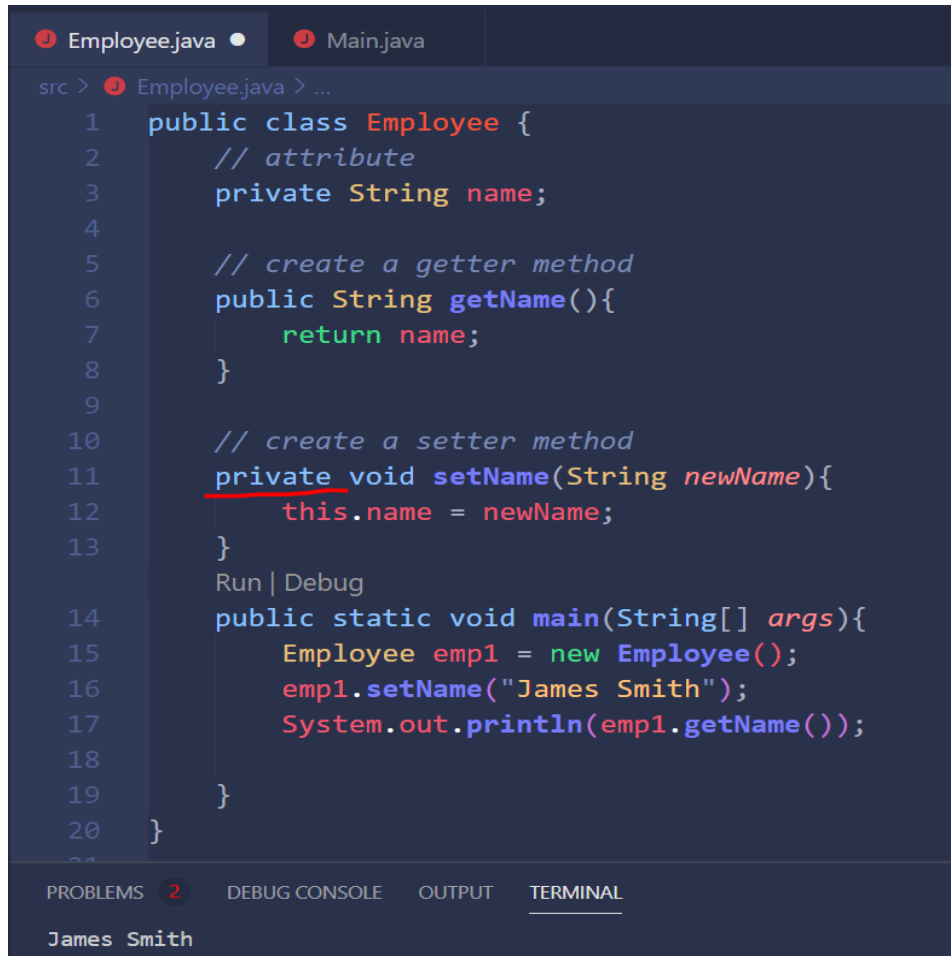The **private** attribute in the employee class cannot be accessed in the Main class.

Private Attributes and Methods

```java
public class Employee {
    // attribute
    private String name;

    // create a getter method
    public String getName(){
        return name;
    }

    // create a setter method
    public void setName(String newName){
        this.name = newName;
    }
    // Run | Debug
    public static void main(String[] args){
        Employee emp1 = new Employee();
        emp1.setName("James Smith");
        System.out.println(emp1.getName());

    }
}
```

```java
public class Main {


    // Run | Debug
    public static void main(String[] args){
        Employee emp1 = new Employee();
        //emp1.name = "Jack";
        emp1.setName("James Smith");
        System.out.println(emp1.getName());
    }
}
```

```
PROBLEMS  1    DEBUG CONSOLE    OUTPUT    TERMINAL


James Smith
```

As already seen, the private attribute could not be access directly through .name attribute but can be access through the setter and getter method.

Private Attributes and Methods

Private methods can be access only within the class where they were declared

Private class and methods cannot be access outside the class where they were declared