Introduction to Programming in Java

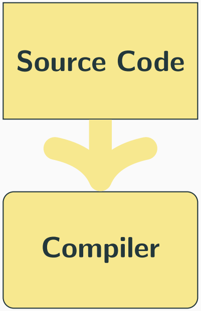
Shirley B. Chu

July 3, 2020

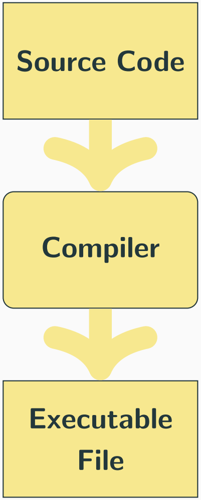
De La Salle University
College of Computer Studies

Compiled Languages

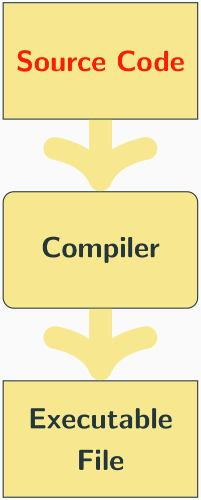
Compiled Languages



Compiled Languages

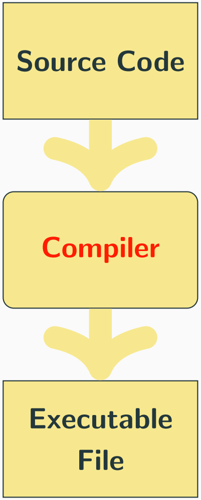


Compiled Languages



high-level programming languages,
e.g. C, C++, Pascal

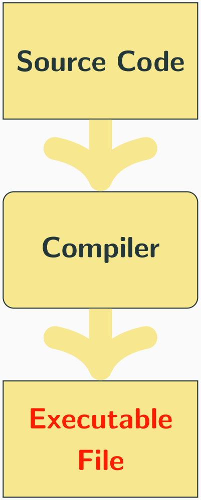
Compiled Languages



high-level programming languages,
e.g. C, C++, Pascal

A software program that translates high level language program into an executable machine language program.

Compiled Languages

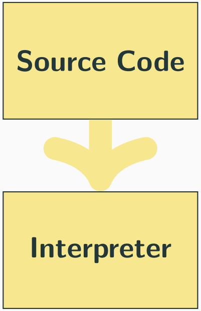


high-level programming languages,
e.g. C, C++, Pascal

A software program that translates high level language program into an executable machine language program.

may be executed many times but on one type of computer only

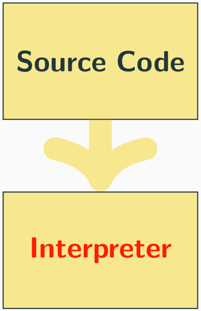
Interpreted Languages



high-level programming languages,
e.g. Java, Python

translates and executes one instruction at a time

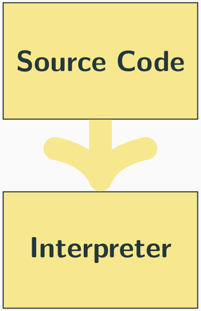
Interpreted Languages



high-level programming languages,
e.g. Java, Python

translates and executes one instruction at a time

Interpreted Languages

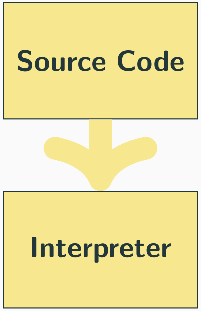


high-level programming languages,
e.g. Java, Python

translates and executes one instruction at a time

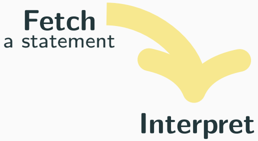
Fetch
a statement

Interpreted Languages

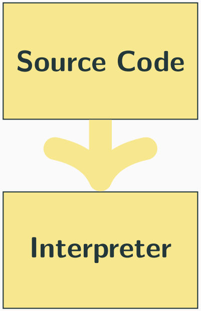


high-level programming languages,
e.g. Java, Python

translates and executes one instruction at a time

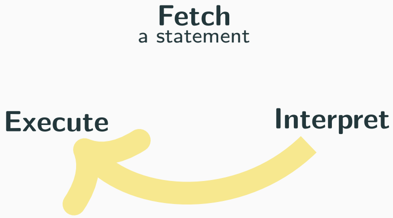


Interpreted Languages

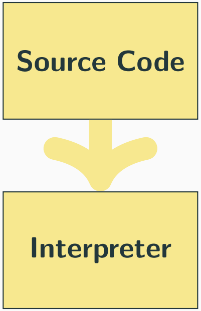


high-level programming languages,
e.g. Java, Python

translates and executes one instruction at a time

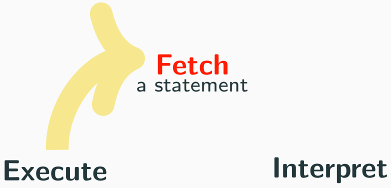


Interpreted Languages

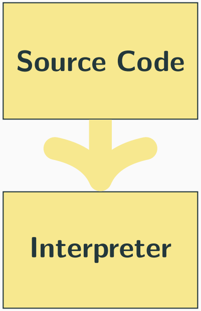


high-level programming languages,
e.g. Java, Python

translates and executes one instruction at a time

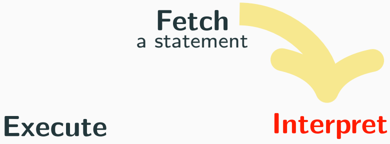


Interpreted Languages

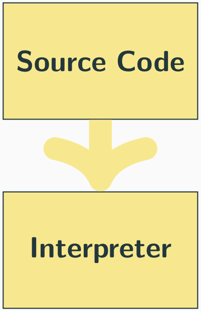


high-level programming languages,
e.g. Java, Python

translates and executes one instruction at a time

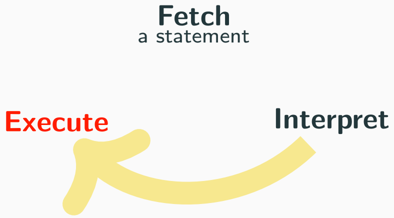


Interpreted Languages

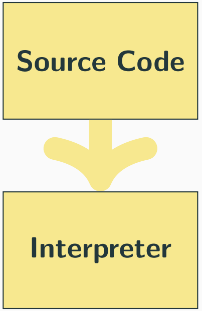


high-level programming languages,
e.g. Java, Python

translates and executes one instruction at a time

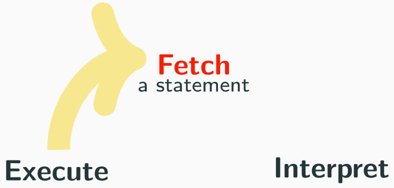


Interpreted Languages

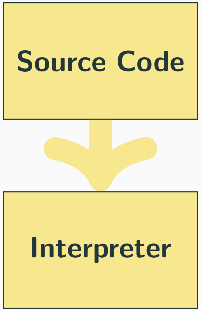


high-level programming languages,
e.g. Java, Python

translates and executes one instruction at a time

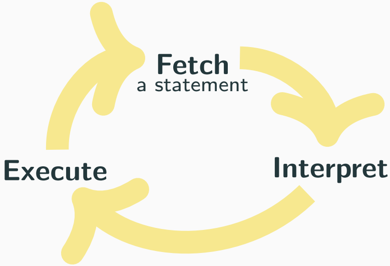


Interpreted Languages



high-level programming languages,
e.g. Java, Python

translates and executes one instruction at a time



Structure

- A Java application consists of **one or more** classes.
- All lines of code to be executed must be in a class.

Class Definition

```
public class SimpleApp
{
}
```

- Class name is `SimpleApp`
- Class names in Java starts with an uppercase letter.
- File name must be `SimpleApp.java`
- The keyword `class` is used to declare a new Java class.
- The keyword `public` is an access modifier. It indicates accessibility or scope.

Structure

- A Java application consists of **one or more** classes.
- All lines of code to be executed must be in a class.

Class Definition

```
public class SimpleApp
{
}
```

- Class name is `SimpleApp`
- Class names in Java starts with an uppercase letter.
- File name must be `SimpleApp.java`
- The keyword `class` is used to declare a new Java class.
- The keyword `public` is an access modifier. It indicates accessibility or scope.

Structure

- A Java application consists of **one or more** classes.
- All lines of code to be executed must be in a class.

Class Definition

```
public class SimpleApp
{
}
```

- Class name is `SimpleApp`
- Class names in Java starts with an uppercase letter.
- File name must be `SimpleApp.java`
- The keyword `class` is used to declare a new Java class.
- The keyword `public` is an access modifier. It indicates accessibility or scope.

Structure

- A Java application consists of **one or more** classes.
- All lines of code to be executed must be in a class.

Class Definition

```
public class SimpleApp
{
}
```

- Class name is `SimpleApp`
- Class names in Java starts with an uppercase letter.
- File name must be `SimpleApp.java`
- The keyword `class` is used to declare a new Java class.
- The keyword `public` is an access modifier. It indicates accessibility or scope.

Structure

- A Java application consists of **one or more** classes.
- All lines of code to be executed must be in a class.

Class Definition

```
public class SimpleApp
{
}
```

- Class name is `SimpleApp`
- Class names in Java starts with an uppercase letter.
- File name must be `SimpleApp.java`
- The keyword `class` is used to declare a new Java class.
- The keyword `public` is an access modifier. It indicates accessibility or scope.

Structure

- A Java application consists of **one or more** classes.
- All lines of code to be executed must be in a class.

Class Definition

```
public class SimpleApp
{
}
```

- Class name is `SimpleApp`
- Class names in Java starts with an uppercase letter.
- File name must be `SimpleApp.java`
- The keyword `class` is used to declare a new Java class.
- The keyword `public` is an access modifier. It indicates accessibility or scope.

Structure

- A Java application consists of **one or more** classes.
- All lines of code to be executed must be in a class.

Class Definition

```java
public class SimpleApp
{
}
```

- Class name is `SimpleApp`
- Class names in Java starts with an uppercase letter.
- File name must be `SimpleApp.java`
- The keyword `class` is used to declare a new Java class.
- The keyword `public` is an access modifier. It indicates accessibility or scope.

Structure

- A Java application consists of **one or more** classes.
- All lines of code to be executed must be in a class.

Class Definition

```
public class SimpleApp
{
}
```

- Class name is `SimpleApp`
- Class names in Java starts with an uppercase letter.
- File name must be `SimpleApp.java`
- The keyword `class` is used to declare a new Java class.
- The keyword `public` is an access modifier. It indicates accessibility or scope.

`main()` method

- The `main()` method (function) is the entry point into the Java application.

`main()` method

- The `main()` method (function) is the entry point into the Java application.
- Each java class can declare at most one `main()` method.

`main()` method

- The `main()` method (function) is the entry point into the Java application.
- Each java class can declare at most one `main()` method.
- The signature of this method is:

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

main() method

- The main() method (function) is the entry point into the Java application.
- Each java class can declare at most one main() method.
- The signature of this method is:

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

Class Definition

```
public class SimpleApp
{




}
```

`main()` method

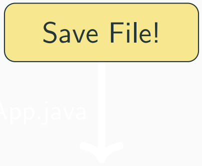
- The `main()` method (function) is the entry point into the Java application.
- Each java class can declare at most one `main()` method.
- The signature of this method is:

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

Class Definition

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

Compiling and Running Java Programs



Source Code: SimpleApp.java

Compiling and Running Java Programs

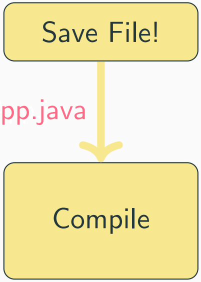
Source Code SimpleApp.java

Compiling and Running Java Programs

Save File!

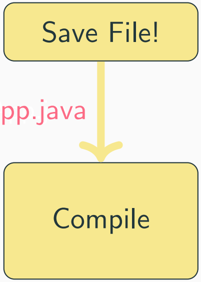
Source Code: SimpleApp.java

Compiling and Running Java Programs



Source Code: SimpleApp.java

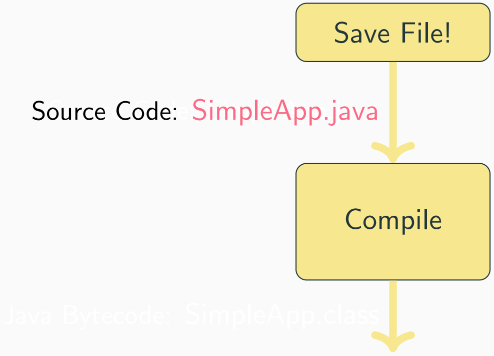
Compiling and Running Java Programs



Source Code: SimpleApp.java

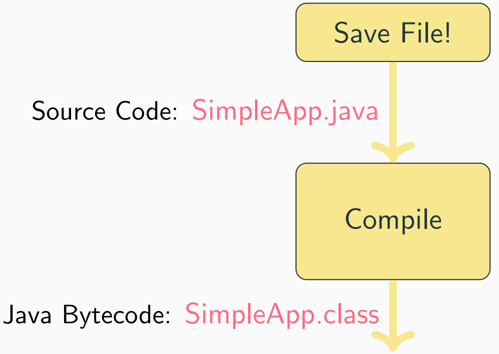
```
javac SimpleApp.java
```

Compiling and Running Java Programs



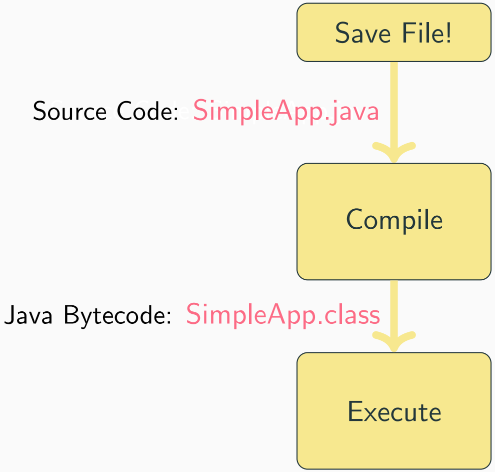
```
javac SimpleApp.java
```

Compiling and Running Java Programs



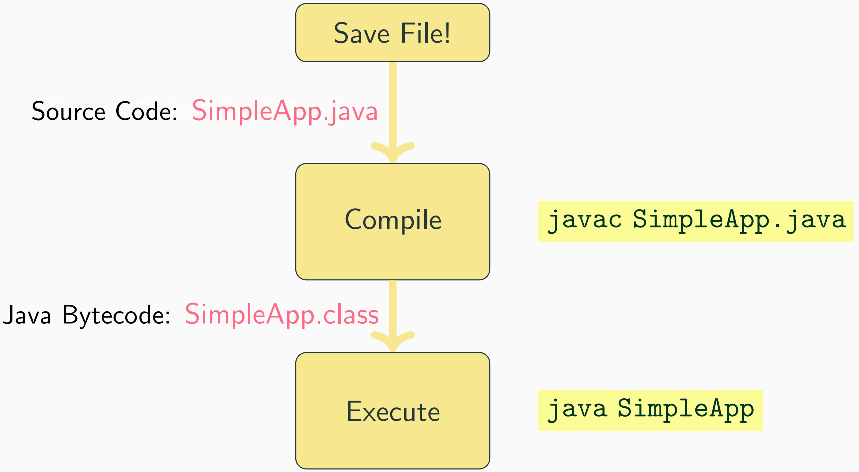
```
javac SimpleApp.java
```

Compiling and Running Java Programs



```
javac SimpleApp.java
```

Compiling and Running Java Programs



```
javac SimpleApp.java
```

```
java SimpleApp
```

Write Once Run Anywhere

- Java is an interpreted language.

Write Once Run Anywhere

- Java is an interpreted language.
- The Java Compiler translates the source code (`.java` file) into machine code (*bytecode*).

Write Once Run Anywhere

- Java is an interpreted language.
- The Java Compiler translates the source code (`.java` file) into machine code (*bytecode*).
- Java bytecode is the machine language for the Java Virtual Machine (JVM).

Write Once Run Anywhere

- Java is an interpreted language.
- The Java Compiler translates the source code (`.java` file) into machine code (*bytecode*).
- Java bytecode is the machine language for the Java Virtual Machine (JVM).

Write Once Run Anywhere

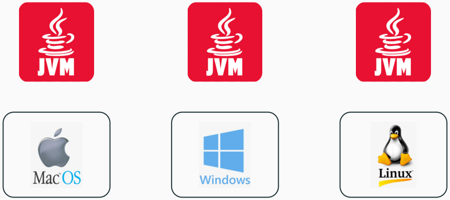
- Java is an interpreted language.
- The Java Compiler translates the source code (`.java` file) into machine code (*bytecode*).
- Java bytecode is the machine language for the Java Virtual Machine (JVM).

Write Once Run Anywhere

- Java is an interpreted language.
- The Java Compiler translates the source code (`.java` file) into machine code (*bytecode*).
- Java bytecode is the machine language for the Java Virtual Machine (JVM).

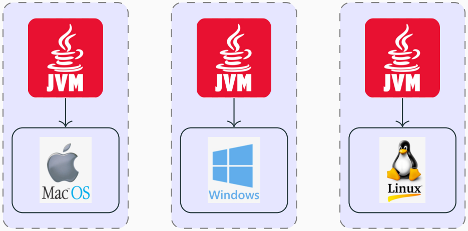
Write Once Run Anywhere

- Java is an interpreted language.
- The Java Compiler translates the source code (`.java` file) into machine code (*bytecode*).
- Java bytecode is the machine language for the Java Virtual Machine (JVM).



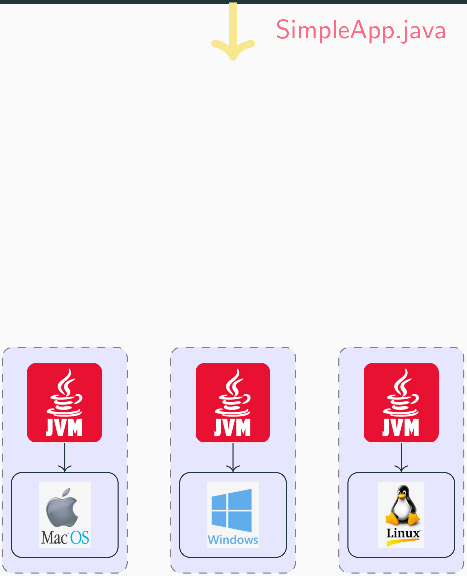
Write Once Run Anywhere

- Java is an interpreted language.
- The Java Compiler translates the source code (`.java` file) into machine code (*bytecode*).
- Java bytecode is the machine language for the Java Virtual Machine (JVM).



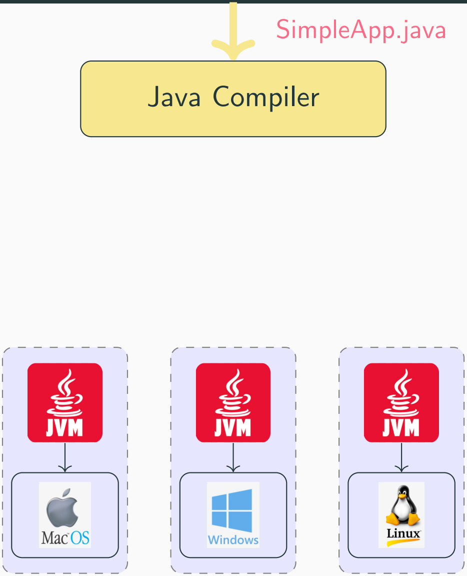
Write Once Run Anywhere

- Java is an interpreted language.
- The Java Compiler translates the source code (`.java` file) into machine code (*bytecode*).
- Java bytecode is the machine language for the Java Virtual Machine (JVM).



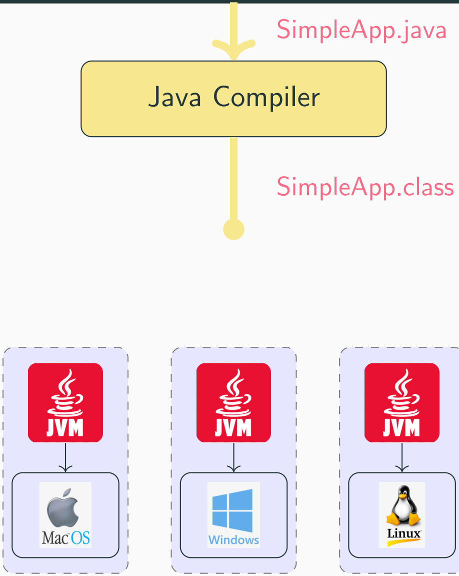
Write Once Run Anywhere

- Java is an interpreted language.
- The Java Compiler translates the source code (`.java` file) into machine code (*bytecode*).
- Java bytecode is the machine language for the Java Virtual Machine (JVM).



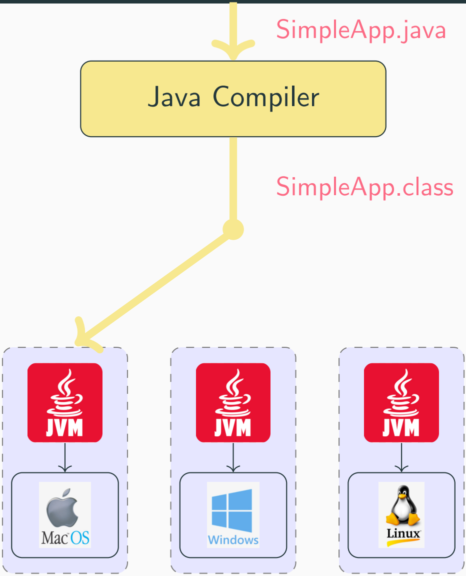
Write Once Run Anywhere

- Java is an interpreted language.
- The Java Compiler translates the source code (`.java` file) into machine code (*bytecode*).
- Java bytecode is the machine language for the Java Virtual Machine (JVM).



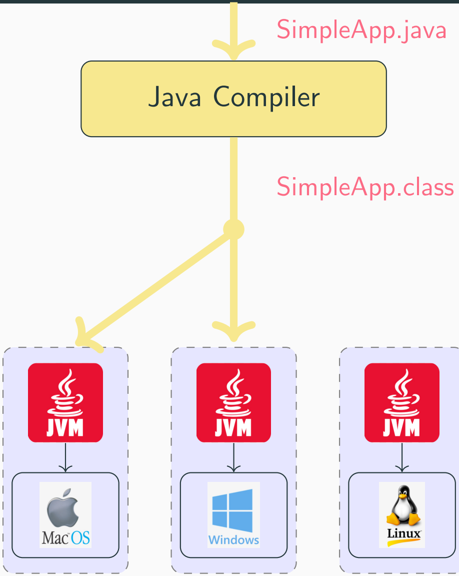
Write Once Run Anywhere

- Java is an interpreted language.
- The Java Compiler translates the source code (`.java` file) into machine code (*bytecode*).
- Java bytecode is the machine language for the Java Virtual Machine (JVM).



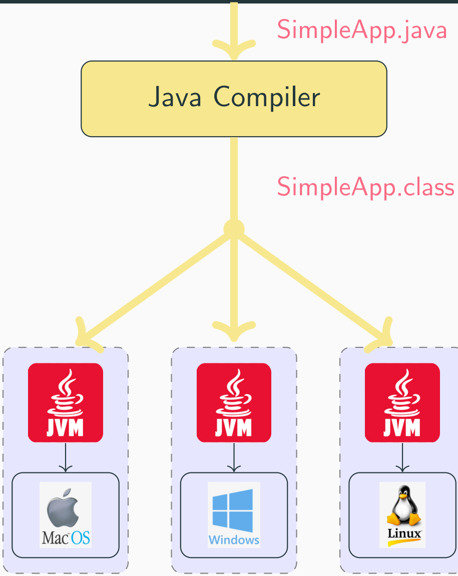
Write Once Run Anywhere

- Java is an interpreted language.
- The Java Compiler translates the source code (`.java` file) into machine code (*bytecode*).
- Java bytecode is the machine language for the Java Virtual Machine (JVM).



Write Once Run Anywhere

- Java is an interpreted language.
- The Java Compiler translates the source code (`.java` file) into machine code (*bytecode*).
- Java bytecode is the machine language for the Java Virtual Machine (JVM).



Comment

- used for internal documentation
- Types include
    1. line comment `// till end of this line`

Comment

- used for internal documentation
- Types include
    1. line comment
    2. multi-line comment

```
// till end of this line
/* muli-line possible
next line comment
*/
```

Comment

- used for internal documentation
- Types include
    1. line comment
    2. multi-line comment
    3. Javadoc comments

```
// till end of this line
/* muli-line possible
next line comment
*/
/** SimpleApp is the
first class to created
for this course.
* @author My Name
* @version 1.0
*/
```

Comment

- used for internal documentation
- Types include
  1. line comment
  2. multi-line comment
  3. Javadoc comments

```
// till end of this line
/* muli-line possible
next line comment
*/
/** SimpleApp is the
first class to created
for this course.
* @author My Name
* @version 1.0
*/
```

Refer to: How to write Javadoc comments

Data

- Java is a strongly-typed language.

Data

- Java is a strongly-typed language.
Primitive Types	

Data

- Java is a strongly-typed language.
-

Primitive Types	Reference Types

Data

- Java is a strongly-typed language.
-

Primitive Types		Reference Types
`boolean`	`int`	
`byte`	`long`	
`char`	`float`	
`short`	`double`	

Data

- Java is a strongly-typed language.
-

Primitive Types		Reference Types

`boolean`
`byte`
`char`
`short`
	`int`
	`long`
	`float`
	`double`
		all non-primitive types, e.g. classes

Data

- Java is a strongly-typed language.
-

Primitive Types		Reference Types
`boolean`	`int`	all non-primitive types, e.g. classes
`byte`	`long`	
`char`	`float`	
`short`	`double`	
holds exactly one value of its declared type at a time		

Data

- Java is a strongly-typed language.
-

Primitive Types		Reference Types
`boolean` `byte` `char` `short`	`int` `long` `float` `double`	all non-primitive types, e.g. classes
holds exactly one value of its declared type at a time		stores addresses

Output Statements

- Commonly used are

```
System.out.print()
System.out.println()
```

- Examples:

Output Statements

- Commonly used are

```
System.out.print()
System.out.println()
```

- Examples:

```
System.out.print(3 + 5);
```

Output Statements

- Commonly used are

```
System.out.print()
System.out.println()
```

- Examples:

```
System.out.print("Hello");
```

Output Statements

- Commonly used are

```
System.out.print()
System.out.println()
```

- Examples:

```
System.out.print('h' + "ello");
```

Output Statements

- Commonly used are

```
System.out.print()
System.out.println()
```

- Examples:

```
System.out.print();
```

Output Statements

- Commonly used are

  ```
  System.out.print()
  System.out.println()
  ```

- Examples:

  ```
  System.out.print();
  ```

  Error!

Output Statements

- Commonly used are

  ```
  System.out.print()
  System.out.println()
  ```

- Examples:

  ```
  System.out.println();
  ```

Output Statements

- Commonly used are

```
System.out.print()
System.out.println()
```

- Examples:

```
System.out.print("ID" + 119);
```

Output Statements

- Commonly used are

```
System.out.print()
System.out.println()
```

- Examples:

```
System.out.print("ID" + 100 + 19);
```

Output Statements

- Commonly used are

```
System.out.print()
System.out.println()
```

- Examples:

```
System.out.print("ID" + (100 + 19));
```

Output Statements

- Commonly used are

  ```
  System.out.print()
  System.out.println()
  ```

- Examples:

```
System.out.print(100 + 19 + "ID");
```

Output Statements

- Commonly used are

```
System.out.print()
System.out.println()
```

- Examples:

```
System.out.print('A' + 4);
```

Output Statements

- Commonly used are

  ```
  System.out.print()
  System.out.println()
  ```

- Examples:

```
System.out.print("A" + 4);
```

Output Statements

- Commonly used are

```
System.out.print()
System.out.println()
```

- Examples:

```
System.out.print(3 + 5);                           8
System.out.print("Hello");                     Hello
System.out.print('h' + "ello");                hello
System.out.print();                           Error!
System.out.println();
System.out.print("ID" + 119);                  ID119
System.out.print("ID" + 100 + 19);           ID10019
System.out.print("ID" + (100 + 19));           ID119
System.out.print(100 + 19 + "ID");             119ID
System.out.print('A' + 4);                        69
System.out.print("A" + 4);                        A4
```

Input Statements using `Scanner`

`Scanner` class

- may be used to read inputs from the console

Input Statements using `Scanner`

`Scanner` class

- may be used to read inputs from the console
- found in the `java.util` package

Input Statements using `Scanner`

`Scanner` class

- may be used to read inputs from the console
- found in the `java.util` package
- to use,

Input Statements using `Scanner`

`Scanner` class

- may be used to read inputs from the console
- found in the `java.util` package
- to use,
    1. include the package `java.util`

Input Statements using Scanner

Scanner class

- may be used to read inputs from the console
- found in the java.util package
- to use,
    1. include the package java.util

       `import java.util.*;`

Input Statements using `Scanner`

`Scanner` class

- may be used to read inputs from the console
- found in the `java.util` package
- to use,
  1. include the package `java.util`
     ```
     import java.util.*;
     ```
  2. declare a variable of type `Scanner`

Input Statements using `Scanner`

`Scanner` class

- may be used to read inputs from the console
- found in the `java.util` package
- to use,
    1. include the package `java.util`

       ```
       import java.util.*;
       ```
    2. declare a variable of type `Scanner`

       ```
       Scanner kb;
       ```

Input Statements using `Scanner`

`Scanner` class

- may be used to read inputs from the console
- found in the `java.util` package
- to use,
    1. include the package `java.util`
       ```
       import java.util.*;
       ```
    2. declare a variable of type `Scanner`
       ```
       Scanner kb;
       ```
    3. instantiate a `Scanner` object and specify `System.in` as the input source

Input Statements using `Scanner`

`Scanner` class

- may be used to read inputs from the console
- found in the `java.util` package
- to use,
  1. include the package `java.util`
     ```
     import java.util.*;
     ```
  2. declare a variable of type `Scanner`
     ```
     Scanner kb;
     ```
  3. instantiate a `Scanner` object and specify `System.in` as the input source
     ```
     kb = new Scanner (System.in);
     ```

Input Statements using `Scanner`

`Scanner` class

- may be used to read inputs from the console
- found in the `java.util` package
- to use,
  1. include the package `java.util`
     ```
     import java.util.*;
     ```
  2. declare a variable of type `Scanner`
     ```
     Scanner kb;
     ```
  3. instantiate a `Scanner` object and specify `System.in` as the input source
     ```
     kb = new Scanner (System.in);
     ```
  4. call the appropriate methods to read the inputs, e.g.

Input Statements using `Scanner`

`Scanner` class

- may be used to read inputs from the console
- found in the `java.util` package
- to use,
  1. include the package `java.util`

     ```
     import java.util.*;
     ```
  2. declare a variable of type `Scanner`

     ```
     Scanner kb;
     ```
  3. instantiate a `Scanner` object and specify `System.in` as the input source

     ```
     kb = new Scanner (System.in);
     ```
  4. call the appropriate methods to read the inputs, e.g.

     ```
     int nVal = kb.nextInt ();
     ```

Input Statements using `Scanner`

`Scanner` class

- may be used to read inputs from the console
- found in the `java.util` package
- to use,
  1. include the package `java.util`
     ```
     import java.util.*;
     ```
  2. declare a variable of type `Scanner`
     ```
     Scanner kb;
     ```
  3. instantiate a `Scanner` object and specify `System.in` as the input source
     ```
     kb = new Scanner (System.in);
     ```
  4. call the appropriate methods to read the inputs, e.g.
     ```
     int nVal = kb.nextInt ();
     ```
  5. close this `Scanner`

Input Statements using Scanner

Scanner class

- may be used to read inputs from the console
- found in the java.util package
- to use,
  1. include the package java.util
     ```
     import java.util.*;
     ```
  2. declare a variable of type Scanner
     ```
     Scanner kb;
     ```
  3. instantiate a Scanner object and specify System.in as the input source
     ```
     kb = new Scanner (System.in);
     ```
  4. call the appropriate methods to read the inputs, e.g.
     ```
     int nVal = kb.nextInt ();
     ```
  5. close this Scanner
     ```
     kb.close ();
     ```

Simple Sample

The program computes for the sum of two numbers given by the user.

Simple Sample

The program computes for the sum of two numbers given by the user.

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









    }
}
```

Simple Sample

The program computes for the sum of two numbers given by the user.

```
import java.util.*;
public class SimpleSample
{
    public static void main (String[] args)
    {










    }
}
```

Simple Sample

The program computes for the sum of two numbers given by the user.

```
import java.util.*;
public class SimpleSample
{
    public static void main (String[] args)
    {
        Scanner kb = new Scanner (System.in);








    }
}
```

Simple Sample

The program computes for the sum of two numbers given by the user.

```
import java.util.*;
public class SimpleSample
{
    public static void main (String[] args)
    {
        Scanner kb = new Scanner (System.in);
        int nOne, nTwo;









    }
}
```

Simple Sample

The program computes for the sum of two numbers given by the user.

```
import java.util.*;
public class SimpleSample
{
    public static void main (String[] args)
    {
        Scanner kb = new Scanner (System.in);
        int nOne, nTwo;

        System.out.print ("Enter a number:  ");
        nOne = kb.nextInt ();
        System.out.print ("Enter another number:  ");
        nTwo = kb.nextInt ();



    }
}
```

Simple Sample

The program computes for the sum of two numbers given by the user.

```
import java.util.*;
public class SimpleSample
{
    public static void main (String[] args)
    {
        Scanner kb = new Scanner (System.in);
        int nOne, nTwo;

        System.out.print ("Enter a number:  ");
        nOne = kb.nextInt ();
        System.out.print ("Enter another number:  ");
        nTwo = kb.nextInt ();

        System.out.println ("Sum is " + (nOne + nTwo));
        kb.close ();
    }
}
```

Scanner

- Other methods in `Scanner`

```
nextInt ();
nextDouble ();
nextBoolean ();
next ();
nextLine ();
```

- To resolve issues when reading inputs, instead of directly using these methods, read the input as a `String` then convert to the appropriate type, e.g.
  - to read integer inputs,
    ```
    int nOne = Integer.parseInt (kb.nextLine ());
    ```
  - to read real-number inputs,
    ```
    double dVal = Double.parseDouble (kb.nextLine ());
    ```

Simple Sample... modified

The program computes for the sum of two numbers given by the user.

```
import java.util.*;
public class SimpleSample
{
    public static void main (String[] args)
    {
        Scanner kb = new Scanner (System.in);
        int nOne, nTwo;

        System.out.print ("Enter a number:  ");
        nOne = Integer.parseInt (kb.nextLine ());
        System.out.print ("Enter another number:  ");
        nTwo = Integer.parseInt (kb.nextLine ());

        System.out.println ("Sum is " + (nOne + nTwo));
        kb.close ();
    }
}
```

☺ **Thank you!** ☺