Introduction to Java

COMP2603
Object Oriented Programming 1

Week 1

Outline

- Java Platform
  - JVM, API
- Differences with C, C++
- Java Language Features
  - Primitive Data Types: Numeric, Character, Boolean
  - Arithmetic, Relational Operators
  - Class Types: Arrays, Strings
  - Conditional Statements
  - Loops, Switch Constructs

Java Platform

" A platform is the hardware or software environment in which a program runs.

Popular examples of platforms are Microsoft Windows, Linux, Solaris OS, and Mac OS. Most platforms can be described as a combination of the operating system and underlying hardware.

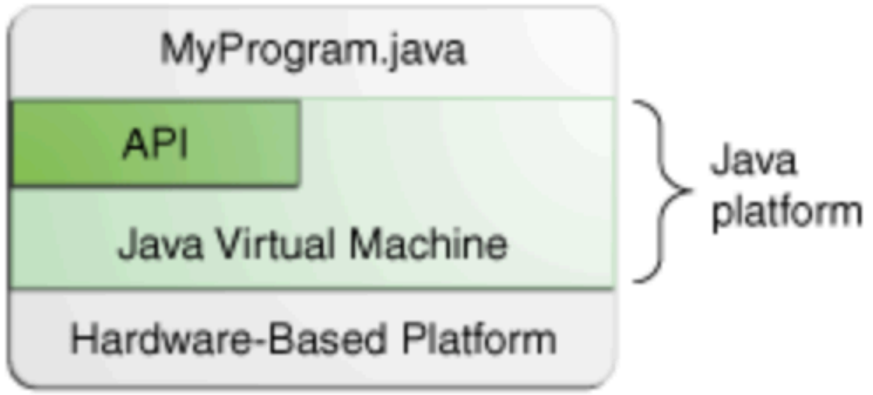
The Java platform differs from most other platforms in that it's a software-only platform that runs on top of other hardware-based platforms. "

https://docs.oracle.com/javase/tutorial/getStarted/intro/definition.html

Java Platform

The Java platform has two components:

- The Java Virtual Machine (JVM)
- The Java Application Programming Interface (API)

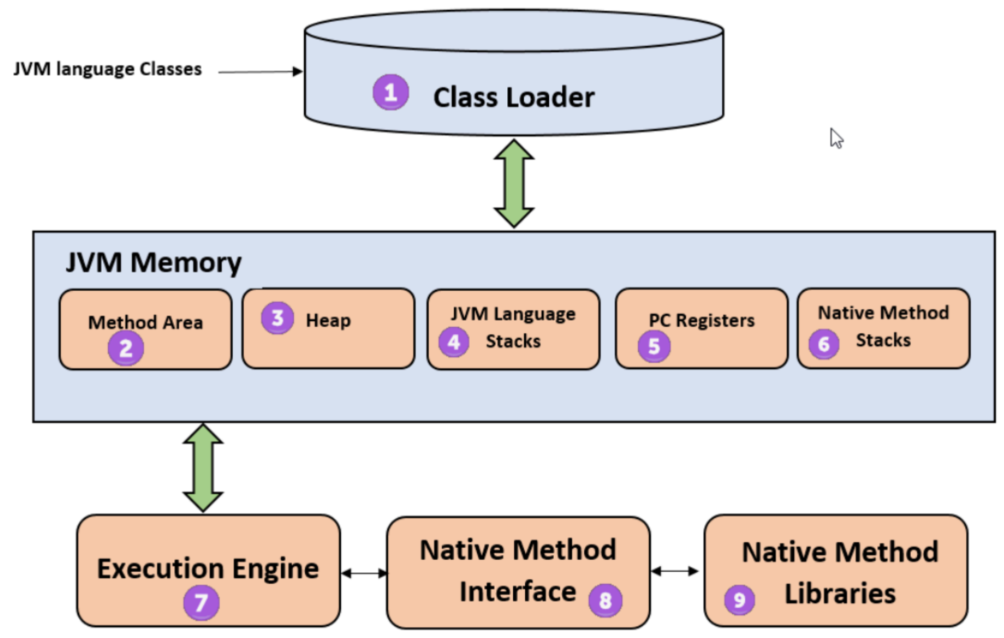


The API and Java Virtual Machine insulate the program from the underlying hardware.

https://docs.oracle.com/javase/tutorial/getStarted/intro/definition.html

Java Virtual Machine

Java Virtual Machine (JVM) is a engine that provides runtime environment to drive the Java Code or applications. It converts Java bytecode into machines language. JVM is a part of Java Run Environment (JRE).



Java Virtual Machine Architecture

https://www.guru99.com/java-virtual-machine-jvm.html

Java Virtual Machine

https://www.guru99.com/java-virtual-machine-jvm.html

Garbage Collector

The Java run-time system manages natural pauses typical during user-driven software use by recovering and compacting fragments of unused memory.

The garbage collector is run as a low-priority thread during idle periods. Unused portions of memory are gathered and reallocated for use during periods of heavy interactive use.

Java API

"Java application programming interface (API) is a list of all classes that are part of the Java development kit (JDK).

It includes all Java packages, classes, and interfaces, along with their methods, fields, and constructors.

These prewritten classes provide a tremendous amount of functionality to a programmer."

http://www.saylor.org/courses/cs101/#1.3.5.3

We will use the latest version, Java 11, in this course:

https://docs.oracle.com/en/java/javase/11/docs/api/index.html

Source files

In the Java programming language, all source code is first written in plain text files ending with the **.java** extension.

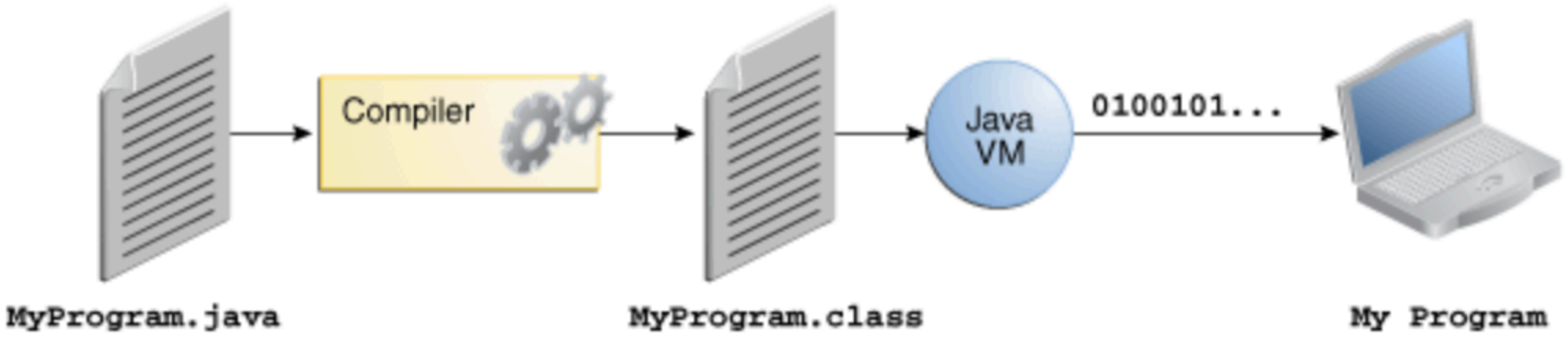
Those source files are then compiled into **.class** files by the javac compiler.

A .class file does not contain code that is native to your processor; it instead contains bytecodes — the machine language of the Java Virtual Machine (Java VM).

The java launcher tool then runs your application with an instance of the Java Virtual Machine.

https://docs.oracle.com/javase/tutorial/getStarted/intro/definition.html

Source files

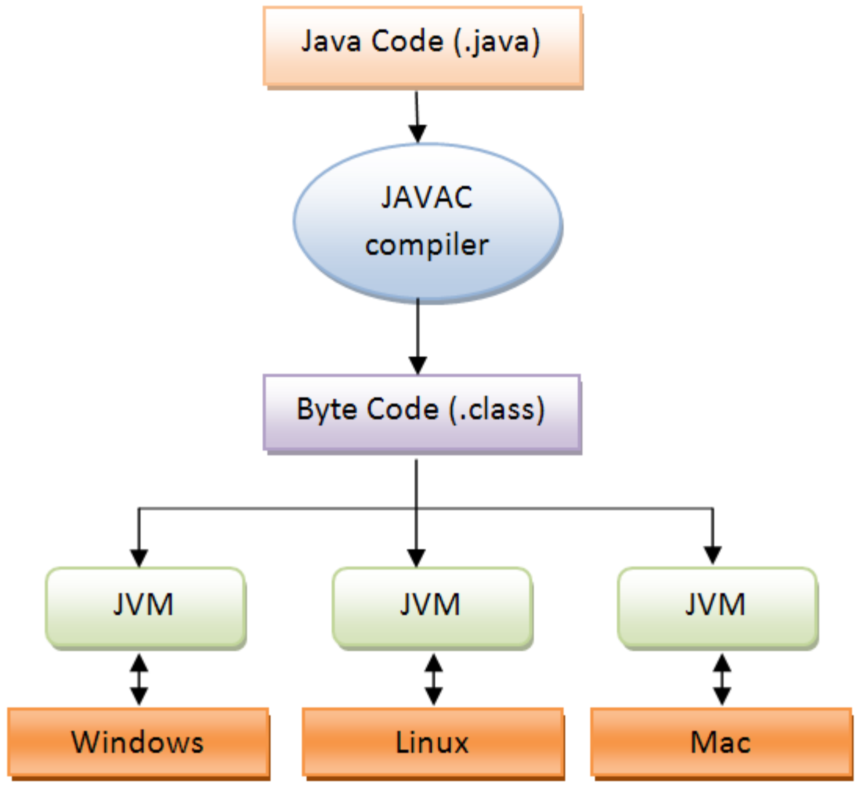


An overview of the software development process.

https://docs.oracle.com/javase/tutorial/getStarted/intro/definition.html

Modern Programming Languages

Most modern programming languages are designed to be relatively easy for people to write and understand. These are called high-level languages. Eg. Pascal, C, C++, HTML, Java.



Features Removed from C and C++

- No More Typedefs, Defines, or Preprocessor
  - #define, typedef
  - Instead of header files - Java source files provide declarations of other classes and methods
- No More Structures or Unions
  - Classes with instance variables are used
- No Enums
  - Classes with variables are used as constants

https://www.oracle.com/java/technologies/simple-familiar.html

Features Removed from C and C++

- No More Functions
  - Object-oriented programming supersedes functional and procedural styles
- No More goto Statements
  - Multi-level break and continue statements used instead
- No More Automatic Coercions
  - Loss of precision would result. Instead explicitly cast.
- No More Pointers
  - Since no structures, arrays and string are full objects, then no need for pointer data types (avoids dangling pointers, trashing of memory).

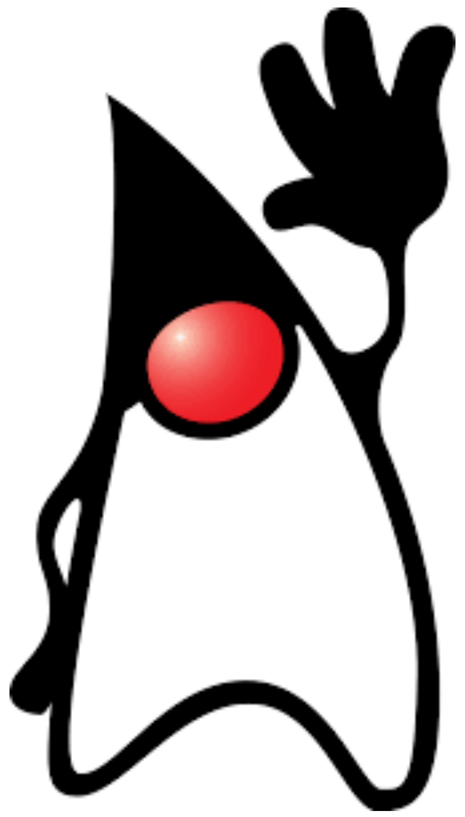
https://www.oracle.com/java/technologies/simple-familiar.html

Object-Oriented Programming

Java was designed for object-oriented programming. Several important concepts will be covered in the next few weeks:

- Encapsulation (information hiding)
- Abstraction

Java Language Features

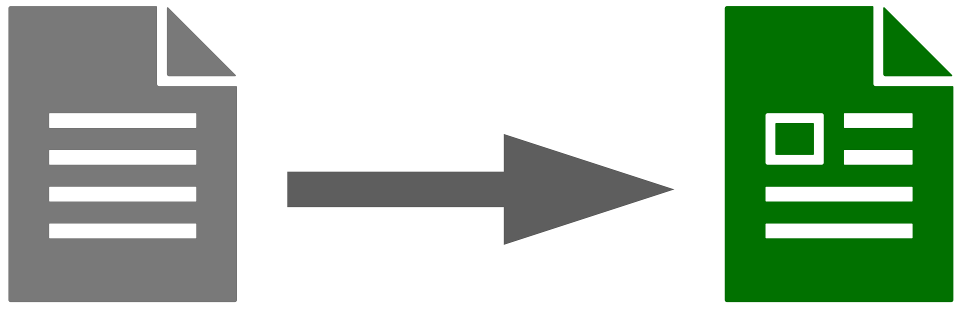


Java Program Format

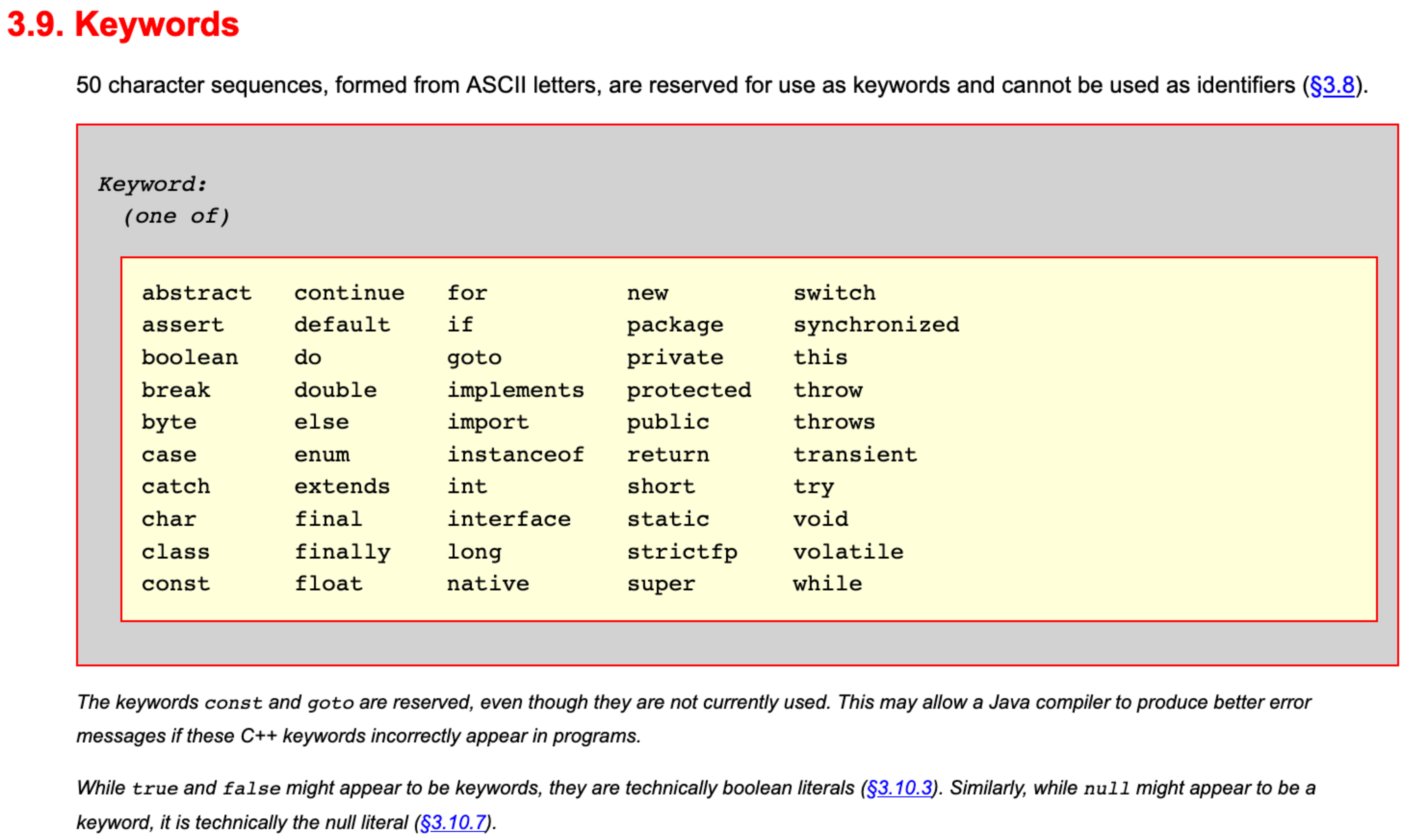
The name of your class

```
//import statements

public class ClassName{
    public static void main(String[] args){
      // code goes here
    }
}
```



Java Keywords



3.9. Keywords

50 character sequences, formed from ASCII letters, are reserved for use as keywords and cannot be used as identifiers (§3.8).

```
Keyword:
  (one of)

  abstract   continue   for          new         switch
  assert     default    if           package     synchronized
  boolean    do         goto         private     this
  break      double     implements   protected   throw
  byte       else       import       public      throws
  case       enum       instanceof   return      transient
  catch      extends    int          short       try
  char       final      interface    static      void
  class      finally    long         strictfp    volatile
  const      float      native       super       while
```

The keywords `const` and `goto` are reserved, even though they are not currently used. This may allow a Java compiler to produce better error messages if these C++ keywords incorrectly appear in programs.

While `true` and `false` might appear to be keywords, they are technically boolean literals (§3.10.3). Similarly, while `null` might appear to be a keyword, it is technically the null literal (§3.10.7).

https://docs.oracle.com/javase/specs/jls/se8/html/jls-3.html

Variables

Variables in a program store data such as numbers and letters. These data items are called **values.**

```
int numberOfBaskets; //declaration
numberOfBaskets = 10; //initialisation
```

Variables must be declared before they can be used. The variable's **type** determines the kind of value that the variable can hold.

Conventions and naming rules: start with lowercase letter, follow camelCase, and use meaningful names.

Types of Variables

There are two main kinds of types in Java: **primitive** types and **class** types.

Primitive Type: simple, indecomposable values e.g. int, double, char, float

Example:

```
int numberOfBaskets;
char symbol = ‘A’;
```

Class Type: specifies a type for an object that has both data and methods.

Example:

```
Calculator calculator;
```

Java Program with Variables

```
//import statements

public class MathDemo{
    public static void main(String[] args){
      int operator1 = 10;
      int operator2 = 20;
    }

}
```

At-Home Exercise

Primitive Types

Type	Kind of Value	Example Description	Fill in Code for Given Example
byte	integer	Stores the number of students sleeping in class	
double	floating-point number	Stores the GPA value that students dream of when they are sleeping in class	
char	single Unicode character	Stores the letter grade that students abhor but which can be avoided if they didn't sleep in class	
boolean	true or false	Stores the student response of whether they have ever signed into a lecture and then went back to sleep	
short	integer	Stores the time in milliseconds that it could take students to complete this worksheet if they type at 216 WPM	
int	integer	Stores the time in milliseconds that it should take students to complete this worksheet	
long	integer	Stores the time in milliseconds that it actually takes students to complete this worksheet, because they were sleeping in class	
float	floating-point number	Stores the probability of anyone doing this worksheet	

The String Class

Strings of characters are treated differently from values of primitive types. There is no primitive type for strings in Java.

The String class is used to store and process strings of characters.

Example 1:

```
String greeting; //declaration
greeting = “Hello, pleased to meet you”; //initialisation
```

Example 2:

```
String name = “Alice”; //declaration and initialisation
```

https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/String.html

Java Program with Strings

```
//import statements

public class StringDemo{
    public static void main(String[] args){
      String bookTitle = “Flowers for Algernon”;
      String author;
      author = “Daniel Keyes”;
      System.out.println(bookTitle);
      System.out.println(author);
    }
}
```

Java Program with Strings

```
//import statements

public class StringDemo{
    public static void main(String[] args){
      String bookTitle = “Flowers for Algernon”;
      String author;
      author = “Daniel Keyes”;
      System.out.println(bookTitle);
      System.out.println(author);
    }
}
```

Output:
```
Flowers for Algernon
Daniel Keyes
```

Java Program with Strings

```
//import statements

public class StringDemo{
    public static void main(String[] args){
      String bookTitle = “Flowers for Algernon”;
      String author;
      author = “Daniel Keyes”;
      System.out.println(“The book, ” + bookTitle +
      “, was written by “ + author +”.”);
    }
}
```

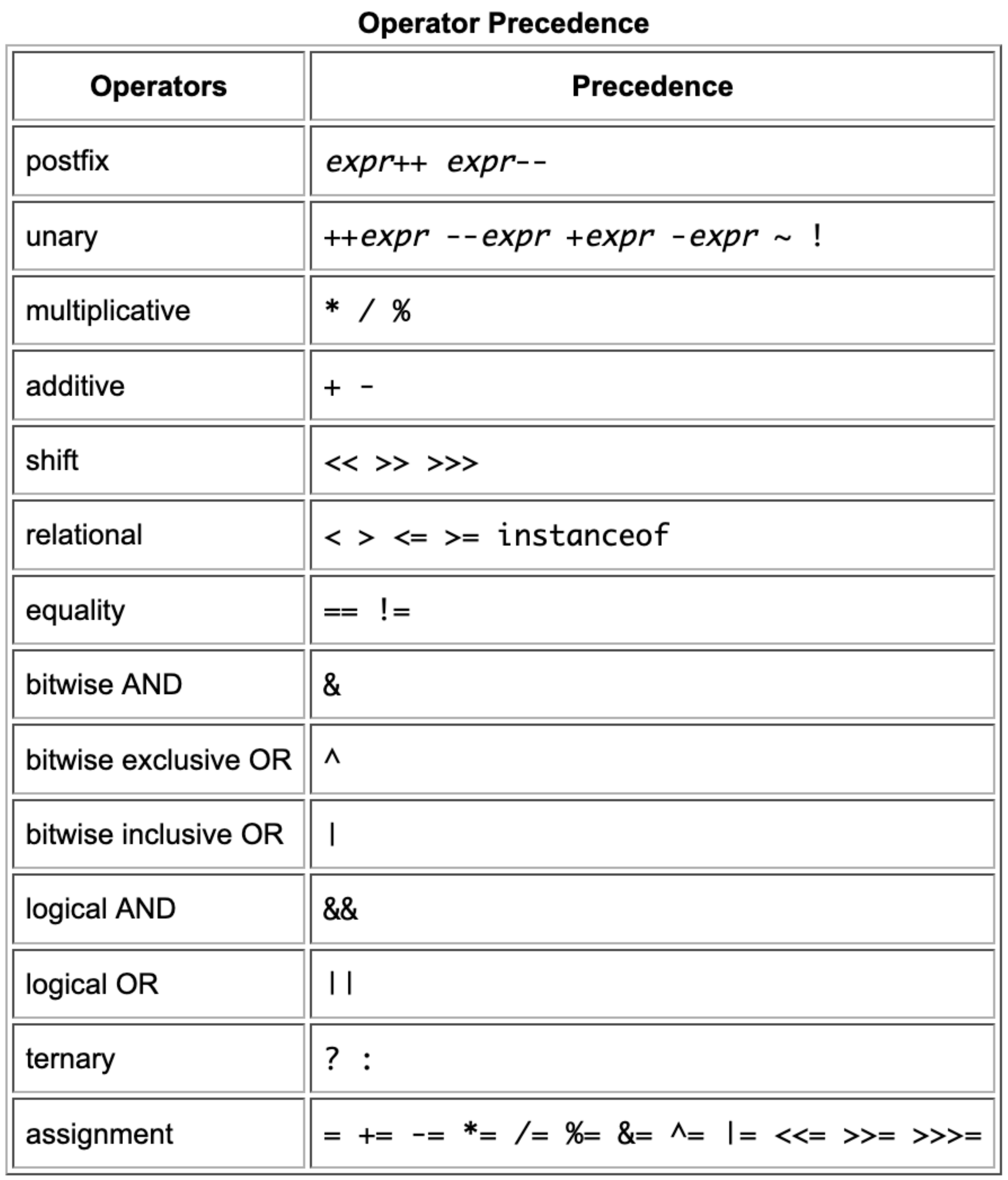
Java Program with Strings

```
//import statements

public class StringDemo{
    public static void main(String[] args){
      String bookTitle = “Flowers for Algernon”;
      String author;
      author = “Daniel Keyes”;
      System.out.println(“The book, ” + bookTitle +
      “, was written by “ + author +”.”);
    }
}
```

Output:
The book, Flowers for Algernon, was written by Daniel Keyes.

Java Operators



Operator Precedence

Operators	Precedence
postfix	*expr*++ *expr*--
unary	++*expr* --*expr* +*expr* -*expr* ~ !
multiplicative	* / %
additive	+ -
shift	<< >> >>>
relational	< > <= >= instanceof
equality	== !=
bitwise AND	&
bitwise exclusive OR	^
bitwise inclusive OR	|
logical AND	&&
logical OR	||
ternary	? :
assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/operators.html

Java Program with Operators

```
public class MathReportDemo{
    public static void main(String[] args){
      int operator1 = 10;
      int operator2 = 250;
      int result = operator2 – operator 1;
      String outputMessage = “The result is: ”;
      //concatenation of Strings with a variable
      String output = outputMessage + result;
      System.out.println(output);
    }
}
```

Java Program with Operators

```
public class MathReportDemo{
    public static void main(String[] args){
      int operator1 = 10;
      int operator2 = 250;
      int result = operator2 - operator 1;
      String outputMessage = “The result is: ”;
      //concatenation of Strings with a variable
      String output = outputMessage + result;
      System.out.println(output);
    }
}
```

Output:
The result is: 240

Expressions, Statements, and Blocks

Operators: used in building expressions, which compute values

Expressions : the core components of statements

Statements may be grouped into blocks

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/expressions.html

Expressions

An expression is a construct made up of variables, operators, and method invocations, which are constructed according to the syntax of the language, that evaluates to a single value. Examples of expressions have been used in the examples so far.

Avoid ambiguous expressions.

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/expressions.html

Statement

Statements are roughly equivalent to sentences in natural languages. A statement forms a complete unit of execution. The following types of expressions can be made into a statement by terminating the expression with a semicolon (;).

- Assignment expressions
- Any use of ++ or --
- Method invocations
- Object creation expressions

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/expressions.html

Block

A block is a group of zero or more statements between balanced braces and can be used anywhere a single statement is allowed.

Java Program with Expressions

```
//import statements

public class MathDemo{
    public static void main(String[] args){
        int operator1 = 200;
        int operator2 = 2000;
        String output = “”;
        //Expression
        int result = (operator2 - operator1)/100;
        //Avoid: int result = operator2 - operator1/100;
        output = “The result is: ” + result;
        System.out.println(output);
    }
}
```

Control Flow Statements

These employ decision making, looping, and branching, enabling your program to conditionally execute particular blocks of code.

- Decision-making statements (if-then, if-then-else, switch)
- Looping statements (for, while, do-while)
- Branching statements (break, continue, return)

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/flow.html

Java Program with Control Flow Statements

```
//import statements

public class MathDemo{
    public static void main(String[] args){
       int operator1 = 200;
       int operator2 = 2000;
       if(operator1 > operator2){
          System.out.println(“Operator 1 is larger”);
       }
       else{
          System.out.println(“Operator 2 is larger”);
       }

    }
}
```

Java Program with Control Flow Statements

```
//import statements

public class MathDemo{
     public static void main(String[] args){
        int operator1 = 200;
        int operator2 = 2000;
        String output = “”;
        int result = (operator2 - operator 1)/100;
        output = output + “The result is: ”;
        if( result%2 == 0){
           output += “    ”;
        }
        else{
           output += “     ”;
        }
        System.out.println(output);
     }
}
```

Fill in appropriate words to complete the statements

Java Program with Control Flow Statements

```
//import statement

public class MathDemo{
     public static void main(String[] args){
         int operator1 = 200;
         int operator2 = 2000;
         String output = “”;
         int result = (operator2 – operator 1)/100;
         output = “The result is: ”;
         if( result%2 == 0){
             output += “even”;
         }
         else{
             output += “odd”;
         }
         System.out.println(output);

     }
}
```

Methods

A method is an action that an object is a capable of performing. Asking an object to perform that method is called invoking the method or calling the method.

Example:

```
    calculator.add(3,4);
```

Syntax:

```
    objectName.methodName(arguments..);
```

dot

Simple Input from the Screen

The Scanner class is typically used for reading input data.

Example:

```
Scanner screen = new Scanner(System.in);
 int i = screen.nextInt();
```

https://docs.oracle.com/javase/7/docs/api/java/util/Scanner.html

Simple Output to the Screen

The System class is typically used for printing output data to the screen.

Example:

```
    System.out.println(“Hello”);
```

https://docs.oracle.com/javase/7/docs/api/java/io/PrintStream.html

Exercise One

3 mins

Write a simple Java program that prints the statement:

Hello World

Your class should be named appropriately.

Class structure

```
//import statements

public class ClassName{
     public static void main(String[] args){
        // your code goes here
     }
}
```

First Java Program

Write a simple Java program that prints the statement:

Hello World

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

Exercise Two

5 mins

Write a simple Java program that accepts a user's name from the screen, stores the value in a String variable and prints a greeting with the format:

Hello <username>

Class structure

```
//import statements

public class ClassName{
     public static void main(String[] args){
         // your code goes here
     }
}
```

Program 2

```
import java.util.Scanner;

public class SecondProgram{
    public static void main(String[] args){
      Scanner screen = new Scanner(System.in);
      String name = screen.next();
      System.out.print(“Hello ”);
      System.out.print(name);
      System.out.println(“Hello ” + name);
    }
}
```

Summary

Today, you learned about:

- History of the Java language
- Important features and comparisons to other languages
- Java language basics (variables, classes, methods)
- How to write a simple Java program
- Java API and how to use it (Scanner)

Homework: Read Language Basics and all subsections

References - Required Reading

- https://docs.oracle.com/javase/tutorial/getStarted/intro/definition.html
- https://www.oracle.com/java/technologies/language-environment.html
- https://docs.oracle.com/javase/specs/jls/se15/html/index.html
- https://docs.oracle.com/javase/tutorial/java/nutsandbolts/index.html