# AP CS A (Java) Training

**Unit 1: Using Objects and Methods**

Updated Curriculum (2025-2026)

# Meeting Etiquette Guide

Camera ON: Encouraged

Write questions in chat or unmute!

Contribute in session

# Meet the Trainers

## Jackson Riche

- Jupiter High School
- Currently teaching AP CS Principles, AP CS A, AICE Computer Science & Cybersecurity
- Started teaching CS  20 years ago
- Email: jackson.riche@palmbeachschools.org

## Yanet Cabrera

- Wellington High School
- Currently teaching AP CS Principles, AP CS A, Algebra 2 Honors
- MA.Ed in Mathematics Education
    - Started teaching CS 4 years into teaching
- Self-taught CS knowledge and took FTCE Prep course to take and pass FTCE CS Certification
- Email: yanet.cabrera@palmbeachschools.org

# Introductions

Tell us about yourself:

- What is your name?
- What school do you teach at?
- What courses are you expecting to teach this year?

**Attendees**

# Agenda for each day

| Time | Description |
| --- | --- |
| **9:00 AM** | Welcome! |
| **10:30 AM** | Short 10 minute break |
| **12:00 PM** | Lunch 1 hour |
| **2:30 PM** | Short 10 minute break |
| **4:00 PM** | End of session |

# AP CS A Units of Study

| Units of Instruction | Approximate Multiple-Choice Exam Weighting |
|---|---|
| **Unit 1:** Using Objects and Methods | **15–25%** |
| **Unit 2:** Selection and Iteration | **25–35%** |
| **Unit 3:** Class Creation | **10–18%** |
| **Unit 4:** Data Collections | **30–40%** |

# What's Going to Chance?

- Adding topics on text files and data sets.
- Removing the inheritance unit (Unit 9). This will more closely align the course with introductory college courses and allow teachers to cover other topics in more detail.
- Consolidating Units 1-8 and 10 into 4 units.

# AP CS A Exam Breakdown

**Multiple-Choice Section**

- 42 questions—an increase from 40.
- 4 answer choices per question—a decrease from 5.
- 55% of overall exam score—an increase from 50%.

**Free-Response Section**

- 4 shortened questions with 25 scored points—a decrease from 36 scored points.
- Removing arrays from Question 3 (Data Analysis with ArrayList). This question will focus only on ArrayLists.
- 45% of overall exam score—a decrease from 50%.

# Summary: 2024-2025 vs 2025-2026

| Aspect | Before (2024–25) | After (2025–26) |
|---|---|---|
| Units | 10 (Units 1–8, 9 Inheritance, 10) | 4 consolidated units |
| Inheritance/Polymorphism | Required | Removed from required content |
| New topics | — | File I/O, data sets, AI/social ethics |
| MCQs | 40 questions, 5 choices, 50% score | 42 questions, 4 choices, 55% score |
| FRQs | 4 questions, 36 points, 50% score | 4 questions, 25 points, 45% score |
| FRQ focus | Arrays & ArrayLists in Q3 | Only ArrayLists in Q3 |

**Software** is a collection of instructions that is run by a computer.

An **integrated development environment** , or **IDE**, is a software application for writing, compiling, testing, and debugging program code.

eclipse

Apache NetBeans IDE

Visual Studio

# AP CS A Curriculum and IDE's

- Teachers use different programming environments(for coding in java) below are some popular ones:
  - Local IDEs(integrated development environment): Dr.Java, Eclipse, jGrasp, many more
  - Online IDEs: JDoodle, CodeHS, many more

- Teachers also use different curriculums, some popular ones are:
  - Code.org, CodeHS, **CSAwesome**, Project STEM, many more

# Units 1:
# Using Objects and Methods

# 1.1 Introduction to Algorithms, Programming, and Programming Concepts

## Learning Objectives:

- Represent patterns and algorithms found in everyday life using written language or diagrams
- Explain the code compilation and execution process.
- Identify types of programming errors.

# Students will be able to represent patterns and algorithms found in everyday life using

- Algorithm is a finite set of step-by-step instructions to solve a problem or complete a task
- It can be represented in many ways:
  - With graphics (**Ex. flowcharts**)
  - Pseudocode(**Designed primarily for Human Understanding**)
  - Program code(**Designed for computers**) **Ex. Java, Python, JavaScript, etc**)

# Key characteristics of algorithm

**Finite**: It must end after a limited number of steps.

**Well-defined**: Each step must be clear and unambiguous.

**Input**: It may take one or more inputs.

**Output**: It produces at least one output.

**Effective**: Each step must be basic enough to be performed exactly.

# An algorithm to find calculate and display the sum of 2 numbers

1. Begin
2. Input firstNum
3. Input secondNum
4. Add firstNum and secondNum
5. Store the result from #4 into a variable sum
6. Display the content variable sum
7. Stop

# Sequencing, Selection, and iteration

- **Algorithms** can be implemented through sequencing, selection, and iteration.
  - **Sequencing** means that each step of the algorithm is completed one at a time.
  - **Selection** means that decisions can be made based on certain condition(s) (**if/else)**
  - **Iteration** means that step(s) in an algorithm can be repeated (**loops**).

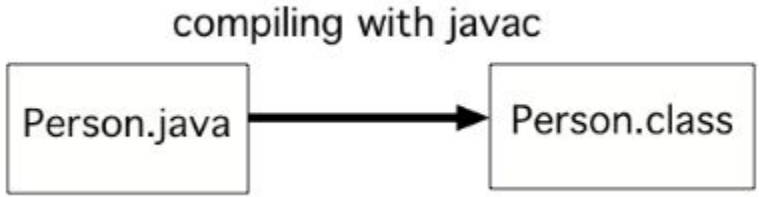
# Students will be able to explain the code compilation and execution process.

- Code can be written in any text editor; however, an Integrated Development Environment (**IDE)** is often used to write programs because it provides tools for a programmer to write, compile, and run code.
- A compiler checks code for some errors. Errors detectable by the compiler need to be fixed before the program can be run.

# Java

Java is a **programming language**, which means that we can use Java to tell a computer what to do.

Computers don't actually speak Java, so we have to **compile** (translate) Java source files (they end in **.java**) into class files (they end in **.class**).




The source file is something humans can read and edit, and the class file is code that a computer can understand and can run.

# Java Terminology

- **class**:
  A description of a type of objects. (Animal class, Human class, Employee class, Car class)

- **statement**: An executable piece of code that represents a complete command to the computer.
  - every basic Java statement ends with a semicolon **;**

- **method**: A named sequence of statements that can be executed together to perform a particular action or computation (in CSP, we called this a procedure)

# Structure of a Java program

- Every executable Java program consists of a **class**, called the **driver class**,
  - that contains a **method** named **`main`**,
    - that contains the **statements** (commands) to be executed.
- Example:

```
public class className {
    public static void main(String[] args) {
        statement;
        statement;
        ...
        statement;
    }
}
```

**class**: a program

**method**: a named group of statements

**statement**: a command to be executed

# Skeleton of a Java program Recap

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

    }
}
```

MyProgram.java

Everything in the **.java** file must exist within the program's **class**.

If the class is MyProgram, the java file must be named **MyProgram.java**

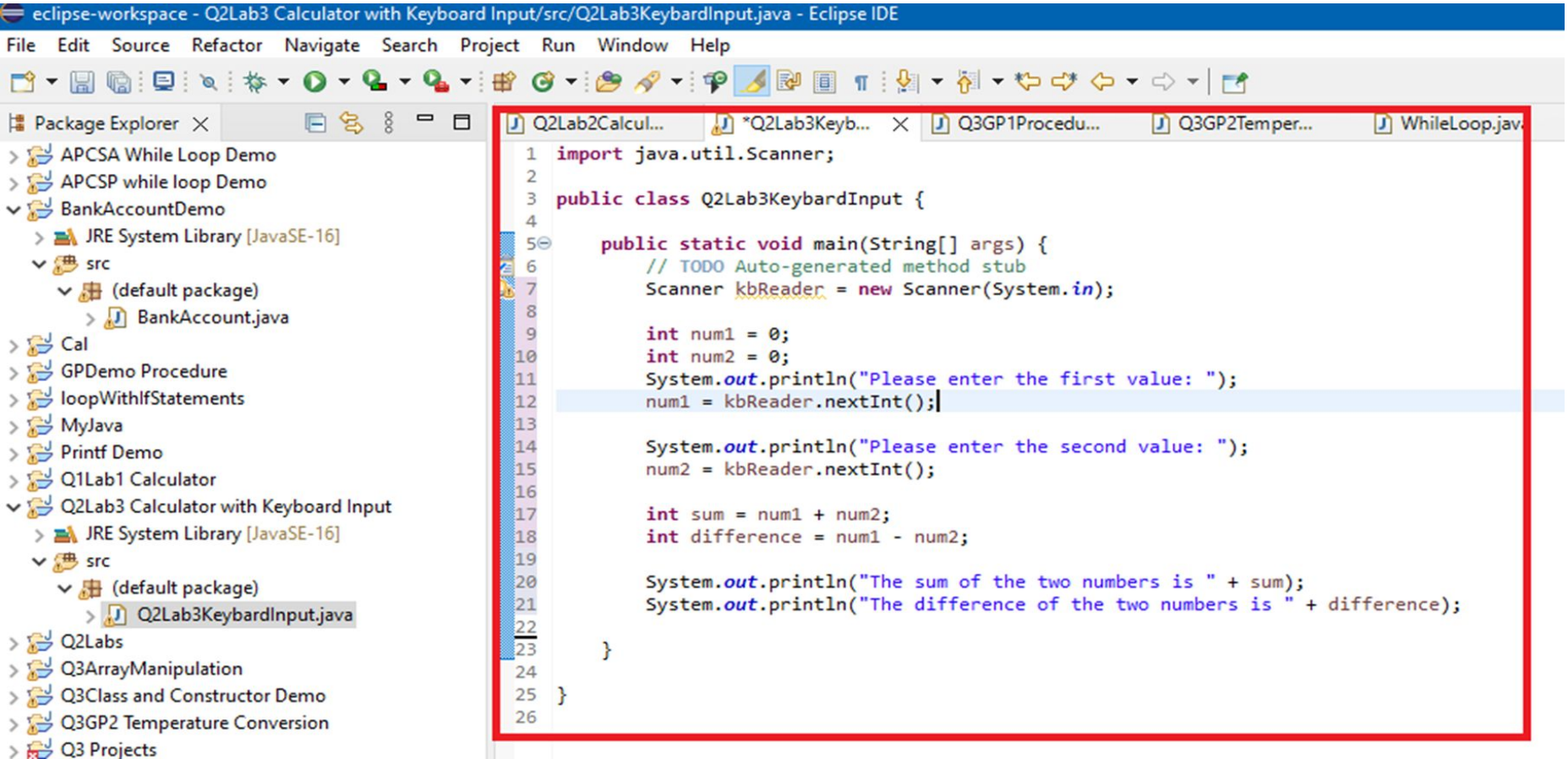
The **main** method is the point where java programs start their execution.

Anything between the curly brackets in the main method will be executed.
*Without a main method, you will get an ERROR!!*

# THINGS TO NOTE...

- **Syntax** is the rules for how a programmer must write code for a computer to understand.
- We write **statements** to tell Java what to do.
- Java is **case-sensitive**, which means **`Hello`** and **`hello`** refer to different things in a program.
- We use **camel case** as the standard convention to capitalize words within names, such as **`myPainter`** or **`MyConsole`**.
- Java has **keywords**, which are words that have a predefined meaning, like **`public`** and **`class`**.

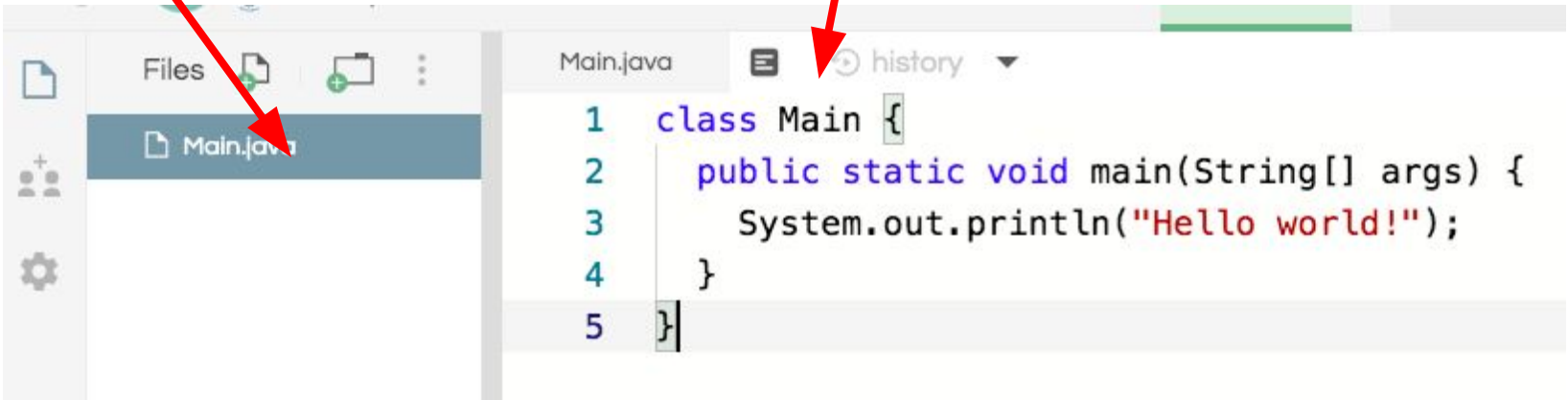
The screen below shows a program written on an IDE that provides pretty-printing, line numbering, formatting, auto-error checking, etc.



# File Naming

The name of the class has to match up with the name of the file.

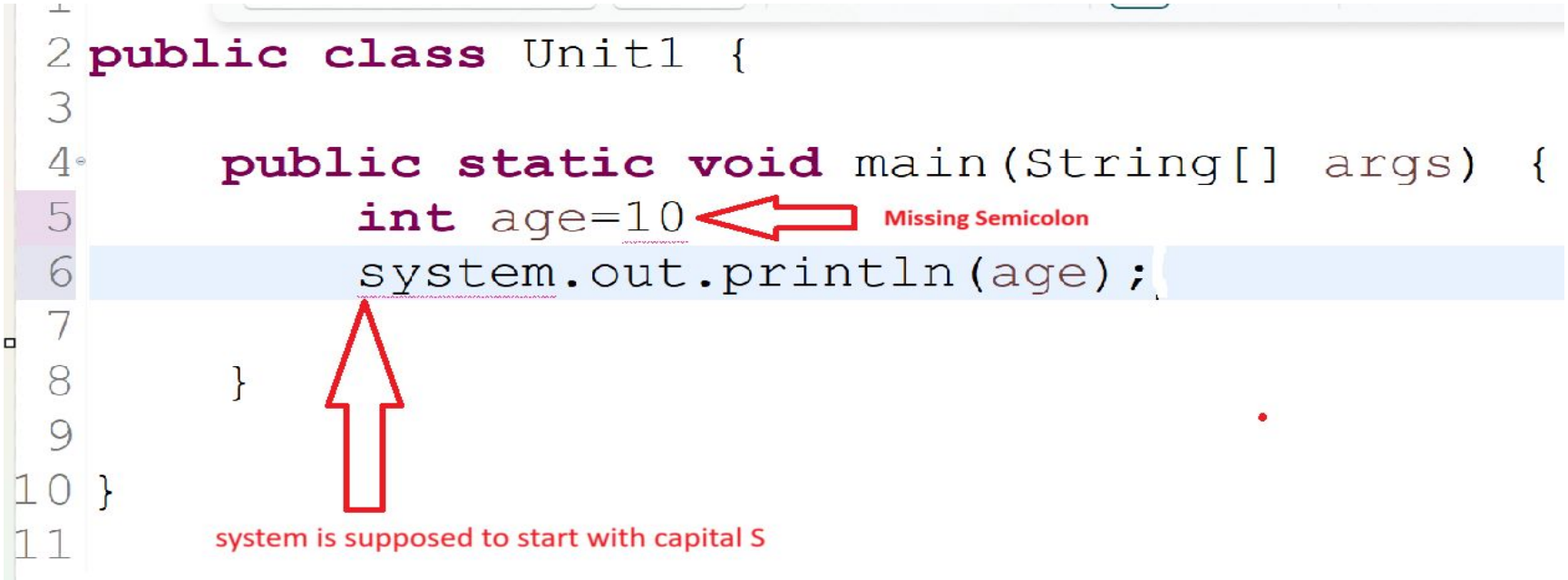
For example, the class below is called **`Main;`** therefore, the name of the file is **Main.java**



# Students will be able to identify the types of programming Errors

- A **syntax error** is an error in the program due to the rules of the programming language not being followed. These errors are detected by the compiler.

Ex.



# Identify the types of programming Errors

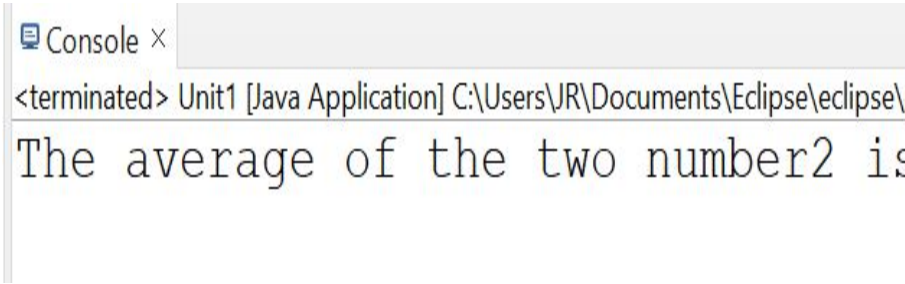
A **logicerror** is a mistake in the algorithm or program that causes it to behave incorrectly or unexpectedly.

These errors are detected by testing the program with specific data to see if it produces the expected outcome.

Ex.

```
1
2 public class Unit1 {
3
4     public static void main(String[] args) {
5         int score1=89;
6         int score2=75;
7         int score3=92;
8         int average=score1+score2+score3/3;
9         System.out.println("The average of the two number2 is "+average);
10
11    }
12 }
```

**Output**



# Identify the types of programming Errors

A **run-time error** is an error that occurs during the execution of of the program. In the other words, it causes a program to crash as a result of an operation that is impossible for the computer to carry out.

Ex.

```
1
2 public class Unit1 {
3
4     public static void main(String[] args) {
5         int x=89;
6         int y=2;
7         int z=(x+y)/(y-2);
8         System.out.println("The average of z is "+z);
9
10    }
11 }
12
```

Console ×

<terminated> Unit1 [Java Application] C:\Users\JR\Documents\Eclipse\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.5.v20221102-0933\jre\

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at Unit1.main(Unit1.java:7)
```

# Identify the types of programming Errors

In Java, run-time errors are often referred to as **exceptions** which are errors that occurs as a result of an unexpected problem detected by the compiler while the program is running. It automatically interrupts the normal flow of the program's execution.

# 1.2 Variables and Data Types

## Learning Objectives:

- Identify the most appropriate data type category for a particular specification.
- Develop code to declare variables to store numbers and Boolean values.

# Data Types

Programming uses a number of different data types. A data **type** determines the type of value (e.g. integers, floats, etc..) and a set of operations (e.g. +, -, *, /, etc..)  we can do on them.

Data types can be categorized as either **primitive** or **reference**.

The primitive data types used in this course define the set of operations for numbers and Boolean(true or false) values.

**Reference variables or object variables** hold a reference(or address) to an object of a class(more on this later).

# Primitive Data Types

A primitive data type specifies the size and type of variable values, and it has no additional methods. There are eight primitive data types in Java:

| Data Type | Size | Description | Default Value (for attributes) |
|---|---|---|---|
| `byte` | 1 byte | Stores whole numbers from -128 to 127 | 0 |
| `short` | 2 bytes | Stores whole numbers from -32,768 to 32,767 | 0 |
| `int` | 4 bytes | Stores whole numbers from -2,147,483,648 to 2,147,483,647 | 0 |
| `long` | 8 bytes | Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 | 0L |
| `float` | 4 bytes | Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits | 0.0f **(f must be there)** |
| `double` | 8 bytes | Stores fractional numbers. Sufficient for storing 15 decimal digits | 0.0 or 0.0d |
| `boolean` | 1 bit | Stores true or false values | false |
| `char` | 2 bytes | Stores a single character/letter or ASCII values | '0' |

# Primitive types

Only 3 primitive types are covered on the AP Computer Science A exam, and they are:

- **int** - which store integers (whole numbers like 3, -76, 20393)

- **double** - which store floating point numbers (decimal numbers like 6.3, -0.9, and 60293.93032)
- **boolean** - which store boolean values (either true or false).

# ARE STRINGS PRIMITIVE? NO!

A **`String`** is an object data type that we often use in our programs. When we create a **`String`**, we are actually creating an object that is an instance of the **`String`** class which represents a collection of characters. Enclosing your character string within double quotes will automatically create a new String object; for example, **`String x = "this is a string ";`**

The **`String`** class is available in all programs by default.

## String

**Category:** java.lang

The `String` class represents a sequence of characters. There exist a variety of methods within the `String` class for identifying specific characters within the string, for comparing strings, and searching for substrings.

### Method Details

String

`String(String original)`

Creates a `String` object that represents the same sequence of characters as the argument

Parameters

| Name | Type | Description |
|---|---|---|
| original | String | a `String` that the `String` object will copy |

Examples

```
String tempString = new String("hello");
```

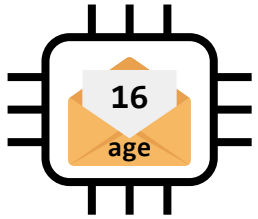
# Receipt example

What's bad about the following code?

```
public class Receipt {
    public static void main(String[] args) {
        // Calculate total owed, assuming 8% tax / 15% tip
        System.out.println("Subtotal:");
        System.out.println(38 + 40 + 30);
        System.out.println("Tax:");
        System.out.println((38 + 40 + 30) * .08);
        System.out.println("Tip:");
        System.out.println((38 + 40 + 30) * .15);
        System.out.println("Total:");
        System.out.println(38 + 40 + 30 +
                           (38 + 40 + 30) * .08 +
                           (38 + 40 + 30) * .15);
    }
}
```

- The subtotal expression `(38 + 40 + 30)` is repeated
- So many `println` statements
- We will use **variables** to solve the above problems.

# Variables




- **variable**: A piece of the computer's memory that is given a name and type, and can store a value.
  - Like preset stations on a car stereo, or cell phone speed dial:

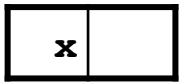

  - Steps for using a variable:
    - *Declare* it - state its name and type
    - *Initialize* it - store a value into it
    - *Use* it - print it or use it as part of an expression

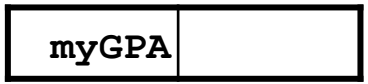
# Declaration

- **variable declaration**: Sets aside memory for storing a value.
  - Variables must be declared before they can be used.
  - Java is a **strongly typed** programming language because every variable must be declared with a data type. A variable cannot start off life without knowing the range of values it can hold, and once it is declared, the data type of the variable cannot change.
- Syntax:

  ```
  dataType variableName;
  ```

  - The variableName is also called an *identifier*.

  - `int x;` 


  - `double myGPA;` 

# Naming Variables


**Variable names must follow certain rules:**

1. They must start with a letter, $, or _
   a. **`numApples`** OR **`$numApples`** OR **`_numApples`**
   b. NOT **`5Apples`**
2. The rest of name can have letters, numbers or _
   a. **`numApples5`** OR **`num_Apples`**
3. Written in **lowerCamelCase**
   a. The first word in a variable name should be lowercase and every other word should be Uppercase.

numApples

~~numapples~~

~~Numapples~~

# Keywords

- **keyword**: An identifier that you cannot use to name a variable because it already has a reserved meaning in Java.

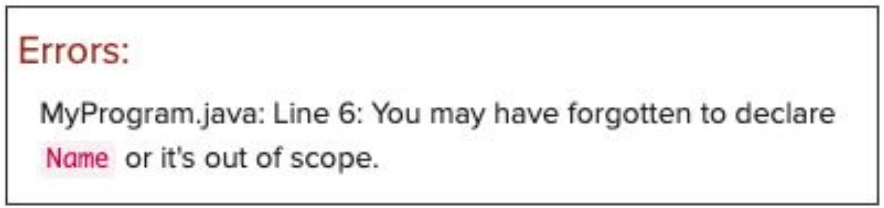
| | | | | |
|---|---|---|---|---|
| `abstract` | `default` | `if` | `private` | `this` |
| `boolean` | `do` | `implements` | `protected` | `throw` |
| `break` | `double` | `import` | **`public`** | `throws` |
| `byte` | `else` | `instanceof` | `return` | `transient` |
| `case` | `extends` | `int` | `short`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

# Other important things to note

- The name of the variable should describe the data it holds. A name like **`score`** helps make your code easier to read.
- A name like **`x`** is not a good variable name in programming, because it gives no clues as to what kind of data it holds.
- Do not name your variables crazy things like **`thisIsAReallyLongName`**, especially on the AP exam. You want to make your code easy to understand, not harder.
- Variable names are **case sensitive.**

  Example: **`name`** is a different variable than **`Name`.**

```
String name = “Karel”;
Name = “New Name!”;
```



# Data type practice

Identify any variables and their data type for each scenario below.

1. Lucy needs to keep track of her store's inventory and how much money she has made.

   ```
   double money      int inventory
   ```

2. Alex wants to write a program to keep track of his family members and some of their characteristics and their health. He wants to collect their age, weight, height and calories they consume.

   ```
   int age     double weight     int height     int calories
   ```

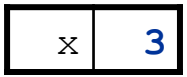
# Assignment

- **assignment**: Stores a value into a variable.
  - The value can be an expression; the variable stores its result.

- Syntax:

```
variableName = expression;
```

  - ```
    int x;
    x = 3;
    ```




  - ```
    double myGPA;
    myGPA = 1.0 + 2.25;
    ```

# Using variables

- Once given a value, a variable can be used in expressions:

```
int x;
x = 3;
System.out.println("x is " + x);      // x is 3

System.out.println(5 * x - 1);        // 14
```

string concatenation:
string + number = concatenated string(more on this later)

- You can assign a value more than once:

```
int x;
x = 3;
System.out.println(x + " here");      // 3 here

x = 4 + 7;
System.out.println("now x is " + x); // now x is 11
```

# Declaration/initialization

- A variable can be declared/initialized in one statement.

- Syntax:

  **dataType variableName = value;**

  - **double myGPA = 3.95;**


  - **int x = (12 - 3) * 2;**

# Assignment and algebra

- Assignment uses = , but it is not an algebraic equation.

  = means, *"store the value at right in variable at left"*

The right side expression is evaluated first, and then its result is stored in the variable at left.

- What happens here?

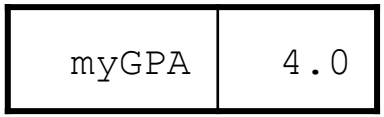
```
int x = 3; // x declared and initialized as 3
x = x + 2; // x gets what it was before plus 2, so now x is 5
```

# Assignment and Types

- The data type in the declaration must match the assigned value type!
  - `int x = 2.5;    // ERROR: incompatible types`

- An **`int`** value can be stored in a **`double`** variable.
  - The value is automatically converted into the equivalent real number.

  - **`double myGPA = 4;`**



# Compiler(syntax) errors

- Order matters.
  - ```
    int x;
    7 = x; // ERROR: should be x = 7;
    ```

- A variable can't be used until it is assigned a value.
  - ```
    int x;
    System.out.println(x);   // ERROR: x has no value
    ```

- You may not declare the same variable twice.
  - ```
    int x;
    int x;                   // ERROR: x already exists
    ```

  - ```
    int x = 3;
    int x = 5;               // ERROR: x already exists
    ```

# Printing a variable's value

- Use concatenation + to print a string and a variable's value on one line.
  - ```
    double grade = (95.1 + 71.9 + 82.6) / 3.0;
    System.out.println("Your grade was " + grade);

    int students = 11 + 17 + 4 + 19 + 14;
    System.out.println("There are " + students +
                       " students in the course.");
    ```

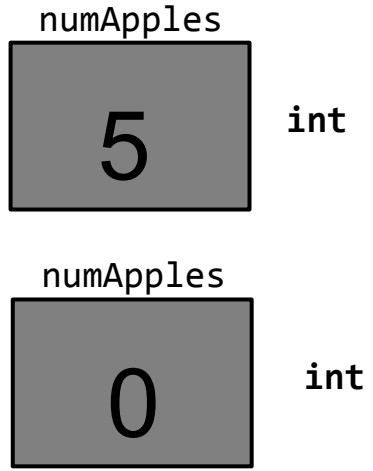
    - Output:

      ```
      Your grade was 83.2
      There are 65 students in the course.
      ```
      ```

# Variables vs Constants

Variable values can be altered after they've been initialized

```
int numApples;
numApples = 5;
numApples = 0;
```




Declaring a variable final prevents it from being altered.

```
final int numApples = 5;
numApples = 0;
```

Errors:

MyProgram.java: Line 7: variable numApples might already have been assigned

# FINAL

- The keyword **`final`** can be used in front of a variable declaration to make it a constant that cannot be changed. **Constants are traditionally capitalized.**

```
public class TestFinal
{
   public static void main(String[] args)
   {
     final double PI = 3.14;
     System.out.println(PI);
     PI = 4.2; // This will cause a syntax error
   }
}
```

# 1.3 Expressions and Output

## Learning Objectives:

- Develop code to generate output and determine the result that would be displayed.
- Develop code to utilize string literals and determine the result of using string literals.
- Develop code for arithmetic expressions and determine the result of these expressions.

- **Develop code to generate output and determine the result that would be displayed.**

System.out.print vs. System.out.println

- System.out. Print and and System.out.println is used to display information on the computer screen.
- System.out.println moves the cursor to the next line after the information has been printed on the screen whereas System.out.print prints in the information on the screen and keeps the cursor at the end of the line.

```
2  public class Unit1 {
3
4      public static void main(String[] args) {
5          System.out.println("Welcome");
6          System.out.print("to Computer");
7          System.out.println(" Science A");
8      }
9  }
10
```

Console ×

<terminated> Unit1 [Java Application] C:\Users\JR\Documents\Eclipse\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.

```
Welcome
to Computer Science
```

# System.out.println

The “System” in **System.out.println()** must be capitalized. And the command line must end with a semicolon (**;**).

Code:

```
public class Welcome{
   public static void main(String[] args){
      System.out.println("Hi there!");
      System.out.println(”Welcome to APCS A!");
   }
}
```

Output:

```
Hi There!
Welcome to APCS A!
```

# System.out.print

Do you see why there are two lines of output?

Code:

```
public class SecondClass{
   public static void main(String[] args){
      System.out.print("Hi there!");
      System.out.println(”Welcome to APCS A!");
      System.out.print(”We will learn Java!");
   }
}
```

Output:

```
Hi There!Welcome to APCS A!
We will learn Java!
```

# Indent Nicely!

```
public class Welcome{ public static void main(String[]
args){ System.out.println("Hi there!”
);System.out.println(”Welcome to APCS A!");}}
```

The code above will compile and run correctly. Java ignores whitespaces. But it is very hard to read, please make an effort to indent nicely!

```
public class Welcome{
    public static void main(String[] args){
        System.out.println("Hi there!");
        System.out.println(”Welcome to APCS A!");
    }
}
```

# string literals

- Develop code to utilize string literals and determine the result of using string literals.

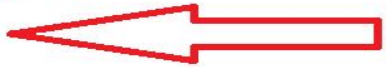
- Literal is the code representation of a fixed value.
- String literal is a set of characters enclosed in double quotes.
- Escape sequences are special sequences of characters that can be included in a string.
- They start with a \ and have a special meaning in Java.
- There many escape sequences in Java; however, the only only ones used in this course are
  - \" for double quote
  - \\ for a backslash
  - \n for a newline

# Create String objects

- String objects can be created by using string literals

```
public static void main(String[] args) {
    String greeting="Hello World!";

}
```



- By calling the String class constructor.

```
public static void main(String[] args) {
    String greeting=new String("Hello World!");

}
```

# Strings can be concatenated using the + or += operator resulting in a new String object.

```
public static void main(String[] args) {
    String s1="Hello";
    String s2="World!";
    String greeting1=s1+" "+s2; //the new string will be
                               //Hello World!

    s1+=" "+s2;
    System.out.println(s1);//will print Hello World!


}
```

Primitive values can be concatenated with a String object. This will result to a String object

```
public static void main(String[] args) {
    String s1="Hi";
    int x=5;
    String message=s1+x;
    System.out.println(message);//will display "Hi5"

}
```

# Escape Sequences in Java

Escape sequences start with a \ and have a special meaning in Java. This course uses the following escape sequences: \", \\, \n

**Ex.**

```
public static void main(String[] args) {
System.out.println("Hello World\"");//prints Hello World"

}
```

```
public static void main(String[] args) {
System.out.println("Hello World\\");//prints Hello World\

}
```

```
public static void main(String[] args) {
System.out.println("Hello\nWorld");//prints Hello
                                    //       World

}
```

# AP Practice Question

How do the escape sequences in the following code snippet format the printed output?

```
String specialItem = "Today: October 22\nwe will be serving \"dinosaur\" cookies.";
System.out.println(specialItem);
```

I. They add a new line.
II. They add quotations.
III. They add a backslash after the date.
IV. They format the word **`dinosaur`** in italics.

A. I only
B. II only
C. IV only
D. I and II
E. I and III

# Coding Examples of Escape Sequences

```
2 public class Unit1 {
3
4     public static void main(String[] args) {
5         System.out.println("A double quote \" is printed ");
6         System.out.println("One backlash \\ is printed");
7         System.out.print("This create \n a new line");
8     }
9 }
10
```

Console ×

<terminated> Unit1 [Java Application] C:\Users\JR\Documents\Eclipse\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.

```
A double quote " is printed
One backlash \ is printed
This create
 a new line
```

- Develop code for arithmetic expressions and determine the result of these expressions.

# Arithmetic Operations

Numbers are values and we can do arithmetic operations on these values. Arithmetic operators are used in mathematical expressions in the same way that they are used in algebra.

Arithmetic expressions, which consist of numeric values, variables, and operators, include expressions of type and in and double

The following table lists the arithmetic operators.
Assume integer variable **A** holds **10** and variable **B** holds **20**, then:

| Operator | Description | Example |
|---|---|---|
| + (Addition) | Adds values on either side of the operator. | **A + B** will give **30** |
| - (Subtraction) | Subtracts right-hand operand from left-hand operand | **A - B** will give **-10** |
| * (Multiplication) | Multiplies values on either side of the operator. | **A * B** will give **200** |
| / (Division) | Divides left-hand operand by right-hand operand | **B / A** will give **2** |
| % (Modulus) | Divides left-hand operand by right-hand operand and returns remainder. | **B % A** will give **0** |

An **expression** is a combination of **operands** and operators that evaluate to a single value.

```
4 + 6
```

10

```
21 - 3 * 2.0
```

15.0

```
"Hi " + "there"
```

"Hi there"

Operators can be used to construct **compound expressions** , or a combination of expressions.

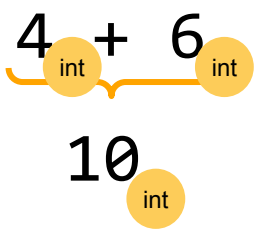
# Evaluating Arithmetic operators

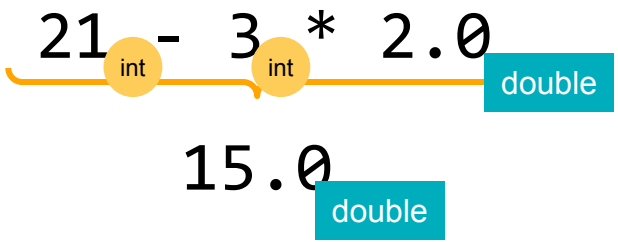
- As a program runs, its expressions are *evaluated*.
  - **`System.out.println(3 * 4);`** prints **`12`** NOT **`3 * 4`**

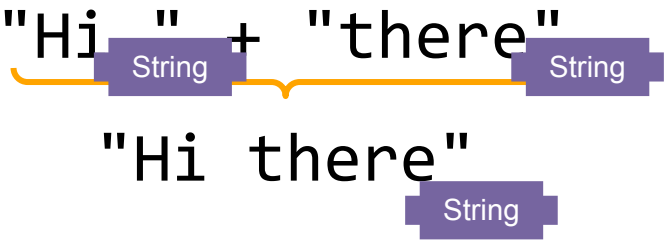
  - How would we print the text **`3 * 4`** ?
    - **`System.out.println(“3 * 4”);`**

# Evaluating Arithmetic operators

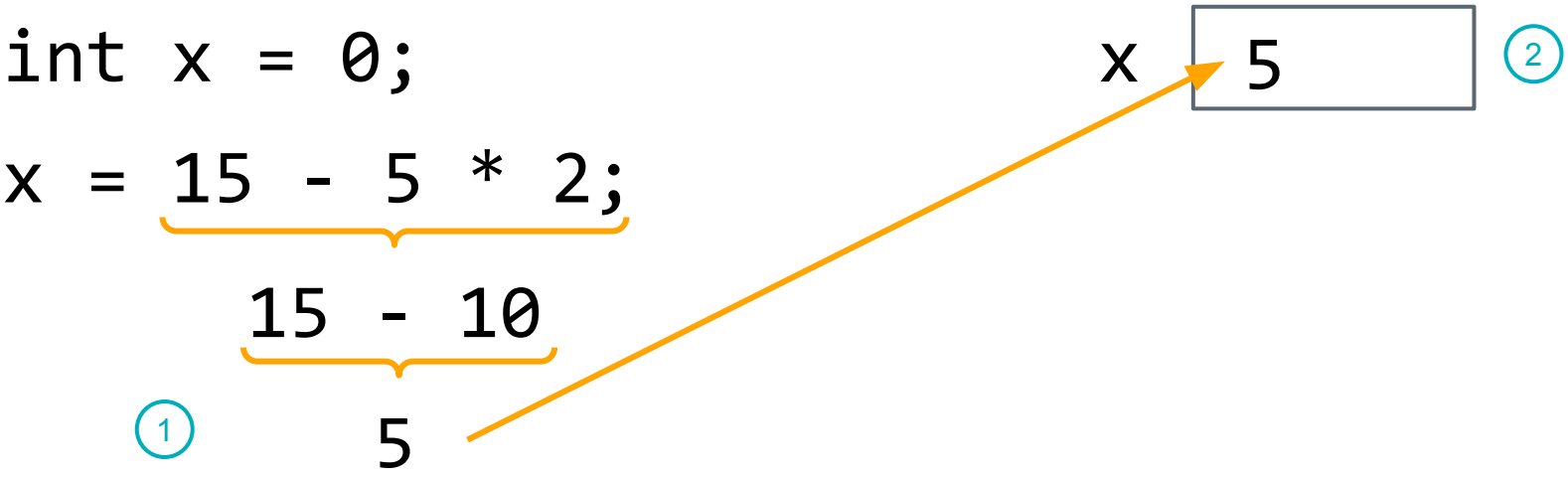
- As a program runs, its expressions are *evaluated*.
  - **`System.out.println(3 * 4);`** prints **`12`** NOT **`3 * 4`**
  - How would we print the text **`3 * 4`** ?
    - **`System.out.println("3 * 4");`**
- When we evaluate an expression, we need to pay attention to both the value and type of the operand.



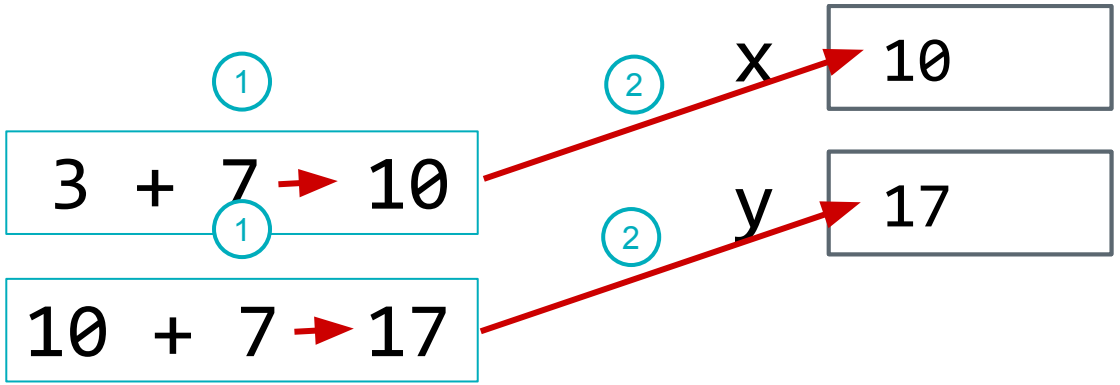



When evaluating expressions, we evaluate the statement on the right, then store the value in the variable on the left.



When we evaluate multiple expression statements in a row, we use the updated value stored in each variable for future expression evaluations.

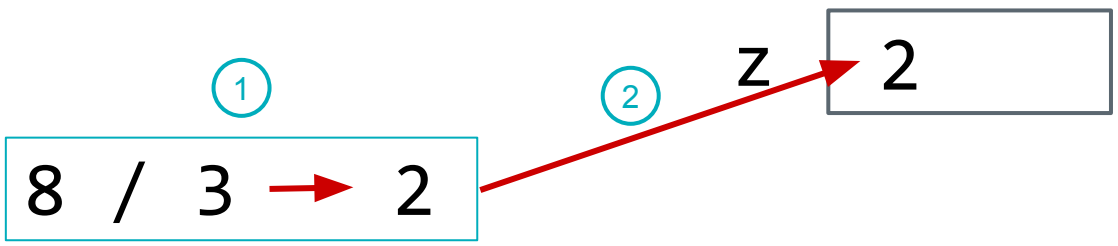
```
1  int x = 3;
2  int y = 7;
3  x = x + y;
4  y = x + y;
```



Dividing two int values will result in an int value.

The decimal portion is **truncated** , or cut off from the end.

```
1  int x = 8;
2  int y = 3;
3  int z = x / y;
```



```
int a = 7;
int b = 0;
int c = a / b;
```

Dividing an **`int`** by **`0`** will result in an **`ArithmeticException`**.

Console

```
[EXCEPTION] Your code hit an exception while it was running.
Exception message: java.lang.ArithmeticException: / by zero
```

```
double a = 7.0;
double b = 0.0;
double c = a / b;
```

If you try to divide a **`double`** by **`0`**, Java assigns the result to **`Infinity`** because of the way it handles decimal numbers.

Console

```
Infinity
```

# Integer remainder with %

The percent sign operator (**%**) is the **mod** (modulo) or remainder operator. The mod operator (**x % y**) returns the remainder after you divide **x** (first number) by **y** (second number) so **5 % 2** will return **1** since **2** goes into **5** two times with a remainder of **1**. Remember long division when you had to specify how many times one number went into another evenly and the remainder? That remainder is what is returned by the modulo operator.

**Examples:**

**5 % 2 = 1**
since
**5 / 2 = 2 Remainder 1**

```
  2 R 1
2 | 5
    4
    1
```

**15 % 5 = 0**
since
**15 / 5 = 3 Remainder 0**

```
  3 R 0
5 | 15
    15
    0
```

**2 % 3 = 2**
since
**2 / 3 = 0 Remainder 2**

```
  0 R 2
3 | 2
    0
    2
```

# Applications of % operator

- Obtain last digit of a number: **`230857 % 10`** is **`7`**
- Obtain last 4 digits: **`658236489 % 10000`** is **`6489`**
- Test for divisibility. For example, is a number even or odd? Is a number a multiple of 3?

```
// a number is even if it has no remainder
// when divided by 2.
if(number % 2 == 0){
    ...
}

// multiple of 3
if(number % 3 == 0){
    ...
}
```

# Precedence

- **precedence**: Order in which operators are evaluated.
  - Generally operators evaluate left-to-right.
    `1 - 2 - 3` is `(1 - 2) - 3` which is `-4`

  - But `* / %` have a higher level of precedence than `+ -`

    ```
    6 + 8 / 2 * 3
    6 +   4   * 3
    6 +     12   is 18
    ```

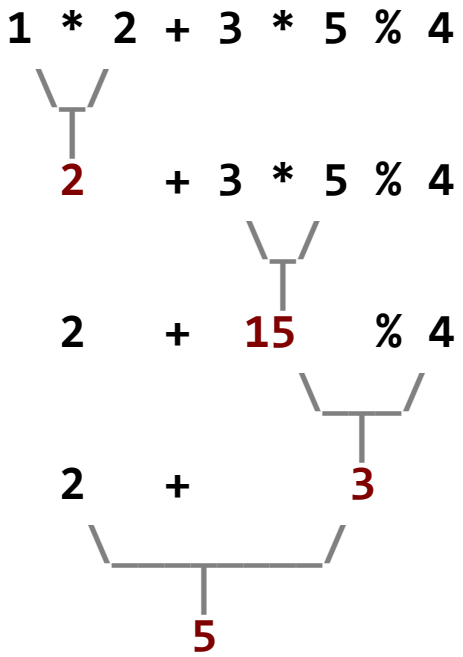
  - Parentheses can force a certain order of evaluation:
    `(1 + 3) * 4` is **16**

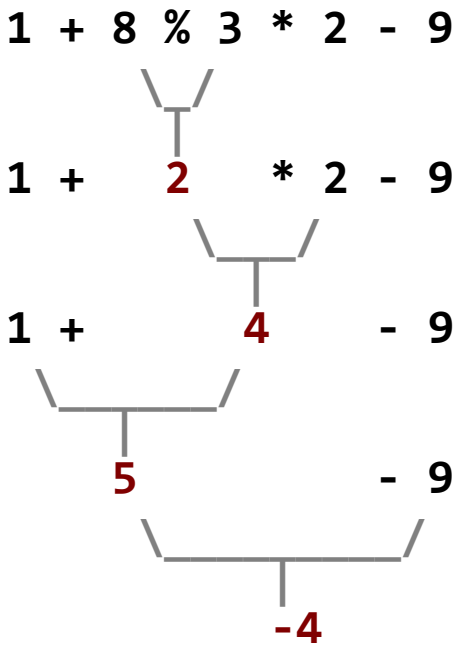
  - Spacing does not affect order of evaluation
    `1+3 * 4-2` is **11**

# Order of Operations in Java

| Precedence | Operator | Description |
|---|---|---|
| First | `( )` | parenthesis |
| Second | `++ -- ! (type)` | unary operators, logical not, typecasting |
| Third | `* / %` | multiplication, division, modulus |
| Fourth | `+ -` | addition, subtraction, string concatenation |
| Fifth | `< <= >= >` | relational operators for greater/lesser |
| Sixth | `== !=` | relational operators for equality |
| Seventh | `&&` | logical and |
| Eighth | `\|\|` | logical or |
| Ninth | `= += -= *= /= %=` | assignment operator |

# Precedence examples



# Practice Question 1

What value is the result of evaluating this expression?

```
(5 * 3 - 4 / 2 ) % 3
   (15 - 4 / 2 ) % 3
   (15 - 2 ) % 3
         13 % 3
            1
```

**(A)** 1

**(B)** 2

**(C)** 3

**(D)** 0

# Practice Question 2

Consider the following code segment.

```
int a = 3;
int b = 10;
double c = a + a / b;
```

← 3 + 3 / 10 = 3 + 0(int) = 3, stored as double 3.0

What is the value of c after this code segment is executed?

A. 0
B. 0.6
C. 3
D. 3.0
E. 3.3

# Real numbers (type double)

- Examples: **`6.022`**, **`-42.0`**, **`2.143`**
  - Placing **`.0`** or **`.`** after an integer makes it a **`double`**.

- The operators **`+ - * / % ()`** all still work with **`double`**.
  - **`/`** produces an exact answer: **`15.0 / 2.0`** is **`7.5`**

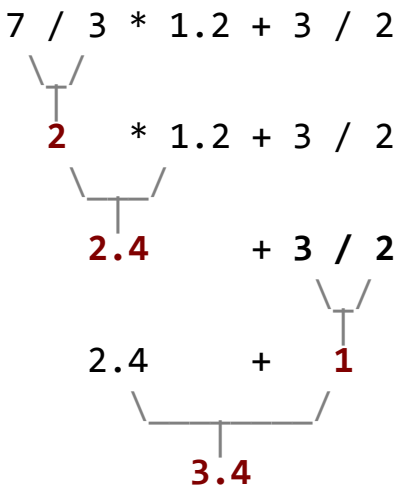
  - Precedence is the same: **`()`** before **`* / %`** before **`+ -`**

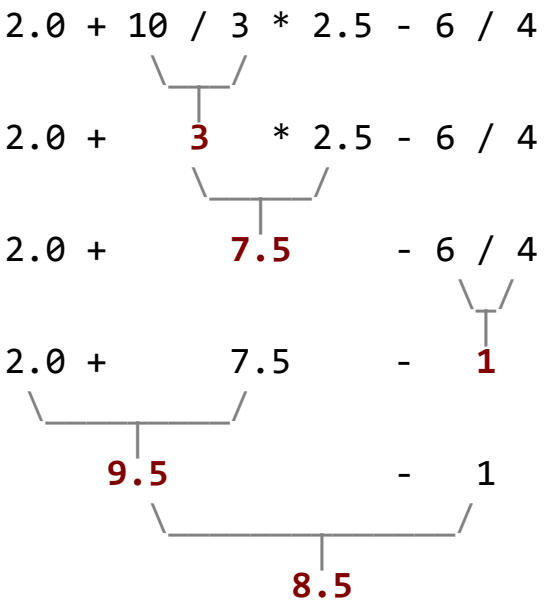
# Mixing types

- When **`int`** and **`double`** are mixed, the result is a **`double`**.

  **`4.2 * 3`** is **`12.6`**

- The conversion is per-operator, affecting only its operands.




**`3 / 2`** is **`1`** above, not **`1.5`**

# 1.4 Assignment statements and Input

## Learning Objectives:

- Develop code for assignment statements with expressions and determine the value that is stored in the variable as a result of these statements.
- Develop code to read input

- Develop code for assignment statements with expressions and determine the value that is stored in the variable as a result of these statements.

# Assignment Operators

Assignment operators are used to assign values to variables.

In the example below, we use the assignment operator (**=**) to assign the value **10** to a variable called **x**:

Example

```
int x = 10;
```

**Shortcuts can be used to modify a variable's value before assigning it.** In the example below we use the addition assignment operator (**+=**) to add a value to a variable:

Example

```
int x = 10;
x += 5;
```

**A list of all assignment 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 |

# Code Tracing

**Code Tracing** is a technique used to simulate a dry run through the code or pseudocode line by line by hand as if you are the computer executing the code. Tracing can be used for debugging or proving that your program runs correctly or for figuring out what the code actually does.

Trace tables can be used to track the values of variables as they change throughout a program. To trace through code, write down a variable in each column or row in a table and keep track of its value throughout the program. Some trace tables also keep track of the output and the line number you are currently tracing.

| | |
|---|---|
| x: | |
| y: | |
| z: | |
| output: | |

or

| x | y | z | output |
|---|---|---|---|
| | | | |

# Code Tracing Example 1

Trace through the following code:

```
int x = 0;
int y = 5;
int z = 1;
x++;
y -= 3;
z = x + z;
x = y * z;
y %= 2;
z--;
```

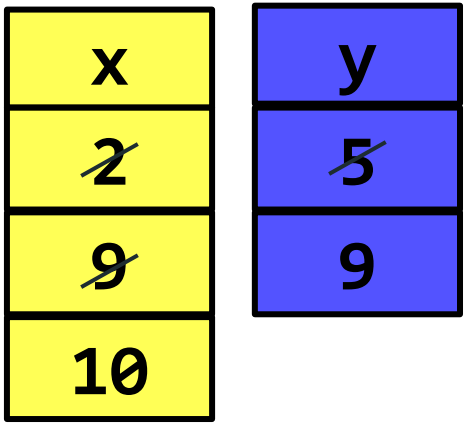
x: ~~0~~ ~~1~~ 4

y: ~~5~~ ~~2~~ 0

z: ~~1~~ ~~2~~ 1

# Code Tracing Example 2

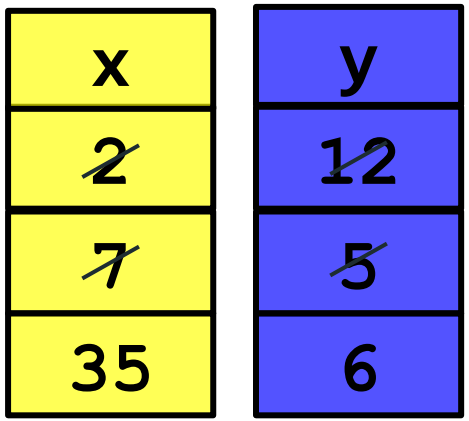
```
int x = 2;
int y = 5;
x = y + 4;
y = x;
x = x + 1;
System.out.print(x + y);
```




Output:
**19**

# Code Tracing Example 3

```
int x = 2;
int y = 12;
x += 5;
y -= x;
x *= y;
y++;
System.out.println(x);
System.out.println(y);
```




Output:

35

6

# Practice Question 1

Explain the values of the variables **mystery1**, **mystery2** and **mystery3**.

```
public static void main(String[] args){
    int num = 159;
    int mystery1 = num % 10;
    int mystery2 = num / 10 % 10;
    int mystery3 = num / 100;
}
```

Answer: **mystery1** is the ones digit, **mystery2** is the tens digit and **mystery3** is the hundreds digit of num. So, since **num** is **159**,

**mystery1 = 9**, **mystery2 = 5** and **mystery3 = 1**.

# Live Coding and assignment of Lab#1

# Develop code to read input

- Input can come in a variety of forms, such as tactile, audio, visual, or text.
- The Scanner class is one way to obtain text input from the keyboard.
- Students will be not be tested on any of form of user input mentioned above. However, using the Scanner to read text input from the user is extremely important when students are completing coding assignments on the computer.

- The Scanner class is one way to obtain text input

```java
import java.util.Scanner;

public class Unit1 {

    public static void main(String[] args) {
        Scanner input =new Scanner(System.in);
        System.out.println("How old are you? ");//Prompt the user for the age
        int age=input.nextInt();//Read the value and store it into variable gpa
        System.out.println("Please enter your gpa ");//prompt the user for the gpa
        double gpa=input.nextDouble();//Read the value and store it into variable gpa

        System.out.println("You are "+age+" Years old");//Display the content of age
        System.out.println("Your gpa is "+gpa);//Display the content of gpa
    }
}
```

Console

<terminated> Unit1 [Java Application] C:\Users\JR\Documents\Eclipse\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.5.v20221102-0933\jre\bin\javaw.exe (Jun 24, 2

```
How old are you?
20
Please enter your gpa
3.6
You are 20 Years old
Your gpa is 3.6
```

- The Scanner class is one way to obtain text input from the keyboard

```java
import java.util.Scanner;

public class StringKBinput{

    public static void main(String[] args) {
        Scanner input =new Scanner(System.in);
        System.out.println("Please your fullName ");
        String fullName=input.nextLine();
        System.out.println("Please your first name ");
        String name=input.next();

        System.out.println("Your full name is "+fullName);//Display your fullname
        System.out.println("Your first name is "+name);//Display your firstname
    }
}
```

Console ×

<terminated> StringKBinput [Java Application] C:\Users\JR\Documents\Eclipse\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.5.v20221102-0933\jre\bin\jav

```
Please your fullName
Jackson Riche
Please your first name
Jackson
Your full name is Jackson Riche
Your first name is Jackson
```

# 1.5 Casting and Ranges of Variables

## Learning Objectives:

- Develop code to cast primitive values to different primitive types in arithmetic expressions and determine the value that is produced as a result.
- Describe conditions when an integer is casting to a value that is out of range.
- Describe the conditions that limit accuracy of expressions

- Develop code to cast primitive values to different primitive types in arithmetic expressions and determine the value that is produced as a result.

## Recap: Doubles and Integers

When **doubles** and **ints** are divided, the outcome will be a **double**.

```
int first = 2;
double second = 4.0;
System.out.println(second / first);
```

→ `2.0`

What if we want this to be an **int**??

What if you want your **int** to be a **double**, or your **double** to be an **int**? The answer is **typecasting**! Casting is turning something of one type into another type!

# Type casting

- **type cast**: A conversion from one type to another.
  - To promote an **`int`** into a **`double`** to get exact division from **`/`**
  - To truncate a **`double`** from a real number to an integer

- Syntax:

```
(dataType)expression
```

Examples:

```
double result = (double)19/5; //3.8
int result2 = (int)result; //3
```

# More about type casting

- Type casting has high precedence and ***only casts the item immediately next to it***.
  - `double x = (double) 1 + 1 / 2;  // 1.0`
  - `double y = 1 + (double) 1 / 2;  // 1.5`

- You can use parentheses to force evaluation order.

  `double average = (double)(a + b + c) / 3;`
  - The code above cast the sum `(a + b + c)` into a double.

- A conversion to **`double`** can be achieved in other ways.

  `double average = 1.0 * (a + b + c) / 3;`

# Order of Operations in Java

| Precedence | Operator | Description |
|---|---|---|
| First | `( )` | parenthesis |
| Second | `++ -- ! (type)` | unary operators, logical not, typecasting |
| Third | `* / %` | multiplication, division, modulus |
| Fourth | `+ -` | addition, subtraction, string concatenation |
| Fifth | `< <= >= >` | relational operators for greater/lesser |
| Sixth | `== !=` | relational operators for equality |
| Seventh | `&&` | logical and |
| Eighth | `\|\|` | logical or |
| Ninth | `= += -= *= /= %=` | assignment operator |

# Java Type Casting

Type casting is when you convert one primitive data type to another type. There are two types:

- **Widening Casting (automatically)** - converting a smaller type to a larger type size
  `int` → `double`

  Example:

```java
int myInt = 9;
double myDouble = myInt; // Implicit casting: int to double
System.out.println(myInt);      // Outputs 9
System.out.println(myDouble);   // Outputs 9.0
```

- **Narrowing Casting (manually)** - converting a larger type to a smaller size type
  `double` → `int`

  Example:

```java
double myDouble = 9.78;
int myInt = (int) myDouble; // Manual casting: double to int
System.out.println(myDouble);   // Outputs 9.78
System.out.println(myInt);      // Outputs 9
```

# Practice Question

Consider the following code segment.

```
double num = 13 / 4;
System.out.print(num);
System.out.print(" ");
System.out.print((int) num);
```

What is printed as a result of executing the code segment?

A. `3 3`
B. `3.0 3`
C. `3.0 3.0`
D. `3.25 3`
E. `3.25 3.0`

# Integer.MAX_VALUE VS. Integer.MIN_VALUE

- The constant **Integer.MAX_VALUE** holds the value of the largest possible int value.
- The constant **Integer.MIN_VALUE** holds the value of the lowest possible int value.

- Integer values in Java are represented by values of type int, which are stored using a finite amount of byte(**4 bytes=32 bits**)

- So, an int value must be within the range of **Integer.MIN_VALUE** and **Integer.MAX_VALUE**

- If an expression is evaluated to an int value that is that outside of the allowed range, we would have what is called as **"overflow".**

- The result would still be an int value but the value that was expected

# Integer.MAX_VALUE VS. Integer.MIN_VALUE

```
1
2  public class OVeralow {
3
4      public static void main(String[] args) {
5          System.out.println(Integer.MIN_VALUE);//will print -2147483648
6
7          System.out.println(Integer.MAX_VALUE);// will print 2147483647
8
9          System.out.println(Integer.MAX_VALUE + 1);//overflow( will print -2147483648 )
10
11         }
12 }
13
```

```
<terminated> OVeralow [Java Application] C:\Users\JR\Dc
-2147483648
2147483647
-2147483648
```

# Describe the conditions that will limit the accuracy of expressions

- Computers allot a specific amount of memory to store data based on the data type.
- Computers represent decimal numbers using binary approximation.
- Some decimal numbers( **for example (⅓=0.1333333..), (⅙=0.1666666)**) cannot be represented exactly in binary form. Loss of information will occur.
    - This will cause rounding or precision errors when performing calculations.
- To avoid rounding errors that naturally occur, use int values.

# Describe the conditions that will limit the accuracy of expressions

Ex.

```
10
11              double value=0.6 + 0.3;
12              System.out.println(value);
13
14              }
15  }
16
```

Console ×

<terminated> OVeralow [Java Application] C:\Users\JR\Documents\Eclipse\eclipse

```
0.8999999999999999
```

# Describe the conditions that will limit the accuracy of expressions

```
10
11          double value=0.1 + 0.2;
12          System.out.println(value);
13
14          }
15  }
16
```

Console ×

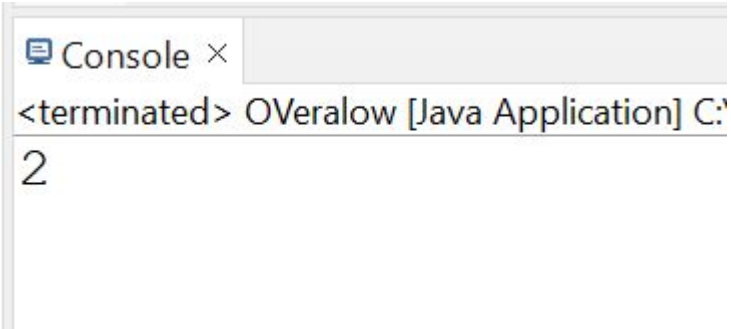
<terminated> OVeralow [Java Application] C:\Users\JR\Documents\Eclipse\eclipse\

```
0.30000000000000004
```

# Describe the conditions that will limit the accuracy of expressions

## Integer division in Java:

```
1
2 public class OVeralow {
3
4     public static void main(String[] args) {
5
6         int value=5;
7         System.out.println(value/2);
8
9         }
10 }
11
```



# Describe the conditions that will limit the accuracy of expressions

- Whenever we are using decimal types in java, the limitation of binary approximation can lead to rounding errors which will affect the accuracy of expressions.
- Recommended solutions for the purpose of this course, use int type whenever possible.

# 1.6 (Compound Assignments and Operators)

## Learning Objectives:

Develop code for assignment statements with compound assignment operators and determine the value that is stored in the variable as a result.

- Operators can be used to construct **compound expressions** , or a combination of expressions.
- Compound assignment operators **+=, -=, *=, /=.** and **%=** can be used in place of the assignment operator in numeric expressions.
- The post increment operator **++** and post decrement operator **--** are used to **add 1** or **subtract 1** from the stored value of a numeric variable. The new value is then assigned to the variable.

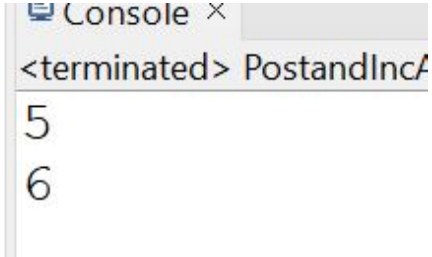
# Compound Operators

```
2  public class CompoundOps {
3
4      public static void main(String[] args) {
5
6          int w=9;
7          int x=5;
8          int y=3;
9          int z=0;
10         x+=10;
11         y-=x;
12         z%=3;
13         w*=2;
14         System.out.println("x:"+x);
15         System.out.println("y:"+y);
16         System.out.println("z:"+z);
17         System.out.println("w:"+w);
18         w/=3;
19         System.out.println("w:"+w);
20     }
21 }
```

```
<terminated> CompoundOps [Java Application] C:\
x:15
y:-12
z:0
w:18
w:6
```

# Post INCREMENT(++)

```
2 public class PostandIncAndDec {
3
4     public static void main(String[] args) {
5         int x=5;
6         System.out.println(x++);
7         System.out.println(x);
8
9     }
10
11 }
12
```



# POST DECREMENT (--)

```
2  public class PostandIncAndDec {
3
4      public static void main(String[] args) {
5          int x=5;
6          System.out.println(x--);
7          System.out.println(x);
8
9      }
10
11 }
12
```

Console ×

<terminated> PostandIncAndDec [Java App

```
5
4
```

# 1.7 (Application Program Interface (API) and Libraries

*Learning Objectives:*

Identify the attributes and behaviors of a class found in the libraries contained in an API.

Key things to know for this section:

- Libraries are collections of classes
- An application programming interface(API) specification informs the programmer how to use those classes.
- Documentation found in API specification and libraries is essential to understanding the attributes and behaviors of a class defined by the API.
- A class defines a specific reference type
- Classes in the APIs and libraries are group into packages.
- Existing classes and class libraries can be utilized to create objects/instances
- Attributes refer to the data related to the class and are stored in variables.
- Objects have attributes and behaviors
- Attributes describe the objects/instances
- Behaviors refer to what the objects/instances of the class can do(or what can be done with them).
- Behaviors are defined by objects/instances.

# Java API, Classes, Packages, and Libraries

- The **Java API** is a collection of prewritten classes and interfaces
- The Java API is actually blueprint(plan) and documentation of all built-in classes, interfaces, along with their methods.
  - The Math, Scanner, and String classes are port of the Java API
- A **class** a blueprint for creating objects
  - It defines the attributes(**variables/fields**) and methods(**behaviors**)which represent the behavior and state of the the object.
- A package contains multiple classes and interfaces
  - The Java API is organized into packages
  - **For ex**.: the **Java.util** package contains utility classes such as Scanner, ArrayList, and Random classes.
  - **java.io** package contains classes for input, and output, such as FileReader and BufferedReader

# Java API, Classes, Packages, and Libraries

- The Java library is a collection of packages and classes that provides additional functionalities.
- It can be part of the standard Java API or others added by developers.
- A library:
  - Contains packages
- A package:
  - Contains classes and interfaces
- The Java API if the main standard library provided by Java, which contains all the packages and classes.

# Java API, Classes, Packages, and Libraries

## In summary:

- **Library:** Like a *jewelry box*
- **Package:** Like a *drawer* in the jewelry box for organizing your jewelries.
- **Class:** One of your jewelries, such a watch or necklace.
- **Java API:** The *instruction manual* that tells you what jewelries are in the jewelry box and how to use them.

# 1.8 (Documentation with comments)

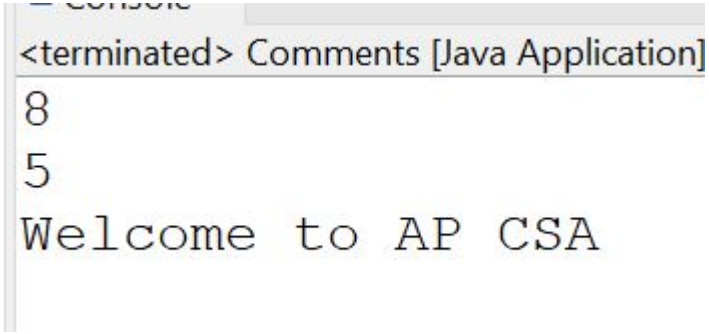
## Learning Objectives:

Describe the functionality and use of code trough comments

- **Comments** are written for both the original programmer and other programmers to understand the code and its functionality.
- Comments are ignored by the compiler and are not executed when the program is run.
- There are three types of comments in Java:
  - // which is used to comment one line at a time
  - /* */ which is used to comment out a block of code
  - /** */ which are javadoc comments and are used to create API documentation.
    - The Javadoc type of comments are usually all other the exam.
- A **precondition** is a condition that must be true just before the execution of a method in order for it to behave as expected.
  - In this situation, we are not expecting expecting that the method will check to ensure that preconditions are met
- A **postcondition** is a condition that must always be true after the execution of of a method.
  - Postconditions describe what is being returned or the current value of the attributes of an object.

# EXAMPLES OF COMMENTS

```
2 public class Comments {
3
4     public static void main(String[] args) {
5         int a=8, b=5;//declaring and initializing to 8 and 5 respectively
6         int c=a-7;
7         String message="Welcome to AP CSA";
8
9         System.out.println(a);
10        System.out.println(b);
11        System.out.println(message);
12
13
14    }
15
16 }
17
```



# EXAMPLES OF COMMENTS

```
public class Comments {

    public static void main(String[] args) {
        int a=8, b=5;//declaring and initializing to 8 and 5 respectively
        int c=a-7;
        String message="Welcome to AP CSA";

        System.out.println(a);
        /*
        System.out.println(b);
        System.out.println(message);
        */

    }

}
```

# Javadocs

Java has a tool called **Javadocs** that comes with the Java JDK that will pull out all of these comments to make API documentation of a class as a web page. It is created by writing comments inside `/** */` and using `@` tags.

Description of the code segment

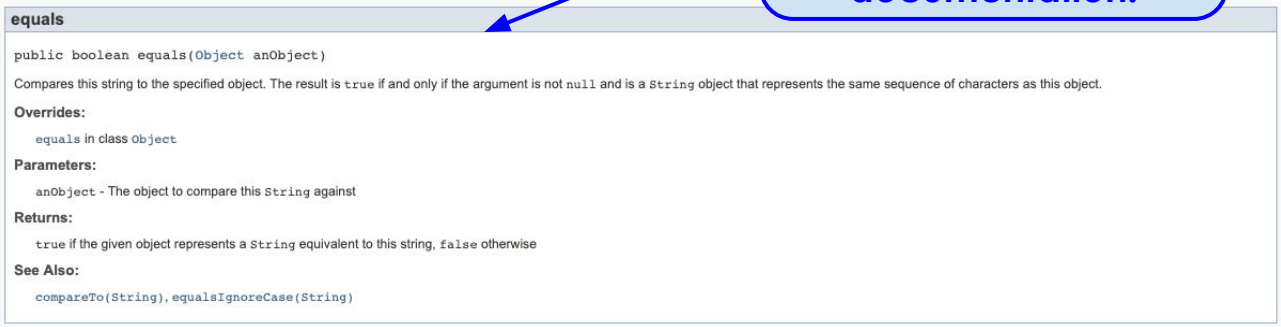
**@param** tags identify the parameters and explain what they represent

```
/**
 * Launch a URL from an intent.
 *
 * @param url           The url from the intent.
 * @param referer       Optional referer URL to be used.
 * @param headers       Optional headers to be sent when opening the URL.
 * @param externalAppId External app id.
 * @param forceNewTab   Whether to force the URL to be launched in a new tab or to fall
 *                      back to the default behavior for making that determination.
 * @param isRendererInitiated Whether the intent is originally from browser renderer process.
 * @param initiatorOrigin Origin that initiates the intent.
 * @param intent        The original intent.
 */
private Tab launchIntent(
        LoadUrlParams loadUrlParams, String externalAppId, boolean forceNewTab, Intent intent) {
```

This code format makes it easy for developers to create documentation, because the Javadocs program automatically translates the Javadoc comments into the API format. Here we can see that the documentation for equals is being translated from the Javadoc format into the proper API documentation. Let's take a look at how we can implement this ourselves.

```
/**
   Compares this string to the specified object. The result is true if and only if the argument is not null
   and is a String object that represents the same sequence of characters as this object.
   @param anObject - The object to compare this String against
   @return true if the given object represents a String equivalent to this stri
   @see compareTo(String), equalsIgnoreCase(String)
*/
public boolean equals(Object anObject)
{
}
```

**Automatically converted to this format in documentation!**



**equals**

`public boolean equals(Object anObject)`

Compares this string to the specified object. The result is `true` if and only if the argument is not `null` and is a `String` object that represents the same sequence of characters as this object.

**Overrides:**

`equals` in class `Object`

**Parameters:**

`anObject` - The object to compare this `String` against

**Returns:**

`true` if the given object represents a `String` equivalent to this string, `false` otherwise

**See Also:**

`compareTo(String)`, `equalsIgnoreCase(String)`

# Examples of comments (Javadoc)

```
/**
 * Constructor to create an employee object with its
 * initial name and salary
 * @param empName the name of the employee
 * @param sal the current salary of the employee
 */
    public Employee(String empName, double sal) {
        name=empName;
        salary=sal;

    }
```

# Examples of precondition and postcondition

```
2  public class Bank {
3  /**
4   * To calculate the sum of a number plus 5;
5   * @number the positive initial number
6   * @return the sum of number + 5
7   */
8      public int getSum5(int number) {
9  //Pre-condition, number must be non-negative
10         if(number <0) {
11             System.out.println("Number is negative. Try again");
12         return 0;
13         }
14
15         int sum=0;
16         sum=number + 5;
17 //Postcondition: calculated sum must be greater than 10
18         if(sum>10) {
19         return sum;
20         }
21         else {
22         return -1;
23         }
24     }
```

# More Examples postcondition

```
2  public class PrePostCondition {
3
4      //post-condition: When computePower is called,
5      //The result is guaranteed to calculate of a given number
6      public double  computeSquareroot(int number ) {
7          return Math.sqrt(number);
8      }
9
10     public static void main(String[] args) {
11
12         PrePostCondition cond=new PrePostCondition();
13         System.out.println(cond.computeSquareroot(49));//print 7.0
14         System.out.println(cond.computeSquareroot(24));//print 4.898979485566356
15         }
16     }
17
```

```
Console
<terminated> PrePostCondition [Java Application] C:\Users\JR
7.0
4.898979485566356
```

# In Summary

| Condition | Responsibility | When | Example |
| --- | --- | --- | --- |
| Precondition | Caller | Before method | `sum >= 0` **in getSum5** |
| Postcondition | Method | After execution | Return is **sqrt(number)** |

# 1.9 (Method Signatures)

*Learning Objectives:*

1. Identify the correct method to call based on documentation and method signature
2. Describe how to call methods.

- **Important terms to know for this section:**
  - Method
  - Method signature
  - Parameter
  - Void method
  - Non-void method
  - Argument
  - Method definition/implementation
  - Method calling
  - Procedure abstraction

# (Method Signatures)

- The method signature is the part of the method that contain its name along with the parameter list(types or order). Neither the return type or the access modifier is part of the method signature.
- Ex.
  - **computeSum()**
  - **computeSum(int num1)**
  - **computeSum(int num1, int num2)**
  - **computeSum(int num1, String message)**

# METHOD HEADER VS. METHOD SIGNATURE

## Method header:

**Public static void computeSum(int num1, int num2)**

## Method signature:

**computeSum(int num1, int num2)**

**Method header** = Full declaration line before the body.

**Method signature** = Just the method name and parameter types (order matters). Return type and modifiers are **not** part of the signature.

# 1.10 (Calling Class Methods)

*Learning Objectives:*

Develop code to call class methods and determine the result of those calls.

- Class methods are associated with the class, not the instance of the class(object).
- The keyword static is included in the method header before the method name.
- Class methods are usually called using the name of the class with the dot (.) operator.
- When the method call happens in the defining class, using the class name is optional in the call.

# What is a method?

- **A method is a block of code with a name which can be called to be executed.**
- **We call a method by referring to its name**

```
public class Calculator {

//This is an example of a class method definition

   public static void computeSum(int num1,int num2) {

       int sum=num1+num2;

       System.out.println("The sum of the two numbers is "+sum);

   }

   }
```

# Calling class methods

```
2 public class Calculator {
3
4     public static void computeSum(int num1,int num2) {
5         int sum=num1+num2;
6         System.out.println("The sum of the two numbers is "+sum);
7     }
8
9     public static void main(String[] args) {
10        Calculator.computeSum(5, 3);//Calling the class method computeSum
11                                    //which is defined on lines 4 through 7
12        computeSum(10, 7);//Another way to call it since it was defined inside
13                          //inside the class Calculator
14    }
15 }
```

```
<terminated> Calculator (3) [Java Application] C:\Users\JR\Documents\
The sum of the two numbers is 8
The sum of the two numbers is 17
```

# Class methods vs. Instance Methods

| Feature | Class Method ( `static` ) | Instance Method |
| --- | --- | --- |
| Belongs to | The **class** | The **instance (object)** |
| Shared by all instances? | ✅ Yes | ✅ The definition is shared, but it acts on different instance data |
| Access instance variables? | ❌ No | ✅ Yes |
| Call syntax | `ClassName.method()` | `object.method()` |

# 1.11 (Math Class)

*Learning Objectives:*

Develop code to write expressions that incorporate calls to built-in mathematical libraries and determine the value that is produced as a result.

- Class methods are associated with the class, not the instance of the class(object).

**Call static methods (Methods of Math Class)**

Below are the methods of the class required for this course.

- `int abs(int x)` — Returns the absolute value of an `int` value
- `double abs(double x)` — Returns the absolute value of a `double` value
- `double pow(double base, double exponent)` — Returns the value of the first parameter raised to the power of the second parameter
- `double sqrt(double x)` — Returns the positive square root of a `double` value
- `double random()` — Returns a double value greater than or equal to 0.0 and less than 1.0

The values returned from `Math.random` can be manipulated to produce a random `int` or `double` in a defined range.

## **Call static methods (**Methods of  Math Class**)**

```
3⊖    public static void main(String[] args) {
4
5 System.out.println(Math.abs(-5));//will print 5
6 System.out.println(Math.abs(-5.0));//will print 5.0
7 System.out.println(Math.pow(5.0, 2.0));//will print 25.0
8 System.out.println(Math.sqrt(49.0));//will print 7.0
9 System.out.println(Math.random());//will print a random number
10                                   //from 0.0(inclusive) to 1(exclusive).
11
12
13    }
```

## Call static methods (Methods of Math Class)

The values returned from `Math.random` can be manipulated to produce a random `int` or `double` in a defined range.

**Ex.**

```
3    public static void main(String[] args) {
4 //This statement will print a random number from 0 to 4
5  System.out.println((int)(Math.random()*5));
6 //This statement will print a random number from 1 to 5
7  System.out.println((int)(Math.random()*5)+1);
8 //This statement will print a random number from 2 to 11
9  System.out.println((int)(Math.random()*10)+2);
10
11
12     }
```

# ASSIGNMENT OF LAB#2

# 1.12 (Objects: Instances of Classes)

*Learning Objectives:*

1. Explain the relationship between a class and an object
2. Develop code to declare variables to store reference types

# Classes vs. Objects

- Consider, for example, an **Employee** class designed for a company's software application to handle employees' information and roles.
- The software developer will design a blueprint/class by taking into consideration the attributes/characteristics that all employees will have in common:
  - Each employee of the company will have a ***name, id, phone, salary/payRate, isFullTime, isActive,* etc**. These are called **attributes** which are used to describe each employee object.
  - When an employee object is created from the Employee class, it will have its own copy of these attributes attached to it.
  - Each employee of the company will also have a set of behaviors/actions**, such as clockIn( ), clockOut(), checkSchedule(), requestDayOff(), getName(), changeName() etc**. These are referred to as **methods** which are used to define a set of behaviors for each employee object.
  - For example, an employee called Jone Doe which was created from the Employee class may need to request for a day off, so the **requestDayOff()** method from the application will allow him to request for that day off.

# Class as a type

- A **class** is like a **blueprint** from which an object is created.
- We can create many objects from the class.
- **Objects** are specific **instance** of the general class blueprint.
- The differences among these objects are the attribute values (**data**) that define each objects' state.
- **For example**, a class **Student** might be used to create many student objects.
  - All these objects will have attributes that are generally common to  a student (e.g., name, course enrolled, studID, etc).
  - But each object(student) will have its own values for these attributes, depending on which student it represents. (Later we will see how we can define new classes.)

# What does a Class Define?

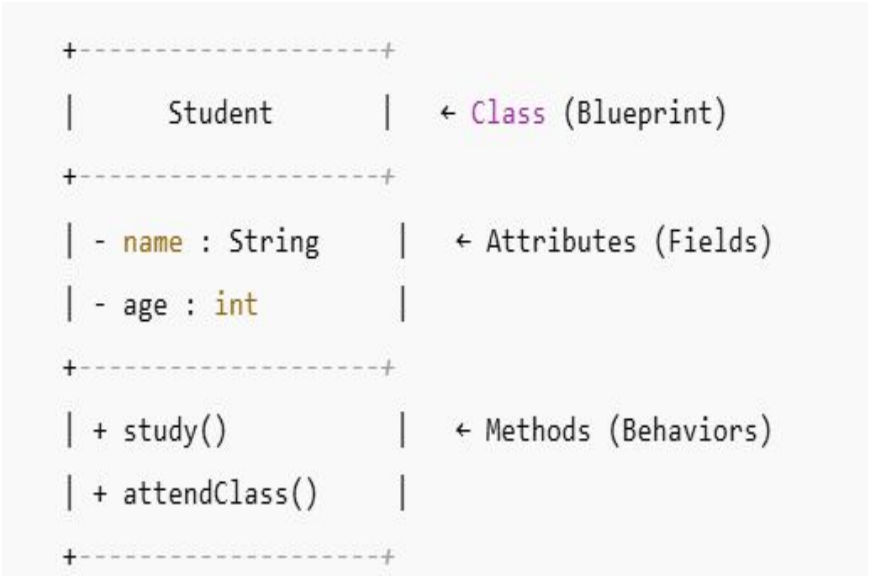
- The **attributes** that describe the objects created from the class:
  - An object's attributes are the data values that describe the object.
  - These are called **instance variables**

- The **behaviors** for the objects created from the class:
  - An object's behaviors are defined by a set of actions associated with the object.
  - For example, methods may enable you to access or change an object's attribute values, or to ask the object to perform a task.
  - These are called **methods**

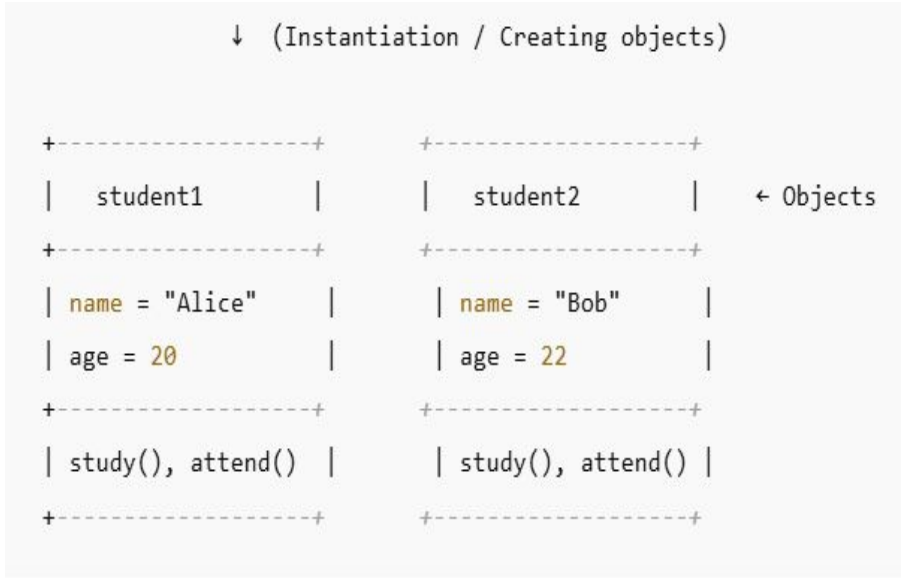
- The **constructor** which is used to create an object of the class and is also used to initialize the attributes of the object that gets created.

# Relationship between classes and objects (UML diagram)

## Class




## Object



# Classes vs. Objects (attributes)

- Below is an example of the declaration of some attributes inside the Employee class.

```
1
2  public class Employee {//beginning of class
3      //lines 5 to 8 show examples of some attributes
4      //declared for the Employee class.
5      String name, id;
6      double salary;
7      boolean isFullTime;
8      double payRate;
9
10     public static void main(String[] args) {//beginning of the main method
11
12
13
14     }//end of the main method
15
16 }//end of class
17
```

# Classes vs. Objects (methods)

- Examples of two methods declared within the Employee class

```
2  public class Employee {//beginning of class
3      //lines 5 to 8 show examples of some attributes
4      //declared for the Employee class.
5      String name, id;
6      double salary;
7      boolean isFullTime;
8      double payRate;
9
10     //Examples of 2 defined methods within the Employee class to retrieve and change the name
11     public String getName() {
12         return name;
13     }
14     public void setName(String newName) {
15         name=newName;
16     }
17     public static void main(String[] args) {//beginning of the main method
18
19
20     }//end of the main method
21
22 }//end of class
```

# 1.13 (Objects Creation and storage (Instantiation)

*Learning Objectives:*

1. Identify using its signature the correct constructor being call.
2. Develop code to declare variables of the correct types to hold object references
3. Develop code to create an object by calling a constructor

# Constructors ( What is it? )

- In order to understand constructors, we need to understand what a method is. A method, ***which is often called function or procedure in other programming languages***, is a set of instructions with a name can be called anywhere in a program (see slide #7 for code example of methods)
- A constructor is a specialized method used to create objects from a class.
  - It must have the same name as the class
  - It must not have any return type
  - The signature of a constructor contains of its name and the list of its parameter Types (if any).
  - A default constructor is a constructor that has no parameter.
  - It is not necessary to define a default constructor as Java will create one for you.

# Constructors ( example )

Lines 11 to 15 show a constructor defined without parameters(default constructor, and it is used to initialize some of the attributes of the class Employee.

```
2  public class Employee {//beginning of class
3      //lines 5 to 8 show examples of some attributes
4      //declared for the Employee class.
5      String name, id;
6      double salary;
7      boolean isFullTime;
8      double payRate;
9
10     //Defining a default constructor for the Employee class
11      public Employee() {
12          name="Bryan Singer";
13          id="A12345";
14          isFullTime=false;
15      }
16
17
18     //Examples of 2 defined methods within the Employee class to retrieve and change the name
19     public String getName() {
20         return name;
21     }
```

# Constructors( without parameters )

- Calling a constructor without parameter

```
25      public static void main(String[] args) {//beginning of the main method
26          //calling a constructor without parameter
27          Employee emp1=new Employee();//calling the default constructor defined on line 11
28                                       //this will create an employee whose name is Bryan Singer
29      }//end of the main method
30
31  }//end of class
```

# Constructors ( with parameters )

```
29  public static void main(String[] args) {//beginning of the m
30      //calling a constructor without parameter
31      Employee emp1=new Employee();//calling the default const
32      //this will create an employee whose name is Bryan Singe
33      Employee emp2=new Employee("Wilna Brown","A23456");
34      //Calling the constructor Employee as defined on line 33
35      //this will create a full-time employee whose name
36      // is Wilna Brown
37
38  end of class
```

**How many objects can you create from a class in Java?**

A. 1

B. 10

C. 1000

D. As many as you need

**What are the data or properties of an object called?**

A. attributes

B. methods

C. class

D. object

**What specifies the behavior for objects of a class in Java?**

A. attributes
B. methods
C. class
D. object

**What best describes the purpose of a class's constructor?**

A. Determines the amount of space needed for an object and creates the object

B. Names the new object

C. Return to free storage all the memory used by this instance of the class.

D. Initialize the instance variables in the object

**A student has created a `Dog` class. The class contains variables to represent the following.**

- A String variable called `breed` to represent the breed of the dog
- An int variable called `age` to represent the age of the dog
- A String variable called `name` to represent the name of the dog

The object `pet` is declared as type Dog. Which of the following descriptions is accurate?

**A. An attribute of the name object is String.**
**B. An attribute of the pet object is name.**
**C. An instance of the pet class is Dog.**
**D. An attribute of the Dog instance is pet.**
**E. An instance of the Dog object is pet.**

**A student has created a `Party` class. The class contains variables to represent the following.**

- An int variable called `numOfPeople` to represent the number of people at the party.
- A boolean variable called `discoLightsOn` to represent whether the disco ball is on.
- A boolean variable called `partyStarted` to represent whether the party has started.

The object `myParty` is declared as type Party. Which of the following descriptions is accurate?

**A. boolean is an attribute of the myParty object.**
**B. myParty is an attribute of the Party class.**
**C. myParty is an instance of the Party class.**
**D. myParty is an attribute of the Party instance.**
**E. numOfPeople is an instance of the Party object.**

# 1.14 (Calling Instance Methods)

*Learning Objectives:*

1. Develop code to call instance methods and determine the result of these calls.

- Instance methods are called on objects of the class.
- The dot(.) operator is used along with the object name to call instance methods.
- A method call on a null reference will result in a NullPointerException.

# Important terms to know for this section

- Methods
- Method definition
- Method calling
- Instance methods
- Class methods
- Parameter
- Formal parameter
- Argument

# What is a method?

- **A method is a block of code with a name which can be called to be executed.**
- **We call a method by referring to its name**

```
public class Calculator {

//This is an example of an instance method definition

    public void computeSum(int num1,int num2) {

        int sum=num1+num2;

        System.out.println("The sum of the two numbers is "+sum);

    }

    }
```

# Calling instance methods

```
2  public class Calculator {
3
4      public void computeSum(int num1,int num2) {
5          int sum=num1+num2;
6          System.out.println("The sum of the two numbers is "+sum);
7      }
8
9      public static void main(String[] args) {
10
11         Calculator cal=new Calculator();
12         cal.computeSum(5, 3);//Calling instance method computeSum
13                              //which is defined on line 4 through 7
14     }
15 }
```

```
<terminated> Calculator (3) [Java Application] C:\Users\JR\Document
The sum of the two numbers is 8
```

# Calling instance methods( without parameters )

```
public void printSalary() {
    System.out.println("Your current salary is "+salary);
}

public static void main(String[] args) {//beginning of
    //calling a constructor without parameter
    Employee emp1=new Employee();//calling the default
    //this will create an employee whose name is Bryan
    Employee emp2=new Employee("Wilna Brown","A23456");
    //Calling the constructor Employee as defined on li
    //this will create a full-time employee whose name
    //is Wilna Brown
    //Printing the salary for emp1
    emp1.printSalary();
```

# Calling instance methods( with parameters )

```
29  public static void main(String[] args) {//beginning of the main m
30      //calling a constructor without parameter
31      Employee emp1=new Employee();//calling the default constructo
32      //this will create an employee whose name is Bryan Singer
33      Employee emp2=new Employee("Wilna Brown","A23456");
34      //Calling the constructor Employee as defined on line 33
35      //this will create a full-time employee whose name
36      // is Wilna Brown
37      System.out.println("Name of employee before change:");
38      System.out.println(emp1.getName());//call method w/o paramete
39      emp1.setName("Wilna Jonhson");//call method with parameter
40      System.out.println("Name of employee before change:");
41      System.out.println(emp1.getName());
42       }//end of main method
```

```
Name of employee before change:
Bryan Singer
Name of employee after change:
Wilna Jonhson
```

# Calling instance methods( with return value )

```
public String getName()  {
     return name;
}

//Method to increase salary by a percentage
public double increaseSalary(double percentInc)  {
     salary=salary+ salary*percentInc;
     return salary;
}
```

```
38 public static void main(String[] args) {//beginning of the
39     //calling a constructor without parameter
40     Employee emp1=new Employee();//calling the default con
41     //this will create an employee whose name is Bryan Sin
42     Employee emp2=new Employee("Wilna Brown","A23456");
43     //Calling the constructor Employee as defined on line
44     //this will create a full-time employee whose name
45     //is Wilna Brown
46
47     System.out.println(emp2.getName());
48
```

# Class methods vs. Instance Methods

| Feature | Class Method ( `static` ) | Instance Method |
| --- | --- | --- |
| Belongs to | The **class** | The **instance (object)** |
| Shared by all instances? | ✅ Yes | ✅ The definition is shared, but it acts on different instance data |
| Access instance variables? | ❌ No | ✅ Yes |
| Call syntax | `ClassName.method()` | `object.method()` |

# 1.15 (String Manipulation)

*Learning Objectives:*

1. Develop code to create string objects and determine the result of creating and combining strings
2. Develop code to call methods on string objects and determine the result of calling these methods

# Calling String methods (Recall Java API)

- Application program interface (**API**) and libraries simply complex programming tasks by allowing software developers to use them without the need to recreate them when creating software applications.
- Java has many libraries with built-in methods that are made available to programmers to use. Those methods have already been tested.
- Programmers need to read the documentation for those APIs and libraries in order to have a good understanding of the attributes and behaviors of an object of a class.
- Those classes in the APIs and libraries are grouped into **packages.**
- The String class is part of the **java.lang** package.
- Classes in the Java.lang package are available by default.

# Recall: Strings

- **String**: A sequence of characters to be printed.
  - Starts and ends with a " quote " character.
    - The quotes do not appear in the output!!
  - Examples:


```
"hello"
"This is a string.  It's very long!"
```

- String Restrictions:
  - May not span multiple lines.

```
"This is not
a legal String."
```

  - May not contain a " character.

```
"This is not a "legal" String either."
```

# The String object

- Each character in a String object is represented by an integer value. The first character of a String is always at index 0 whereas the last character is located at the length of the string subtracted by 1.

**Ex.**


- As shown above, the total number of characters in the string is 13, but the last character (**!**) is at index 12.
- Any attempt to access an index value outside the range of this string will result in an indexOutOfBoundsExceptions.
- A String object can be concatenated with an object reference which leads to a call to the toString() method of the referenced object.

# Calling String methods

The following String methods are those that will be used for this course.

- `String(String str)` — Constructs a new `String` object that represents the same sequence of characters as `str`
- `int length()` — Returns the number of characters in a `String` object
- `String substring(int from, int to)` — Returns the substring beginning at index `from` and ending at index `to - 1`
- `String substring(int from)` — Returns `substring(from, length())`
- `int indexOf(String str)` — Returns the index of the first occurrence of `str`; returns –1 if not found
- `boolean equals(String other)` — Returns `true` if `this` is equal to `other`; returns `false` otherwise
- `int compareTo(String other)` — Returns a value < 0 if `this` is less than `other`; returns zero if `this` is equal to `other`; returns a value > 0 if `this` is greater than `other`

# Calling String methods

- `int length()` — Returns the number of characters in a `String` object

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

    String s1="Hello world";
    System.out.println(s1.length());//will print 11
    System.out.println("Welcome".length());//will print 7

}
```

- `boolean equals(String other)` — Returns `true` if `this` is equal to `other`; returns `false` otherwise

```
public static void main(String[] args) {
String s1="Hello";
System.out.println(s1.equals("Hello"));//prints true
System.out.println(s1.equals("hello"));//prints false
System.out.println(s1.equals("Hell"));//prints false

}
```

# Calling String methods

- `String substring(int from)`

The substring method has 2 versions:
The first version is the one shown above. When called, it returns a substring from the character located at valued stored in from up to the last character of string being referenced.
**Ex.**

```
    public static void main(String[] args) {
    String s1="Hello, World!";
//This statement will print the entire string:Hello, World!
    System.out.println(s1.substring(0));
//This statement will print:ello, World!
    System.out.println(s1.substring(1));
//This statement will print:d!
    System.out.println(s1.substring(s1.length()-2));

    }
```

# Calling String methods

- `String substring(int from, int to)` — Returns the substring beginning at index `from` and ending at index `to - 1`

The second version is the one shown above. When called, it returns a substring starting at the character located at the value stored in variable **from** and up to the character located at index **to -1** of the string being referenced.

**Ex.**

```
    public static void main(String[] args) {
    String s1="Hello, World!";
//This statement will print the string:Hel
    System.out.println(s1.substring(0,3));
//This statement will print:o, W
    System.out.println(s1.substring(4,8));
//This statement will print:ello, Wor
    System.out.println(s1.substring(1, s1.length()-3));

    }
```

# Calling String methods

- `int compareTo(String other)` — Returns a value < 0 if `this` is less than `other`; returns zero if `this` is equal to `other`; returns a value > 0 if `this` is greater than `other`

```
public class Test {
    public static void main(String[] args) {
        String a = "Apple";
        String b = "Banana";

        int result = a.compareTo(b);

        System.out.println(result); // Output: negative because "Apple" < "Banana"
    }
}
```

`"Apple".compareTo("Banana")` returns a **negative number** because `"Apple"` comes **before** `"Banana"`

# Calling String methods

- `String(String str)` — Constructs a new `String` object that represents the same sequence of characters as `str`

This is the String constructor which creates a new String object which is a copy of the of the original String object.

```
3       public static void main(String[] args) {
4       String s1="Hello World!";
5       String aStringCopy=new String(s1);
6   //This statement will print:Hello world!
7       System.out.println(s1);
8   //This statement will print:Hello world!
9       System.out.println(aStringCopy);
0
1
2       }
```

# References:

- This training document was prepared using some of the resources from the CollegeBoard, including their AP Computer Science A course page which can be found at the link below:
- https://apcentral.collegeboard.org/courses/ap-computer-science-a?utm_source=chatgpt.com
-