

Java Fundamentals

Student Workbook 1 - Introduction to Java

Version 2.0

Table of Contents

Module 1 Introduction to Programming	1-1
Section 1-1 Thinking About Programming	1-2
Programming	1-3
Section 1-2 Learning to be Precise	1-4
Learning to Be Precise	1-5
Examples: "Program the Monkey"	1-6
Exercises: "Program the Monkey"	1-7
Expanded Command Vocabulary	1-9
Examples: "Smarter Monkey"	1-10
Exercises: "Program the Monkey" - Part 2	1-11
Section 1-3 Dealing with Ambiguity	1-14
Dealing with Ambiguity	1-15
Trying to Be Exact	1-16
Complex Problems	1-19
Exercises	1-20
Module 2 Introducing Java	2-1
Section 2-1 Overview of Java	2-2
Java	2-3
Versions of Java	2-4
Java Bytecode and the Java Virtual Machine (JVM)	2-5
Java Runtime Environment (JRE)	2-6
Java Developer Kit (JDK)	2-7
Compiling Java	2-8
Section 2-2 Configuring your Machine	2-9
Configuring your Development Machine	2-10
Verify Java Configuration	2-11
Module 3 Basic Java Syntax	3-1
Section 3-1 Basic Java Syntax	3-2
Java Syntax	3-3
Javadoc	3-4
Example: Code Written with Javadoc in Mind	3-5
Data Types	3-6
Variables	3-7
Read-only Variables	3-8
Data Types : Numbers	3-9
Data Types : Characters	3-10
Types of Variables	3-11
Static Variables	3-12
Instance Variables	3-13
Default Values	3-14
Exercise	3-15
Section 3-2 Operators and Expressions	3-16
String Concatenation	3-17
Mathematical Operators and Expressions	3-18
Operators and Expressions <i>cont'd</i>	3-19
Pre/Post- Increment and Decrement	3-20
Literals Are Typed	3-21
Widening Issues	3-23
Narrowing Issues	3-25
Type Casting	3-27
Java's <code>Math</code> Class	3-28
Example: Working with the <code>Math</code> Class	3-29
Assignment Operators	3-30
Exercises	3-31
Section 3-3 Writing to the Screen and Reading from the Keyboard	3-33
Writing Text to the Screen	3-34
Formatting Output	3-35
Java Format Specifiers	3-36
Reading Input with Scanner	3-37
Read a Whole Line of Text	3-38

Read Individual Values	3-39
Example: A Calculator	3-40
Mixed Input Types	3-41
Buffered Input	3-42
Line Separators and Numeric Input	3-43
Line Separators and <code>nextLine()</code>	3-44
Dealing With Line Separators	3-45
Consuming a Line Separator	3-46
Exercises	3-47
Section 3-4 Static Methods	3-49
Static Methods	3-50
Declaring a static Method	3-51
Passing Data to Methods	3-52
Returning Data from Methods	3-53
Using a <code>Scanner</code> in Multiple Methods	3-54
Example: Re-declaring the Scanner	3-55
Example: Re-declaring the Scanner <i>cont'd</i>	3-56
Example: Declaring the Scanner at the Class-Level	3-57
Example: Declaring the Scanner at the Class-Level <i>cont'd</i>	3-58
Example: Passing the Scanner	3-59
Example: Passing the Scanner <i>cont'd</i>	3-60
Exercises	3-61
Module 4 Conditionals	4-1
Section 4-1 Conditionals	4-2
Conditionals	4-3
<code>if</code> Statement	4-4
<code>if</code> / <code>else</code>	4-5
<code>if</code> / <code>else</code> Statements <i>cont'd</i>	4-6
Comparing Strings	4-7
Example: Better Calculator?	4-8
Conditional Operator	4-9
Exercises	4-10
Section 4-2 The <code>switch</code> Statement	4-12
<code>switch</code> Statement	4-13
<code>switch</code> Example	4-14
The <code>break</code> statement	4-15
Exercises	4-16

Module 1

Introduction to Programming

Section 1–1

Thinking About Programming

Programming

- **Programming is essentially the process of telling a computer what to do -- step by step -- in a language the computer understands**

- **When you program, you must give precise and detailed instructions**

- "Close only counts in horseshoes and hand grenades"

- **When you program, you must use a language the computer understands**

- There are many, maNY, MANY languages out there

- This week, we will begin our study of Java

- Along the way, you'll learn about some other languages like bash, SQL, XML, JSON and others!

- **Java is a good language to learn**

- Java is used by more than 6 million developers and runs on more than 5.5 billion devices

Section 1–2

Learning to be Precise

Learning to Be Precise

- To demonstrate using precise instructions, we'll spend a little time on a paper/pen coding exercise

- In this first phase, we will make a monkey move to a banana and eat it using a set of specific commands

Note: the ideas for this exercise came from codemonkey.com¹

- Right now, your monkey understands 3 commands

Step *number*

where *number* is the number of squares to move

Example: Step 5

Turn *direction*

where *direction* is left or right

Example: Turn left

Eat banana

This only works if the monkey is standing on an adjacent square to the banana *and* facing it.

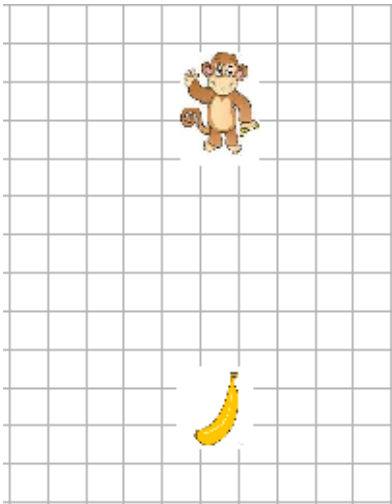
Example: Eat banana

¹ And before there was codemonkey.com, there was a really old programming language called "LOGO", in which you could move "turtles" to draw pictures. It still works, and we'll try it later.

Examples: "Program the Monkey"

Example

Move the monkey to the banana and have him eat it.

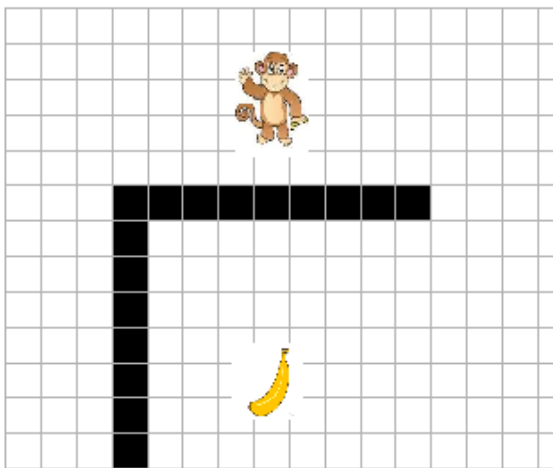


SOLUTION

Step 6
Eat banana

Example

Move the monkey to the banana and have him eat it. Avoid the barrier. Pay attention to the direction the monkey is facing.



SOLUTION

Turn left
Step 5
Turn right
Step 7
Turn right
Step 4
Eat banana

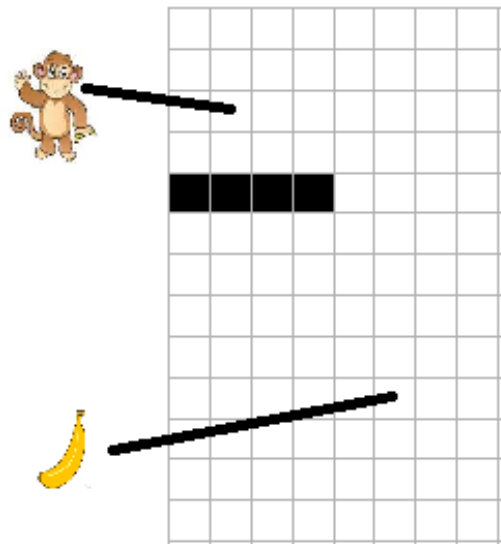
- **Note: Just like programming in real life, there is often more than one way to be successful**

Exercises: "Program the Monkey"

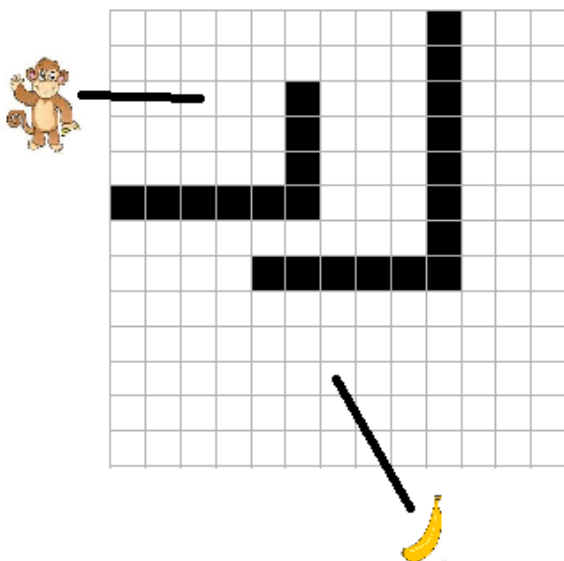
In the next few exercises, we will let you figure out the steps needed to "program your monkey" to move to and eat the banana. We are using lines to show you where the monkey and banana are so that the positions are very precise.

When you finish, talk it over in your group to see how others programmed their monkey.

EXERCISE 1



EXERCISE 2



EXERCISE 3



Expanded Command Vocabulary

- When you are learning a programming language, you start with just a few commands and learn to use them

When you conquer them, you learn more commands and how to use them too!

Before long, you know a LOT about the programming language

- Your monkey understands the commands we discussed before, plus some commands that can be used to pick up an item or drop an item

:

Step *number*

where *number* is the number of squares to move

Turn *direction*

where *direction* is left or right

Pickup *item*

where *item* is banana or basket

When you pick up an item, you must be facing the item in an adjacent square.

Example: Pickup basket

Drop *item*

where *item* is banana or basket

When you drop an item, it stays in the square directly in front of the square you are standing in. NOTE: Only one item can reside on a square, however, if the banana is IN the basket that counts as one item.

Example: Drop basket

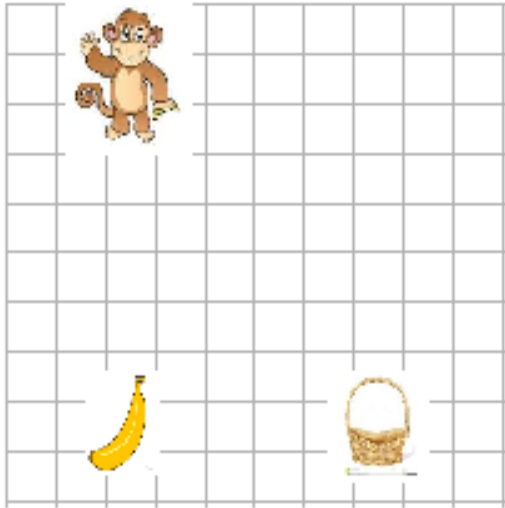
Eat banana

This only works if the monkey is standing on square adjacent to the banana directly facing it.

Examples: "Smarter Monkey"

Example

Make the monkey drop the banana into the basket.



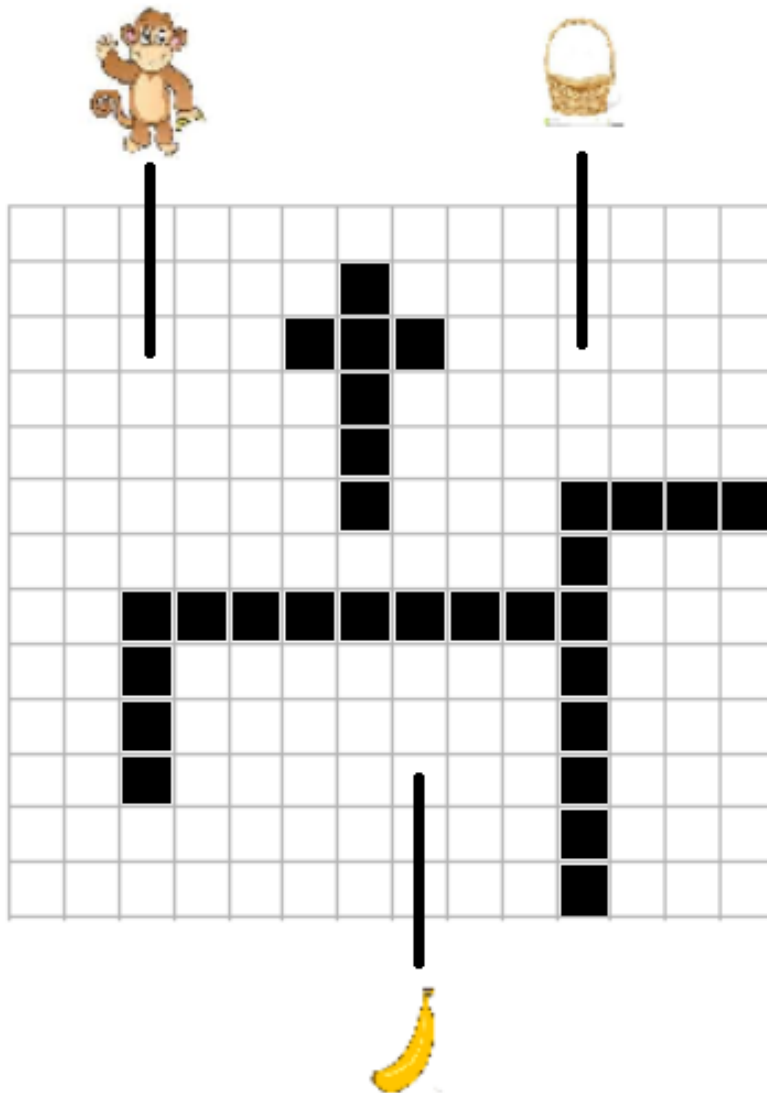
SOLUTION

Step 5
Pickup banana
Turn left
Step 5
Turn right
Drop banana

Exercises: "Program the Monkey" - Part 2

In these exercises, you need to "program your monkey" to put the banana in the basket. Like before, when you finish, talk it over in your group to see how others programmed their monkey.

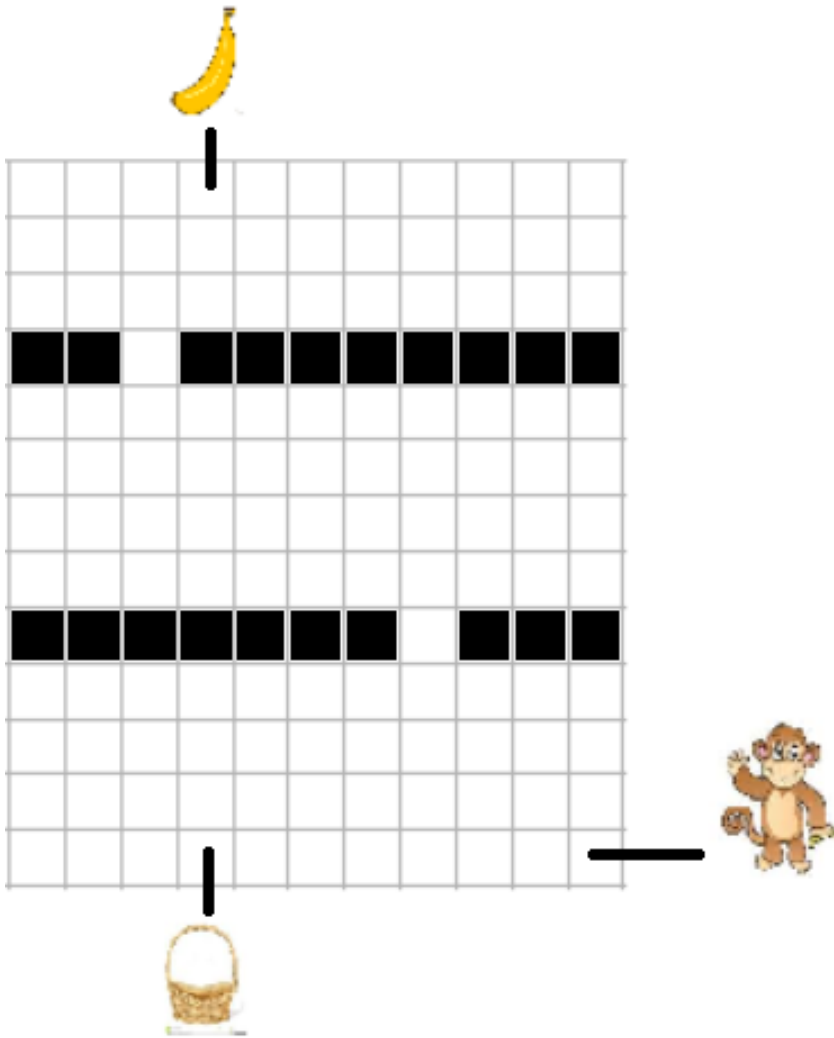
EXERCISE 1



EXERCISE 3

Would your answer change if you knew that the goal was to take the smallest number of steps possible?

The goal here is: *the banana ends up in the basket*. It doesn't say where the basket has to be. Do you pick up the banana and carry it to the basket? Do you take the basket over near the banana?



Section 1–3

Dealing with Ambiguity

Dealing with Ambiguity

- Sometimes, it's harder to generate a list of the tasks you must perform when the process isn't quite as well defined or as visual as working with monkeys!
- In these situations, we often write down the steps that we need to perform in English in order to "get our head around" the problem
- Computers follow your exact instructions -- but learning to be precise is tricky!
- Watch the "Exact Instructions Challenge" on YouTube before proceeding any further!

https://www.youtube.com/watch?v=cDA3_5982h8

Trying to Be Exact

- **How do you make scrambled eggs for two people?**

ATTEMPT 1

```
Get four eggs from refrigerator
Crack eggs into skillet
Scramble eggs
```

- Do you understand the process from our first attempt
- What's wrong with the description above?

- **The write up has some pre-conditions to scrambling eggs, but it doesn't really tell us how to scramble eggs**

ATTEMPT 2

```
Get four eggs from refrigerator
Crack eggs into skillet

Repeat
[
    Stir eggs as they cook
    Wait 30 seconds
] until they are the desired consistency

Turn stove off
```

- Better, but where did the eggs come from?

- **How exact do you have to be?**

- A lot of that depends on how intelligent the reader is

ATTEMPT 3

```
Open the refrigerator
Get four eggs
Close refrigerator
```

```
Crack eggs into skillet
```

```
Repeat
```

```
[
    Stir eggs as they cook
    Wait 30 seconds
] until they are the desired consistency
```

```
Turn stove off
```

- **Of course, if we are programming a computer - it has no preconceived insights and you have to be very exact**

- **You also have to describe possible error conditions**

- For example, what if we are out of eggs?

ATTEMPT 4

```
Open the refrigerator
If we are out of eggs
```

```
[
    Make different breakfast plans
    Close refrigerator
    Exit script
]
```

```
Get four eggs
Close refrigerator
```

```
Crack eggs into skillet
```

```
Repeat
```

```
[
    Stir eggs as they cook
    Wait 30 seconds
] until they are the desired consistency
```

```
Turn stove off
```

- **Not bad -- but are you really being detailed?**

Where did the skillet come from?

-

Will your eggs possibly stick to the pan as you cook?

-

ATTEMPT 5

```
Open the refrigerator
```

```
If we are out of eggs
```

```
[
```

```
    Make different breakfast plans
```

```
    Close refrigerator
```

```
    Exit script
```

```
]
```

```
Get four eggs
```

```
Close refrigerator
```

```
Get skillet from rack above stove
```

```
Spray PAM all around skillet for 3 seconds
```

```
Put skillet on stove
```

```
Crack eggs into skillet
```

```
Repeat
```

```
[
```

```
    Stir eggs as they cook
```

```
    Wait 30 seconds
```

```
] until they are the desired consistency
```

```
Turn stove off
```

- **Can you see your sarcastic friend following the instructions for spraying PAM?**

My sarcastic friend would spray the handle and bottom of the skillet!

-

Oops! Replace with:

```
Spray PAM all around the inside of the skillet for 3 seconds
```

Complex Problems

- **When you are working with complex problems, you really have to think about all aspects of the problem**

- Where did the PAM come from?

- Do they know how to crack eggs?

- What if some of the shell ended up in the pan?

- What did they do with the shells once they put the egg in the skillet?

- Who turned on the stove?

- Stir the eggs with what?

- What is "desired consistency"?

- In my house, one of us likes eggs runny and the rest want them cooked
*into dried hard unappetizing flecks!"

- Is plating the eggs part of the scope of the problem?

- Is cleaning up your mess part of the scope of the problem?

- **Other things that might impact how you write include:**

- The vocabulary of the user (new to the process, experienced)

- The skills of the user (never cooked, casual cook, chef)

- The complexity of the process

- Sometimes, you just won't understand enough about the process to get
*all of the details right at the beginning

- * Software development is an iterative process!

Exercises

EXERCISE 1

Write out the process of how to "brush your teeth". This may take 4-10 minutes if you do a good job.

Then, meet with your group to go over the write-ups. Tweak the write-ups as needed.

Your team will present a "brush your teeth" process to the class as a whole. Your team can select one of the team member's to advance or can blend them together using steps from each team member.

Module 2

Introducing Java

Section 2–1

Overview of Java

Java

- **Java is an object-oriented programming language that has been around since the mid 1990s**

It was originally developed by Sun Microsystems, and has now been acquired by Oracle

- **One of its strengths is that it is *platform independent***

Once translated into Java bytecode, it can run on many different types of computers

Often people refer to this as "write once, run anywhere!"

- **Platform independence is what makes Java a popular choice desktops, mobile devices, servers, IoT, and more**

Versions of Java

- Java has continued to evolve -- Java 17 is the latest "Long Term Support" version

However, many production apps are written using older versions

Version	Release date	End of Free Public Updates ^{[1][5][6][7]}	Extended Support Until
JDK Beta	1995	?	?
JDK 1.0	January 1996	?	?
JDK 1.1	February 1997	?	?
J2SE 1.2	December 1998	?	?
J2SE 1.3	May 2000	?	?
J2SE 1.4	February 2002	October 2008	February 2013
J2SE 5.0	September 2004	November 2009	April 2015
Java SE 6	December 2006	April 2013	December 2018 December 2026 for Azul ^[8]
Java SE 7	July 2011	April 2015	July 2022
Java SE 8 (LTS)	March 2014	January 2019 for Oracle (commercial) December 2030 for Oracle (non-commercial) December 2030 for Azul December 2030 for IBM Semeru At least May 2026 for Eclipse Adoptium At least May 2026 for Amazon Corretto	December 2030 ^[9]
Java SE 9	September 2017	March 2018 for OpenJDK	N/A
Java SE 10	March 2018	September 2018 for OpenJDK	N/A
Java SE 11 (LTS)	September 2018	September 2026 for Azul September 2026 for IBM Semeru At least October 2024 for Eclipse Adoptium At least September 2027 for Amazon Corretto At least October 2024 for Microsoft ^{[10][11]}	September 2026 September 2026 for Azul ^[8]
Java SE 12	March 2019	September 2019 for OpenJDK	N/A
Java SE 13	September 2019	March 2020 for OpenJDK	N/A
Java SE 14	March 2020	September 2020 for OpenJDK	N/A
Java SE 15	September 2020	March 2021 for OpenJDK March 2023 for Azul ^[8]	N/A
Java SE 16	March 2021	September 2021 for OpenJDK	N/A
Java SE 17 (LTS)	September 2021	September 2029 for Azul At least September 2027 for Microsoft At least TBA for Eclipse Adoptium	September 2029 or later September 2029 for Azul
Java SE 18	March 2022	September 2022 for OpenJDK	N/A
Java SE 19	September 2022	March 2023 for OpenJDK	N/A
Java SE 20	March 2023	September 2023 for OpenJDK	N/A
Java SE 21 (LTS)	September 2023	TBA	September 2031 ^[9]

Legend: ■ Old version ■ Older version, still maintained ■ Latest version ■ Future release

from Wikipedia - https://en.wikipedia.org/wiki/Java_version_history

Java Bytecode and the Java Virtual Machine (JVM)

- When a Java program is compiled, it is not translated into the machine code native to the developer's computer

- This would mean it could only run on similar computers

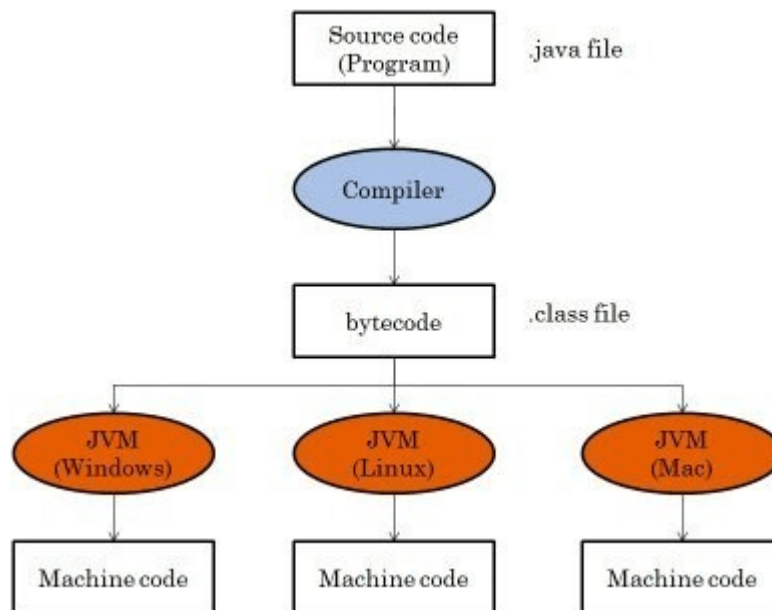
- Instead, Java programs are translated into something called Java *bytecode*

Bytecode is stored in a `.class` file and is eventually executed on a computer by using a JVM

- The Java Virtual Machine (JVM) is the program responsible for loading and executing a Java application

- It executes the Java bytecode instructions

- There are several JVMs available depending on the type of computer you use



Java Runtime Environment (JRE)

- You must install the Java Runtime Environment (JRE) in order to *run* Java applications on your computer
- The JRE is a software layer that sits on top of a computer's native operating system
- It provides the class libraries and other resources (including the JVM) needed to run Java applications
- However, the JRE does not contain the resources a developer needs for creating Java applications

Java Developer Kit (JDK)

- The Java Developer Kit (JDK) is set of tools for developing Java applications

- It includes a CLI, the Java compiler, debuggers, etc.

- If you want to create Java applications, you must install a JDK

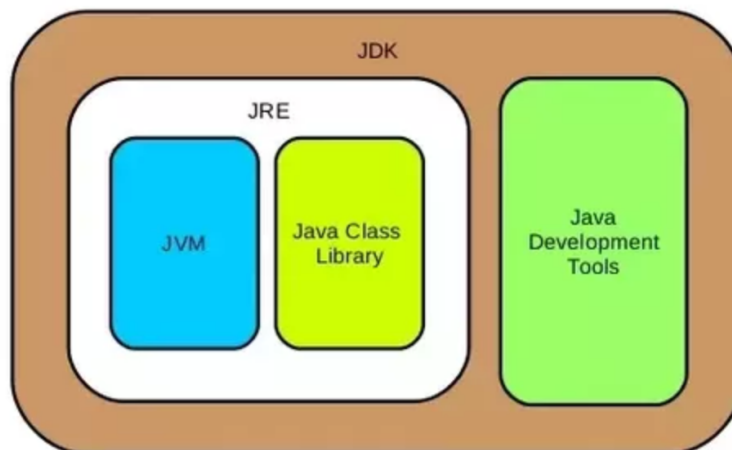
- Developers choose JDKs by Java version and by package or edition—

- Java Standard Edition (Java SE) <-- We want this one!

- Java Enterprise Edition (Java EE)

- Java Mobile Edition (Java ME)

- The JDK also includes a compatible JRE because so that the Java application can be tested and run on the developer's machine



Compiling Java

- **To compile a Java application into bytecode, you use**

`javac`

- `javac` **is a Java compiler**

It confirms the grammar you coded is correct

- It then translates the code into bytecode and places it in a `.class` file

- **If you have syntax errors, the compiler will not generate the bytecode**

- **This is the difference between a compiled language like Java and an interpreted language like JavaScript**

- You can't even try to run the code if there are syntax errors

Section 2–2

Configuring your Machine

Configuring your Development Machine

- If you don't have a JDK, you can download the version you need here

<https://www.oracle.com/java/technologies/downloads/>

- Once you have installed a JDK on your development machine, you will need to do some (hopefully) simple configurations

- **Step 1: Configure the JAVA_HOME environment variable**

On a Windows machine, this will be something like:

C:\Program Files\Java\jdk-17

- **Step 2: Add the bin folder containing Java developer tools to your PATH environment variable**

On a Windows machine, this will be something like:

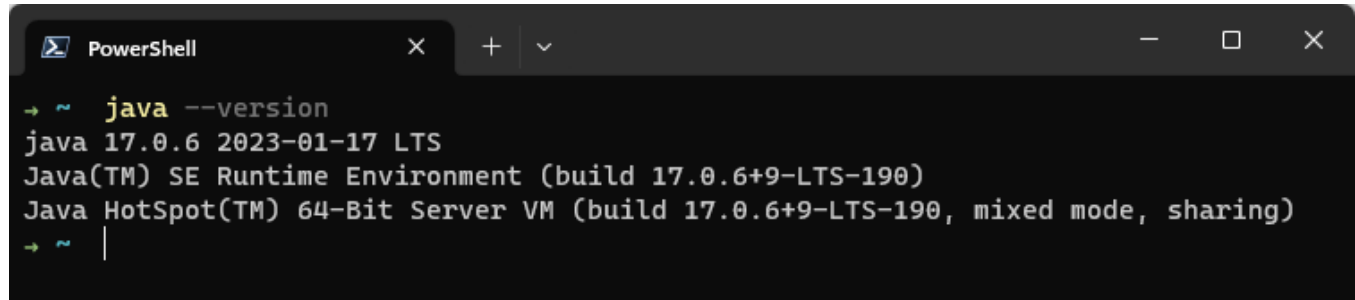
C:\Program Files\Java\jdk-17\bin

- **Step 3 (possible): If your path variable includes an Oracle Java reference, either move it to the bottom of the PATH list or remove it**

Verify Java Configuration

- Open Windows Terminal and execute the following command

```
java --version
```



```
PowerShell
→ ~ java --version
java 17.0.6 2023-01-17 LTS
Java(TM) SE Runtime Environment (build 17.0.6+9-LTS-190)
Java HotSpot(TM) 64-Bit Server VM (build 17.0.6+9-LTS-190, mixed mode, sharing)
→ ~ |
```


Module 3

Basic Java Syntax

Section 3–1

Basic Java Syntax

Java Syntax

- **Java is one of the C-family languages**

Includes C, C++, C#, Java, Kotlin, etc.

- **This means, amongst other things:**

it is case sensitive, everywhere and always (!)

it uses semicolons (;) as command terminators

there are 2 types of comments

single line comments

```
// this line is a comment
```

and multi-line comments

```
/*  
this line is a comment  
this line is also a comment  
*/
```

the `if`, `switch`, `for`, and `while` statements will look very familiar

- **Source code for Java applications is contained in one or more `.java` files**

Unlike most other languages, Java really cares about the name of the file

The file must start with an uppercase letter and match the name of the class it contains

Javadoc

- The Javadoc tool, included in the JDK, generates API documentation from comments found in the source code
- Comments should have a specific style

Syntax

NOTE: a javadoc comment begins with 2 "stars"

```
/**  
 * This is a Javadoc comment  
 */
```

- Javadoc looks for these special comments in front of class, method, and variable declarations
- These comments are commonly made up of two sections:
 - Description of what you are commenting
 - Tags marked with an @ symbol that describe specific metadata

Example: Code Written with Javadoc in Mind

Example

```
/**
 * HelloWorldApp is an example of most people's first
 * Java program
 *
 * @author Dana Wyatt
 */
public class HelloWorldApp {

    public static void main(String[] args) {
        /**
         * The greeting to be displayed in the Console.
         */
        String message = "Howdy Java!";
        display(message);
    }

    /**
     * Displays any message the console window
     *
     * @param message the message displayed
     */
    public static void display(String message) {
        System.out.println(message);
    }
}
```

- You can run Javadoc from the terminal window

Example

```
javadoc HelloWorldApp.java
```

- We won't focus on Javadoc in this course, but it produced the online docs that you'll use all the time

Data Types

- **There are two types of data types in Java:**

- *Primitive* data types include: `boolean`, `char`, `byte`, `short`, `int`, `long`, `float`, and `double`

- *Non-primitive* data types include arrays, classes, and interfaces

- **Primitive types are pre-defined by Java and represent simple values:**

- `boolean` - a true/false value in a *single bit*

- `char` - a single Unicode character (16-bits)

- numbers such `byte`, `short`, `int`, `long`, `float`, and `double` as that have different memory structures and sizes

- * Note: floats and doubles store their values in an imprecise way - they are not usually used for storing currency values that must be precise

- **Non-primitive types are arrays, class types, etc**

- They are built by combining primitive types in various ways

- Java libraries provide many predefined types

- The `String` data type is actually implemented in a Java class

- * That's why its name starts with a capital S!

Programmers also create many classes as they go about developing their application using object-oriented design principles

Variables

- **Java is a statically-typed language and every variable MUST be assigned a data type**

- **Variables in Java are similar to those in other languages**

They hold information and allow a programmer to interact with it by the variable name

Example

```
// hold a whole number
int num1;
long num2;

// hold a number with decimal points
float num3;
double num4;

// hold a Boolean
boolean isHappy;

// hold a string
String word;
```

- **You can declare many variables on the same line if they are of the same type**

You can choose to initialize some and not others

Example

```
int x = 5, y, z = 50;
```

Read-only Variables

- In Java, a read-only variable is declared with the keyword **final**
- It must be initialized when you declare the variable
Any attempt to change it later will be an error

Example

```
final String name = "Dana";  
  
System.out.println("Your name is " + name);  
  
name = name + " Wyatt";  // syntax error!!  
  
System.out.println("Your full name is " + name);
```

Data Types : Numbers

- **Java allows programmers to choose from many primitive data types when working with numbers**

The difference between them boils down to how many bits are allocated in the computer's memory and how does the computer format those bits

- **For example, a `byte` is an 8-bit number**

7 of the bits are used for the number and 1 bit is used to indicate whether the number is positive or negative

This means the range of a `byte` variable is -128 to 127

- **Other whole number types include:**

- a `short` that is 16 bits and its range is -32,768 to 32,767

- an `int` that is 32 bits and its range is -2,147,483,648 to 2,147,483,647

- a `long` that is 64 bits and its range is -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

- **Java's two floating point primitive types are:**

- a `float` that is 32 bits and holds about 6-7 significant digits

* Its range is approximately $\pm 3.40282347E+38F$

- a `double` that is 64 bit and holds about 15 significant digits

* Its range is approximately $\pm 1.79769313486231570E+308$

Data Types : Characters

- **Java also contains a `char` type**

a `char` is 16 bits and holds a Unicode character - it represents a printable character such as the letter "X"

It *can* also be used as an unsigned 16-bit number whose range is 0 to 64,767

Types of Variables

- In Java, there are three types of variables, depending on *where* and *how* you declare them:

- local variables
- static variables
- instance variables

- ***Local variables* are declared inside a method**

They are created when the method is called and are garbage collected when the method ends

- Notice that parameters are local variables, too

Example

When `add()` is called, it created the two parameters `num1` and `num2` as well as the variable `sum`. When the `add()` method finishes running, all three variables are destroyed

```
public int add(int num1, int num2) {  
    int sum;  
    sum = num1 + num2;  
    return sum;  
}
```

Static Variables

- ***Static* variables are allocated once and retain their values the for life of the program**

- A static variable cannot be local
- It must declared in a class
- Must be declared using the `static` modifier

Example

The `counter` variable is created when the application starts running. It stays around for the life of the application. It can be seen by all of the methods in the `MainApp` class because that is where it is declared. However, because it is `private` it cannot be accessed outside of the class.

```
public class MainApp {  
    private static int counter = 0;  
  
    public static void main(String[] args) {  
        counter++;  
        ...  
    }  
  
    public static void anotherMethod() {  
        counter--;  
        ...  
    }  
}
```


Instance Variables

- ***Instance variables belong to an object***

They are created when the object is instantiated and are garbage collected when the object is destroyed

Example

The `name` and `age` variables in the `Person` class are instance variables. Each time a person object is created, those variables are created for that object and stay allocated until that object goes away.

```
class Person {
    private String name;
    private int age;

    public Person(string name, int age) {
        this.name = name;
        this.age = age;
    }
    ...
}

public class MainApp {
    public static void main(String[] args) {
        Person me = new Person("Dana Wyatt", 63);
        Person you = new Person("John Q Student", 28);
        ...
    }
}
```

- **We will see lots of examples of these as the course continues and we become better Java programmers**

Default Values

- **Java provides default values for *class-level* variables based on their data types**

- Numbers default to `0`
- Booleans default to `false`
- Objects default to `null`

- **Knowing this explains why code acts as it does when you forget to initialize variables**

- **However, the compiler does NOT assign *local* variables a default**

- If you try to access a local variable that is *uninitialized*, the compiler will generate a syntax error

- **A best practice says *always* initialize your variables**

Exercise

EXERCISE 1

Using **Notepad** declare variables to hold the following data. Consider what information is being stored with each variable, and how you will use the data.

NOTE: this exercise is a mental/paper exercise and **SHOULD NOT** be completed in IntelliJ.

Declare each variable with the correct data types:

- a vehicle identification number in the range 1000000 - 9999999
- a vehicle make /model (i.e. Ford Explorer)
- a vehicle color
- whether the vehicle has a towing package
- an odometer reading
- a price
- a quality rating (A, B, or C)
- a phone number
- a social security number
- a zip code

(you will have a chance to share your answers through zoom)

Section 3–2

Operators and Expressions

String Concatenation

- Java uses the + operator to concatenate strings

Example

```
public class BuildAStringApp {  
  
    public static void main(String args[]){  
        String word1 = "Hello";  
        String word2 = "World";  
        String greeting;  
  
        greeting = word1 + " " + word2 + "!";  
        System.out.println(greeting);  
    }  
}
```

OUTPUT

Hello World!

- When you Google, you will also find that there is a `concat()` method

Mathematical Operators and Expressions

- Java supports the standard arithmetic operators you find in most languages (+ - * /) and well as the remainder (or *modulo*) operator (%)
- Because Java is strongly typed, the results of arithmetic operations are sometimes surprising

Note: one integer divided by another is an integer!

Example

```
public class BasicIntegerMathApp {  
  
    public static void main(String[] args){  
        int a = 10;  
        int b = 3;  
        int result;  
  
        result = a + b;  
        System.out.println(result);    // displays 13  
  
        result = a - b;  
        System.out.println(result);    // displays 7  
  
        result = a * b;  
        System.out.println(result);    // displays 30  
  
        result = a / b;  
        System.out.println(result);    // displays 3  
  
        result = a % b;  
        System.out.println(result);    // displays 1  
    }  
}
```

Operators and Expressions *cont'd*

- **Floating point math always returns a floating point result**

Example

```
public class BasicFloatingPointMathApp {  
  
    public static void main(String[] args){  
        float a = 10;  
        float b = 3;  
        float result;  
  
        result = a + b;  
        System.out.println(result);    // displays 13.0  
  
        result = a - b;  
        System.out.println(result);    // displays 7.0  
  
        result = a * b;  
        System.out.println(result);    // displays 30.0  
  
        result = a / b;  
        System.out.println(result);    // displays 3.3333333  
  
        result = a % b;  
        System.out.println(result);    // displays 1.0  
  
        result = b - (a % b);  
        System.out.println(result);    // displays 2.0  
    }  
}
```

- **Parentheses can be placed around expressions to control the order in which they are evaluated**

Expressions can be a combination of integer and floating point values, but if the result is a float point value, it must be assigned to a floating point variable

Pre/Post- Increment and Decrement

- Like other C-based languages, Java embraces the pre/post- increment and decrement operator

Example

```
int x = 5;
int y;

x++;    // adds 1 to x    (now it is 6)
++x;    // adds 1 to x    (now it is 7)

y = ++x;    // adds 1 to x and then assigns x to y
             // x is now 8 and y is 8

x = 5;

y = x++;    // assigns x to y and then adds 1 to x
             // x is now 6 and y is 5
```


Literals Are Typed

- In Java, literal values (hard coded values) have a data type

Example

```
101    <- an int
101L   <- a long (can use a little l, but it looks like a 1)
7.25   <- a double
7.25f  <- a float
"Hi"   <- a String
'a'    <- a char
true   <- a boolean
```

- Starting in Java SE 7, you can place underscore characters (`_`) between digits in a numerical literal

- This sometimes makes the number more readable

Example

```
long ssn = 111_22_3333L;
long creditCard = 5200_7500_6500_0001L;
double loanBalance = 1_225_570.00;
```

- You can't place underscores:

- At the beginning or end of a number

- Adjacent to a decimal point in a floating point literal

- Prior to an `F` or `L` suffix

- When using literals, you must decide whether you want to use 32-bit floating point values...

Example

```
public class PriceApp {  
  
    public static void main(String args[]){  
        int count = 11;  
        float unitPrice = 7.12f; //this is an error without the f  
        float taxRate = 0.825f;  
        float totalCost;  
  
        totalCost = (count * unitPrice) * (1 + taxRate);  
        System.out.println(totalCost);  
    }  
}
```

- ... or 64 bit ones

Example

```
public class PriceApp {  
  
    public static void main(String args[]){  
        int count = 11;  
        double unitPrice = 7.12;  
        double taxRate = 0.825;  
        double totalCost;  
  
        totalCost = (count * unitPrice) * (1 + taxRate);  
        System.out.println(totalCost);  
    }  
}
```

- Static typing will take some getting used to!

Widening Issues

- When you take a variable or expression and assign it to a variable of a wider data type, there are no errors and it is considered a "safe" process

- What does wider mean?

It means no loss of precision

- `byte -> short -> char -> int -> long -> float -> double`

- For example, assigning a `float` variable to a `double`!

- ... or an `int` variable to a `long`!

Example

```
int myInt = 9;
long myLong;
```

```
float myFloat = 3.8f;
double myDouble;
```

```
// an int fits in a long
myLong = myInt;
```

```
// a float fits in a double
myDouble = myFloat;
```

```
// you can even put an int or long into a float or double
myFloat = myLong;
myDouble = myLong;
```

- Widening isn't about a specific value for a variable and whether it will fit in the new variable

- It is about whether any value in the source data type will fit in the data type of the new variable

Narrowing Issues

- When you take a variable or expression and assign it to a narrower variable, the compiler will generate an error

If you want the assignment to happen, you have to explicitly tell the compiler

- What does narrower mean?

It means there is a "chance" that a value in the source variable might now fit in the narrower variable

`double -> float -> long -> int -> char -> short -> byte`

- For example, assigning a `double` variable to a `float`!

... or a `float` variable to a `int`!

-

Example

```
int myInt;
long myLong = 9;

float myFloat;
double myDouble = 123.4567890123;

// an int won't necessarily fit in a long
myInt = myLong;

// a double won't necessarily fit in a float
myFloat = myDouble;

// a float probably won't necessarily fit in an int
myLong = myFloat;
```

SYNTAX ERRORS

```
Main.java:12: error: incompatible types: possible lossy conversion from long to int
myInt = myLong;
      ^
Main.java:15: error: incompatible types: possible lossy conversion from double to float
myFloat = myDouble;
        ^
Main.java:18: error: incompatible types: possible lossy conversion from float to long
myLong = myFloat;
        ^
3 errors
```

Type Casting

- Sometimes you know that a narrowing assignment is okay and can force the compiler to make the assignment using type casting

- To cast an expression to a narrow type, put the target type in parenthesis in front of the expression

It will suppress the compiler error

Example

```
long myLong = 9;
int myInt;

// you know the range of values in the long variable
// will always fit in the int
myInt = (int) myLong;
```

- This is often needed because you are calling a function whose return type you can't control

For example, you are calling `Math.pow()` that returns a double but you need less precision

* See more details on the next page

Example

```
float num = 2.2;

// if you don't cast the result, you have to store
// the value in a double variable
float result = (float) Math.pow(num, 2);
```

Java's Math Class

- Java has a **Math** class with many helpful predefined methods

- See: https://www.w3schools.com/java/java_ref_math.asp

- The methods are static methods

- This means you invoke them using the name of the class

Example

```
double degrees = 90.0;

double answer = Math.sin(degrees);
```

- Most of the methods have a one or more double parameters and return a double

- But read the documentation because this isn't entirely true

Example

```
double value = 1234.567;

int wholeNumber = Math.round(value);
```

Example: Working with the `Math` Class

Example

```
public class MathApp {  
    public static void main(String[] args) {  
        int natKidCount = 2, brittKidCount = 4;  
        int mostKids = Math.max(natKidCount, brittKidCount);  
        System.out.println(  
            "The biggest family has " + mostKids + " kids");  
    }  
}
```


Assignment Operators

- Java supports a set of assignment operators that are combined with arithmetic operators to make certain operations easier to express
- They include: `+=` `-=` `*=` `/=` `%=`

Example

`// Original technique`

```
int answer = 0;
answer = answer + 10; // 10
answer = answer - 5;  // 5
answer = answer * 10; // 50
answer = answer / 2;  // 25
answer = answer % 10; // 5 (int remainder of 25 / 10)
```

Example

`// Shortcut technique`

```
int answer = 0;
answer += 10; // 10
answer -= 5;  // 5
answer *= 10; // 50
answer /= 2;  // 25
answer %= 10; // 5 (int remainder of 25 / 10)
```

Exercises

In this exercise you will create a simple project that uses java to perform various mathematical calculations. Complete your work in the `pluralsight/workbook-1` folder.

EXERCISE 2

Using IntelliJ create a new Java project named `MathApplication`. Add a new package named `com.pluralsight` with a file named `MathApp.java` structured as follows:

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

        // Question 1:
        // declare variables here
        // then code solution
        // then use System.out.println() to display results
        // ex: System.out.println("The answer is " + answer);

        // REPEAT FOR NEXT EXERCISE

    }
}
```

Step 1

Write code to find answers to the following questions.

NOTES: Coding is not just about solving equations, your code should be legible and meaningful

Use comments and line spacing to visually separate each question.

Take your time to think of and use meaningful variable names (don't just use `num1`)

Print **meaningful** messages to the screen so that the reader has all of the context information for what you are.

QUESTIONS:

1. Create two variables to represent the salary for Bob and Gary, name them `bobSalary` and `garySalary`. Create a new variable named `highestSalary`. Determine whose salary is greater using `Math.max()` and store the answer in `highestSalary`. Set the initial salary variables to any value you want. Print the answer (i.e "The highest salary is ...")
2. Find and display the smallest of two variables named `carPrice` and `truckPrice`. Set the variables to any value you want.
3. Find and display the area of a circle whose radius is 7.25
4. Find and display the square root a variable after it is set to 5.0
5. Find and display the distance between the points (5, 10) and (85, 50)
6. Find and display the absolute (positive) value of a variable after it is set to -3.8
7. Find and display a random number between 0 and 1

Step 2

Push your changes to **GitHub** (always stage, commit and push your changes)

1. `git add -A`
2. `git commit -m "completed Mod3 Exercise 2 - MathApp"`
3. `git push origin main`

Section 3–3

Writing to the Screen and Reading from the Keyboard

Writing Text to the Screen

- You can use both the `System.out.print()` and `System.out.println()` methods to display a line of text

The difference is that the `println()` method places a CR/LF in the output stream after it writes

Example

```
class Program
{
    public static void main(String args[])
    {
        System.out.print("Hello ");
        System.out.print("World!");
    }
}
```

OUTPUT

Hello World!

Example

```
class Program
{
    public static void main(String args[])
    {
        System.out.println("Hello World!");
    }
}
```

OUTPUT

Hello World!

- The two examples above aren't *exactly* the same...

QUESTION: What would happen if you added a `println()` statement to each?

Formatting Output

- There are several ways to format numbers in Java

<https://java2blog.com/format-double-to-2-decimal-places-java/>

- One of the easiest is to use the `String` class' `format()` method

It accepts a format string (more on this on the next page)

Example

```
float subtotal = 22.87;
float tax = subtotal * 0.0825f;
float totalDue = subtotal + tax;

System.out.println(
    "Total due is: " + String.format("%.2f", totalDue));
```

- Another way is to use the `printf()` method instead of `print()` or `println()`

It accepts a format string also

Example

```
float subtotal = 22.87;
float tax = subtotal * 0.0825f;
float totalDue = subtotal + tax;

System.out.printf("Total due is: %.2f", totalDue);
```

Java Format Specifiers

- Format specifiers begin with a percent character (%)
- They end with a type character that specifies type of data (int, float, etc) being formatted
- In between, you can specify width and precision

Syntax

% [flags] [width] [.precision] **type-character**

- Type characters include:
 - d decimal integer (base 10 number)
 - x or X hexadecimal integer (base 16)
 - f floating point number
 - s string
 - c character

Example

```
int id = 10135;
String name = "Brandon PLYERS";
float pay = 5239.77f;

System.out.printf("%s (id: %d) $%.2f", name, id, pay);
```

Reading Input with Scanner

- You can use the `Scanner` class to read input
 - A whole line at a time, or
 - One formatted value at a time
- The `Scanner` class is defined in `java.util.Scanner` and you will have to import it at the top of the class file
 - * You can import just the `Scanner` class or all classes in the same package

Example

```
java.util.Scanner;    // import the Scanner class
import java.util.*;   // import all classes in the package
```

- If you pass `System.in` as a constructor argument, the `Scanner` will read from the keyboard

Example

```
Scanner scanner = new Scanner(System.in);
```


Read a Whole Line of Text

- Use the `nextLine()` method to read a line of input

Scanner reads the input up to the next line separator (LF or CRLF) and returns it as a String

- The line separator character is discarded
- If the scanner is closed, it throws an `IllegalStateException`

Example

```
import java.util.*;

class Program
{
    public static void main(String[] args)
    {
        Scanner myScanner = new Scanner(System.in);

        System.out.print("Enter your name: ");
        String name = myScanner.nextLine();

        System.out.println("Howdy " + name);
    }
}
```

Read Individual Values

- To read individual data values (numbers, words, booleans, etc.) you can use other `Scanner` methods; for example:

```
-    nextInt()  
-    nextFloat()  
-    nextDouble()  
-    nextBoolean()
```

- By default, the scanner breaks the input into *tokens*, separated by *whitespace* (space, tab, line separators, etc.)
- Scanner reads the next token and returns its value, *if possible*, as the desired type

Scanner may throw an `InputMismatchException` if the input is not compatible with the requested type

Initial whitespace is discarded, but *following* whitespace is left in the stream (more on this in a minute)

Example

```
System.out.println("How old are you? ");  
int age = scanner.nextInt();
```

```
System.out.println("What is your salary? ");  
double salary = scanner.nextDouble();
```

Example: A Calculator

Example

```
import java.util.*;

class CalculatorApp
{
    public static void main(String[] args)
    {
        Scanner myScanner = new Scanner(System.in);

        // get two numbers, add them together, and display the sum

        System.out.print("Enter first number: ");
        int num1 = myScanner.nextInt();

        System.out.print("Enter second number: ");
        int num2 = myScanner.nextInt();

        int sum = num1 + num2;

        System.out.println("The sum is " + sum);
    }
}
```

Mixed Input Types

Example

// File: UserInputApp.java

```
import java.util.*;

public class UserInputApp {

    public static void main(String[] args) {

        // create a scanner
        Scanner scanner = new Scanner(System.in);

        // read name
        System.out.print("What is your name? ");
        String name = scanner.nextLine();

        // read age
        System.out.print("How old are you? ");
        int age = scanner.nextInt();

        // display a greeting
        String greeting;
        if (age <= 21) {
            greeting = "Hi ";
        }
        else {
            greeting = "Hello ";
        }
        System.out.println(greeting + " " + name + "!");
    }
}
```

TRACE OF TERMINAL WINDOW SESSION

```
What is your name? Dana
How old are you? 63
Hello Dana!
```

- It seems to work fine, but we must be careful...

Buffered Input

- **Most operating systems require you to hit the ENTER key to send input on through to the program**

This actually inserts a line separator (Usually CRLF) in the input buffer, which must be handled by the scanner

Example

```
Scanner scanner = new Scanner(System.in);
```

```
System.out.print("How old are you? ");  
int age = scanner.nextInt();
```

TRANSCRIPT OF RUNNING CODE:

```
How old are you? 63↵    ↵ where the return arrow  
                        is the ENTER key (CRLF)
```

- **In this example, `nextInt()` reads 63 correctly, and leaves the following CRLF in the input buffer**

- **Does this cause a problem?**

It depends...

-

Line Separators and Numeric Input

- If another numeric input method is called (`nextInt()`, `nextFloat()`, etc), it discards the leftover CRLF as whitespace before reading the next token

It's looking for numbers -- so an extra CRLF doesn't affect numeric input

Example

```
Scanner scanner = new Scanner(System.in);

System.out.print("Enter number 1: ");
int num1 = scanner.nextInt();

System.out.print("Enter number 2: ");
int num1 = scanner.nextInt();
```

TRANSCRIPT OF RUNNING CODE:

```
Enter number 1: 13↵
Enter number 2: 145↵    □ This ignores the CRLF left in the
                        buffer after reading 13 and
                        successfully reads 145

// HOWEVER, now there is a different CRLF
// left in the buffer after 145
```

Line Separators and nextLine()

- If the next read is performed with `nextLine()`, it stops at the first CRLF it finds!

Example

```
Scanner scanner = new Scanner(System.in);

System.out.print("Enter number 1: ");
int num1 = scanner.nextInt();

System.out.print("Enter number 2: ");
int num2 = scanner.nextInt();

System.out.println("What do you want to do? ");
System.out.print("    Enter 'add' or 'subtract': ");
String action = scanner.nextLine();

System.out.println("Preparing to do math... ");
```

TRANSCRIPT OF RUNNING CODE:

```
Enter number 1: 13↵
Enter number 2: 145↵
What do you want to do?
    Enter 'add' or 'subtract': Preparing to do math...

// The system didn't wait for the user to enter
// data because the CRLF in the buffer after 145
// marked the end of the input when nextLine()
// went to read its data
```

Dealing With Line Separators

- **So how do you fix this?**

- Carefully and on a case-by-case basis

- **If you read a mix of numbers and text, whose job is it to get rid of the extra CRLF???**

- The code reading the number (that left CRLF in the buffer)?

- The code reading text (that doesn't want a CRLF before its read)?

- **But --**

If the code reading an int gets rid of the CRLF, it would be wasted energy if the next piece of code also reads an int

If the code reading text tries to get rid of the CRLF and its not there, then that could be a problem

- **There's no great answer! But someone has to decide. So, here is my answer for the code above...**

- If will forcibly consume the CRLF that is causing the problem!

Consuming a Line Separator

Example

```
Scanner scanner = new Scanner(System.in);

System.out.print("Enter number 1: ");
int num1 = scanner.nextInt();

System.out.print("Enter number 2: ");
int num2 = scanner.nextInt();
scanner.nextLine();    // "eat" the leftover CRLF

System.out.println("What do you want to do? ");
System.out.print("    Enter 'add' or 'subtract': ");
String action = scanner.nextLine();

System.out.println("Preparing to do math...");
```

TRANSCRIPT OF RUNNING CODE:

```
Enter number 1: 13↵
Enter number 2: 145↵
What do you want to do?
    Enter 'add' or 'subtract': add↵
Preparing to do math...
```

Exercises

In these exercise you will create basic console applications which prompt the user for information and then perform calculations based on the user input. Remember to do your work in the `workbook-1` directory.

Remember to be conscious of the format and readability of your code.

- * Use meaningful variable names
- * Use whitespace and lines to organize your code and thoughts so that others can easily understand your code
- * Don't try to do multiple things on a single line of code - when in doubt opt for code readability

EXERCISE 3

Create a Java application named **BasicCalculator** that reads in two floating point numbers and then asks the user whether they want to add, subtract, multiply or divide the two numbers.

Step 1

Perform the requested operation and display the results.

```
Enter the first number: 5
Enter the second number: 12

Possible calculations:
(A)dd
(S)ubtract
(M)ultiply
(D)ivide
Please select an option: M

5 * 12 = 60
```

Step 2

Push your changes to GitHub

```
git add -A
git commit -m "Completed Mod3 - Exercise 3 - Basic Calculator"
git push origin main
```

EXERCISE 4

Step 1

Create a Java application named **PayrollCalculator** that prompts the user to enter:

- their name
- their hours worked (a floating point number)
- their pay rate (a floating point number) Calculate their gross pay.

Display the employee's name and their gross pay.

Step 2

Push your changes to GitHub

```
git add -A
git commit -m "Completed Mod3 - Exercise 4 - Payroll Calculator"
git push origin main
```

(Optional)

You learned about `if` statements in your pre-work, so although we have not discussed them in class, you should be able to figure out how to pay 1.5x overtime after 40 hours. You can use W3Schools guide as a reference

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

Push your changes to GitHub

```
git add -A
git commit -m "Updated Payroll Calculator with overtime pay"
git push origin main
```

Section 3–4

Static Methods

Static Methods

- You can add additional static methods to your application class besides the required `main()` method

- Static methods do not require us to instantiate the class

Here as we are first learning Java, they allow us to break our application logic up into more manageable "chunks"

Eventually, we will transition to fully object-oriented coding

- Since the class may not be instantiated, a static method is not allowed to access instance variables (!)

It can only access local variables or static variables

Declaring a static Method

- If a method doesn't return anything, it uses the word **void** as the return type

Example

MainApp.java

```
public class MainApp {  
  
    public static void main(String[] args) {  
        foo();  
        moo();  
    }  
  
    public static void foo() {  
        System.out.println("Foo");  
    }  
  
    public static void moo() {  
        System.out.println("Moo");  
    }  
}
```

OUTPUT

```
Foo  
Moo
```

Passing Data to Methods

- You can pass data (called arguments) to a method in Java

They are received in parameters in the order they are passed

Arguments and parameters don't have to have the same name, but they should be of the same type

Example

MainApp.java

```
public class MainApp {  
  
    public static void main(String[] args) {  
        int a = 4, b = 9, c = 10, d = 3;  
  
        addAndDisplay(a, b);  
        addAndDisplay(b, c);  
        addAndDisplay(a, d);  
    }  
  
    public static void addAndDisplay(int num1, int num2) {  
        int sum = num1 + num2;  
        System.out.printf("%d + %d = %d", num1, num2, sum);  
    }  
}
```

OUTPUT

```
4 + 9 = 13  
9 + 10 = 19  
4 + 3 = 7
```

Returning Data from Methods

- **Java requires you to specify the type of data that will be returned from a method**

Instead of `void`, you would specify the data type of the value being returned

Example

MainApp.java

```
public class MainApp {

    public static void main(String[] args) {
        int a = 4, b = 9, c = 10, d = 3;
        int sum;

        sum = addAndDisplay(a, b);
        display(a, b, sum);

        sum = addAndDisplay(b, c);
        display(b, c, sum);

        sum = addAndDisplay(a, d);
        display(a, d, sum);
    }

    public static int add(int num1, int num2) {
        int sum = num1 + num2;
        return sum;
    }

    public static void display(int num1, int num2, int sum) {
        System.out.printf("%d + %d = %d", num1, num2, sum);
    }
}
```

OUTPUT

```
4 + 9 = 13
9 + 10 = 19
4 + 3 = 7
```


Using a Scanner in Multiple Methods

- **If you want to read data from the keyboard in multiple methods, you have three choices**
- **You could declare a new `Scanner` object in each method**
 - It is easy to copy and paste the `Scanner` declaration over and over
 - This leads to duplicate code (yuck!)
- **You could declare one `Scanner` object at the class level**
 - However, because it is accessed by static methods, the `Scanner` object must be assigned to static instance
 - A static class member is shared by all instances of the class
- **You could pass the `Scanner` object from `main` to any method that needs to use it**
 - This requires an extra parameter but is fairly easy

Example: Re-declaring the Scanner

Example

```
import java.util.*;

public class InputApp {

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print(
            "What do you want to do (1-add, 2= subtract) ? ");

        int command = scanner.nextInt();
        if (command == 1) {
            doAdd();
        }
        else if (command == 2) {
            doSubtract();
        }
        else {
            System.out.printf(
                "%d -- Invalid command!", command);
        }
    }

    public static void doAdd() {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter 1st number: ");
        double num1 = scanner.nextDouble();

        System.out.print("Enter 2nd number: ");
        double num2 = scanner.nextDouble();

        double sum = num1 + num2;
        System.out.printf("%f + %f = %f", num1, num2, sum);
    }

    public static void doSubtract() {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter 1st number: ");
        double num1 = scanner.nextDouble();
```

Example: Re-declaring the Scanner *cont'd*

```
        System.out.print("Enter 2nd number: ");
        double num2 = scanner.nextDouble();

        double difference = num1 - num2;
        System.out.printf(
            "%f + %f = %f", num1, num2, difference);
    }
}
```

TRACE OF TERMINAL WINDOW SESSION

What do you want to do (1=add, 2= subtract) ? 1

Enter 1st number: 25.25
Enter 2nd number: 5.25
25.25 + 5.25 = 30.500000

Example: Declaring the Scanner at the Class-Level

Example

```
import java.util.*;

public class InputApp {

    static Scanner scanner = new Scanner(System.in);

    public static void main(String[] args) {
        System.out.print(
            "What do you want to do (1=add, 2= subtract) ? ");

        int command = scanner.nextInt();
        if (command == 1) {
            doAdd();
        }
        else if (command == 2) {
            doSubtract();
        }
        else {
            System.out.printf(
                "%d -- Invalid command!", command);
        }
    }

    public static void doAdd() {
        System.out.print("Enter 1st number: ");
        double num1 = scanner.nextDouble();

        System.out.print("Enter 2nd number: ");
        double num2 = scanner.nextDouble();

        double sum = num1 + num2;
        System.out.printf("%f + %f = %f", num1, num2, sum);
    }

    public static void doSubtract() {
        System.out.print("Enter 1st number: ");
        double num1 = scanner.nextDouble();
```

Example: Declaring the Scanner at the Class-Level *cont'd*

```
System.out.print("Enter 2nd number: ");
double num2 = scanner.nextDouble();

double difference = num1 - num2;
System.out.printf(
    "%f - %f = %f", num1, num2, difference);
}
}
```

TRACE OF TERMINAL WINDOW SESSION

What do you want to do (1=add, 2= subtract) ? 2

```
Enter 1st number: 25.25
Enter 2nd number: 5.25
25.25 - 5.25 = 20.000000
```

Example: Passing the Scanner

Example

```
import java.util.*;

public class InputApp {

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print(
            "What do you want to do (1-add, 2= subtract) ? ");

        int command = scanner.nextInt();
        if (command == 1) {
            doAdd(scanner);
        }
        else if (command == 2) {
            doSubtract(scanner);
        }
        else {
            System.out.printf(
                "%d -- Invalid command!", command);
        }
    }

    public static void doAdd(Scanner scanner) {

        System.out.print("Enter 1st number: ");
        double num1 = scanner.nextDouble();

        System.out.print("Enter 2nd number: ");
        double num2 = scanner.nextDouble();

        double sum = num1 + num2;
        System.out.printf("%f + %f = %f", num1, num2, sum);
    }

    public static void doSubtract(Scanner scanner) {

        System.out.print("Enter 1st number: ");
        double num1 = scanner.nextDouble();
```

Example: Passing the Scanner *cont'd*

```
System.out.print("Enter 2nd number: ");
double num2 = scanner.nextDouble();

double difference = num1 - num2;
System.out.printf(
    "%f - %f = %f", num1, num2, difference);
}
```

TRACE OF TERMINAL WINDOW SESSION

What do you want to do (1=add, 2= subtract) ? 1

Enter 1st number: 25.25
Enter 2nd number: 5.25
25.25 + 5.25 = 30.500000

Exercises

In this exercise you will refactor the code from your previous payroll calculator exercise. Refactoring is the process of updating existing code and breaking it into smaller methods to make it more modular and easier to maintain.

EXERCISE 5 (Optional)

Before you start making changes, make a plan to consider how you will break up your code. Use your notebook to diagram your plan. Talk to the people in your breakout room to discuss ideas about how you might do it.

Step 1

Code your changes

When you finish, we will get together as a class and examine some of the different approaches people used.

Step 2

Push your changes to GitHub

```
git add -A
git commit -m "Completed Mod3 - Exercise 3 - Basic Calculator"
git push origin main
```


Module 4

Conditionals

Section 4–1

Conditionals

Conditionals

- **Conditional logic lets us do different things depending on the value of an expression**
- **Conditionals include:**
 - `if` and `if/else` statements
 - `switch` statements
 - conditional operator (sometimes called the ternary operator)
 -

if Statement

- The if statement executes some code if a condition expression evaluates as **true**

- The condition is surrounded by parentheses

- Braces are only *required* around the code if there is more than one statement, but best practice is to always have them

Syntax

```
if (condition) {  
    // statement(s)  
}
```

- The condition is often expressed by comparing values:

== != < <= > >=

The equality and inequality operators can be used with both primitive types and objects

The other comparison operators only work with primitive types that can be represented in numbers

Example

```
String name = "Ezra";  
int age = 17;  
  
double price = 25.00;  
  
if (age < 18) {  
    price = price * .9;  
}  
  
System.out.println("Price of admission: " + price);
```

if / else

- The **else** statement can be used to write alternative code that executes when the condition is false

If the else condition has more than one statement, you must surround them with curly braces

- else must always follow an if

Example

```
String name = "Ezra";  
int age = 17;
```

```
double price;
```

```
if (age < 18) {  
    price = 22.50;  
}  
else {  
    price = 25.00;  
}
```

```
System.out.println("Price of admission: " + price);
```

if / else Statements *cont'd*

- You can string together **if / else if / else** statements for more complex logic

Example

```
String name = "Ezra";
int age = 17;

double price;

if (age < 18) {
    price = 18.00;
}
else if (age < 65) {
    price = 25.00;
}
else {
    price = 18.00;
}

System.out.println("Price of admission: " + price);
```

- You can build complex conditions by using the logical operators: **&&** **||** **!**

Example

```
String name = "Ezra";
int age = 17;

double price;

if (age < 18 || age >= 65) {
    price = 18.00;
}
else {
    price = 25.00;
}
```

Comparing Strings

- The `==` operator tests whether both operands are the same (identical) object

This doesn't work reliably for Strings, since the string you read as input is a different object from the one that holds a literal value

- The String class' `equals()` method tests whether the strings contain the same characters in the same order

Example

```
String homeState = "Texas";
String contactPhone;

if (homeState.equals("Texas") || homeState.equals("Kansas")) {
    contactPhone = "800-555-5555";
}
else {
    contactPhone = "855-555-5555";
}
```

Example: Better Calculator?

Example

```
import java.util.*;

public class InputApp {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);
        System.out.print(
            "What do you want to do (add, subtract) ? ");

        String command = scanner.nextLine();
        if (command.equals("add")) {
            doAdd(scanner);
        }
        else if (command.equals("subtract")) {
            doSubtract(scanner);
        }
        else {
            System.out.printf(
                "%s -- Invalid command!", command);
        }
    }

    public static void doAdd(Scanner scanner) {
        System.out.print("Enter 1st number: ");
        double num1 = scanner.nextDouble();

        System.out.print("Enter 2nd number: ");
        double num2 = scanner.nextDouble();

        double sum = num1 + num2;
        System.out.printf("%f + %f = %f", num1, num2, sum);
    }

    public static void doSubtract(Scanner scanner) {
        System.out.print("Enter 1st number: ");
        double num1 = scanner.nextDouble();

        System.out.print("Enter 2nd number: ");
        double num2 = scanner.nextDouble();

        double difference = num1 - num2;
        System.out.printf(
            "%f + %f = %f", num1, num2, difference);
    }
}
```


Conditional Operator

- The conditional operator (`?:`) in Java is like a shorthand version of `if/else`

- A similar operator is found in all C-family languages

Syntax

```
variable = (condition) ? expressionTrue : expressionFalse;
```

Example

```
String name = "Ezra";  
int age = 17;  
  
double price = (age < 18) ? 22.50 : 25.00;  
  
System.out.println("Price of admission: " + price);
```

Exercises

In these exercises you will create a Sandwich Shop application. You will allow users to order a simple sandwich and you will calculate the price. The customer may receive a discount based on their age.

Remember to create your projects in the `pluralsight/java-development/workbook-1` folder.

EXERCISE 1

Create a Java application named **SandwichShop**. This will be a point of sales application to calculate the cost of a sandwich.

Step 1

Prompt the user for the size of the sandwich (1 or 2):

- 1: Regular: base price \$5.45
- 2: Large: base price \$8.95

Prompt the user for their age:

- Student (17 years old or younger) – receive a 10% discount
- Seniors (65 years old or older) – receive a 20% discount

Display the cost of the sandwich to the screen

Step 2

Push your changes to GitHub

```
git add -A
git commit -m "Completed Mod4 - Exercise 1 - Sandwich Shop"
git push origin main
```

EXERCISE 2

Modify the **SandwichShop** from the last exercise to allow customers to order a "loaded" sandwich (double everything).

Step 1

Prompt the user for the size of the sandwich (1 or 2):

- 1: Regular: base price \$5.45
- 2: Large: base price \$8.95

Prompt the user if they would like the sandwich "loaded" (yes/no). If so there is an additional cost for a loaded sandwich

- Regular: \$1.00
- Large: \$1.75

Prompt the user for their age:

- Student (17 years old or younger) – receive a 10% discount
- Seniors (65 years old or older) – receive a 20% discount

Display the cost of the sandwich to the screen

Step 2

Push your changes to GitHub

```
git add -A
git commit -m "Added Loaded option for the sandwich shop"
git push origin main
```

Section 4–2

The `switch` Statement

switch Statement

- The **switch** statement evaluates an expression and executes the code in a matching **case** block

- If there is no match, the `default` block executes

- Code in the case executes until a `break` statement is encountered or the switch statement ends

Syntax

```
switch(expression) {  
    case value-1:  
        // code block  
        break;  
    case value-2:  
        // code block  
        break;  
    default:  
        // code block  
}
```

switch Example

- Switch statements can get pretty long if the expression has many unique values

- It's still easier to read than the equivalent if/else construct

Example

```
int dayNumber = 3;
String description = "";

switch (dayNumber) {
    case 0:
        description = "Sunday";
        break;
    case 1:
        description = "Monday";
        break;
    case 2:
        description = "Tuesday";
        break;
    case 3:
        description = "Wednesday";
        break;
    case 4:
        description = "Thursday";
        break;
    case 5:
        description = "Friday";
        break;
    case 6:
        description = "Saturday";
        break;    // optional break -- okay without it
}
```

The break statement

- If a **case** block doesn't end with a **break**, then code falls into the next block

This is intentional; it can make your switch statement more compact when several values lead to the same behavior

Example

```
int dayNumber = 3;
String description = "";

switch (dayNumber) {
    case 0:
    case 6:
        description = "Weekend";
        break;
    case 1:
    case 2:
    case 3:
    case 4:
    case 5:
        description = "Weekday";
        break;
}
```

Exercises

In this exercise you will create a CLI application for a rental car company. You will prompt the user to answer questions about their selected options and then calculate the cost of the rental.

You are not required to use only `switch` statements in this exercise. You can choose to use `if/else` or `switch` statements as you deem appropriate.

EXERCISE 1

Step 1

IMPORTANT: DO NOT just start coding - take time to plan the flow of your application in your notebook.

Read the requirements below and consider how you would calculate the cost if you did not have a computer. Practice several scenarios (with different options) in your notebook before you create your project in IntelliJ.

Take a picture or screenshot of your notebook and save the image to your project directory (after you create the project).

Step 2

Create a Java application named `RentalCarCalculator` that estimates the cost of reserving a rental car. Prompt the user for the following information:

- pickup date (store as a string)
- number of days for the rental
- whether they want an electronic toll tag at \$3.95/day (yes/no)
- whether they want a GPS at \$2.95/day (yes/no)
- whether they want roadside assistance at \$3.95/day (yes/no)
- their current age

Calculate and display:

- basic car rental
- options cost
- underage driver surcharge
- total cost

The basic car rental is \$29.99 per day. There is a 30% surcharge on the basic car rental for drivers under 25. All taxes have already been incorporated into the fees shown.

Step 3

Push your changes to GitHub

```
git add -A
git commit -m "Completed Mod4 - Exercise 1 - Rental Car Calculator"
git push origin main
```