Computer Programming in Java
Bush School *CPJava Fall 2020*

Welcome to CPJava!
Start in Remote Schooling format
Follow in Hybrid format in Oct

Chandru Narayan
Bicyclist Astronomer Engineer
Teaching Math and CS

A unique first course in programming with some advanced topics!

1. Visual and Project-based learning
2. Computing and the Web
3. Programming using Java
4. Simulate Natural Systems
5. Machine Learning
6. Develop Games
7. Learn Math & Physics Concepts
8. AP Exam preparation (optional)

Expectations of Learning

- Advanced Java Programming
- Object Oriented including inheritance
- How to make and maintain your own website
- How to use Git and Git Hub
- Searching and sorting
- Recursion
- Robust code design and style
- APCS A topics
- Code Vectors, Newton's Laws, Machine Learning etc
- + Fun stuff like Asteroids, Super Mario Games and Computer Art that they don't teach you in college!

CPJava course is online!

- The complete 1-year coursework is online
- We'll start at the top and work our way to the bottom

You are here!

Grading Policy

Grades are cumulative for the full-year

<u>Fall Semester</u>

40% Projects

40% Classwork / Assignments

20% Student Portfolio

<u>Spring Semester</u>

TBD

How to succeed

Simply write lots of of Java code

Publish your work to the web (Github)

You cannot learn programming by reading or watching!

Make lots of mistakes - truly the best way to learn to code!

Working cooperatively with your peers

Don’t be afraid to try new things!

Tools/Resources for CPJava

Zoom - Live and Recorded Class Sessions

CPJava website - Lesson Plans and Projects

Bush Portal - Course Syllabus, Schedule and Grades

CPJava Google Classroom - Classroom code nnqro5m

Processing IDE Download - Java Editor and Compiler

Github - Projects and Portfolio Submissions

repl.it - Code Exercise Submissions

CPJava Slack Channel - Instructor and Student Communications (eMail is not preferred)

CSAwesome Online Course - Additional/Detail topics

System Requirements

A robust Internet connection

A laptop Computer - Mac or Windows (Chromebook is not sufficient)

Slides show Mac screens - Windows will look slightly different

Let's write first program!

Download Processing

Write and run your first program!

(do steps in numerical sequence noted)

Submitting your GitHub URL

- Sign in to GitHub and go to your profile

- Copy the URL of your p

- Sign in to Google Classroom *for GitHub.com* assignment. Choose *Add* | *Link* to submit the link to Google classroom

Tutoring

- Peer tutoring through CSF:
  - appointments can be schedule at lowellcsf.weebly.com
  - drop in tutoring in the library every school day from Block 1 to 4:45 PM after school
- Lowell's after school computer science clubs:
  - Girls Who Code, Mondays room 334
  - App Development club, Wednesdays room 334
  - Byte Lab, Thursdays room 238
  - People Who Code (aka Host Club) Thursdays room 334

in class, but. . .

- *Please* do not through away food or wrappers in the classroom
- Use the composting bins in the hallway

Lowell has a mouse problem

Java Basics

- Loops
- if statements
- Functions (aka “methods”)

- It’s expected that you have done programming with these already in some language
- Over the next two weeks we’ll go over how these work in Java

A basic Java "Hello World" program

- Click on the link to the "Four 4s Challenge" of just go to apcslowell.github.io/APJavaBasics/
- It might look complicated at first, but . . .

Hello.java

```
1  public class Hello {
2
3      public static void main(String[] args) {
4          System.out.println("Hello World!");
5      }
6  }
```

A basic Java program

- It turns out that this line is much more important than the others
- So we can ignore everything except the code circled in green

Hello.java

```
1  public class Hello {
2
3      public static void main(String[] args) {
4          System.out.println("Hello World!");
5      }
6  }
```

Statements

- The circled code is an example of a *statement*
- It's like an English sentence
- A semi-colon marks the end of a Java statement

Hello.java

```
1  public class Hello {
2
3      public static void main(String[] args) {
4          System.out.println("Hello World!");
5      }
6  }
```

String

- A **String** is a collection of letters, digits, punctuation and/or spaces
- The beginning and end of the **String** are marked with double quotes (")

Hello.java

```
1  public class Hello {
2
3      public static void main(String[] args) {
4          System.out.println("Hello World!");
5      }
6  }
```

- `System.out.print()` vs `System.out.println()`
  - `System.out.println()` prints first and then goes to the next line

Java beta Run

Goodbye.java

```
1  public class Goodbye {
2
3      public static void main(String[] args) {
4          System.out.println("Top");
5          System.out.println("Bottom");
6      }
7  }
```

Powered by trinket

```
Top
Bottom
```

print() vs println()

- `System.out.print()` prints, but it does NOT go to the next line
- If we change the first statement to `System.out.print() "Bottom"` is printed on the same line as `"Top"`

print() vs println()

- Changing the second statement to `System.out.print()` doesn't change the output since nothing is printed after `"Bottom"`

Java beta ▶ Run

Goodbye.java

```
1 public class Goodbye {
2
3     public static void main(String[] args) {
4         System.out.println("Top");
5         System.out.print("Bottom");
6     }
7 }
```

Powered by trinket

```
Top
Bottom
```

Comments

- Comments have no effect on the

ey make

and your

at you

Hello.java

```
1  public class Hello {
2
3      public static void main(String[] args) {
4          // generate some simple output
5          System.out.println("Hello, World!");
6      }
7  }
```

Escape Sequences

- In Java, Escape Sequences begin with a backslash \
- *A good way to remember the difference between a backslash and a forward slash is that a backslash leans backwards (\ ), while a forward slash leans forward (/)*

Common Java Escape Sequences

- `\n` Insert a newline in the text at this point

Java beta Run

Goodbye.java

```
1 public class Goodbye {
2   public static void main(String[] args) {
3     System.out.print("Top\nBottom");
4   }
5 }
```

Powered by trinket

```
Top
Bottom
```

Common Java Escape Sequences

- `\t` Insert a tab in the text at this point

Common Java Escape Sequences

- \' Insert a single quote character
- \" Insert a double quote character
- \\ Insert a backslash character

Formatting Java code

- In Java programs, some spaces are required
- For example, you need at least one space between keywords
- The program below is not legal

```
publicclassGoodbye{

    publicstaticvoidmain(String[] args) {
        System.out.print("Goodbye, ");
        System.out.println("cruel world");
    }
}
```

Formatting Java code

- But most other spaces are optional
- For example, this program is legal but hard to read

```
public class Goodbye { public static void main(String[] args)
{ System.out.print("Goodbye, "); System.out.println
("cruel world");}}
```

Formatting for easier reading

- The blank space (also called "white space") commonly used to format code is:
  - Indentation inside of **{ }**
  - One statement per line

```
public class Goodbye {
  public static void main(String[] args) {
    System.out.print("Goodbye, ");
    System.out.println("cruel world");
  }
}
```

Debugging

called “debugging”

- It’s good to work around other programmers when you are learning a new programming language
- Asking for help with debugging is a part of learning

often get an error message

Errors

- When you are learning a new programming language, errors are a fact of life
- Errors are ok, just fix them and move on

In this case I made a *syntax* error

Syntax error

- *Syntax* is the grammar and spelling of a computer language
- Here I forgot the double quotes around my name

- This time I misspelled my name
- The computer doesn't know my name, so the program runs incorrectly without an error message

Logic error

(Run time) Exceptions

- Sometimes a logic error crashes the computer and stops the running program
- Here I made the logic error of dividing by zero

Java beta Run Copy

Hello.java

```
1 public class Hello {
2
3     public static void main(String[] args) {
4         // generate some simple output
5         System.out.println(1/0);
6     }
7 }
```

Powered by trinket

```
Exception in thread "main"
java.lang.ArithmeticException: / by zero
        at Hello.main(Hello.java:5)
```

Arithmetic in Java

- Addition
- Subtraction
- Multiplication
- Division

Literals vs. Expressions

- Double quotes around text tells Java it is an expression
- Java will **`print`** an expression exactly as written

Literals vs Expressions

- Here's an expression `"4/4"`
- Java **`prints`** it in exactly the same form

Java beta Run

Hello.java

```
1  public class Hello {
2
3      public static void main(String[] args) {
4          System.out.println("4/4");
5          System.out.println(4/4);
6      }
7  }
```

Powered

```
4/4
1
```

Literals vs. Expressions

- Here's an expression `4/4`
- Java evaluates it to get an answer 1
- And then **`prints`** it

Java beta ▶ Run

Hello.java

```
1  public class Hello {
2
3      public static void main(String[] args) {
4          System.out.println("4/4");
5          System.out.println(4/4);
6      }
7  }
```

Powered

4/4

1

Four 4s challenge

- Use exactly four **4**'s to write an expression that evaluates to every integer from 1 to 10, using only + − * / and ()
- No decimals, factorials, square roots, exponents, etc.

Four 4s challenge

- Print 10 expressions that use arithmetic and four 4s that evaluate to 1 through 10
- Here's one way to do the first

Four 4s challenge

- Here's another way to do the first
- If you have extra time, try to get 11, 12, 13, etc.

Chapter 2

- Declarations
- Initializations
- Comments
- `%` (modulus) and Integer division
- `+` and `String`s

Variables

- The value of a variable can *change* (think *vary*)
- A Java variable has size limits for its ***type*** (for example, integers are limited to values between -2,147,483,648 and 2,147,483,647)
- Every Java variable has a ***type*** and a ***name***

Variables: Parking space analogy

- Like a parking space, a variable can store a value (think *vehicle*) until you need it later

Variables: Parking space analogy

- Parking spaces are often labeled so you can find your car later when you need it
- Lets say that **`G210`** is the *name* of this parking space

Variables: Parking space analogy

- In addition to a ***name***, parking spaces can have a ***type*** that sets size limits

- We are trying to print an integer that is too large for Java's integer size

What is the error?

Java beta Run

Hello.java

```
1 public class Hello {
2
3     public static void main(String[] args) {
4         // generate some simple output
5         System.out.println(1231231231231231231232313);
6     }
7 }
```

Copy

Powered by trinket

```
Hello.java:5: error: integer number too large:
1231231231231231231232313

System.out.println(1231231231231231231232313);
                   ^

1 error
```

Primitive data types

"primitive" types:

- `int`
- `float`
- `double`
- `boolean`
- `char`

Primitive data types

- **`double`** a decimal value with up to 15 digits
- A **`boolean`** can only hold values that evaluate to either **`true`** or **`false`**
- **`char`** holds a single letter, digit, space or punctuation mark and must be enclosed in *single quotes*, like this: **`'G'`**

float vs. double

- `float` is short for *floating point*, another name for *decimal point*
- `double` gets its name because of its size, it has about twice as many digits as a `float`

Java beta Run

Hello.java

```
1  public class Hello {
2
3      public static void main(String[] args) {
4          double dNum = 5/9.0;
5          System.out.println(dNum);
6          System.out.println((float)dNum);
7      }
8  }
```

Powered by trinket

```
0.5555555555555556
0.5555556
```

String variables

- The beginning and end of the text is marked with double quotes (")

+ and Strings

- Using `+` with `String`s isn't addition arithmetic, it's called *concatenation*
- That's a fancy word that means making bigger `String`s out of little ones

+ and `Strings`

- Java executes from left to right so, `1 + 2` is `3`,
- `"llo"`
- `"Hello1" + 2`

```
System.out.println(1 + 2 + "Hello");
// the output is 3Hello

System.out.println("Hello" + 1 + 2);
// the output is Hello12
```

- We could make `11` with four 4s by putting an empty `String` in the middle
- (`String` *concatenation is not allowed in the rules of the four 4s challenge though)*

Hello.java

```
1 public class Hello {
2
3     public static void main(String[] args) {
4         System.out.println(4/4+""+4/4);
5     }
6 }
```

Power

11

called a *declaration*

- Here's an example declaration

A Java variable declaration

```
int num;
```

- The first word of the declaration is the type
- The second word is the name that the programmer chooses

Variable names and keywords

- start with a number
- contain a space
- have a special meaning in Java

- Java has about 50 reserved words or *keywords* that you are not allowed to use as variable names such as **`public`**, **`class`**, **`static`**, **`void`**, **`int`** ...

one word

camelCase

- **camelCase** capitalizes the first letter of each word except the first word
- Examples: **`firstName`**, **`numberOfDogs`**
- Java variable names are case-sensitive, so **`firstName`** is not the same as **`firstname`** or **`FirstName`**.

What's the error?

Java beta Run Copy

Hello.java

```
1 public class Hello {
2
3     public static void main(String[] args) {
4         int num;
5         System.out.println(num);
6     }
7 }
```

Powered by tinket

```
Hello.java:5: error: variable num might not
have been initialized
        System.out.println(num);
                           ^
1 error
```

Variable initialization

- After a Java variable is declared, it needs to be *initialized*

Java beta Run Copy

Hello.java

```
1 public class Hello {
2
3     public static void main(String[] args) {
4         int num;
5         System.out.println(num);
6     }
7 }
```

Powered by trinket

```
Hello.java:5: error: variable num might not
have been initialized
        System.out.println(num);
                           ^
1 error
```

Variable Initialization

- The English word "initial" means *first*
- We initialize a variable by assigning ("setting it equal to") its *first* value

with one line of code:

Declare *and* initialize

```
int num = 3;
```

- Just remember that the one line of code is doing two different steps: the ***declaration*** and the ***initialization***

Line Comments

- Tells the computer to ignore some text
- Used to:
  - Write notes to yourself or other programmers
  - Temporarily disable code

Arithmetic operators

- Addition
- Subtraction
- Multiplication
- Division
- Modulus (also *Mod* or *Modulo*) calculates the remainder of dividing two integers

Modulus and integer division

Remember how you did math in grade school?

$$5\overline{)8}$$

64

Modulus and integer division

Remember how you did math in grade school?

$$\begin{array}{r} 1 \\ 5\overline{)\,8} \end{array}$$

65

Modulus and integer division

Remember how you did math in grade school?

$$
\begin{array}{r}
1 \\
5\overline{)\,8} \\
5 \\
\hline
\end{array}
$$

66

Modulus and integer division

Remember how you did math in grade school?

$$
\begin{array}{r}
1 \\
5\overline{)\,8} \\
5 \\
\hline
3
\end{array}
$$

67

Modulus and integer division

The modulus operator gives the remainder of an integer division expression

8 / 5

8 % 5

Kahoot!

Integer Division in Java

$$
\begin{array}{r}
0 \\
8\overline{)5} \\
0 \\
\hline
5
\end{array}
$$

In class: Chapter 2 ex #2 & 1

- Then find a partner do ex 1: Stump the Chump

books.trinket.io/thinkjava

Hello.java

```
3   public static void main(String[] args) {
4     ?? day = "Wednesday";
5     ?? date = 22;
6     ?? month = "August";
7     ?? year = 2018;
8     System.out.println("American Format:");
9     System.out.print(day + ", ");
10    System.out.print(month + " ");
11    //your code here
12    System.out.println("European Format:");
13    // more of your code here
14  }
```

Powered by trinket

```
American Format:
Wednesday, August 22, 2018
European Format:
Wednesday 22 August 2018
```

Exercise 1

If you are using this book in a class, you might enjoy this exercise. Find a partner and play "Stump the Chump":

Start with a program that compiles and runs correctly. One player looks away while the other player adds an error to the program. Then the first player tries to find and fix the error.

In class: integers vs. decimals

- When the types are mixed, the Java uses the larger type

Hello.java

```
1  public class Hello {
2
3      public static void main(String[] args) {
4          System.out.println(9/5);
5          System.out.println(9.0/5);
6          System.out.println(9/5.0);
7          System.out.println(9.0/5.0);
8      }
9  }
```

Power

```
1
1.8
1.8
1.8
```

In class: Ch2 #3 Arithmetic Operators

- Do calculations to display:
  - Time in seconds since last midnight
  - Seconds remaining until midnight tonight
  - Percentage of day passed (hint: you will need to use at least one decimal number)

books.trinket.io/thinkjava

Hello.java

```java
public class Hello {

    public static void main(String[] args) {
      int hour = 18, minute = 58, second = 10;

      System.out.println("Time since midnight: " + ??);
      System.out.println("Time remaining today: " + ??);
      System.out.println("Percentage of day passed: " + ??);

    }
}
```

Powered by trinket

```
Time since midnight: 68290 seconds
Time remaining today: 18110 seconds
Percentage of day passed: 0.7903935185185185%
```

Chapter 3: Input with Scanner

- Add scanner code to program
- `Scanner in = new Scanner(System.in);`
  - create a **`new Scanner`**
- `int someInteger = in.nextInt();`
  - Get an integer and store it in **`someInteger`**

Echo.java

```
1  import java.util.Scanner;
2
3  public class Echo {
4
5      public static void main(String[] args) {
6          Scanner in = new Scanner(System.in);
7
8          System.out.print("Type an integer: ");
9          int someInteger = in.nextInt();
10         System.out.println("You typed: " + someInteger);
11
12     }
13 }
```

Powered by trinket

```
Type an integer:  42
You typed: 42
```

Chapter 3: The types must match

```
String someInteger = in.nextInt();
```

- Java won't let you store an **`int`** in a String

Echo.java

```java
import java.util.Scanner;

public class Echo {

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

        System.out.print("Type an integer: ");
        String someInteger = in.nextInt();
        System.out.println("You typed: " + someInteger);

    }
}
```

Powered by trinket

```
Echo.java:9: error:
incompatible types: int
cannot be converted to
String
        String someIntege
= in.nextInt();

^
1 error
```

An online time converter

www.tools4noobs.com/online_tools/seconds_to_hh_mm_ss/

- 10,000 seconds is 2 hours 46 minutes and 10 seconds

10,000%(60*60) is 2,800

Using modulus

2,800 / 60 is 46 (minutes)

2,800%60 is 40 (seconds)

In class: Chapter 3 Exercise 3

seconds converter

You can use the Echo program in chapter 3 as a starting point

Echo.java

```java
import java.util.Scanner;

public class Echo {

    public static void main(String[] args) {
        int seconds;
        Scanner in = new Scanner(System.in);

        System.out.print("Type a number of seconds: ");
        seconds = in.nextInt();
        System.out.println(seconds + " seconds is:");
        int hours = seconds/(60 * 60);
        seconds = seconds%(60*60);
        System.out.println(hours + " hours");
        //Your java code here
    }
}
```

Powered by trinket

```
Type a number of seconds:
10000
10000 seconds is:
2 hours
46 minutes
and 40 seconds
```

Chapter 4: Functions

- Functions should do just one job or task
- Functions in Java are identified with parenthesis (“methods”)
- The parenthesis may or may not have something inside called an argument

```
System.out.println("Hello World");
in.nextInt();
```

no argument

argument

Functions can be used for an *effect* or *value*

- `System.out.println()` has an *effect*, it causes something to be displayed
- `in.nextInt()` is used to get the *value* of what the user typed
- Other functions that we will learn about in a couple days can be used to calculate mathematical *values*

`void` functions (aka methods)

You can *define* (create) your own functions that are used for effect with the Java keyword **`void`**

Java beta ▶ Run

NewLine.java

```java
public class NewLine {

    public static void blah() {
        System.out.println("blah, blah, blah");
    }

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

Powered by trinket

```
blah, blah, blah
```

`void` functions (aka methods)

- Functions are like paragraphs of computer code
- Every Java program has a **`main`** function
- There can be many other functions

Java beta ▶ Run

NewLine.java

```
1  public class NewLine {
2
3      public static void blah() {
4          System.out.println("blah, blah, blah");
5      }
6
7      public static void main(String[] args) {
8          blah();
9      }
10 }
```

Powered by trinket

```
blah, blah, blah
```

void functions (aka methods)

- You can *call* (use) your function by typing its name without **void**

Java beta Run

NewLine.java

```java
public class NewLine {

    public static void blah() {
        System.out.println("blah, blah, blah");
    }

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

Powered by trinket

```
blah, blah, blah
```

void functions (aka methods)

- Note that the order the functions are *defined* is unimportant
- Java always runs the **main** function first

Java beta ▶ Run

NewLine.java

```java
public class NewLine {

    public static void main(String[] args) {
        blah();
    }

    public static void blah() {
        System.out.println("blah, blah, blah");
    }

}
```

Powered by trinket

```
blah, blah, blah
```

void functions (aka methods)

- The order of the function *calls* is important
- Functions can be reused as many times as you want

Java beta Run

NewLine.java

```
1  public class NewLine {
2
3      public static void main(String[] args) {
4          blah();
5          boring();
6          blah();
7      }
8
9      public static void blah() {
10         System.out.println("blah, blah, blah");
11     }
12     public static void boring() {
13         System.out.println("boring, boring, boring");
14     }
15 }
```

Powered by trinket

```
blah, blah, blah
boring, boring, boring
blah, blah, blah
```

- The function call has matching *arguments*
- The values in variables **h** & **m** are copied into variables **hour** & **minute**

Java beta Run

PrintTime.java

```
1  public class PrintTime {
2
3      public static void printTime(int hour, int minute) {
4          System.out.print(hour);
5          System.out.print(":");
6          System.out.println(minute);
7      }
8
9      public static void main(String[] args) {
10         int h = 7;
11         int m = 35;
12         printTime(h, m);
13     }
14 }
```

Powered b

```
7:35
```

Arguments must match Parameters in number, order and type

What's the problem?

- Order, the parameters are **`int String`** order but the arguments are in **`String int`** order

Java beta ▶ Run

Main.java

```
1  public class Main {
2
3      public static void someFunction(int num, String word){
4          System.out.println(num + ", " + word);
5      }
6      public static void main(String[] args) {
7          int n = 1;
8          String w = "one";
9          someFunction(w,n);
10     }
11
12 }
```

Powered by trinket

```
Main.java:9: error:
incompatible types: String
cannot be converted to int
        someFunction(w,n);
                     ^
Note: Some messages have been
simplified; recompile with -
Xdiags:verbose to get full
output
1 error
```

Variables can only be used int the function where they are declared

What's the problem?

- **n** and **w** were declared in the **main** function and are not available in **someFunction**

Main.java

```
1  public class Main {
2
3      public static void someFunction(int num, String word){
4        System.out.println(n + ", " + w);
5      }
6      public static void main(String[] args) {
7          int n = 1;
8          String w = "one";
9          someFunction(n,w);
10     }
11
12 }
```

Powered by trinket

```
Main.java:4: error: cannot
find symbol
      System.out.println(n +
", " + w);
                          ^
  symbol:   variable n
  location: class Main
Main.java:4: error: cannot
find symbol
      System.out.println(n +
", " + w);
```

Now the program is correct

- The arguments match the parameters in number order and type
- The correct variables are used in each function

Main.java

```
1  public class Main {
2
3      public static void someFunction(int num, String word){
4        System.out.println(num + ", " + word);
5      }
6      public static void main(String[] args) {
7          int n = 1;
8          String w = "one";
9          someFunction(n,w);
10     }
11
12 }
```

Powered by trinket

```
1, one
```

In class: Chapter 4 Exercise 3

Go to Google classroom and click on the link

https://trinket.io/embed/java/481075601e

Submit the link to your finished program by choosing *Share* | *Link*

trinket Java beta Run Share

PrintTime.java

```java
public class PrintTime {

    public static void printEuropean(String day,int date,String month, int year) {
        //your java code
    }
    public static void printAmerican(String day,int date,String month, int year) {
        //your java code
    }
    public static void main(String[] args) {
        String dy = "Friday";
        String mn = "August";
        int dt = 24;
        int yr = 2018;
        //function call to printAmerican
        //function call to printEuropean
    }
}
```

Powered by trinket

```
Friday, August 24, 2018
Friday, 24 August, 2018
```

Submit to google classroom by choosing *Share* | *Link* and submitting the link to google classroom

Kahoot!

End of first week AP Java

Chapter 5

- Logical operators
- Relational operators
- Short circuit evaluation
- Distributing the ! (De Morgan's laws)
- We'll skip recursion and stack diagrams for now

Relational Operators

- Is Greater Than
- Is Greater Than or Equal
- Is Less Than
- Is Less Than or Equal
- Is Not equal
- Is Equal to

Logical operators

- And
- Or
- Not
- Logical operators are sometimes also called *Boolean* operators

Truth Tables

&&	T	F
T	T	F
F	F	F

||	T	F
T	T	T
F	T	F

!	T	F
	F	T

- Truth tables show the results combining expressions with **&&** **||** or **!**

Logical Oper

- What would be p

&&	T	F
T	T	F
F	F	F

||	T	F
T	T	T
F	T	F

!	T	F
	F	T

```
int nCount = 150;
System.out.println((0<nCount)&&(nCount<=100));
```

Logical Oper

- What would be p

&&	T	F
T	T	F
F	F	F

||	T	F
T	T	T
F	T	F

!	T	F
	F	T

```
int nCount = 150;
System.out.println((0<nCount)&&(nCount<=100));
```

Hello.java

```
public class Hello {
  public static void main(String[] args) {
    int nCount = 150;
    System.out.println(( 0 < nCount ) && ( nCount <= 100 ));
  }
}
```

Powered by trinket

```
false
```

Logical Oper

&&	T	F
T	T	F
F	F	F

||	T	F
T	T	T
F	T	F

!	T	F
	F	T

- Now what would be

```
int nCount = 50;
System.out.println((0<nCount)&&(nCount<=100));
```

Logical Oper

&&	T	F
T	T	F
F	F	F

||	T	F
T	T	T
F	T	F

!	T	F
	F	T

- What would be p

```
int nCount = 50;
System.out.println((0<nCount)&&(nCount<=100));
```

Hello.java

```
1
2 public class Hello {
3   public static void main(String[] args) {
4     int nCount = 50;
5     System.out.println(( 0 < nCount ) && ( nCount <= 100 ));
6   }
7 }
```

Powered by trinket

```
true
```

How would Java evaluate these expressions?

```
//true
char cGrade = 'B';
('A' <= cGrade) && (cGrade
//true
```

&&	T	F
T	T	F
F	F	F

||	T	F
T	T	T
F	T	F

!	T	F
	F	T

```
int nAge = 16;
! (19 >= nAge && nAge >= 13)
//false
int nAge = 16;
19 < nAge || nAge < 13
//false
```

Short circuit evaluation

- **`true ||`** ***`anything`*** is always **`true`**, so Java doe[illegible]uate the ex[illegible]***ng***

- Likewise, **`false &&`** ***`anything`*** is always false
- Ignoring the ***`anything`***, when possible, is called **short circuit evaluation**

Short circuit evaluation

- What does `-3 < 0 || 5*2-17/2 < 1128` evaluate

&&	T	F
T	T	F
F	F	F

||	T	F
T	T	T
F	T	F

!	T	F
	F	T

- true
- Its not necessary to evaluate the second condition because the first condition is `true` and `true || anything` is always `true`

Short circuit evaluation

What does

`1==0 && 274/17 > 7846/461`

&&	T	F
T	T	F
F	F	F

||	T	F
T	T	T
F	T	F

!	T	F
	F	T

- Its not necessary to evaluate the second condition because the first condition is `false` and `false && anything` is always `false`

Distributing the !

- The rules for distributing the `!` are known as De Morgan's laws

`!(A && B)` *is the same as* `!A || !B`

`!(A || B)` *is the same as* `!A && !B`

Distributing the !

is the same as

```
x >= 5 || y != 3
```

```
!(x >= 1 || y != 7)
```

is the same as

```
x < 1 && y == 7
```

code in the curly braces (called a block)

`if` statements

executes

```
if (x > 0) {
    System.out.println("x is positive");
}
```

if else statements

```
if (x > 0) {
    System.out.println("x is positive");
}
else {
    System.out.println("x is NOT positive");
}
```

if else if statements

```
} else if (temp > 50) {
System.out.println("Warm");
} else if (temp > 50) {
System.out.println("Cool");
} else {
System.out.println("Cold");
}
```

if statements

is the same as

```
if (x > 0)
    System.out.println("x is positive");
```

More short circuit evaluation

Java beta Run Copy

NewLine.java

```
1  public class NewLine {
2
3      public static void main(String[] args) {
4          if(1/0 != 0){
5              System.out.println("Hello World!");
6          }
7      }
8  }
```

Powered by trinket

```
Exception in thread "main"
java.lang.ArithmeticException: / by
zero
        at
NewLine.main(NewLine.java:4)
```

because of the divide by zero exception in the `if(1/0 != 0)`

More short circuit evaluation

- Because `1==1 || anything` is `true`, Java does not execute the code that would cause an exception

More short circuit evaluation

Java beta ▶ Run Copy

NewLine.java

```
1  public class NewLine {
2
3      public static void main(String[] args) {
4          if(1/0 != 0 || 1==1){
5              System.out.println("Hello World!");
6          }
7      }
8  }
```

```
Powered by trinket
Exception in thread "main"
java.lang.ArithmeticException: / by
zero
        at
NewLine.main(NewLine.java:4)
```

code that causes the exception executes first so

Hello World!

is not printed

In class: Chapter 5 Ex 1, 2 and 6

Write your answers in the google doc, save and submit

Think Java Chapter 5

File Edit View Insert Format Tools Add-ons Help Last edit was 3 days ago

100% Normal text Verdana 12

Complete the following 3 exercises (#1,2 and6) from Chapter 5 of Think Java and submit to Google Classroom

Exercise 1

Logical operators can simplify nested conditional statements. For example, can you rewrite this code using a single `if` statement?

```
if (x > 0) {
    if (x < 10) {
        System.out.println("positive single digit number.");
    }
}
```

Exercise 2 For the following program:

1. What do you predict will be the output of the following program?

```
public static void zoop(String fred, int bob) {
    System.out.println(fred);
```

In class: Chapter 5 Exercise 1

- Combine the two `if` statements into one using a logical operator
- You can use a program like the one below to check your answer

NewLine.java

```java
public class NewLine {

  public static void main(String[] args) {
    int x = 10;
    if (x > 0) {
      if (x < 10) {
        System.out.println("positive single digit number.");
      }
    }
  }
}
```

Matching names & curly braces

- Your program must have a **`main`** function
- You need matching curly braces

NewLine.java

```
1  public class NewLine {
2
3    public static void main(String[] args) {
4      int x = 8;
5      if(x > 0 && x < 10){
6          System.out.println("positive single digit");
7      }
8    }
9  }
```

Powered by trinket

```
positive single digit
```

```
int buzz = 2;
zoop("just for", bizz);
clink(2 * buzz);
}
public static void clink(int fork) {
System.out.print("It's ");
zoop("breakfast ", fork);
}
public static void ping(String strangStrung) {
System.out.println("any " + strangStrung + "more ");
}
```

Chapter 5 Exercise 2

```
int buzz = 2;
zoop("just for", bizz);
clink(2 * buzz);
}
public static void clink(int fork) {
System.out.print("It's ");
zoop("breakfast ", fork);
}
public static void ping(String strangStrung) {
System.out.println("any " + strangStrung + "more ");
}
```

Chapter 5 Exercise 2

Output:

just for

```
int buzz = 2;
zoop("just for", bizz);
clink(2 * buzz);
}
public static void clink(int fork) {
System.out.print("It's ");
zoop("breakfast ", fork);
}
public static void ping(String strangStrung) {
System.out.println("any " + strangStrung + "more ");
}
```

Chapter 5 Exercise 2

Output:

just for

```
int buzz = 2;
zoop("just for", bizz);
clink(2 * buzz);
}
public static void clink(int fork) {
System.out.print("It's ");
zoop("breakfast ", fork);
}
public static void ping(String strangStrung) {
System.out.println("any " + strangStrung + "more");
}
```

Chapter 5 Exercise 2

Output:

just for

any not more

```
zoop("just for", bizz);
clink(2 * buzz);
}
public static void clink(int fork) {
System.out.print("It's ");
zoop("breakfast ", fork);
}
public static void ping(String strangStrung) {
System.out.println("any " + strangStrung + "more ");
}
```

Chapter 5 Exercise 2

Output:

just for

any not more

Chapter 5 Exercise 2

```
        zoop("just for", bizz);
        clink(2 * buzz);
        }
    public static void clink(int fork) {
    System.out.print("It's ");
        zoop("breakfast ", fork);
        }
  public static void ping(String strangStrung) {
System.out.println("any " + strangStrung + "more ");
        }
```

Output:

just for
any not more
It's

Chapter 5 Exercise 2

```
        int buzz = 2;
        zoop("just for", bizz);
        clink(2 * buzz);
        }
    public static void clink(int fork) {
        System.out.print("It's ");
        zoop("breakfast ", fork);
        }
    public static void ping(String strangStrung) {
    System.out.println("any " + strangStrung + "more ");
        }
```

Output:

just for
any not more
It's breakfast

```
        int buzz = 2;
      zoop("just for", bizz);
        clink(2 * buzz);
        }
public static void clink(int fork) {
      System.out.print("It's ");
      zoop("breakfast ", fork);
        }
public static void ping(String strangStrung) {
System.out.println("any " + strangStrung + "more ");
        }
```

Chapter 5 Exercise 2

Output:

just for
any not more
It's breakfast
!

In class: Chapter 5 Exercise 2

NewLine.java

```java
public class NewLine {

  public static void zoop(String fred, int bob) {
    System.out.println(fred);
    if (bob == 5) {
        ping("not ");
    } else {
        System.out.println("!");
    }
  }
  public static void main(String[] args) {
      int bizz = 5;
      int buzz = 2;
      zoop("just for", bizz);
      clink(2 * buzz);
  }
  public static void clink(int fork) {
      System.out.print("It's ");
      zoop("breakfast ", fork);
  }
  public static void ping(String strangStrung) {
      System.out.println("any " + strangStrung + "more ");
  }

}
```

Powered by trinket

```
just for
any not more
It's breakfast
!
```

It’s not just for breakfast anymore

```
        System.out.println("ik");
        baffle(quince);
    System.out.println("boo-wa-ha-ha");
        }
    }

public static void main(String[] args) {
    zippo("rattle", 13);
    }

}
```

Chapter 5 Exercise 6

```
        System.out.println("ik");//1
        baffle(quince);
    System.out.println("boo-wa-ha-ha");
        }
    }

public static void main(String[] args) {
    zippo("rattle", 13);
    }

}
```

Chapter 5 Exercise 6

```
        System.out.println("ik");//1
            baffle(quince);
    System.out.println("boo-wa-ha-ha");
            }
        }

public static void main(String[] args) {
        zippo("rattle", 13);
            }

        }
```

Chapter 5 Exercise 6

```
        System.out.println("ik");//1
            baffle(quince);
    System.out.println("boo-wa-ha-ha");
            }
        }

public static void main(String[] args) {
        zippo("rattle", 13);
            }

        }
```

Chapter 5 Exercise 6

```
        System.out.println("ik");//1
            baffle(quince);
    System.out.println("boo-wa-ha-ha"); //4

                }
            }

public static void main(String[] args) {
            zippo("rattle", 13);
                }

            }
```

Chapter 5 Exercise 6

In class: Chapter 5 Exercise 6

Java beta Run

NewLine.java

```java
public class NewLine {

    public static void baffle(String blimp) {
        System.out.println(blimp);
        zippo("ping", -5);
    }

    public static void zippo(String quince, int flag) {
        if (flag < 0) {
            System.out.println(quince + " zoop");
        } else {
            System.out.println("ik");
            baffle(quince);
            System.out.println("boo-wa-ha-ha");
        }
    }

    public static void main(String[] args) {
        zippo("rattle", 13);
    }

}
```

Powered by trinket

```
ik
rattle
ping zoop
boo-wa-ha-ha
```

Ch 6 value functions & `return`

- Functions with another type other than `void` make a value of that type and `return` it
- We'll skip the other stuff in Chapter 6 (recursion, JavaDoc tags and overloading) for now

An `absoluteValue()` function

- Note `double` and `return` statements

```
public static double absoluteValue(double x) {
    if (x < 0) {
        return -x;
    } else {
        return x;
    }
}
```

A program using `absoluteValue()`

Two function definitions

`absoluteVaue()` and **`main()`**

Java beta Run

NewLine.java

```
1  public class NewLine {
2
3      public static double absoluteValue(double x) {
4          if (x < 0) {
5              return -x;
6          } else {
7              return x;
8          }
9      }
10     public static void main(String[] args) {
11         System.out.println(absoluteValue(-99) );
12         System.out.println(absoluteValue(0) );
13         System.out.println(absoluteValue(37) );
14     }
15 }
```

Powered by

```
99.0
0.0
37.0
```

A program using `absoluteValue()`

- Three function calls to `absoluteVaue()`

Java beta Run

NewLine.java

```java
public class NewLine {

  public static double absoluteValue(double x) {
      if (x < 0) {
          return -x;
      } else {
          return x;
      }
  }
  public static void main(String[] args) {
    System.out.println(absoluteValue(-99) );
    System.out.println(absoluteValue(0) );
    System.out.println(absoluteValue(37) );
  }
}
```

Powered by

```
99.0
0.0
37.0
```

Unreachable "dead" code

- unreachable because it is *after* **`return`**
- **`return`** "goes back" and no other code in **`absoluteValue()`** is executed

Java beta ▶ Run Copy

NewLine.java

```
1  public class NewLine {
2
3    public static double absoluteValue(double x) {
4        if (x < 0) {
5            return -x;
6        } else {
7            return x;
8        }
9        System.out.println("This line is dead.");
10   }
11   public static void main(String[] args) {
```

Powered by trinket

```
NewLine.java:9: error: unreachable statement
        System.out.println("This line is dead.");
        ^
```

- An `isSingleDigit()` function
  A `boolean` function that `return`s `true` or `false`

Java beta Run

NewLine.java

```java
public class NewLine {

  public static boolean isSingleDigit(int x) {
    if (x > -10 && x < 10) {
      return true;
    } else {
      return false;
    }
  }
  public static void main(String[] args) {
    System.out.println(isSingleDigit(-99) );
    System.out.println(isSingleDigit(0) );
    System.out.println(isSingleDigit(37) );
    System.out.println(isSingleDigit(7) );
  }
}
```

Powered

```
false
true
false
true
```

A program using `isSingleDigit()`

- Four function calls to `isSingleDigit()`

Java beta Run

NewLine.java

```java
public class NewLine {

  public static boolean isSingleDigit(int x) {
    if (x > -10 && x < 10) {
        return true;
    } else {
        return false;
    }
  }
  public static void main(String[] args) {
    System.out.println(isSingleDigit(-99) );
    System.out.println(isSingleDigit(0) );
    System.out.println(isSingleDigit(37) );
    System.out.println(isSingleDigit(7) );
  }
}
```

Powered

```
false
true
false
true
```

- A different `isSingleDigit()`. Not better, just another way of doing the same thing

Java beta Run

NewLine.java

```
1  public class NewLine {
2
3    public static boolean isSingleDigit(int x) {
4      return x > -10 && x < 10;
5    }
6
7    public static void main(String[] args) {
8      boolean result = isSingleDigit(-99);
9      System.out.println(result);
10     result = isSingleDigit(0);
11     System.out.println(result);
12   }
13 }
```

Powered

```
false
true
```

Storing the return value

- Again, just another way of doing the same thing

Java beta Run

NewLine.java

```
1  public class NewLine {
2
3    public static boolean isSingleDigit(int x) {
4      return x > -10 && x < 10;
5    }
6
7    public static void main(String[] args) {
8      boolean result = isSingleDigit(-99);
9      System.out.println(result);
10     result = isSingleDigit(0);
11     System.out.println(result);
12   }
13 }
```

Powered

```
false
true
```

- *Syntax* is the grammar and spelling of a computer language

Review: Syntax error

- Here I forgot the double quotes around my name which results in an error message

Review: Logic error

- This time I misspelled my name
- The computer doesn't know my name, so the program runs incorrectly without an error message

Ch 6 Exercise 1: Will this cause an error message? Is it a mistake?

- Try experimenting

Java beta Run

NewLine.java

```
1  public class NewLine {
2
3    public static boolean isSingleDigit(int x) {
4      return x > -10 && x < 10;
5    }
6
7    public static void main(String[] args) {
8      isSingleDigit(-99);
9      isSingleDigit(0);
10   }
11 }
```

Ch 6 Exercise 2: Write a `isDivisable()` function

- Hint: use modulus

Java beta Run

NewLine.java

```java
public class NewLine {

  public static boolean isDivisable(int n, int m) {
    //your code here
  }

  public static void main(String[] args) {
    System.out.println(isDivisable(10,2));
    System.out.println(isDivisable(10,3));
    System.out.println(isDivisable(12,3));
    System.out.println(isDivisable(12,5));
  }
}
```

Powered b

```
true
false
true
false
```

Ch6 Exercise 3: write isTriangle()

- If any of the three lengths is greater than the sum of the other two, you cannot form a triangle.

Java beta Run

NewLine.java

```java
public class NewLine {

  public static boolean isTriangle(int a, int b, int c) {
    //your code here
  }

  public static void main(String[] args) {
    System.out.println(isTriangle(3,4,5));
    System.out.println(isTriangle(3,4,8));
    System.out.println(isTriangle(12,3,11));
    System.out.println(isTriangle(12,3,5));
  }
}
```

Powered b

```
true
false
true
false
```

In class: Chapter 6 Exercises 1, 2, 3 & 5

Think Java Chapter 6

File Edit View Insert Format Tools Add-ons Help All changes saved in Drive

100% Normal text Merriweather 12

Complete the following 4 exercises (#1,2, 3 and 5) from Chapter 6 of Think Java, save your work and submit to Google Classroom

Exercise 1

If you have a question about whether something is legal, and what happens if it is not, a good way to find out is to ask the compiler. Answer the following questions by trying them out.

1. What happens if you invoke a value method and don't do anything with the result; that is, if you don't assign it to a variable or use it as part of a larger expression?
2. What happens if you use a void method as part of an expression? For

Ch 6 Exercise 1

1. What happens if you invoke (call) a value method and don't do anything with the result?

A. No error message is shown, but it is still a mistake to call a function that returns a value and not use the returned value

Java beta Run

NewLine.java

```
1  public class NewLine {
2
3    public static boolean isSingleDigit(int x) {
4      return x > -10 && x < 10;
5    }
6
7    public static void main(String[] args) {
8      isSingleDigit(-99);
9      isSingleDigit(0);
10   }
11 }
```

Ch 6 Exercise 1

2. What happens if you use a void function as part of an expression?

A. You get an error message that **`'void' type is not allowed here`**

NewLine.java

```
1  public class NewLine {
2
3      public static void one() {
4          System.out.print("one");
5      }
6
7      public static void main(String[] args) {
8          System.out.println(one() + 7);
9      }
10 }
```

Powered by trinket

```
NewLine.java:8: error: 'void' type
not allowed here
        System.out.println(one() +
7);
                                ^

1 error
```

Ch 6 Exercise 1

What if the `one()` function returned an **`int`**, would that still cause an error message?

A. No

NewLine.java

```
1  public class NewLine {
2
3      public static int one() {
4          return 1;
5      }
6
7      public static void main(String[] args) {
8          System.out.println(one() + 7);
9      }
10 }
11
```

Powered by trinket

```
8
```

Ch 6 Exercise 1

What if the `one()` function returned an **`String`**, would that still cause an error message?

A. No

NewLine.java

```
1  public class NewLine {
2
3      public static String one() {
4          return "one";
5      }
6
7      public static void main(String[] args) {
8          System.out.println(one() + 7);
9      }
10 }
11
```

Powered by trinket

```
one7
```

Ch 6 Exercise 2: Write a `isDivisable()` function

- Hint: use modulus

NewLine.java

```java
public class NewLine {

  public static boolean isDivisable(int n, int m) {
    if(n % m == 0)
      return true;
    else
      return false;
  }

  public static void main(String[] args) {
    System.out.println(isDivisable(10,2));
    System.out.println(isDivisable(10,3));
    System.out.println(isDivisable(12,3));
    System.out.println(isDivisable(12,5));
  }
}
```

Powered by trinket

```
true
false
true
false
```

Ch 6 Exercise 2: Write a `isDivisable()` function

- A better version

NewLine.java

```java
public class NewLine {
    public static boolean isDivisable(int n, int m) {
      if(m == 0)
        return false;
      else if(n%m==0)
        return true;
      else
        return false;
    }

    public static void main(String[] args) {
        System.out.println(isDivisable(10,2));
        System.out.println(isDivisable(10,0));
        System.out.println(isDivisable(10,3));
    }
}
```

Powered b

```
true
false
false
```

Ch6 Exercise 3: write isTriangle()

- If any of the three lengths is greater than the sum of the other two, you cannot form a triangle.

NewLine.java

```java
public class NewLine {

  public static boolean isTriangle(int a, int b, int c) {
    if(c > a + b)
      return false;
    else if(b > a + c)
      return false;
    else if(a > b + c)
      return false;
    else
      return true;
  }

  public static void main(String[] args) {
    System.out.println(isTriangle(3,4,5));
    System.out.println(isTriangle(3,4,8));
    System.out.println(isTriangle(12,3,11));
    System.out.println(isTriangle(12,3,5));
  }
}
```

Powered by trinket

```
true
false
true
false
```

Ch 6 Exercise 6: Find the output

```
public static boolean isHoopy(int x) {
    boolean hoopyFlag;
    if (x % 2 == 0) {
        hoopyFlag = true;
    } else {
        hoopyFlag = false;
    }
    return hoopyFlag;
}
public static boolean isFrabjuous(int x) {
    boolean frabjuousFlag;
    if (x > 0) {
        frabjuousFlag = true;
    } else {
        frabjuousFlag = false;
    }
    return frabjuousFlag;
}
```

Ch 6 Exercise 6: Find the output

```
public static boolean isHoopy(int x) {
        boolean hoopyFlag;
        if (x % 2 == 0) {
          hoopyFlag = true;
          } else {
         hoopyFlag = false;
          }
        return hoopyFlag;
        }
public static boolean isFrabjuous(int x) {
        boolean frabjuousFlag;
          if (x > 0) {
        frabjuousFlag = true;
          } else {
       frabjuousFlag = false;
          }
      return frabjuousFlag;
        }
```

Output:

true

Ch 6 Exercise 6: Find the output

```
public static boolean isHoopy(int x) {
    boolean hoopyFlag;
    if (x % 2 == 0) {
        hoopyFlag = true;
    } else {
        hoopyFlag = false;
    }
    return hoopyFlag;
}

public static boolean isFrabjuous(int x) {
    boolean frabjuousFlag;
    if (x > 0) {
        frabjuousFlag = true;
    } else {
        frabjuousFlag = false;
    }
    return frabjuousFlag;
}
```

Output:

true

true

Ch 6 Exercise 6: Find the output

```
}
}
public static boolean isHoopy(int x) {
boolean hoopyFlag;
if (x % 2 == 0) {
hoopyFlag = true;
} else {
hoopyFlag = false;
}
return hoopyFlag;
}
public static boolean isFrabjuous(int x) {
boolean frabjuousFlag;
if (x > 0) {
frabjuousFlag = true;
} else {
frabjuousFlag = false;
}
return frabjuousFlag;
}
```

true

true

ping!

pong!

Ch 6 Exercise 6: Find the output

Java beta Run

Hello.java

```java
public class Hello {

  public static void main(String[] args) {
    boolean flag1 = isHoopy(202);
    boolean flag2 = isFrabjuous(202);
    System.out.println(flag1);
    System.out.println(flag2);
    if (flag1 && flag2) {
        System.out.println("ping!");
    }
    if (flag1 || flag2) {
        System.out.println("pong!");
    }
  }
  public static boolean isHoopy(int x) {
      boolean hoopyFlag;
      if (x % 2 == 0) {
          hoopyFlag = true;
      } else {
          hoopyFlag = false;
      }
      return hoopyFlag;
  }
  public static boolean isFrabjuous(int x) {
      boolean frabjuousFlag;
      if (x > 0) {
          frabjuousFlag = true;
      } else {
          frabjuousFlag = false;
      }
      return frabjuousFlag;
  }

}
```

Powered by trinket

```
true
true
ping!
pong!
```