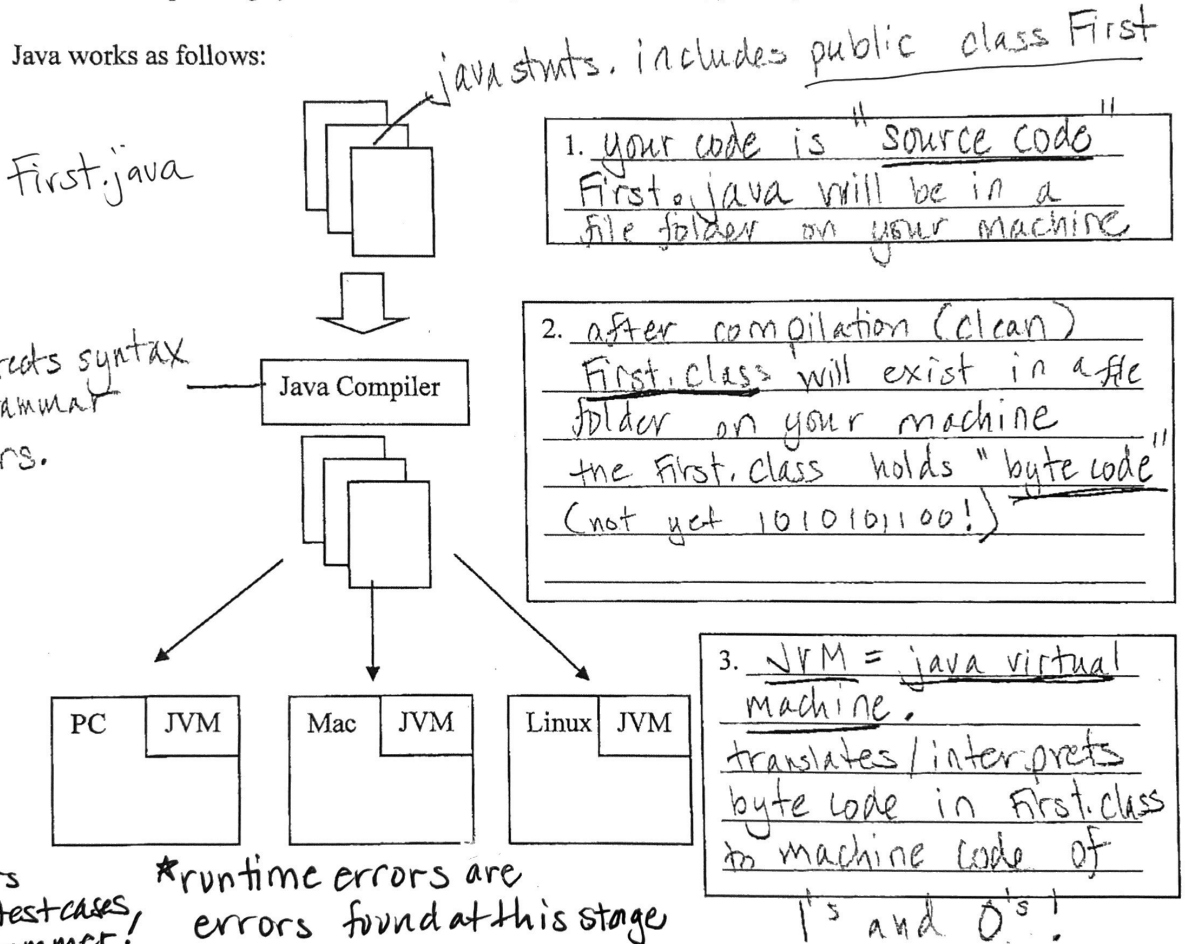
Compiling and Running Java Programs.

A compiler is a program that translates code to machine language that the central processing unit (the computer's "brain") can understand. In many languages, the source code must be customized for different operating systems and then compiled for certain types of processors.

Java works as follows:



java stmts. includes public class First

1. your code is "source code" First.java will be in a file folder on your machine

★ detects syntax or grammar errors.

2. after compilation (clean) First.class will exist in a file folder on your machine the First.class holds "byte code" (not yet 10101011 00!)

3. JVM = java virtual machine. translates/interprets byte code in First.class to machine code of 1's and 0's!

★ logic errors found by test cases and programmer!

★ runtime errors are errors found at this stage

The JRE (Java Runtime Environment) includes the JVM and java standard class files. It is the JRE that you see occasionally updated on your computer.

byte code created on one type of OS is portable to any other OS due to the JVM! makes java awesome! ☺

KEY

AP Computer Science A
Unit 3. Boolean Expressions. Notes

Comments are statements that help describe a program but are ignored by the computer. There are two types of comments.

1) **Single line comments**. Anything to the right of // is a comment.

```
double rate;            // text that describes the variable's purpose
```

2) **Multiple-line comments**. Anything between a /* and the closing */ is a comment.

```
/* This program was written by Jennifer Somebody
   August 2019.         */
```

Types of Errors.

Compiler Errors aka syntax **Errors**. A compiler is software that checks if a program correctly follows a programming language's rules. For example, are there semicolons where they are supposed to be? Are only integer values being assigned to ints and so on. Compilers do other things, but this is all we are concerned about here.

Example 1.

```
int x = 8.9;            // compiler error
```

In Java, a program must be free of compiler errors or it will not run.

Run-time Errors occur while your program is running. They may be due to something the user does or some flaw in the code that is only exposed when it runs.

Example 2. Given the following code is part of a larger program and compiles, a run-time error will occur if we enter a double bc we are using nextInt() method.

```
Scanner s = new Scanner( System.in );
System.out.println( "Enter a number" );
int n = s.nextInt();
```

$$7\overline{)10}\ \ \begin{matrix}1\\0\end{matrix}$$

Example 3. The following code compiles because the expression on the right evaluates to an int. However, a run-time error occurs because we cannot divide by 0

```
int x = 10 / 0;
System.out.println( x );
```

"java.lang.ArithmeticException"

Logic Errors occur when your code compiles and runs but the output is incorrect. These are generally the programmer's fault. this is why we have test cases.

"George Boole" Boolean algebra

Boolean Expressions. A Boolean expression is either true or false.

For example: x > 8

If x is greater than or equal to 8, then the expression is true. Otherwise the expression is false. ! means not

Here are 6 operators used in Boolean expressions < <= > >= != ==

1. If *num* is 10, what is the value of this expression? true	num >= 10
2. Give a value for *y* that makes this expression false? 5 3. Give a value for *y* that makes this expression true? anything other than 5	y != 5

if and if-else Statements. These statements are called control statements because they control whether a block of code is executed or not. Some texts refer to them as conditionals because they represent a condition that may be true or false.

10 * .99 = 9.9 tops int casts to 9

Example 1. 0 .. 99

```
int x = (int) ( 10 * Math.random() );
if ( x > 4 ) {
        x++;
        System.out.print( x );
}
System.out.println( "Done" );
```

x can hold 0 to 9

boolean gt4 = x>4; if (gt4)

block

The if must be followed by parentheses. if ( )

The ( ) must contain a conditional statement

If the block is more than one statement, you must have { } open close curly brackets.

In the above example, if x has a value of 7, what is printed? 8Done

Example 2. 0 .. 2 ; 0 .. 99

```
int z = (int) ( 3 * Math.random() );
if ( z == 2 )
        z++;
else
        z--;
System.out.println( z );
```

z holds 0, 1, or 2

if z = 0 or 1

new: z/z → z/3; 1 → 0; 0 → -1 } code trace

In this example, the curly braces were NOT required because the block of code following the if statement and the code following the *else* keyword were not more than 1 stmt

List the numbers that might be printed in the above example. 3, 0, or -1

Do exercises 1 to 19. you only need a { } block if 2 or more statements follow an if or else!

Block Scope. If a variable is declared with a block of code, it is only allowed to be used in that { } block.

Example of code with a compiler error due to block scope.

```
int a = (int)(10*Math.random()) + 1;
if ( a >= 5 ){ // block begins
        int b = a + 10;          // b declared in block
        System.out.println( a + b );
} // block ends
System.out.println( a + b );    // compiler error: cannot find symbol
```

☆ If you need to use a variable within a block AND use it after the block, then declare it before the block.

One possible alternative approach that does NOT generate a compiler error.

```
int a = (int)(10*Math.random()) + 1;          // 1 to 10
int b = 0;  // declared outside the block
if ( a >= 5 ){  // block begins
        b = a + 10;
        System.out.println( a + b );
} // block ends
System.out.println( a + b );    // compiler error: cannot find symbol
```

if-else if Statements. The basic if - else structure can be expanded to handle multiple conditionals using else-if statements.

Important. You only execute the code associated with the true conditional clause

Example.

```
int h = (int)( 50 * Math.random() );      // h holds 0 to 49
if ( h % 7 == 0 )                         // h multiple of 7 including 0
        System.out.print( "R" );
else if ( h < 22 )                        // less than 22, != 7, !=14, != 21
        System.out.print( "O" );
else if ( h > 10 )                        //                22 to 48, != 28, 35, 42, 49
        System.out.print( "C" );
else
        System.out.print( "K" );          // not a way to get here.
```

If h is 14, what is printed? R

If h is 19, what is printed? O

What value(s) of h will cause R to be printed? multiples of 7

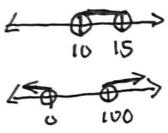
Note. Not every *if*, or *else if*, needs to end with an *else*.

Do exercises 20 to 33.

```
if (n < 3)
    n++;
if (n > 7)
    n--;
```

&& means "and" || means "or"

Logical Operators. More complicated Boolean expressions can be composed using logical operators && (AND) and || (OR)



x > 10 && x < 15 is true when x greater than 10 and x less than 15

y < 0 || y > 100 is true when y less than 0 or y greater than 100

1. What values of *b* will cause S to be printed?
3, 4, 5, 6
	// b is an int
	if (b >= 3 && b < 7)
	System.out.println("S");

2. What values of *c* will cause V to be printed?
$\mathbb{Z}$ - all integers
	// c is an int
	if (c >= 3 || c < 7)
	System.out.println("V");

3. What values of *d* will cause Z to be printed?
$d \in \mathbb{Z} \mid [8, \infty)$
	// d is an int
	if (d > 0 && d > 7)
	System.out.println("Z");

There is a third logical operator that is useful is certain situations: the NOT operator !

boolean a = !true; a holds false

boolean z = !false; z holds true

if (!ap)

```
int n = 10;
if (!(n < 10))
    System.out.print("n is 10 or greater!");
```

Nested If Statements. When there is an if statement within another if statement, that is known as *nested* if statements.

Example.

```
int k = (int) (15 * Math.random());    // 0 .. 14
if (k % 2 == 0) {
    if (k % 3 == 0)
        System.out.println("A");
    else
        System.out.println("B");
} else
    System.out.println("C");
```

$\mathbb{Z}$ integers

For what values of *k* is A printed? even integers < 15 multiples of 6 so 6 ≠ 12, 0

For what values of *k* is B printed? even integers except 6 ≠ 12 so 0, 2, 4, 8, 10, 14

For what values of *k* is C printed? odd numbers < 15

Comparing String Objects. A compiler error occurs if you try to compare objects using > < >= <=. And usually, you will not get the result you want if you use == to compare objects. (We will discuss why this is in unit 5.) To compare strings, we will use the equals and compareTo methods.

Do exercises 34 to the end.

you will be comparing the storage location in memory because strings are objects.