JAVA INTRODUCTION

PROGRAMMING OVERVIEW

3 Types of programming languages
Machine, Assembly, High-Level

Java — *High-level, Object-Oriented, Strongly-typed language*

STRONGLY-TYPED = cannot perform operations that are not allowed for or are not compatible with a datatype

Flow of Control — *Sequential Execution, Method Call, Selection, Looping*

Variables — *named locations in memory where we can store values, help us refer to data in a program*

To declare variables = need datatype & identifier

Object-Oriented Programming Concepts

Encapsulation
Reusability
Classes
Objects

Classes vs. Objects

CLASS = generic blueprint
OBJECT = specific item created using the blueprint

Objects are instances of classes!

Approaches for getting an instance of a class... { *constructor method, factory method (premade), static constants* }

PRIMITIVE DATA TYPES

boolean — *true or false*

int, **byte**, **long**, **short** — *Integer Data Types*

double, **float** — *Floating-Point Data Types*

char — *Single character*

PRECISION

short | int | long | float | double

IDENTIFIERS

Symbolic names you assign to classes, methods, and data

Start with A-Z, a-z, _, $, or many Unicode characters

Underscores typically only for constants, $ for programmatically-generated variables, and camelcase (e.g. firstExamScore) used for all other variable identifiers

No spaces are allowed

Cannot be a Java reserved word

Case-sensitive

Can contain an almost unlimited number of letters and/or digits

Identifiers of objects are called object references

WRAPPER CLASSES

used to convert primitives to objects & vice versa

Unboxing — *when Java provides conversion from wrapper class object to primitive*

e.g.: When an Integer object is used in an arithmetic expression, the int value is automatically used

Autoboxing — *when Java automatically converts a primitive to a wrapper class object*

e.g.: Assigning an int to an Integer object reference will convert the int to an Integer

int --> Integer
short --> Short
float --> Float
long --> Long
double --> Double
byte --> Byte
boolean --> Boolean
char --> Character

Helpful Static Methods

Integer.parseInt(String s)
returns the int value of s

Integer.valueOf(String s)
returns the Integer value of s

NOTABLE VOCABULARY

IDE — *Integrated Development Environment*

Package — *collection of related classes that can be imported into programs*

API — *Application Programming Interface, tells how to use a class*

Pseudocode — *tool used by programmers to design a program w/o worrying about syntax*

```
//Pseudocode for calculating sum of two numbers
read firstNumber, read secondNumber, set total to (firstNumber - secondNumber), output total
```

Methods — *operations for a class*

Constructors, Accessor, Mutator, & other operations that perform class-related functions on the data

Algorithms — *a process or set of rules to be followed in problem-solving operations*

Members of a Class — *methods and fields*

+ — *String concatenation operator, allows us to print data along with Strings*

Be sure to add a space to the end of a String literal before concatenating a value = allows for more readable output

= — *assignment operator, used to assign values to data*

```
datatype variable1 = data
(assignment goes right to left: read "variable1 gets the value of data")
```

this — *the implicit parameter, java keyword used if two identifiers are the same and clarification is needed*

static — *Java keyword that signifies a member belongs to the class as a whole and is not unique to just one object*

final — *Java keyword used to state a variable is constant and cannot be changed*

new — *Java keyword used in constructor methods for instantiating objects*

Expression — *consists of operators and operands that evaluate to a single value*

Instance Variables — *data associated with an object of a class, can be variables and constants of any primitive type or objects of*

ARITHMETIC OPERATIONS

Binary Operators

+	*addition*
-	*subtraction*
*	*multiplication*
/	*division*
%	*modulus (remainder after division)*

PRECEDENCE

( ) parentheses
++, -- postfix
++, -- prefix
**, /, %*
+, -
*=, +=, -=, *=, %=*

Shortcut Operators

Prefix - change variable and then use it after
++a *--a*

Postfix - use variable and then change it after
a++ *a--*

++a = a++ = a+1
--a = a-- = a-1

Type Casting

IMPLICIT- when compiler automatically performs a data type promotion (less precise to more precise)
(e.g. arithmetic promotion of operands)

EXPLICIT - when instruct compiler specifically to convert the type of variable

Notables

Explicit type casting = need to put (desiredDataType) in front of the value to convert

```
int num1 = 90;
int num2 = 94;
double avg = ((double)num1+num2)/2;
```

Any promoted variable is not permanently changed = its type remains the same after the calculation is performed

When calculations of mixed data types are performed, lower precision operands are promoted to the type of the operand with the higher precision

Integer division discards any fractional part
(e.g. 6/4 is 1 with integer division)

4/0 --> exception generated
4.0/0.0 --> Infinity
0.0/0.0 --> Not a Number (NaN)

USING CLASSES & OBJECTS

Declaring Variables *MUST DO before use!*

```
dataType identifier;          int bestGrade;          // in general
ClassName objreference;       Student student1;       // for objects
```

Instantiating Variables *(assigning values)*

```
dataType identifier = initialValue;
double average = 98.7;    // for primitives
ClassName objref = new ClassName(arguments);
Student student1 = new Student("Emma", 21, teacher1);
// instantiating an object with alternate constructor
Student student1 = new Student( ); //default constructor
//Note: default constructors have no arguments!
```

Calling Methods for Objects

```
objectReference.methodName(arguments);
student1.setExamScores(98,83,96);
/* when calling a method, only include values in the argument
list = including data types will cause an error */
```

Accessing Static Constants

```
ClassName.staticConstant
Color background = Color.BLUE;
```

Calling Static Methods

```
ClassName.staticMethodName(arguments);
int absoluteVal = Math.abs(someNumber);
```

Notables

- Declare each variable only once
- Cannot call methods on object references that have not been assigned values
- When calling a method that takes no arguments, include empty parentheses after method name

ACCESS MODIFIERS

keywords that specify where a class or member can be used

public	can be used by methods of the same class & methods of other classes
private	can be used by methods of the same class ONLY
protected	can be used by methods of the same class, methods of subclasses, & methods in the same package
(none)	can be used by methods of the same package

ARGUMENTS = data passed when a method called, actual values

PARAMETERS = variables in method definitions

USER-DEFINED CLASSES

Defining a Class

```
accessModifier class ClassName
{ // class definition goes here }
```

ClassName = noun starting with capital letter

Defining Constructors

special methods for creating objects

```
accessMod ClassName(parameters)
{ // initialize object data here }
```

Default Constructor = has no parameters

```
public Cat()
{
    name = "Cosmo";
    age = 4;
}
```

You should provide at a minimum a default constructor & a constructor that accepts initial values for all instance variables

Use public access modifie for constructors usually

Alternate Constructor = has one or more parameters

```
public Cat(String catName, int catAge)
{
    name = catName;
    age = catAge;
}
```

Defining Variables

```
accessMod dataType identifier;      // general
accessMod final dataType ID;        // if a constant
private int age;        private final int MAX_NUM;
/* use private as variable access modifier for better
encapsulation & write identifiers of constants in all
capital letters */
```

Note: Providing constructors is optional. If you don't write any, the compiler provides a default constructor that autoinitializes a instance variables

byte, short, int, long --> 0
double, float --> 0.0
char --> null character
boolean --> false
object references --> null

Writing Methods

```
accessModifier returnType methodName(parameters) // method header
{ //method body here }  /* methods with a return type other than void = need a
                         return statement in method body */
accessModifier static returnType methodName(parameters)
{ //method body here }          // method definition for static method
public returnType getInstanceVariable( )
{ return instanceVariable; }    // method definition for accessor method
public void setInstanceVariable(dataTypeOfVar newValue)
{ //validate newValue, then assign } // method definition for mutator method
```

OVERLOADING METHOD: same name but with different parameter list

OVERRIDING METHOD: same method header but different method body

Access/Call	Static	Non-Static
Instance Variables	NO	YES
Static Variables	YES	YES
Non-Static Variables	NO	YES
Static Methods	YES	YES
Reference this	NO	YES
Constructor methods	YES	YES

METHOD SIGNATURE = includes the method name and data typ number, and order of any method parameters

Importing Classes for Use

To use a pre-existing Java class in your program that is not in the java.lang package, you need to import it.

Do this ABOVE class statement!

```
import packageName.ClassName;
//imports a single class
import packageName.*;
//imports all classes in package
```

ERRORS

- **COMPILER =** *does not run due to syntax error*
- **RUN-TIME =** *runs but crashes due to class misuse*
- **LOGIC =** *runs but unexpected or incorrect output*

COMMENTS

Comments in Java are ignored by the compiler

/* com ment */	multi-line
//comment	single line

Javadoc

tool provided by JDK

```
/**
* Javadoc comment1
* comment2...
*/
```

Use comments to explain yo code, and be sure to format code with enough white spa to increase readability!

HELPFUL PRE-EXISTING CLASSES

String

NOTE: NOT a primitive data type but used frequently, so Java lets us instantiate String objects without usi the keyword new

RETURNS	METHOD
int	length()
String	toUpperCase()
String	toLowerCase()
char	charAt(int index)
int	indexOf(String substring)
int	indexOf(char searchChar)
String	substring(int start, int end)

REMEMBER THAT THE FIRST INDEX OF A STRING IS 0, NOT 1!

Use escape sequences inside String literals to represent certain characters

CHARACTER	ESCAPE SEQ.
newline	\n
tab	\t
double quotes	\"
single quotes	\'
backslash	\\

```
"Name: " + name + "\nID: " + id
```

Name: Bob
ID: 129323

DecimalFormat

```
DecimalFormat id = new DecimalFormat(pattern);
```

PATTERN SYMBOL	MEANING
0	Required digit
#	Optional digit
.	Decimal point
,	Comma separator
$	Dollar sign
%	Multipy by 100, add %

```
DecimalFormat myFormat = new DecimalFormat("$#0.00");
myFormat.format(78.6666);
//formatted String = $78.67
```

PrintStream

```
System.out.print(stringToPrint);
System.out.println(stringToPrint);
/* second one prints newline character after stringToPrint */
```

NumberFormat

can be used to format numbers for output (specifically currency and percentages) with methods (called factory methods) that can be used instead of a constructor to create objec

RETURNS	METHOD
NumberFormat	getCurrencyInstance()
NumberFormat	getPercentageInstance()
String	format(double number)

The format method works the same for DecimalFormat and NumberFormat

```
NumberFormat money = NumberFormat.getCurrencyInstance();
money.format(myPayCheck);
```

Math

Math.E Math.PI

RETURNS	STATIC METHOD
arg. data type	Math.abs(num)
double	Math.log(double num)
arg. data type	Math.min(arg a, arg b)
arg. data type	Math.max(arg a, arg b)

Random *generates pseudorandom numbers*

```
Random myRand = new Random();
```

nextInt() method returns a random int ranging from 0 up to but not including the argument passed

```
Random rand = new Random();
int num1 = rand.nextInt(end-start+1) + start;
int num2 = rand.nextInt(41) + 10;
//num2 is between 10 and 50 inclusively
```

Scanner

divides input into sequences of characters (called tokens) using delimiters (default delimiter is any whitespace character)

RETURNS	METHOD
int	nextInt()
double	nextDouble()
String	next()
String	nextLine()

When reading from the keyboard, provide user with clear prompts = describe data & restrictions

```
Scanner scan = new Scanner(System.in);
int age = scan.nextInt();
```

JOptionPane *(annoying pop-ups)*

```
JOptionPane.showInputDialog(null, Object prompt);
JOptionPane.showMessageDialog(null, Object msg);
```