JAVA/OOPS

Books:

- Herbert Schildt, “Java the Complete Reference”, 9th Edition.
- “Head First Java”, O’Reilly Publication
- Edward G. Finegan, Robert Liguori, “OCA Java SE 8 Programmer I Study Guide”, Oracle Press.
- Kathy Sierra, “OCA/OCP Java SE 7 Programmer I & II Study Guide”, Oracle Press.

Other readings and relevant websites:

- http://www.w3schools.com/
- http://www.javatpoint.com/java
- https://www.tutorialspoint.com/java/
- http://www.nptelvideos.com/java
- https://www.geeksforgeeks.org/object-oriented-programming-oops-concept-in-java/
- Java Development Kit (JDK) and IDE (NetBeans or Eclipse)
- Coding Ninjas (online platform- https://www.codingninjas.com/)

Topics:

- Java Basics: Basic syntax, identifiers, keywords, Java data types & operators
- Control Statements: decision constructs, using loop constructs, command line arguments
- Working with Arrays: creating and using arrays (1-D, 2-D and multi-dimensional arrays), jagged arrays
- Introduction to Object Oriented Programming: benefits and application of OOP, basic concepts and characteristics of OOP, abstraction, data hiding, static and dynamic binding, encapsulation, inheritance and polymorphism, procedural programming vs object-orientedprogramming.

- Objects and Classes: basics of objects and classes, structure of a class, definition of class members, member variable and member function, role of constructors and methods in class, define an object.
- Access Control Modifiers: access control, method overloading, constructors, constructor overloading, use of this and static.
- Inheritance: working with inheritance, inheritance basics & types, using super, method overriding, dynamic method dispatch, final keyword.
- Abstract Methods & Classes, Packages & Interfaces: built-in packages and user defined packages, interfaces: declaration, implementation, extending classes and interfaces.
- Strings, StringBuffer, StringBuilder & StringTokenizer: introduction, immutable string, methods of String class, StringBuffer class & StringBuilder class, toString method, StringTokenizer class.
- Exception Handling: exception handling fundamentals, exception types, try and catch, multiple catch clauses, nested try, throw, throws and finally, creating custom exception.
- Multithreading: Java thread model, main thread, creating thread by implementing runnable and extending thread class, creating multiple threads, using isAlive() and join(), thread priorities, synchronization.
- Generics: introduction, generic example, generic class, generic method, generic constructor and generic interfaces.

- ## Basic Syntax:

About Java programs, it is very important to keep in mind the following points.

- **Case Sensitivity -** Java is case sensitive, which means identifier Helloand hello would have different meaning in Java.
- **Class Names -** For all class names the first letter should be in Upper Case. If several words are used to form a name of the class, each inner word's first letter should be in Upper Case.

  Example:

class MyFirstJavaClass

- **Method Names -** All method names should start with a Lower Case letter. If several words are used to form the name of the method, then each inner word's first letter should be in Upper Case.

  Example:

  public void myMethodName()

- **Program File Name -** Name of the program file should exactly match the class name. When saving the file, you should save it using the class name (Remember Java is case sensitive) and append '.java' to the end of the name (if the file name and the class name do not match, your program will not compile).

  Example:

  Assume 'MyFirstJavaProgram' is the class name. Then the file should be saved as 'MyFirstJavaProgram.java'

- **public static void main(String args[]) -** Java program processing starts from the main() method which is a mandatory part of every Java program.

Java Identifiers:

All Java components require names. Names used for classes, variables, and methods are called identifiers. In Java, there are several points to remember about identifiers. They are as follows:

- All identifiers should begin with a letter (A to Z or a to z), currency character ($) or an underscore (_).
- After the first character, identifiers can have any combination of characters.
- A key word cannot be used as an identifier.
- Most importantly, identifiers are case sensitive.
- Examples of legal identifiers: age, $salary, _value, __1_value.
- Examples of illegal identifiers: 123abc, -salary.

- **List of Java Keywords:**

A list of Java keywords or reserved words are given below:

- **abstract**: Java abstract keyword is used to declare an abstract class. An abstract class can provide the implementation of the interface. It can have abstract and non-abstract methods.
- **boolean**: Java boolean keyword is used to declare a variable as a boolean type. It can hold True and False values only.
- **break**: Java break keyword is used to break the loop or switch statement. It breaks the current flow of the program at specified conditions.
- **byte**: Java byte keyword is used to declare a variable that can hold 8-bit data values.
- **case**: Java case keyword is used with the switch statements to mark blocks of text.
- **catch**: Java catch keyword is used to catch the exceptions generated by try statements. It must be used after the try block only.
- **char**: Java char keyword is used to declare a variable that can hold unsigned 16-bit Unicode characters
- **class**: Java class keyword is used to declare a class.
- **continue**: Java continue keyword is used to continue the loop. It continues the current flow of the program and skips the remaining code at the specified condition.
- **default**: Java default keyword is used to specify the default block of code in a switch statement.
- **do**: Java do keyword is used in the control statement to declare a loop. It can iterate a part of the program several times.
- **double**: Java double keyword is used to declare a variable that can hold 64-bit floating-point number.
- **else**: Java else keyword is used to indicate the alternative branches in an if statement.
- **enum**: Java enum keyword is used to define a fixed set of constants. Enum constructors are always private or default.
- **extends**: Java extends keyword is used to indicate that a class is derived from another class or interface.

- **final**: Java final keyword is used to indicate that a variable holds a constant value. It is used with a variable. It is used to restrict the user from updating the value of the variable.
- **finally**: Java finally keyword indicates a block of code in a try-catch structure. This block is always executed whether an exception is handled or not.
- **float**: Java float keyword is used to declare a variable that can hold a 32-bit floating-point number.
- **for**: Java for keyword is used to start a for loop. It is used to execute a set of instructions/functions repeatedly when some condition becomes true. If the number of iteration is fixed, it is recommended to use for loop.
- **if**: Java if keyword tests the condition. It executes the if block if the condition is true.
- **implements**: Java implements keyword is used to implement an interface.
- **import**: Java import keyword makes classes and interfaces available and accessible to the current source code.
- **instanceof**: Java instanceof keyword is used to test whether the object is an instance of the specified class or implements an interface.
- **int**: Java int keyword is used to declare a variable that can hold a 32-bit signed integer.
- **interface**: Java interface keyword is used to declare an interface. It can have only abstract methods.
- **long**: Java long keyword is used to declare a variable that can hold a 64-bit integer.
- **native**: Java native keyword is used to specify that a method is implemented in native code using JNI (Java Native Interface).
- **new**: Java new keyword is used to create new objects.
- **null**: Java null keyword is used to indicate that a reference does not refer to anything. It removes the garbage value.
- **package**: Java package keyword is used to declare a Java package that includes the classes.
- **private**: Java private keyword is an access modifier. It is used to indicate that a method or variable may be accessed only in the class in which it is declared.

- **protected**: Java protected keyword is an access modifier. It can be accessible within the package and outside the package but through inheritance only. It can't be applied with the class.
- **public**: Java public keyword is an access modifier. It is used to indicate that an item is accessible anywhere. It has the widest scope among all other modifiers.
- **return**: Java return keyword is used to return from a method when its execution is complete.
- **short**: Java short keyword is used to declare a variable that can hold a 16-bit integer.
- **static**: Java static keyword is used to indicate that a variable or method is a class method. The static keyword in Java is mainly used for memory management.
- **strictfp**: Java strictfp is used to restrict the floating-point calculations to ensure portability.
- **super**: Java super keyword is a reference variable that is used to refer to parent class objects. It can be used to invoke the immediate parent class method.
- **switch**: The Java switch keyword contains a switch statement that executes code based on test value. The switch statement tests the equality of a variable against multiple values.
- **synchronized**: Java synchronized keyword is used to specify the critical sections or methods in multithreaded code.
- **this**: Java this keyword can be used to refer the current object in a method or constructor.
- **throw**: The Java throw keyword is used to explicitly throw an exception. The throw keyword is mainly used to throw custom exceptions. It is followed by an instance.
- **throws**: The Java throws keyword is used to declare an exception. Checked exceptions can be propagated with throws.
- **transient**: Java transient keyword is used in serialization. If you define any data member as transient, it will not be serialized.
- **try**: Java try keyword is used to start a block of code that will be tested for exceptions. The try block must be followed by either catch or finally block.
- **void**: Java void keyword is used to specify that a method does not have a return value.

- **volatile**: Java volatile keyword is used to indicate that a variable may change asynchronously.
- **while**: Java while keyword is used to start a while loop. This loop iterates a part of the program several times. If the number of iteration is not fixed, it is recommended to use the while loop.

Primitive Data Types:

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

Data Type	Size	Description
byte	1 byte	Stores whole numbers from -128 to 127
short	2 bytes	Stores whole numbers from -32,768 to 32,767
int	4 bytes	Stores whole numbers from -2,147,483,648 to 2,147,483,647
long	8 bytes	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4 bytes	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits
double	8 bytes	Stores fractional numbers. Sufficient for storing 15 decimal digits

boolean	1 bit	Stores true or false values
char	2 bytes	Stores a single character/letter or ASCII values

- ## Numbers Data Types:

Primitive number types are divided into two groups:

- **Integer types:**
  stores whole numbers, positive or negative (such as 123 or -456), without decimals. Valid types are byte, short, int and long. Which type you should use, depends on the numeric value.

- **Floating point types:**
  represents numbers with a fractional part, containing one or more decimals. There are two types: float and double.

- ## Boolean Types:

A boolean data type is declared with the boolean keyword and can only take the values true or false.

- ## Characters:

The char data type is used to store a **single** character. The character must be surrounded by single quotes, like 'A' or 'c':

- ## Non-Primitive Data Types:

Non-primitive data types are called **reference types** because they refer to objects.The main difference between **primitive** and **non-primitive** data types are:

- Primitive types are predefined (already defined) in Java. Non-primitive types are created by the programmer and is not defined by Java (except for String).
- Non-primitive types can be used to call methods to perform certain operations, while primitive types cannot.
- A primitive type has always a value, while non-primitive types can be null.
- A primitive type starts with a lowercase letter, while non-primitive types starts with an uppercase letter.
- The size of a primitive type depends on the data type, while non-primitive types have all the same size.

• Operators in Java:

Operator in Java is a symbol that is used to perform operations. For example: +, -, *, / etc.There are many types of operators in Java which are given below:

- Unary Operator
- Arithmetic Operator
- Shift Operator
- Relational Operator
- Bitwise Operator
- Logical Operator
- Ternary Operator
- Assignment Operator

Javatpoint Link:

https://www.javatpoint.com/operators-in-java

GeeksforGeeks Link:

https://www.geeksforgeeks.org/operators-in-java/

- # Java Control Statements/Control Flow in Java:

Java compiler executes the code from top to bottom. The statements in the code are executed according to the order in which they appear.
However, Java provides statements that can be used to control the flow of Java code. Such statements are called control flow statements. It is one of the fundamental features of Java, which provides a smooth flow of program.

Java provides three types of control flow statements.

- **Decision Making statements:**
  - if statements
  - switch statement
- **Loop statements:**
  - do while loop
  - while loop
  - for loop
  - for-each loop
- **Jump statements:**
  - break statement
  - continue statement

Javatpoint Link:

https://www.javatpoint.com/control-flow-in-java

Edureka Link:

https://www.edureka.co/blog/control-statements-in-java/#:~:text=A%20control%20statement%20in%20java,a%20set%20of%20two%20statements.

- **An array in Java:**

is a group of like-typed variables referred to by a common name. Arrays in Java work differently than they do in C/C++. Following are some important points about Java arrays.

- In Java, all arrays are dynamically allocated. (discussed below)
- Since arrays are objects in Java, we can find their length using the object property length. This is different from C/C++, where we find length using sizeof.
- A Java array variable can also be declared like other variables with [] after the data type.
- The variables in the array are ordered, and each has an index beginning from 0.

- **One-Dimensional Arrays:**

The general form of a one-dimensional array declaration is

type var-name[];

OR

type[] var-name;

An array declaration has two components: the type and the name. type declares the element type of the array. The element type determines the data type of each element that comprises the array. Like an array of integers, we can also create an array of other primitive data types like char, float, double, etc., or user-defined data types (objects of a class). Thus, the element type for the array determines what type of data the array will hold.

- **Multidimensional Arrays(2d):**

Multidimensional arrays are arrays of arrays with each element of the array holding the reference of other arrays. These are also known as Jagged Arrays.

A multidimensional array is created by appending one set of square brackets ([]) per dimension.

- **Jagged Array in Java:**

A jagged array is an array of arrays such that member arrays can be of different sizes, i.e., we can create a 2-D array but with a variable number of columns in each row. These types of arrays are also known as Jagged arrays.

GeekforGeeks Link:

https://www.geeksforgeeks.org/arrays-in-java/

- **OOPs (Object-Oriented Programming System):**

Javatpoint Link:

https://www.javatpoint.com/java-oops-concepts

Object means a real-world entity such as a pen, chair, table, computer, watch, etc. Object-Oriented Programming is a methodology or paradigm to design a program using classes and objects. It simplifies software development and maintenance by providing some concepts:

- Object
- Class
- Inheritance
- Polymorphism
- Abstraction
- Encapsulation

Apart from these concepts, there are some other terms which are used in Object-Oriented design:

- Coupling
- Cohesion

- Association
- Aggregation
- Composition

• Objects and Classes: basics of objects and classes:

Java is an object-oriented programming language.

Everything in Java is associated with classes and objects, along with its attributes and methods. For example: in real life, a car is an object. The car has attributes, such as weight and color, and methods, such as drive and brake.

A Class is like an object constructor, or a "blueprint" for creating objects.

• Create an Object:

In Java, an object is created from a class. We have already created the class named Main, so now we can use this to create objects.

• Using Multiple Classes:

You can also create an object of a class and access it in another class. This is often used for better organization of classes (one class has all the attributes and methods, while the other class holds the main() method (code to be executed)).

W3school Link:

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

• Access Control Modifiers:

As the name suggests access modifiers in Java helps to restrict the scope of a class, constructor, variable, method, or data member. There are four types of access modifiers available in java:

1. **Private**: The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
2. **Default**: The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.
3. **Protected**: The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.
4. **Public**: The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

GeeksforGeeks Link:

https://www.geeksforgeeks.org/access-modifiers-java/

Javatpoint Link:

https://www.javatpoint.com/access-modifiers

• Inheritance in Java:

Inheritance in Java is a mechanism in which one object acquires all the properties and behaviors of a parent object. It is an important part of OOPs (Object Oriented programming system).

The idea behind inheritance in Java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of the parent class. Moreover, you can add new methods and fields in your current class also.

Inheritance represents the **IS-A relationship** which is also known as a *parent-child* relationship.

• Why use inheritance in java:

- For Method Overriding (so runtime polymorphism can be achieved).
- For Code Reusability.

- **Terms used in Inheritance:**
- **Class:** A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.
- **Sub Class/Child Class:** Subclass is a class which inherits the other class. It is also called a derived class, extended class, or child class.
- **Super Class/Parent Class:** Superclass is the class from where a subclass inherits the features. It is also called a base class or a parent class.
- **Reusability:** As the name specifies, reusability is a mechanism which facilitates you to reuse the fields and methods of the existing class when you create a new class. You can use the same fields and methods already defined in the previous class.

Javatpoint Link:

https://www.javatpoint.com/inheritance-in-java

GeeksforGeeks Link:

https://www.geeksforgeeks.org/inheritance-in-java/

- **Abstract Classes and Methods:**

Data abstraction is the process of hiding certain details and showing only essential information to the user.
Abstraction can be achieved with either abstract classes or interfaces (which you will learn more about in the next chapter).

The abstract keyword is a non-access modifier, used for classes and methods:

- Abstract class: is a restricted class that cannot be used to create objects (to access it, it must be inherited from another class).
- Abstract method: can only be used in an abstract class, and it does not have a body. The body is provided by the subclass (inherited from).

- ## Introduction to Packages:

A package is a mechanism to group the similar type of classes, interfaces and sub-packages and provide access control. It organizes classes into single unit. In Java already many predefined packages are available, used while programming.
For example: java.lang, java.io, java.util etc.

- ### Advantages of Packages:
- Packages provide code reusability, because a package has group of classes.
- It helps in resolving naming collision when multiple packages have classes with the same name.
- Package also provides the hiding of class facility. Thus other programs cannot use the classes from hidden package.
- Access limitation can be applied with the help of packages.
- One package can be defined in another package.

- ### Types of Packages:

There are two types of packages available in Java.

- **Built-in packages:**
  Built-in packages are already defined in java API. For example: java.util, java.io, java,lang, java.awt, java.applet, java.net, etc.
- **User defined packages:**
  The package we create according to our need is called user defined package.

- ## Interface:

An interface is a reference type in Java. It is similar to class. It is a collection of abstract methods.

- Interface is similar to a class, but it contains only abstract methods.
- By default the variables declared in an interface are public, static and final.
- Interface is a mechanism to achieve full abstraction.
- An interface does not contain any constructor.

Syntax:

```
interface InterfaceName
{
  public void method1();
  public void method2();
  <type>variableName = value;
}
```

- ## Extending interfaces:

An interface has to extend another interface as it happens in class. It cannot implement another interface.

Differences between Abstract class and Interface

Abstract class	Interface
It cannot support multiple inheritances.	Interface supports multiple inheritances.
It contains both abstract and non abstract method.	It contains only abstract method.
Abstract class is the partially implemented class.	Interface is fully unimplemented class.
It can have main method and constructor.	It cannot have main method and constructor.
It can have static, non-static, final, non-final variables.	It contains only static and final variable.

TutorialRide.com link:

https://www.tutorialride.com/core-java/packages-interfaces-in-java.htm

Javatpoint Link:

https://www.javatpoint.com/package#:~:text=A%20java%20package%20is%20a,io%2C%20util%2C%20sql%20etc.

- ## Java String:

In Java, string is basically an object that represents sequence of char values. An array of characters works same as Java string. For example:

```
char[] ch={'j','a','v','a','t','p','o','i','n','t'};
String s=new String(ch);
```

is same as:

```
String s="javatpoint";
```

Java String class provides a lot of methods to perform operations on strings such as compare(), concat(), equals(), split(), length(), replace(), compareTo(), intern(), substring() etc.

The java.lang.String class implements *Serializable*, *Comparable* and *CharSequence* interfaces.

- ## StringBuffer:

It is a peer class of String that provides much of the functionality of strings. The string represents fixed-length, immutable character sequences while StringBuffer represents growable and writable character sequences. StringBuffer may have characters and substrings inserted in the middle or appended to the end. It will automatically grow to make room for such additions and often has more characters preallocated than are actually needed, to allow room for growth.

StringBuffer(): It reserves room for 16 characters without reallocation

Example:

StringBuffer s = new StringBuffer();

GeeksforGeeks Link:

https://www.geeksforgeeks.org/stringbuffer-class-in-java/

- ## StringBuilder:

StringBuilder in Java represents a mutable sequence of characters. Since the String Class in Java creates an immutable sequence of characters, the StringBuilder class provides an alternative to String Class, as it creates a mutable sequence of characters. The function of StringBuilder is very much similar to the StringBuffer class, as both of them provide an alternative to String Class by making a mutable sequence of characters. However, the StringBuilder class differs from the StringBuffer class on the basis of synchronization. The StringBuilder class provides no guarantee of synchronization whereas the StringBuffer class does. Therefore this class is designed for use as a drop-in replacement for StringBuffer in places where the StringBuffer was being used by a single thread (as is generally the case). Where possible, it is recommended that this class be used in preference to StringBuffer as it will be faster under most implementations.

- **StringBuilder():** Constructs a string builder with no characters in it and an initial capacity of 16 characters.
- **StringBuilder(int capacity):** Constructs a string builder with no characters in it and an initial capacity specified by the capacity argument.
- **StringBuilder(CharSequence seq):** Constructs a string builder that contains the same characters as the specified CharSequence.
- **StringBuilder(String str):** Constructs a string builder initialized to the contents of the specified string.

GeeksforGeeks Link:

https://www.geeksforgeeks.org/stringbuilder-class-in-java-with-examples/

javatpoint Link:

https://www.javatpoint.com/StringBuilder-class

- ## StringTokenizer in Java:

The **java.util.StringTokenizer** class allows you to break a String into tokens. It is simple way to break a String. It is a legacy class of Java.

It doesn't provide the facility to differentiate numbers, quoted strings, identifiers etc. like StreamTokenizer class. We will discuss about the StreamTokenizer class in I/O chapter.

In the StringTokenizer class, the delimiters can be provided at the time of creation or one by one to the tokens.

- ### Constructors of the StringTokenizer Class:

There are 3 constructors defined in the StringTokenizer class.

Constructor	Description
StringTokenizer(String str)	It creates StringTokenizer with specified string.
StringTokenizer(String str, String delim)	It creates StringTokenizer with specified string and delimiter.
StringTokenizer(String str, String delim, boolean returnValue)	It creates StringTokenizer with specified string, delimiter and returnValue. If return value is true, delimiter characters are considered to be tokens. If it is false, delimiter characters serve to separate tokens.

JavaTpoint Link:

https://www.javatpoint.com/string-tokenizer-in-java

- **Java Exception Handling:**

In the tutorial, we will learn about different approaches of exception handling in Java with the help of examples.

In the last tutorial, we learned about Java exceptions. We know that exceptions abnormally terminate the execution of a program.

This is why it is important to handle exceptions. Here's a list of different approaches to handle exceptions in Java.

- try...catch block
- finally block
- throw and throws keyword
- **Major reasons why an exception Occurs:**
    - Invalid user input
    - Device failure
    - Loss of network connection
    - Physical limitations (out of disk memory)
    - Code errors
    - Opening an unavailable file

Programiz Link:

https://www.programiz.com/java-programming/exception-handling

GeeksforGeeks Link:

https://www.geeksforgeeks.org/exceptions-in-java/

- **Multithreading in Java:**

Multithreading is a Java feature that allows concurrent execution of two or more parts of a program for maximum utilization of CPU. Each part of such program is called a thread. So, threads are light-weight processes within a process.

Threads can be created by using two mechanisms :

- Extending the Thread class
- Implementing the Runnable Interface
- **Thread creation by extending the Thread class:**
  We create a class that extends the java.lang.Thread class. This class overrides the run() method available in the Thread class. A thread begins its life inside run() method.

- **Multitasking:**

Multitasking is a process of executing multiple tasks simultaneously. We use multitasking to utilize the CPU. Multitasking can be achieved in two ways:

- Process-based Multitasking (Multiprocessing)
- Thread-based Multitasking (Multithreading)

GeeksforGeeks Link:

https://www.geeksforgeeks.org/multithreading-in-java/#:~:text=Multithreading%20is%20a%20Java%20feature,Extending%20the%20Thread%20class

Javatpoint Link:

https://www.javatpoint.com/multithreading-in-java

- **Java – Generics:**

It would be nice if we could write a single sort method that could sort the elements in an Integer array, a String array, or an array of any type that supports ordering.

Java Generic methods and generic classes enable programmers to specify, with a single method declaration, a set of related methods, or with a single class declaration, a set of related types, respectively.

Generics also provide compile-time type safety that allows programmers to catch invalid types at compile time.

Using Java Generic concept, we might write a generic method for sorting an array of objects, then invoke the generic method with Integer arrays, Double arrays, String arrays and so on, to sort the array elements.

- **Generic Classes:**

A generic class declaration looks like a non-generic class declaration, except that the class name is followed by a type parameter section.

As with generic methods, the type parameter section of a generic class can have one or more type parameters separated by commas. These classes are known as parameterized classes or parameterized types because they accept one or more parameters.

Tutorialspoint Link:

https://www.tutorialspoint.com/java/java_generics.htm

GeeksforGeeks Link:

https://www.geeksforgeeks.org/generics-in-java/

Javatpoint Link:

https://www.javatpoint.com/generics-in-java
