# Java Terminology

**CHAPTER 1**

**Class** - blueprint or a template for creating different objects which defines its properties and behaviors

**GUI** – Graphical User Interface. A console to display output and get user input

**IDE** – Integrated Development Environment. Like Eclipse or NetBeans

**Java Compiler** – translates Java source code into *platform-independent* format known as *bytecode*. Files that contain bytecode have the .class extension

**JDK** – Java Development Kit

**JRE** – Java Runtime Environment. Includes all the software needed to run bytecode

**Packages** - organize classes that belong to the same category or provide similar functionality

**SDK** – Software Development Kit

**CHAPTER 2**

**Access Modifier** – *public* and *private* keywords are access modifiers that control the scope of the class

**API** – Application Programming Interface, provides all classes that are included as a part of the *Java Development Kit* (JDK)

**Block of Code** – code within curly braces

**Constructor** - special methods that are called when an object is instantiated. In other words, when you use the new keyword. ... A Java class constructor initializes instances (objects) of that class. Typically, the constructor initializes the fields of the object that need initialization

**Identifier** – any name that you create in Java, these can be names of classes, methods, variables, etc.

**Keyword** – a word that’s reserved by the Java language, you can use keywords as identifiers

**Method** – a block of code that performs a task

**Object** – they have states and behaviors. When you create an *object* from a Java class, you are creating an instance of the class. Then you can use the methods of the class by calling them from the object

**Relational Operators** – use to compare two number operands

**Static Methods** - By declaring a method using the static keyword, you can call it without first creating an object because it becomes a class method (a method that belongs to a class rather than an object). To code a static method, you code an *access modifier*, the *static keyword*, its *return type*, its *name*, and a *parameter* list

**Token** – an entry that a user makes. A user can make two or more tokens by using whitespace (spaces, tab, or return)

**CHAPTER 3**

**Implicit Casts** – performed automatically and can be used to convert data with a less precise type to a more precise type

**Narrowing Conversion** – casts data from a more precise type to a less precise type

**CHAPTER 4**

**Block Scope** – any variables defined within a block of code

**Break Statement** – breaks out of the loop by causing the program execution to jump to the statement that follow the loop

**Continue Statement** – continues a loop by causing execution to jump to the top of the loop

**Short-Circuit Operators** – when there are two operators, the second one is only performed if needed

**CHAPTER 5**

**Arguments** – if a method includes parameters, you pass arguments to the parameters

**Exception** – an object that contains information about an error that has occurred. When an error occurs in a method, the method *throws* an exception

**Exception Handling** – code a *try/catch* *statement* to *catch* and handle any exceptions that are *thrown*

**Subclass** – all exceptions are subclasses of the Exception class. The Exception class represents the most general type of exception

**CHAPTER 6**

**Breakpoint** – causes program execution to stop before the line that contains the breakpoint is executed

**Debug** – find and fix all the errors that are found when the application is tested

**JAR Files** – Java Archive, stores all the classes and resources needed by your application

**Logic Errors** – statements that don’t cause syntax or runtime errors, but produce the wrong results

**Runtime Errors** – don’t violate any syntax rules, but they throw *exceptions* that stop the execution of the application

**Step Into** – (F7) steps through the code one statement at a time, including statements in called methods

**Step Over** – (F8) steps through the code one statement at a time, skipping over all methods

**Syntax Errors** – violate the rules on how Java statements must be written

**CHAPTER 7**

**Business Tier** – provides interface between the database tier and the presentation tier

**Database Tier** – responsible for all the database access that’s required by the application

**Minus Sign (-)** – marks private

**Plus Sign (+)** – marks public

**Presentation Tier** – handles the details of the applications user interface

**Three-Tiered Application** – consists of a presentation tier, a business tier, and a database tier

**UML** – Unified Modeling Language, used to describe the classes and objects of an object-oriented application

**CHAPTER 8**

**Abstract Class** – class that can be inherited by other classes but that you can’t use to create an object. To declare an abstract class, code the abstract keyword in the class declaration

**Abstract Method** – code the abstract keyword in the method declaration and you omit the method body. Abstract methods cannot have private access

**Access Modifiers** - specify the accessibility of the members declared by a class

**Annotation** – a standard way to provide information about your code. When you override a method, you can add the @Override annotation to the method

**Base Class/Parent Class/Superclass** - a class that another class inherits

**Child Class/Derived Class/Subclass** – a class that inherits from an existing class

**Extend** – a subclass can extend the superclass by adding new fields and methods to the superclass

**Final Class** – create to prevent a class from being inherited

**Final Parameter** – prevent a method from assigning a new value to a parameter

**Inheritance** – lets you create a new class based on an existing class. Then, the new class *inherits* the fields and methods of the existing class

**Polymorphism** – lets you treat objects of different types as if they were the same type by referring to a superclass that’s common for both objects. One benefit is that you can write generic code that’s designed to work with the superclass. Then you can use code with instances of any class that’s derived from the superclass

**Protected Members** – are accessible to the current class, to other classes in the same package, and to subclasses

**CHAPTER 9**

**Clone** – create a new instance of the object that contains all the same data as the first object

**DAO** – Data Access Object

**Default Methods** – work much like regular methods in a class. They can define static methods, as a result, abstract classes have fewer advantages over interfaces

**Implements** – a class that *implements* an interface can use any constants defined by the interface. In addition, it must provide an implementation for each abstract method defined by the interface. If it doesn’t, the class must be declared as abstract

**Inheritance** – mechanism wherein a new class is derived from an existing class. When a child class is extending a superclass

**Interface** – can define one or more methods. These methods are automatically public and abstract. As a result, the interface only specifies the method signatures, not any code that implements the methods

**Multiple Inheritance** – a class can inherit more than one class

**CHAPTER 10**

**Enumeration** – is a special Java type used to define collections of constants. More precisely, a Java enumeration type is a special kind of Java class. An enumeration can contain constants, methods etc. It contains a set of related constants. The constants are defined with the int type and are assigned values from 0 to the number of constants in the enumeration minus 1;

**HTML Tag** – Use this tag to display text in monospaced format. Most likely to use <code> tag

**Javadoc Comment** – begins with /\*\* and ends with \*/. Use these to describe a class and its public and protected fields, constructors, and methods

**Library** – can store one or more packages the each contains one or more classes

**Package** – can store one or more classes, it can also store interfaces

**Project Jigsaw** – divides the Java platform into a set of independent modules

**Static Import** – lets you simplify references to the constants in an enumeration. Use a *static import* to import all of the constants of an enumeration or all of the static fields and methods of a class

**CHAPTER 11**

**Array** – an object that contains one or more items called *elements*. Each element is a primitive type such as an int or a double or an object such as String or a custom type

**Enhanced Foreach Loop** – simplifies the code required to loop through arrays

**Index** – to refer to elements in an array, use an *index* that ranges from 0 to one less than the number of elements in the array

**CHAPTER 12**

**Array List** – more efficient than a linked list for accessing individual elements randomly. However, less efficient than a linked list when inserting elements into the middle of the list

**Autoboxing** – automatically add and remove the wrapper classes for primitive types whenever necessary. Used by collections

**Collection** – an object that can hold other objects. Collections are similar to arrays, but are more flexible to use

**Collection Framework** – interface based, which means that each class in the collection framework implements one of the interfaces defined by the framework

**Dequeue** – operation that retrieves the element that’s in front of the queue

**Diamond Operator** – you can omit the type from within the brackets that follow the constructor if the compiler can infer the type from the context. This empty set of brackets is known as the *diamond operator*.

**Enqueue** – operations that adds an element to the end of the queue

**Generics** – refers to a feature that lets you create typed collections

**Linked List** – the LinkedList class provides an efficient way to insert elements into the middle of a list. However, a linked list is less efficient when accessing elements

**Map** – a collection that contains values that are associated with keys. It stores key-value pairs where you can use a key to retrieve a value. Two commonly used classes that implement maps are HashMap and TreeMap

**Queue** – a type of collection that lets you access elements of a *first-in*, *first-out* (FIFO) basis

**Typed Collection** – a collection that can hold objects of only a certain type

**Wrapper Class** – an integer class. Allows the collection to store an array of integer values

**CHAPTER 13**

**Regular Expression** - special sequence of characters that helps you match or find other strings or sets of strings, using a specialized syntax held in a pattern

**Immutable** – a string that has a fixed length and cannot be edited. They can’t grow or shrink

**Mutable** – a string where you can edit the characters

**Padding** – adding spaces to a string to a specified length

**CHAPTER 15**

**Absolute Path** – specify the entire path for a file

**Autoflush Feature** – flushes the buffer each time the println() method is called

**Buffer** - a block of memory into which you can write data, which you can then later read again. This memory block is wrapped in a NIO *buffer* object, which provides a set of methods that makes it easier to work with the memory block

**Buffered Stream** – adds a block of internal memory called a *buffer* to a stream. Use to make disk processing more efficient

**Checked Exceptions** – the compiler checks these exceptions to make sure that you have handled them. As a result, you must handle these exceptions before you can compile your code

**DAO** – Data Access Object

**Flush** – sends all data in the buffer to the I/O device. One way to do that is to a use try-with-resources statement to automatically close the I/O stream after you use it

**I/O Operations** – input and output operations (file I/O)

**Input File** – file that is read by a program

**Output File** – file that is written by a program

**Relative Path** – specify the path of the file relative to the directory that contains the class that identifies the path

**Stream** – flow of data from one location to another. To write data to a file from internal storage, use an *output stream*. To read from a file into internal storage, use an *input stream*

**Text Files** – to read and write text files, use *character streams*. To read and write *binary files*, use *binary streams*

**Try-With-Resources** – statement is a special type of try statement that you can use to initialize an object that uses system resources. This statement automatically closes the object and releases its resources when it’s done executing

**UTF** – Universal Text Format. The writeUTF() method writes a two-byte number that indicates the number of bytes in the string, then it writes the characters using *Universal Text Format*