Scene Graph

The scene graph depicts a hierarchical tree of nodes that represents all of the visual elements of the application's user interface.
Each element in a scene graph is called a node and each node has an ID, style class, and bounding volume.
The root node is the base node and it cannot have any parent nodes. All other nodes in a scene graph have a single parent and zero or more children

Event driven programming

In computer programming, event-driven programming is a programming paradigm in which the flow of the program is determined by events such as user actions (mouse click, key presses) sensor outputs or messages from other programs. Event driven programming is the dominant paradigm used in graphical user interfaces and other applications that are centered on performing certain actions in response to user input.

ExecutorService

The java.util.concurrent.ExecutorService interface represents an asynchronous execution mechanism which is capable of executing tasks in the background. An ExecutorService is thus very similar to a **thread pool**. In fact, the implementation of ExecutorService present in the java.util.concurrent package is a thread pool implementation.

ForkJoinPool

The ForkJoinPool is similar to the **Java ExecutorService** but with one difference. The ForkJoinPool makes it easy for tasks to split their work up into smaller tasks which are then submitted to the ForkJoinPool too. Tasks can keep splitting their work into smaller subtasks for as long as it makes to split up the task.

Blocking Queue

The Java BlockingQueue interface in the java.util.concurrent package represents a queue which is thread safe to put into, and take instances from. A BlockingQueue is typically used to have on thread produce objects, which another thread consumes.

Semaphore

The java.util.concurrent.Semaphore class is a **counting semaphore**. That means that it has two main methods:

- acquire()
- release()

The counting semaphore is initialized with a given number of "permits". For each call to acquire() a permit is taken by the calling thread. For each call to release() a permit is returned to the semaphore. Thus, at most N threads can pass the acquire() method without any release() calls, where N is the number of permits the semaphore was initialized with. The permits are just a simple counter.

CountDownLatch

A java.util.concurrent.CountDownLatch is a concurrency construct that allows one or more threads to wait for a given set of operations to complete.
A CountDownLatch is initialized with a given count. This count is decremented by calls to the countDown() method. Threads waiting for this count to reach zero can call one of the await() methods. Calling await() blocks the thread until the count reaches zero.

CyclicBarrier

The java.util.concurrent.CyclicBarrier class is a synchronization mechanism that can synchronize threads progressing through some algorithm. In other words, it is a barrier that all threads must wait at, until all threads reach it, before any of the threads can continue.

Lock

It is a thread synchronization mechanism just like synchronized blocks. A Lock is, however, more flexible and more sophisticated than a synchronized block.

ReadWriteLock

Allows multiple threads to read a certain resource, but only one to write it, at a time.

Read Lock: If no threads have locked the ReadWriteLock for writing, and no thread have requested a write lock. Thus, multiple threads can lock the lock for reading.

Write Lock: If no threads are reading or writing. Thus, only one thread at a time can lock the lock for writing.

Atomic Variables

the atomic classes make heavy use of compare-and- swap (CAS), an atomic instruction directly supported by most modern CPUs.
Those instructions usually are much faster than synchronizing via locks.
The advice is to prefer atomic classes over locks in case you just have to change a single mutable variable concurrently.
Many atomic classes: AtomicBoolean, AtomicInteger, AtomicReference, AtomicIntegerArray, etc

Atomic Boolean

Provides a boolean variable which can be read and written atomically, and which also contains advanced atomic operations like compareAndSet()

Interface

An **interface** is a reference type in **Java**. It is similar to class. It is a collection of abstract methods. A class implements an **interface**, thereby inheriting the abstract methods of the **interface**. Along with abstract methods, an **interface** may also contain constants, default methods, static methods, and nested types

Override vs Overload

Overloading occurs when two or more methods in one class have the same method name but different parameters.
Overriding means having two methods with the same method name and parameters (i.e., method signature). One of the methods is in the parent class and the other is in the child class.

Concurrent Programming

there are two basic units of execution:
processes and threads
- a computer system normally has many active processes and threads –even for only one execution core
- processing time for a single core is shared among processes and threads through an OS feature called time slicing.
- more and more common for computer systems to have multiple processors or processors with multiple execution cores

Processes

a process has a self-contained execution environment.
- in general it has a complete, private set of basic run-time resources; for example, each process has its own memory space.
- most implementations of the Java virtual machine run as a single process

processes are fully isolated from each other
-to facilitate communication between processes, most operating systems support Inter Process Communication (IPC) resources, such as pipes and sockets. (for communication between processes either on the same system or on different systems)

Threads

are called lightweight processes
- creating a new thread requires fewer resources than creating a new process.
-threads exist within a process — every process has at least one.
-threads share the process's resources, including memory and open files. This makes for efficient, but potentially problematic, communication.

Threads are the units that are scheduled by the system for executing on the processor
- On a single processor, each thread has its turn by multiplexing based on time

- On a multiple processor, each thread is running at the same time with each processor/ core running a particular thread.

a thread is a particular execution path of a process.
- one allows multiple threads to read and write the same memory (no process can directly access the memory of another process).
- when one thread modifies a process resource, the change is immediately visible to sibling threads.

What is a thread

A sequential program is one that when executed has only one single flow of control.
i.e. at any time instant, there is at most only one instruction (or statement or execution point) that is being executed in the program.

A multi-thread program is one that can have multiple flows of control when executed.
At some time instance, there may exist multiple instructions (or execution points) that are being executed in the program
Ex. in a Web browser we may do the following tasks at the same time:
- scroll a page, download an image, play sound, print a page

A thread is a single sequential flow of control within a program.

Default Access Modifier

Default access modifier means we do not explicitly declare an access modifier for a class, field, method, etc.
A variable or method declared without any access control modifier is available to any other class in the same package. The fields in an interface are implicitly public static final and the methods in an interface are by default public.

Public

A class, method, constructor, interface, etc. declared public can be accessed from any other class. Therefore, fields, methods, blocks declared inside a public class can be accessed from any class belonging to the Java Universe.
However, if the public class we are trying to access is in a different package, then the public class still needs to be imported. Because of class inheritance, all public methods and variables of a class are inherited by its subclasses.

Protected

Variables, methods, and constructors, which are declared protected in a superclass can be accessed only by the subclasses in other package or any class within the package of the protected members' class.
The protected access modifier cannot be applied to class and interfaces. Methods, fields can be declared protected, however methods and fields in a interface cannot be declared protected.

Protected access gives the subclass a chance to use the helper method or variable, while preventing a nonrelated class from trying to use it.

Private

Methods, variables, and constructors that are declared private can only be accessed within the declared class itself.
Private access modifier is the most restrictive access level. Class and interfaces cannot be private.
Variables that are declared private can be accessed outside the class, if public getter methods are present in the class.
Using the private modifier is the main way that an object encapsulates itself and hides data from the outside world.

Static

A static method belongs to the class rather than object of a class.
A static method can be invoked without the need for creating an instance of a class.
A static method can access static data member and can change the value of it.
A static method cannot use those fields (or call those methods) which are not static. It can use or call only the static members.
Static fields are shared by all class instances. They are allocated only once in the memory.
If a static field is not initialized, it will take the default value of its type

Inheritance

Inheritance is a mechanism wherein a new class is derived from an existing class. In **Java**, classes may **inherit** or acquire the properties and methods of other classes. A class derived from another class is called a subclass, whereas the class from which a subclass is derived is called a superclass.

Polymorphism

Polymorphism is the ability of an object to take on many forms. The most common use of **polymorphism** in OOP occurs when a parent class reference is used to refer to a child class object. Any **Java** object that can pass more than one IS-A test is considered to be **polymorphic**.

Abstract Method

Abstract classes are **classes** that contain one or more **abstract** methods. An **abstract** method is a method that is declared, but contains no implementation. **Abstract classes** may not be instantiated, and require subclasses to provide implementations for the **abstract** methods.

Java Stream

```
import java.util.Arrays;

import java.util.Collections;

import java.util.List;

import java.util.stream.Collectors;

public class Main
{
	public static void main(String[] args)
	{
		List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8,9,10,11,12,14,15);

		List<Integer> first = numbers.stream().filter(p->((p%3==0)||
(p%7==0))).collect(Collectors.toList());

		Integer third = second.stream().reduce((a,b)->((a+b)%5)).orElse(0); // similar
to .get(); returns a default value if no values are left after the reduce

		List<Integer> fourth = Collections.singletonList(third); // same as
Arrays.asList(third)

		List<Integer> everything =Collections.singletonList(numbers.stream().filter(p-
>((p%3==0)||(p%7==0))).map(p->p-1).reduce((a,b)->((a+b)%5)).orElse(0));

		System.out.println(everything);

	}

}
```