Java Threads: Visual Comparison & Cheatsheet

1. Visual Chart: Java Threading Approaches Comparison

Method	**Version**	**Returns Result**	**Ease**	**Thread Mgmt**	**Best Use**
Extends Thread	1.0	No	Easy	Manual	Quick standalone thread
Implements Runnable	1.0	No	Easy	Manual	Reusability with other classes
ExecutorService	5	No (Runnable)	Medium	Automatic	Multiple tasks/thread pool
Callable + Future	5	Yes	Medium	Automatic	Need result from task
ForkJoinPool	7	Yes	Medium	Auto-balanced	Divide & conquer tasks
CompletableFuture	8	Yes	Flexible	Auto	Async + chaining tasks
Parallel Streams	8	Yes	Simple	Auto	Parallel collection processing
Virtual Thread	21	Yes	Very Easy	Auto	Massive concurrency
Structured Concurrency	21	Yes	Safe	Scoped	Related task groups

Java Threads: Visual Comparison & Cheatsheet

2. Java Threading Cheatsheet

```
1. Create Thread (Simple)
    new Thread(() -> System.out.println("Hi")).start();

2. Using Runnable
    Runnable task = () -> {...};
    new Thread(task).start();

3. ExecutorService
    ExecutorService ex = Executors.newFixedThreadPool(3);
    ex.execute(task); ex.shutdown();

4. Callable + Future
    Future<T> f = ex.submit(() -> returnVal);
    T val = f.get();

5. CompletableFuture
    CompletableFuture.supplyAsync(() -> ...)
                     .thenApply(...)
                     .thenAccept(...);

6. ForkJoinPool
    ForkJoinPool.commonPool().submit(() -> {...});

7. Virtual Thread (Java 21)
    Thread.startVirtualThread(() -> {...});

8. Virtual Thread Executor
    Executors.newVirtualThreadPerTaskExecutor().submit(task);

9. Structured Concurrency (Java 21)
    try (var scope = new StructuredTaskScope.ShutdownOnFailure()) {
        var t1 = scope.fork(() -> task1());
        var t2 = scope.fork(() -> task2());
        scope.join(); scope.throwIfFailed();
    }
```

Java Threads: Traditional vs Virtual Threads (With Real Examples)

Overview: Java Threading Techniques

Java provides various threading models from its early versions to the latest Java 21+.
This PDF covers traditional threads, Runnable, ExecutorService, Callable, CompletableFuture,
ForkJoinPool, virtual threads, and structured concurrency with real examples.

1. Extending Thread Class

```
class MyThread extends Thread {
    public void run() {
        System.out.println("Running in thread: " + Thread.currentThread().getName());
    }
}
public class Main {
    public static void main(String[] args) {
        new MyThread().start();
    }
}
```

2. Implementing Runnable Interface

```
class MyRunnable implements Runnable {
    public void run() {
        System.out.println("Runnable running in: " + Thread.currentThread().getName());
    }
}
public class Main {
    public static void main(String[] args) {
        Thread t = new Thread(new MyRunnable());
        t.start();
    }
}
```

3. ExecutorService with Runnable

```
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

public class Main {
    public static void main(String[] args) {
        ExecutorService executor = Executors.newFixedThreadPool(2);
        executor.execute(() -> System.out.println("Task executed"));
        executor.shutdown();
    }
}
```

4. Callable with Future

```
import java.util.concurrent.*;

public class Main {
    public static void main(String[] args) throws Exception {
        ExecutorService executor = Executors.newSingleThreadExecutor();
        Callable<String> task = () -> "Callable Result";
        Future<String> future = executor.submit(task);
        System.out.println(future.get());
        executor.shutdown();
    }
}
```

5. ForkJoinPool

```
import java.util.concurrent.ForkJoinPool;

public class Main {
    public static void main(String[] args) {
        ForkJoinPool pool = new ForkJoinPool();
        pool.submit(() -> System.out.println("Task in ForkJoinPool")).join();
        pool.shutdown();
    }
}
```

6. CompletableFuture

```
import java.util.concurrent.CompletableFuture;

public class Main {
    public static void main(String[] args) {
        CompletableFuture.supplyAsync(() -> "Step 1")
                         .thenApply(res -> res + " -> Step 2")
                         .thenAccept(System.out::println);
    }
}
```

7. Parallel Streams

```
import java.util.stream.IntStream;
```

```
public class Main {
    public static void main(String[] args) {
        IntStream.range(1, 5).parallel().forEach(i ->
            System.out.println(i + " in " + Thread.currentThread().getName())
        );
    }
}
```

8. Virtual Threads (Java 21+)

```
public class Main {
    public static void main(String[] args) {
        for (int i = 0; i < 5; i++) {
            Thread.startVirtualThread(() -> {
                                                System.out.println("Virtual   thread:   "   +
Thread.currentThread().getName());
            });
        }
    }
}
```

9. Virtual Thread ExecutorService

```
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

public class Main {
    public static void main(String[] args) {
        ExecutorService executor = Executors.newVirtualThreadPerTaskExecutor();
        executor.submit(() -> System.out.println("Virtual Executor Task"));
        executor.shutdown();
    }
}
```

10. Structured Concurrency (Java 21+)

```
import java.util.concurrent.StructuredTaskScope;

public class Main {
    public static void main(String[] args) throws Exception {
        try (var scope = new StructuredTaskScope.ShutdownOnFailure()) {
            var user = scope.fork(() -> "User Data");
            var order = scope.fork(() -> "Order Info");

            scope.join();
            scope.throwIfFailed();

            System.out.println(user.result() + ", " + order.result());
        }
    }
}
```