ISSN 2658-5014 (Print) ISSN 2686-7044 (Online)
http://siit.ugatu.su 2025. Vol. 7, No. 5(24)
SYSTEMS ENGINEERING AND INFORMATION TECHNOLOGIES
Scientific paper

UDC 004.8 DOI: 10.54708/2658-5014-SIIT-2025-no5-p136 EDN: ITOSME

# Actual Concurrency Patterns: An Initial Collection

A. V. Prutzkow

Concurrency has become a key performance tool in programming. Programmers write concurrent programs faster when use patterns. Concurrency patterns are collected in books of the 2000s and reflect the state of the art of that time. Actual issues explore the concurrency patterns, but do not systematized them like object-oriented design patterns. The purpose of the study is to stimulate discussion of concurrency patterns in programming by their classifying and initial collecting. Patterns fall into six types: (1) Synchronized Data Processing, (2) Task Decomposition, (3) Data Decomposition, (4) Class/Method Structural Patterns, (5) Thread Control, and (6) Architectural Patterns. Instances of the types are: (1) Shared/Atomic Method, Resource Pool, Copy on Write Collection, Lock Striping, (2) Pipeline, (3) Data Item Enumeration, Divide and Conquer (Fork/Join), (4) Before/After, Condition with Volatile, (5) Invoke All, Invoke Any, (6) Producer–Consumer, Readers–Writers, Publisher–Subscriber. These patterns are collected because they are (1) basic, or (2) actual, or (3) typical instances of the pattern types. The patterns don not depend on the programming language, but we used the Java programming language in the listings.

*concurrency; patterns; threads; synchronization; task decomposition; data decomposition; Java*

## Introduction

Design patterns simplify program development by providing ready-made structures for program elements, reducing development time and increasing reusability [Tho25]. At the same time, the classes used in patterns are more coupled and less cohesive [Moh16]. Patterns constitute a new level of abstraction [San97]. Object-oriented design patterns are systematized in [Gam95]. In [Bug17], it is suggested to read this book with a good amount of skepticism. Currently, concurrency is a way to improve program performance. One of the first publications on concurrency patterns is [San97]. [Mat05, Sch00] are devoted to concurrency patterns. Patterns of programs are collected in [San11, Sot09]. However, these books present the state of the art of the 2000s. Concurrency has evolved significantly since then, especially in the Java programming language. In what follows, we will consider concurrency in this language. Fundamental principles of concurrency are explored in [Goe06], book of the 2000s as well. Actual concurrent patterns are listed in [Zap21] and are inspired by [Goe06], but do not form a system. [Ast17, Bob24] discuss the patterns but do not focus on them.

## Motivation

While writing a monograph on concurrency in Java [Pru26], we discovered that there is no system of concurrency patterns similar to [Gam95]. This forced us to collect concurrency patterns from various sources.

## The Purpose of the Study

The purpose of the study is to stimulate discussion of concurrency patterns in programming by their classifying and initial collecting, as in [San97].

Prutzkow A. V. Actual Concurrency Patterns: An Initial Collection // СИИТ. 2025. Т. 7, № 5(24). С. 136-145. DOI: 10.54708/2658-5014-SIIT-2025-no5-p136. EDN: ITOSME.

Prutzkow A. V. “Actual Concurrency Patterns: An Initial Collection” // SIIT. 2025. Vol. 7, no. 5(24), pp. 136-145. DOI: 10.54708/2658-5014-SIIT-2025-no5-p136. EDN: ITOSME (In Russian).

## Terminology

### Thread or Task?

During the evolution of concurrency in Java, the task, not the thread, became its fundamental element. A thread is a tool for executing tasks. However, we will continue to consider the thread the fundamental element of concurrency.

### Concurrency vs Parallelism

Concurrency and parallelism are not the same thing.

Concurrency is the simultaneous execution of threads using shared resources.

Parallelism is the simultaneous execution of threads that exclusively (under the problem conditions) use resources. Parallelism is difficult to implement in practice. If threads execute on unconnected computers, these computers may be connected to the same power supply system. Then, the power supply system becomes a shared resource. Under the problem conditions, the power supply may be insignificant, so this caveat is included in the definition of "parallelism".

## Types of Concurrency Patterns

Concurrency patterns fall into the following types:

- Patterns of Synchronized Data Processing are for thread-safe data modifying;
- Patterns of Task Decomposition are for breaking down tasks into parts;
- Patterns of Data Decomposition are for breaking down data into parts for parallel processing;
- Class/Method Structural Patterns are structures of class or method;
- Thread Control Patterns;
- Architectural Patterns are structures of program; the solution to the problem is reduced to the architecture determined by the pattern.

## The Structure of Pattern Description

We provide the following structure of the pattern description (similar to [Mat05]):

- Name: pattern name;
- Uses: used patterns or an elementary pattern;
- Problem: the problem that the pattern is used to solve;
- Solution: a solution to the problem using Java tools or an abstract description of the solution.
- Discussion: discussion of the pattern and solution.

## Patterns of Synchronized Data Processing

### Shared/Atomic Method

Name: Shared/Atomic Method (Atomic Compound Actions [Zap21]).

Uses: It's an elementary pattern.

Problem: Executing a method with a non-atomic sequence of actions over an object results in an inconsistent state of that object.

Solution: Use synchronization: low-level (the synchronized statement, the synchronized block), or high-level (ReentrantLock), or classes with atomic methods (AtomicLong, etc., LongAdder, etc.).

Discussion: An object's state becomes inconsistent due to a prohibited sequence of changes. To maintain a consistent state, it is necessary to restrict actions that change it. This restriction is achieved by converting the sequence of actions into an atomic command.

### Resource Pool

Name: Resource Pool [Zap21] (Shared-Resource [San97]).

Uses: Shared/Atomic Method.

Problem: Threads use objects to process data. The number of threads is significantly greater than the number of objects. How is to organize the getting and returning of objects?

Solution: Getting and returning of objects is a duty of the dispatcher (listing 1). There are two methods of the dispatcher: get() (lines 8–17) and return() (lines 19–22). Threads call the methods. Use low-level synchronization with the wait() and notify()/notifyAll() methods (listing 1), or the ReentrantLock and Condition classes, or the Semaphore class [Zap21], or tread-safe collections.

Discussion: We use the pattern in our practice, but it's rarely mentioned in the literature.

Listing 1

**The ObjectDispatcher class**

```
001  public class ObjectDispatcher<T> {
002     private Queue<T> objects;
003
004     public ObjectDispatcher(Queue<T> objects) {
005        this.objects = objects;
006     }
007
008     public synchronized T get() {
009        if (this.objects.size() == 0) {
010           try {
011              this.wait();
012           } catch (InterruptedException e) {
013              Thread.currentThread().interrupt();
014           }
015        }
016        return this.objects.poll();
017     }
018
019     public synchronized void return(T object) {
020        this.objects.add(object);
021        this.notify();
022     }
023  }
```

### Copy-on-Write Collection

Name: Copy on Write Collection [Goe06, Eva22].

Uses: Shared/Atomic Method.

Problem: Threads share a data collection. Threads iterate over the collection more often than they modify it. What data structure for the collection will provide the best performance?

Solution: If a copy of the collection is created each time it is modified, then traversal of the collection does not need to be synchronized, since the iterators will go over the original collection. The pattern is implemented in the CopyOnWriteArrayList and CopyOnWriteArraySet classes. Shared/Atomic Method is used when copying a set when it is modified.

Discussion: The collection is traversed more often than it is modified – this condition narrows the scope of the pattern, but makes the CopyOnWriteArrayList class thread-safe and linearizable [Haw12]. Actions over an object are linearizable if every action is atomic in a moment between its call and return [Sco24].

### Lock Striping

Name: Lock Striping [Eva22, Goe06].

Uses: Shared/Atomic Method.

Problem: Threads modify a data collection. It is necessary to improve performance while maintaining synchronization.

Solution: To improve performance, it's necessary to split the synchronization of the entire collection into parts. If one thread accesses one part of the collection, another thread can access another part of the collection in parallel. Such data collection are non-blocking. Use non-blocking collections such as ConcurrentHashMap, ConcurrentSkipListMap, ConcurrentSkipListSet, ConcurrentLinkedDeque, or ConcurrentLinkedQueue. These collections never throw ConcurrentModificationException. The Shared/Atomic Method pattern is used to synchronize access to a part of the collection.

Discussion: The principles of non-blocking collections are explored in [Sco24].

## Patterns of Task Decomposition

### Pipeline

Name: Pipeline [Bob24, Mat05, Sot09].

Uses: Resource Pool (in some implementations).

Problem: It is necessary to perform the stages of solving the problem in parallel.

Solution: To solve a problem, a pipeline of problem-solving stages is formed. Each stage processes a unit of data and passes it to the next stage (fig. 1). The next stage processes another unit of data in parallel. Java does not provide tools for implementing this pattern.

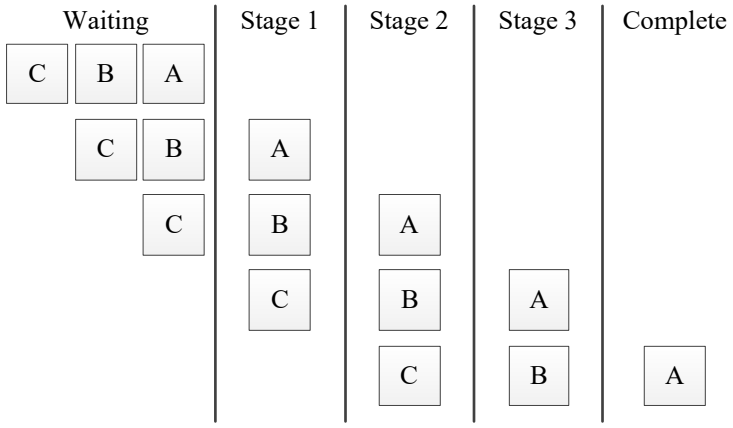


**Fig. 1.** The pipeline processes data units $A$, $B$, and $C$ processed in 3 stages [Sot09].

Discussion: This pattern implemented in [Cho15] and in a generic and reusable parallel pattern interface (GRPPI) for the C++ language [Ast17], and has no tools in Java yet. The reason is the difficulty of its implementation, despite its clarity. The Pipeline pattern is the only task decomposition pattern that we found.

## Patterns of Data Decomposition

### Data Item Enumeration

Name: Data Item Enumeration.

Uses: Shared/Atomic Method.

Problem: Data needs to be processed in parallel.

Solution: Data can be processed in parallel if the data is sequential collection and is divided into independent ranges (fig. 2). The processing results are aggregated using the Shared/Atomic Method pattern.

If the data is not sequential, it can be enumerated. Let's assume there is a tree (fig. 3) with folders and files. The file paths are pre-stored in a list. The Data Item Enumeration pattern is then applied. The way for transforming the data and enumerating it depends on the structure of the data and

the problem. Java provides an iterator for part of a collection – an object of the Spliterator interface. Streams are transformed into parallel ones by the parallel() method.

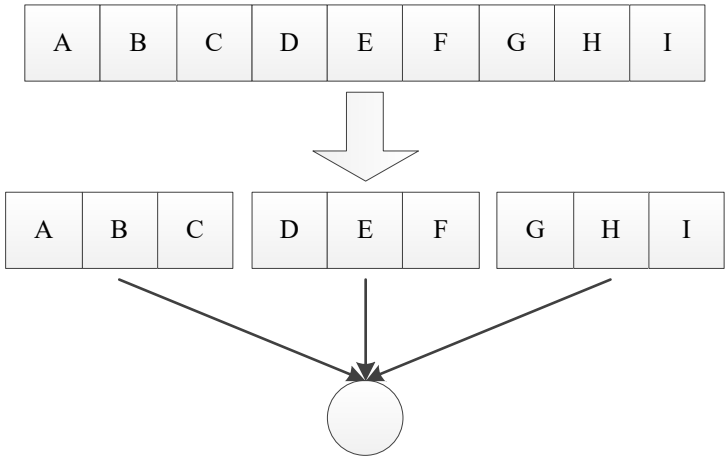


**Fig. 2.** Data decomposition into parts, their parallel processing, and aggregation of the results.

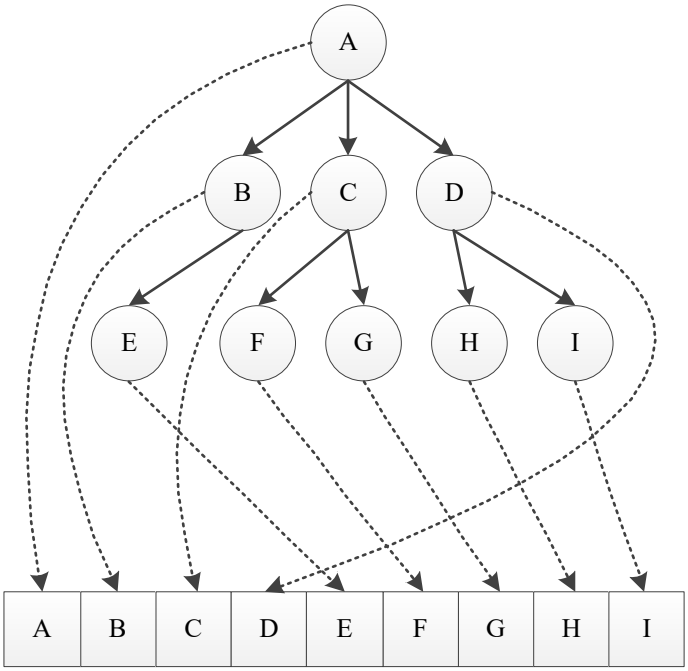


**Fig. 3.** Transforming a tree into an list for parallel processing.

Discussion: Decomposing data into independent parts and processing them in parallel makes the problem embarrassingly parallel [Bob24, Sot09]. An embarrassingly parallel problem is that:

$$P \gg S,$$

where $P$ are the stages of problem solving that can be performed in parallel; $S$ are the stages of problem solving performed sequentially.

Embarrassingly parallel problems have the following properties [Bob24, Eva24]:

- no or minimal synchronization;
- use of data decomposition rather than task decomposition;
- no or weak dependency between subtasks;
- infinite time acceleration due to parallel processing.

**Divide and Conquer (Fork/Join)**

Name: Divide and Conquer (Fork/Join) [Bob24, Far03, Mat05].

Uses: Depending on implementation.

Problem: It's impossible to immediately split the data into independent parts, but for each part, it's known whether it can be split or processed. How can data be processed in parallel?

Decision: At each step, a decision is made whether to compute the data or to divide it (fork) (fig. 4). In the first case, the data parts are computed and results are waited (join), obtained results are aggregated.

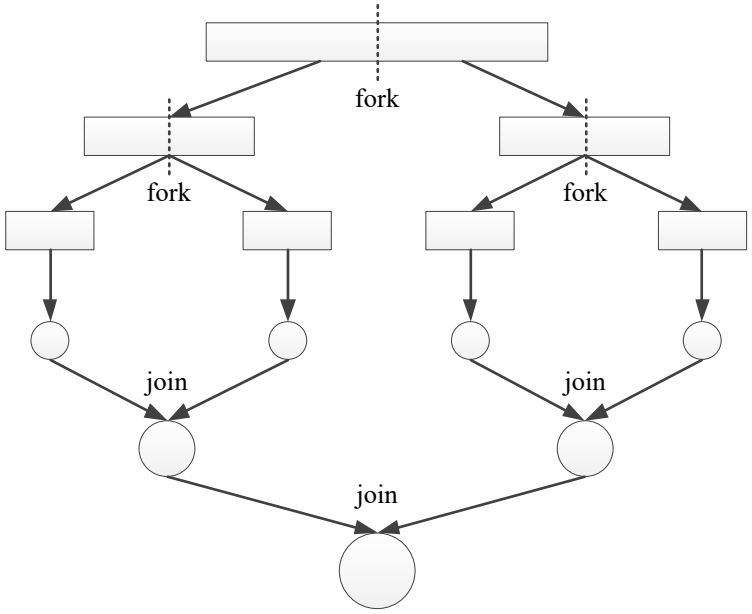


**Fig. 4.** The Divide and Conquer pattern of task solution [Urm19].
Rectangles represent the task and subtasks, circles represent results.

Use the ForkJoinPool class. The RecursiveTask and RecursiveAction classes have a compute() method that implements the Divide and Conquer pattern. The compute() method has a general structure (listing 2) [Sha18, Urm19].

Discussion: The size of the data to be processed is determined by the overall processing performance.

Listing 2

**The Structure of the compute() Method**

```
001  if (task is small enough or no longer divisible) {
002    compute the task
003  } else {
004    divide the task into subtasks
005    launch the subtasks asynchronously (the fork stage)
006    wait for the completion of all subtasks (the join stage)
007    combine the results of all subtasks
008  }
```

## Class/Method Structural Patterns

**Before/After**

Name: Before/After (Layering Policy Control, Before/After Control [Lea99]).

Uses: It's an elementary pattern.

Problem: Before and after an action, the preceding and following actions must be completed.

Solution: Action before() (listing 3, line 1) is executed before the action. The after() action (line 5) is executed after the action in the finally part in case an exception occurs.

Discussion: The pattern is used for synchronization by the ReentrantLock class.

Listing 3

**An implementation of the Before/After pattern**

```
001  before();
002  try {
003     action();
004  } finally {
005     after();
006  }
```

### Condition with Volatile

Name: Condition with Volatile (Shutdown with Volatile [Eva22]).

Uses: It's an elementary pattern.

Problem: A thread executes actions in a loop. The loop can be terminated externally. What structure should the thread class have to ensure that terminating the loop is thread-safe?

Solution: Introduce a boolean field with the volatile modifier (listing 4, line 2), add it as a loop termination condition (lines 9–11), and a method to set the value that terminates the loop (lines 4–6).

Discussion: The pattern is used for thread impact: stopping, suspending, resuming.

Listing 4

**Typical implementation of the Condition with Volatile pattern**

```
001  public class TaskExecutor implements Runnable {
002     private volatile boolean condition = false;
003
004     public void setConditionToTrue() {
005        this.condition = true;
006     }
007
008     public void run() {
009        while (!this.condition) {
010           // commands
011        }
012     }
013  }
```

## Thread Control Patterns

### Invoke All

Name: Invoke All [Vee24] (Task Convergence [Zap21]).

Uses: Depending on implementation.

Problem: Threads need to be started to get results from them.

Solution: Java has several tools for solving this problem. Use the ExecutorService, or CompletionService, or CompletableFuture, or CyclicBarrier, or CountDownLatch, or StructuredTaskScope classes/interfaces.

Discussion: This pattern is used when solving embarrassingly parallel problems.

### Invoke Any

Name: Invoke Any [Vee24].

Uses: Depending on implementation.

Problem: Threads need to be started to produce the first result. The results of the other threads are irrelevant.

Solution: Use the ExecutorService, CompletionService, CompletableFuture, or StructuredTaskScope classes/interfaces. When the result is obtained, the other threads are terminated.

Discussion: This pattern is used for solving embarrassingly parallel problems as well.

## Architectural Patterns

### Producer–Consumer

Name: Producer–Consumer [Abr99, Bob24].

Uses: Resource Pool (in some implementations).

Problem: Producers create objects, and consumers process them. How can producers and consumers be connected?

Solution: Use a thread-safe queue with a dispatcher for adding and obtaining data [Goe06]. Producers push objects onto the queue, and consumers pull them. There is general implementation in [Mal19].

Discussion: The pattern decouples producers and consumers “in time, space, and synchronization” [Eug03] in full. The pattern implemented and in hardware [Pra13].

### Readers–Writers

Name: Readers–Writers [Bob24, Dow09].

Uses: Depending on implementation.

Problem: Readers and writers work with the same data collection. Readers only read data, while writers modify data. How is to provide high performance in their work?

Solution: Use the ReentrantReadWriteLock class [Mat05].

Discussion: A parameter of this pattern is the priorities of readers and writers. Possible approaches are discussed in [Car05, Sco24]. The final decision is made after performance measurements.

### Publisher–Subscriber

Name: Publisher–Subscriber [Sch00, Urm19].

Uses: Depending on implementation.

Problem: Publisher creates objects. The subscriber requests them from the publisher. How is to organize their interaction?

Solution: Use the Flow class, its nested interfaces Processor, Publisher, Subscriber, Subscription, and the implementation class SubmissionPublisher [Fer17].

Discussion: The Publisher-Subscriber pattern differs from Producer–Consumer pattern as follows:

- the publisher and subscriber interact directly without a communication channel;
- the subscriber interacts with a specific publisher, the publisher knows about subscribers only in general, for example, through the interface.

Various programs based on the Publisher – Subscriber pattern have been developed [Laz22].

## Conclusion

The patterns above are (1) basic, or (2) actual, or (3) typical instances of pattern types. There are other patterns, but they do not meet the conditions.

We hope this article will become the beginning of a discussion of concurrency patterns, their collection, and the discovering of new patterns.

## Список литературы | References

[Abr99] Abraham U. Models for Concurrency. CRC Press, 1999. Abraham U. Models for Concurrency. CRC Press, 1999.

[Ast17] del Rio Astorga D. et al. A Generic Parallel Pattern Interface for Stream and Data Processing // Concurrency and Computation: Practice and Experience. 2017. Vol. 29. DOI: 10.1002/cpe.4175.

[Bob24] Bobrov K. Grokking Concurrency. Manning, 2024.

[Bug17] Bugayenko Ye. Elegant Objects. CreateSpace, 2017.

[Car05] Carver R. H., Tai K. Modern Multithreading: Implementing, Testing, and Debugging Multithreaded Java and C++/Pthreads/Win32 Programs. Wiley, 2005.

[Cho15] Chow J. et al. Pipeline Pattern in an Object-Oriented, Task-Parallel Environment // Concurrency and Computation: Practice and Experience. 2015. Vol. 27(5). P. 1273–1291. DOI: 10.1002/cpe.3305.

[Dow09] Downey A. The Little Book of Semaphores. Green Tea Press, 2009.

[Eug03] Eugster P. Th. et al. The Many Faces of Publish/Subscribe // ACM Computing Surveys. 2003. Vol. 35(2). P. 114–131.

[Eva22] Evans B. et al. The Well-Grounded Java Developer. 2nd ed. Manning, 2022.

[Eva24] Evans B., Gough J. Optimizing Cloud Native Java Practical Techniques for Improving JVM Application Performance. 2nd ed. O'Reilly, 2024.

[Far03] Farchi E. et al. Concurrent bug patterns and how to test them. In: Proceedings International Parallel and Distributed Processing Symposium. IEEE, 2003.

[Fer17] Fernández González J. 2nd ed. Mastering Concurrency Programming with Java 9. Packt, 2017.

[Gam95] Gamma E. et al. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995.

[Goe06] Goetz B. et al. Java Concurrency in Practice. Addison-Wesley, 2006.

[Haw12] Hawkins P. et al. Concurrent Data Representation Synthesis. In: PLDI'12: Proceedings of the 33rd ACM SIGPLAN Conference on Programming Language Design and Implementation. ACM, 2012. P. 417–428.

[Laz22] Lazidis A. et al. Open-Source Publish-Subscribe Systems: A Comparative Study. In: Advanced Information Networking and Applications. AINA 2022. Lecture Notes in Networks and Systems, vol. 449. Springer, 2022. P. 105–115.

[Lea99] Lea D. Concurrent Programming in Java. 2nd ed. Addison-Wesley, 1999.

[Mal19] Malkhi D. (ed.). Concurrency: The Works of Leslie Lamport. ACM, 2019.

[Mat05] Mattson T. et al. Patterns for Parallel Programming. Addison Wesley, 2005.

[Moh16] Mohammed M. et al. Empirical Insight into the Context of Design Patterns: Modularity Analysis. In: 2016 7th International Conference on Computer Science and Information Technology. IEEE, 2016.

[Pra13] Prat-Pérez A. et al. Producer-Consumer: the Programming Model for Future Many-core Processors. In: ARCS 2013. Lecture Notes in Computer Science, vol. 7767. Springer, 2013.

[Pru26] Пруцков А. В. Многопоточность в Java: шаблоны и инструменты. М.: Курс, 2026. 188 с.

[San11] Sandén B. I. Design of Multithreaded Software. Wiley, 2011.

[San97] Sandén B. I. Concurrent Design Patterns for Resource Sharing. TRI-Ada, 1997.

[Sch00] Schmidt D. et al. Pattern-Oriented Software Architecture. Vol. 2: Patterns for Concurrent and Networked Objects. John Wiley and Sons, 2000.



del Rio Astorga D. et al. A Generic Parallel Pattern Interface for Stream and Data Processing // Concurrency and Computation: Practice and Experience. 2017. Vol. 29. DOI: 10.1002/cpe.4175.

Bobrov K. Grokking Concurrency. Manning, 2024.


Bugayenko Ye. Elegant Objects. SPb: Piter, 2017.


Carver R. H., Tai K. Modern Multithreading: Implementing, Testing, and Debugging Multithreaded Java and C++/Pthreads/Win32 Programs. Wiley, 2005.


Chow J. et al. Pipeline Pattern in an Object-Oriented, Task-Parallel Environment // Concurrency and Computation: Practice and Experience. 2015. Vol. 27. No. 5. P. 1273–1291. DOI: 10.1002/cpe.3305.


Downey A. The Little Book of Semaphores. Green Tea Press, 2009.

Eugster P. Th. et al. The Many Faces of Publish/Subscribe // ACM Computing Surveys. 2003. Vol. 35(2). P. 114–131.

Evans B. et al. The Well-Grounded Java Developer. 2nd ed. Manning, 2022.

Evans B., Gough J. Optimizing Cloud Native Java Practical Techniques for Improving JVM Application Performance. 2nd ed. O'Reilly, 2024.

Farchi E. et al. Concurrent bug patterns and how to test them. In: Proceedings International Parallel and Distributed Processing Symposium. IEEE, 2003.

Fernández González J. 2nd ed. Mastering Concurrency Programming with Java 9. Packt, 2017.

Gamma E. et al. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995.


Goetz B. et al. (Java Concurrency in Practice. Addison-Wesley, 2006.


Hawkins P. et al. Concurrent Data Representation Synthesis. In: PLDI'12: Proceedings of the 33rd ACM SIGPLAN Conference on Programming Language Design and Implementation. ACM, 2012.

Lazidis A. et al. Open-Source Publish-Subscribe Systems: A Comparative Study. In: Advanced Information Networking and Applications. AINA 2022. Lecture Notes in Networks and Systems, vol. 449. Springer, 2022. P. 105–115.

Lea D. Concurrent Programming in Java. 2nd ed. Addison-Wesley, 1999.

Malkhi D. (ed.). Concurrency: The Works of Leslie Lamport. ACM, 2019.

Mattson T. et al. Patterns for Parallel Programming. Addison Wesley, 2005.

Mohammed M. et al. Empirical Insight into the Context of Design Patterns: Modularity Analysis. In: 2016 7th International Conference on Computer Science and Information Technology. IEEE, 2016.

Prat-Pérez A. et al. Producer-Consumer: the Programming Model for Future Many-core Processors. In: ARCS 2013. Lecture Notes in Computer Science, vol. 7767. Springer, 2013.


Prutzkow A. V. Concurrency in Java: Patterns and Tools. Moscow: Kurs, 2026. 188 p. (In Russian).


Sandén B. I. Design of Multithreaded Software. Wiley, 2011.

Sandén B. I. Concurrent Design Patterns for Resource Sharing. TRI-Ada, 1997.

Schmidt D. et al. Pattern-Oriented Software Architecture. Vol. 2: Patterns for Concurrent and Networked Objects. John Wiley and Sons, 2000.

[Sco24] Scott M., Brown T. Shared-Memory Synchronization. 2nd ed. Springer, 2024.

[Sha18] Sharan K. Java Language Features: With Modules, Streams, Threads, I/O, and Lambda Expressions. Apress, 2018.

[Sot09] Sottile M. et al. Introduction to Concurrency in Programming Languages. CRC Press, 2009.

[Tho25] Thomas P. The Role of Design Patterns in Facilitating Code Reuse and Reducing Development Time. ResearchGate, 2025.

[Urm19] Urma R.-G. et al. Modern Java in Action. Lambdas, Streams, Functional and Reactive Programming. Manning, 2019.

[Vee24] Veen R., Vlijmincx D. Virtual Threads, Structured Concurrency, and Scoped Values. Apress, 2024.

[Zap21] Zapparoli L. H. GitHub – LeonardoZ/java-concurrency-patterns: Concurrency Patterns and features found in Java, through multithreaded programming. Threads, Locks, Atomics and more [webpage]. 2021. URL: https://github.com/LeonardoZ/java-concurrency-patterns.

Scott M., Brown T. Shared-Memory Synchronization. 2nd ed. Springer, 2024.

Sharan K. Java Language Features: With Modules, Streams, Threads, I/O, and Lambda Expressions. Apress, 2018.

Sottile M. et al. Introduction to Concurrency in Programming Languages. CRC Press, 2009.

Thomas P. The Role of Design Patterns in Facilitating Code Reuse and Reducing Development Time. ResearchGate, 2025.

Urma R.-G. et al. Modern Java in Action. Lambdas, Streams, Functional and Reactive Programming. Manning, 2019.

Veen R., Vlijmincx D. Virtual Threads, Structured Concurrency, and Scoped Values. Apress, 2024.

Zapparoli L. H. GitHub – LeonardoZ/java-concurrency-patterns: Concurrency Patterns and features found in Java, through multithreaded programming. Threads, Locks, Atomics and more [webpage]. 2021. URL: https://github.com/LeonardoZ/java-concurrency-patterns.

## ОБ АВТОРАХ | ABOUT THE AUTHORS

**ПРУЦКОВ Александр Викторович**
Рязанский государственный радиотехнический университет им. В. Ф. Уткина, Липецкий государственный педагогический университет им. П. П. Семенова-Тян-Шанского, Россия.
mail@prutzkow.com ORCID: 0000-0002-4110-5269.
Проф. каф. вычислительной и прикладной математики, проф. каф. информатики, информационных технологий и защиты информации.

**PRUTZKOW Alexander Viktorovich**
Ryazan State Radio Engineering University,
Lipetsk State Pedagogical University,
Russia.
mail@prutzkow.com ORCID: 0000-0002-4110-5269.
Prof., Dept. of Computational and Applied Mathematics, prof., Dept. of Computer Science, Information Technologies, and Information Security.

## МЕТАДАННЫЕ | METADATA

**Заглавие:** Современные шаблоны многопоточности: исходный набор.

**Авторы:** Пруцков А. В.

**Аннотация:** Многопоточность стала основным способом повышения производительности. Разработка многопоточных программ могла бы быть ускорена за счет применения шаблонов. Шаблоны собраны в книгах, изданных в 2000-х годах, и отражают состояние многопоточности того времени. Современные источники содержат шаблоны, но не систематизируют их, как шаблоны проектирования объектно-ориентированных программ. Цель работы – активизировать обсуждение шаблонов многопоточных программ за счет классификации и начального сбора актуальных шаблонов. Шаблоны разделены на следующие типы: (1) Synchronized Data Processing, (2) Task Decomposition, (3) Data Decomposition, (4) Class/Method Structural Patterns, (5) Thread Control, (6) Architectural Patterns. Представителями этих типов являются следующие шаблоны: (1) Shared/Atomic Method, Resource Pool, Copy on Write Collection, Lock Striping, (2) Pipeline, (3) Data Item Enumeration, Divide and Conquer (Fork/Join), (4) Before/After, Condition with Volatile, (5) Invoke All, Invoke Any, (6) Producer–Consumer, Readers–Writers, Publisher–Subscriber. Эти шаблоны собраны, потому что они являются: базовыми, или актуальными, или типичные представители типов шаблонов. Шаблоны не зависят от языка программирования, но в листингах использовался язык программирования Java.

**Ключевые слова:** многопоточность; шаблоны; потоки выполнения; синхронизация; разложение задач; разложение данных; Java.

**Язык:** Английский.

Статья поступила в редакцию 15 июля 2025 г.

**Title:** Actual Concurrency Patterns: An Initial Collection.

**Authors:** Prutzkow A. V.

**Abstract:** Concurrency has become a key performance tool in programming. Programmers write concurrent programs faster when use patterns. Concurrency patterns are collected in books of the 2000s and reflect the state of the art of that time. Actual issues explore the concurrency patterns, but do not systematized them like object-oriented design patterns. The purpose of the study is to stimulate discussion of concurrency patterns in programming by their classifying and initial collecting. Patterns fall into six types: (1) Synchronized Data Processing, (2) Task Decomposition, (3) Data Decomposition, (4) Class/Method Structural Patterns, (5) Thread Control, and (6) Architectural Patterns. Instances of the types are: (1) Shared/Atomic Method, Resource Pool, Copy on Write Collection, Lock Striping, (2) Pipeline, (3) Data Item Enumeration, Divide and Conquer (Fork/Join), (4) Before/After, Condition with Volatile, (5) Invoke All, Invoke Any, (6) Producer–Consumer, Readers–Writers, Publisher–Subscriber. These patterns are collected because they are (1) basic, or (2) actual, or (3) typical instances of the pattern types. The patterns don not depend on the programming language, but we used the Java programming language in the listings.

**Key words:** concurrency; patterns; threads; synchronization; task decomposition; data decomposition; Java.

**Language:** English.

The article was received by the editors on 15 July 2025.