https://www.youtube.com/watch?v=C7Cob-zPEWc&ab_channel=theTechology

Watch Ads and Make Money | Earn Money Online

https://www.youtube.com/watch?v=qcPcYC0atwY&ab_channel=theTechology

https://www.youtube.com/shorts/_kQ7SDr_zdM

https://www.youtube.com/watch?v=te3OU9fxC8U&ab_channel=JetBrainsTV

Rust vs Go - Which is Better and Why?

https://www.youtube.com/watch?v=ZJ6dVVobjaI&ab_channel=FedericoTerzi

Software Engineer Ranks Programming Languages

https://www.youtube.com/watch?v=6FEYcBPBGOk&ab_channel=Cl%C3%A9mentMihailescu

The history of Rust
https://www.youtube.com/watch?v=79PSagCD_AY&ab_channel=AssociationforComputingMachinery%28ACM%29

Intro

"Rust and Java are two of the most popular programming languages, but which one is best for your next project?

I am tired of superficial hot takes on Java with 0% depth
https://www.youtube.com/watch?v=6FEYcBPBGOk&ab_channel=Cl%C3%A9mentMihailescu

Objectively, Java succeeded at its primary “directive” I guess, write once, debug anywhere or something like that, and today we’ll leave all the memes aside and evaluate both languages in an “unbiased” way:

We will focus on:

1. Compilation
2. Memory Management
3. Concurrency
4. Tooling
5. Common use cases
6. Community/Ecosystem
7. Learning resources
8. Syntax and Style

I added these sections as chapters to this video, so feel free to jump around based on your particular needs.

Stick around, this is going to be awesome, I promise!

Compilation

Both Rust and Java are compiled programming languages, which means that the source code is transformed into an executable form before it can be run. However, the compilation process for Rust and Java are quite different in terms of how the code is transformed and how the resulting executables are structured.

In Rust, the compilation process starts with the Rust compiler, rustc, which takes the source code as input and generates an intermediate representation of the code in the form of LLVM IR. This LLVM IR is then passed to the LLVM compiler, which generates native machine code that can be run on the target platform.

LLVM means Low-Level Virtual Machine

The Rust compiler also performs static analysis of the code to catch common programming errors and ensure that the code is safe and efficient.

In Java, the compilation process starts with the Java compiler, javac, which takes the source code as input and generates bytecode in the form of class files. These class files contain the bytecode for the Java Virtual Machine (JVM), which is a platform-independent execution environment.

However, it is important to note that the JVM is not completely platform-agnostic, as it still depends on the underlying hardware and operating system to provide certain services, such as memory management and thread scheduling. Additionally, the performance of the JVM can vary depending on the underlying hardware and operating system, as some implementations may be more optimized than others.

To run a Java program, the bytecode in the class files must be loaded into the JVM and executed. The JVM translates the bytecode into native machine code on the fly and executes it, making it possible to run Java programs on any platform that has a JVM implementation.
In summary, the compilation process for Rust involves generating native machine code directly from the source code, while the compilation process for Java involves generating platform-independent bytecode that is executed by the JVM.

Memory Management

Java and Rust are both powerful languages, with at least one goal in common: To prevent memory bugs like seg faults, memory leaks or dangling pointers.

But they have radically different approaches to accomplish this.

Rust uses the borrowing and ownership system, while Java uses a garbage collection.

Early versions of Rust had a garbage collector https://youtu.be/olbTX95hdbg?t=1524 but the language designers at Mozilla figured out that if they wanted to replace C++ code with Rust they could not afford the overhead and complexity that a garbage collector brings in.

In Rust, every value has a single owner, and ownership can be transferred between variables through a process called "borrowing." This system helps ensure that values are not used after they have been freed, and that multiple variables do not try to modify the same memory at the same time.

Java, on the other hand, uses a garbage collector to automatically manage memory. The garbage collector periodically scans for objects that are no longer being used and frees the memory they occupy. This approach can make Java programs easier to write and debug, but it can also come at the cost of some performance, as the garbage collector can introduce pauses in the program's execution.

Both Rust and Java offer tools and techniques that can help programmers manage memory more effectively. For example, Rust has features such as "lifetimes" and "reference counting" that can help ensure that memory is used and freed in a controlled manner. Java has options for controlling the garbage collector and optimizing memory usage, such as the "-XX:+UseParallelGC" flag, which enables parallel garbage collection.

There are several common programming errors that can occur in Java that are prevented or made more difficult by Rust's borrowing and ownership system. Here are a few examples:

- Null references: Java allows null references, which can cause runtime errors if a null reference is dereferenced or used in a way that is not expected. Rust does not have null references, and instead uses an option type to represent the possibility of no value.
- Data races: Java's concurrency model is based on threads and locks, which can lead to problems such as data races if shared memory is not protected properly. Rust's borrowing and ownership system helps prevent data races by ensuring that values are accessed in a controlled and safe manner.

It is not easy to create a dangling pointer in Rust or Java, as Rust's borrowing and ownership system prevents references to values from becoming invalid. In Rust, every value has a single owner, and ownership can be transferred between variables through a process called "borrowing." This system ensures that values are not used after they have been freed, and that multiple variables do not try to modify the same memory at the same time.
Here is an example of a code snippet in Rust that tries to create a dangling pointer, but will fail to compile due to a borrow error:
Copy code

```
fn main() {
        // Create a new value and store it in a local variable
        let val = Box::new(42);
```

```
    // Create a new reference and store it in a local variable
    let ref1 = &val; // Try to create a second reference, but this will fail to compile
    let ref2 = &val;
}
```

In this example, the value created on line 4 is stored in a local variable called val, which is wrapped in a Box. On line 7, a new reference called ref1 is created and stores a reference to the value. On line 10, the program tries to create a second reference called ref2, but this will fail to compile with an error similar to the following:

Copy code

error[E0502]: cannot borrow `val` as immutable because it is also borrowed as mutable --> src/main.rs:10:17 | 7 | let ref1 = &val; | --- immutable borrow occurs here ... 10 | let ref2 = &val; | ^^^ mutable borrow occurs here

This error occurs because Rust's borrowing and ownership system does not allow multiple references to the same value at the same time. In order to create a second reference, the first reference would need to be released first. This helps prevent dangling pointers and ensures that references are always valid.

Null pointers in rust

It is not possible to create a null pointer in Rust, as Rust does not have null references. Instead of using null references, Rust uses an option type to represent the possibility of no value.

The Option type in Rust is an enumeration with two variants: Some, which wraps a value, and None, which represents the absence of a value. Here is an example of how the Option type can be used in Rust:

```
fn main() {
    let x: Option<i32> = Some(42);
    let y: Option<i32> = None;

    match x {
        Some(val) => println!("x has a value: {}", val),
        None => println!("x has no value"),
    }

    match y {
        Some(val) => println!("y has a value: {}", val),
        None => println!("y has no value"),
    }
}
```

It is not possible to disable null pointers in Java, as null references are a fundamental part of the Java language. Null references are used to indicate that a reference does not currently refer to an object, and they are supported by many of the language's features and libraries.

However, it is possible to reduce the risk of null pointer errors in Java by using techniques such as the following:

- Using the @Nonnull annotation: This annotation, which is part of the javax.annotation package, can be used to indicate that a method parameter or return value must not be null. The @Nonnull annotation can be used in combination with static analysis tools to help detect and prevent null pointer errors.
- Using the Optional class: The java.util.Optional class is a wrapper class that can be used to represent optional values. It can be used as an alternative to null references and can help prevent null pointer errors by requiring explicit handling of the absence of a value.
- Using the Objects.requireNonNull() method: This method, which is part of the java.util.Objects class, can be used to check if an object is null and throw a NullPointerException if it is. This method can be used to explicitly check for null values and to fail early in cases where a null value is not expected.

There are several static analysis libraries and tools available for Java that can be used to help detect and prevent null pointer errors and other types of defects and vulnerabilities in Java programs. Some examples of static analysis libraries and tools for Java include the following:

- FindBugs: FindBugs is a static analysis tool that can be used to detect and report potential bugs and security vulnerabilities in Java programs. It uses static analysis techniques to analyze Java bytecode and identify patterns that are indicative of defects or vulnerabilities. FindBugs is open source and is available for free.
- PMD: PMD is a static analysis tool that can be used to detect and report a wide range of coding problems in Java programs, including null pointer errors, unused variables, and code that violates coding standards. PMD is open source and is available for free.
- Checkstyle: Checkstyle is a static analysis tool that can be used to enforce coding standards and best practices in Java programs. It can be used to detect and report a wide range of coding problems, including null pointer errors, and can be configured to enforce specific coding standards and rules. Checkstyle is open source and is available for free.

Use cases

SIM Cards

JavaCard had received backing from Visa while MULTOS was endorsed by MasterCard.

Apple Pay

https://support.apple.com/guide/security/apple-pay-component-security-sec2561eb018/web

Web Development (Client) Side:

Since Java 1.0, it was possible to run Java on the browser via Java applets.

As millennial, I grew up with Java and flash games!

Java applets were introduced as part of the Java 1.0 release in 1996 and were designed to be a platform-independent, secure way to deliver interactive content to web browsers. Applets are written in Java and are typically compiled to Java bytecode, which can be executed by the Java Virtual Machine (JVM) on the client side. This allows applets to run in any web browser that has a JVM installed, without the need to install additional software or plugins.
To use a Java applet on a web page, the web page must include an <applet> HTML tag that specifies the applet's codebase, main class, and any required parameters. When the web page is loaded, the browser downloads the applet and its dependencies and runs the applet in a sandbox, providing a secure environment for the applet to execute.
Although Java applets were once a popular way to add interactive and dynamic content to web pages, they are now less commonly used due to security and performance concerns. Applets require the user to have a JVM installed and can be slow to start and run, especially on older or low-end devices. Additionally, applets have a reputation for being prone to security vulnerabilities, and many modern web browsers have disabled or removed support for applets due to these concerns.

Web Development (Server):

Android OS development

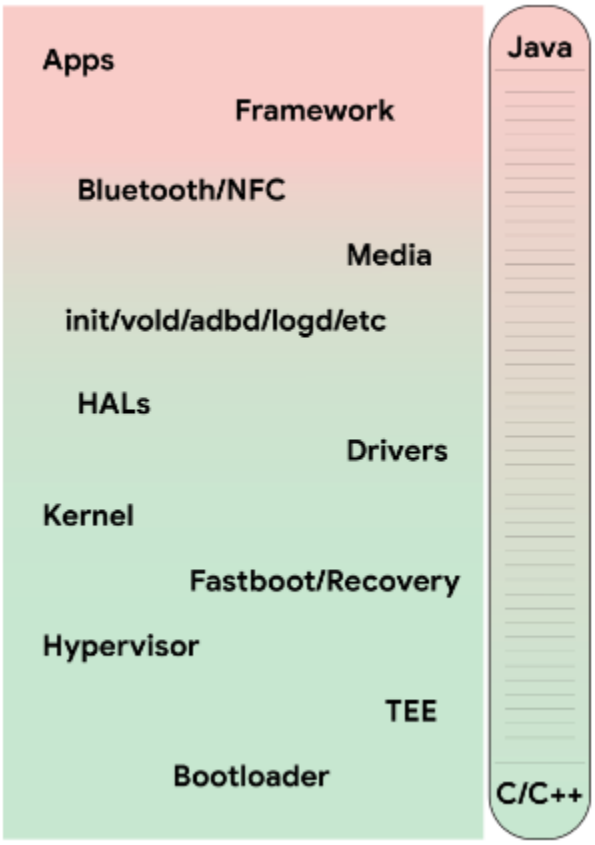
For developing apps, Java and Kotlin are the best options, Unfortunately, for the lower layers of the OS, Java and Kotlin are not an option.

https://security.googleblog.com/2021/04/rust-in-android-platform.html

Lower levels of the OS require systems programming languages like C, C++, and Rust. These languages are designed with control and predictability as goals. They provide access to low level system resources and hardware. They are light on resources and have more predictable performance characteristics.

For C and C++, the developer is responsible for managing memory lifetime. Unfortunately, it's easy to make mistakes when doing this, especially in complex and multithreaded codebases.

Rust provides memory safety guarantees by using a combination of compile-time checks to enforce object lifetime/ownership and runtime checks to ensure that memory accesses are valid. This safety is achieved while providing equivalent performance to C and C++.



Yes, it is possible to write Android apps in Rust, although it is not as common as using Java or Kotlin, which are the primary programming languages for Android development.
Rust is a systems programming language that is known for its safety, performance, and concurrency support, making it well suited for building low-level systems and libraries, as well as for developing high-performance and concurrent applications. However, Rust is not as widely used as Java or Kotlin for building Android apps, and the Android platform does not natively support Rust as a programming language.
To write an Android app in Rust, you can use the rust-android-ndk crate, which provides bindings to the Android Native Development Kit (NDK) and allows you to call into native Android APIs from Rust code. You can then use the cargo-apk crate to build an Android package (APK) from your Rust code, which can be installed and run on an Android device.
Alternatively, you can use a tool such as cargo-apk-pack or cargo-apk-build to automate the process of building an APK from Rust code. These tools allow you to specify the dependencies of your Rust project in a Cargo.toml file, and then build an APK that includes the Rust code and the required dependencies.

Keep in mind that building Android apps in Rust may require more effort and expertise than using Java or Kotlin, as you will need to use the Android NDK and possibly integrate with other tools and frameworks.

Tooling

Toolchain management

cargo is the package manager and build tool for Rust, while rustup is a tool for managing Rust installations and versions.
cargo is used to manage dependencies, build, and run Rust projects. It allows you to specify the dependencies of your project in a Cargo.toml file, and then automatically downloads and builds these dependencies as well as your own project. cargo also provides a variety of commands for building, testing, and publishing Rust packages, as well as for running and debugging Rust programs.
rustup is a tool for installing and managing multiple Rust toolchains and versions on the same machine. It allows you to install different versions of the Rust compiler and its standard library, as well as to switch between these versions easily. rustup also provides a set of command-line utilities for working with Rust projects, such as rustc, rustdoc, and rustfmt.
In general, cargo is used for managing and building Rust projects, while rustup is used for managing Rust installations and toolchains. Both tools are essential for working with Rust, and are typically used together in Rust development.
I hope this helps to clarify the difference between cargo and rustup. If you have any other questions, please don't hesitate to ask.

In Java, there is no equivalent to rustup, as Java does not have a tool for managing multiple installations and versions of the Java runtime and compiler in the same way that Rust does with rustup.
However, there are a few options for managing different versions of the Java runtime and compiler on the same machine.
One option is to use the sdkman tool, which is a command-line utility for managing multiple versions of the Java Development Kit (JDK), as well as other software development kits such as Gradle, Groovy, and Scala. sdkman allows you to install, switch between, and remove different versions of the JDK easily, and provides a set of commands for working with Java projects.
Another option is to use the jenv tool, which is a command-line utility for managing multiple versions of the JDK and Java runtime on the same machine. jenv allows you to specify the JDK or Java runtime that you want to use for a particular project or directory, and provides a set of commands for working with Java projects.
Both sdkman and jenv can be useful for managing multiple versions of the JDK and Java runtime on the same machine, but they do not provide all of the features of rustup, such as the ability to switch between Rust toolchains and versions easily.

I hope this helps to answer your question. If you have any other questions, please don't hesitate to ask.

Syntax and Style

Rust was built by people coming from these 3 camps:

1. C/C++
    a. Assurance that Rust will be good for low level users.
    b. They get a language with a strong type system and a package manager.
2. Scripting language (javascript, ruby)
    a. Appreciation for convenient user interface and experience.
    b. Cargo came directly from npm package manager
    c. A language that was extremely low level in performance.
3. Functional language
    a. Type Systems knowledge.

You can see that it has a very powerful type system that includes:

Traits (haskell)
Enums(
A Macro system

Deployment
Concurrency

Java or Rust do not prevent deadlocks but both
To create a deadlock in Rust, you can use the standard library's std::sync::Mutex type to create mutexes that can be used to synchronize access to shared resources. Here is an example of how you can create a deadlock in Rust:

```
use std::sync::{Mutex, Arc};
use std::thread;

fn main() {
    let lock1 = Arc::new(Mutex::new(1));
    let lock2 = Arc::new(Mutex::new(2));

    let lock1_clone = lock1.clone();
    let lock2_clone = lock2.clone();
```

```
    let t1 = thread::spawn(move || {
        let _guard1 = lock1_clone.lock().unwrap();
        println!("Thread 1 acquired lock 1");
        thread::sleep(std::time::Duration::from_secs(1));
        println!("Thread 1 trying to acquire lock 2");
        let _guard2 = lock2_clone.lock().unwrap();
        println!("Thread 1 acquired lock 2");
    });

    let t2 = thread::spawn(move || {
        let _guard2 = lock2.lock().unwrap();
        println!("Thread 2 acquired lock 2");
        thread::sleep(std::time::Duration::from_secs(1));
        println!("Thread 2 trying to acquire lock 1");
        let _guard1 = lock1.lock().unwrap();
        println!("Thread 2 acquired lock 1");
    });

    t1.join().unwrap();
    t2.join().unwrap();
}
```

```
class Deadlock {
    static Object lock1 = new Object();
    static Object lock2 = new Object();

    public static void main(String[] args) {
        Thread t1 = new Thread(new Runnable() {
            public void run() {
                synchronized(lock1) {
                    try {
                        Thread.sleep(100);
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                    synchronized(lock2) {
                        System.out.println("Thread 1");
                    }
                }
            }
        });
```

```
        Thread t2 = new Thread(new Runnable() {
            public void run() {
                synchronized(lock2) {
                    try {
                        Thread.sleep(100);
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                    synchronized(lock1) {
                        System.out.println("Thread 2");
                    }
                }
            }
        });

        t1.start();
        t2.start();
    }
}
```

Channels are a way of sending and receiving values between threads in Rust. They are implemented in the standard library in the std::sync::mpsc module and can be used to communicate between threads in a thread-safe and efficient way.
To use channels in Rust, you can create a channel using the channel function, which returns a pair of Sender and Receiver values. The Sender value can be used to send values over the channel, and the Receiver value can be used to receive values from the channel.
Here is an example of how you can use channels in Rust:

```
use std::sync::mpsc;
use std::thread;

fn main() {
    // Create a channel
    let (tx, rx) = mpsc::channel();

    // Spawn a new thread
    let t1 = thread::spawn(move || {
        // Send a value over the channel
        tx.send(1).unwrap();
    });

    // Wait for the thread to finish
```

```
    t1.join().unwrap();

    // Receive the value from the channel
    let value = rx.recv().unwrap();
    println!("Received value: {}", value);
}
```

In this example, a new thread is spawned and sends a value over the channel. The main thread then receives the value from the channel and prints it to the console.
Channels can be used to send values of any type that implements the Send trait. They are also multiple producer, single consumer (MPSC) channels, which means that multiple threads can send values over the same channel, but only one thread can receive values from the channel at a time.
In Java, you can create channels to communicate between threads using the java.util.concurrent.BlockingQueue interface and its implementing classes. A blocking queue is a thread-safe queue that can be used to communicate between threads by allowing one thread to put elements into the queue and another thread to take elements out of the queue.
To create a channel using a blocking queue in Java, you can create an instance of a blocking queue implementation, such as ArrayBlockingQueue or LinkedBlockingQueue, and pass it to the threads that will be communicating over the channel.
Here is an example of how you can create a channel using a blocking queue in Java:

```
import java.util.concurrent.ArrayBlockingQueue;
import java.util.concurrent.BlockingQueue;

public class Main {
    public static void main(String[] args) {
        // Create a blocking queue with a capacity of 10
        BlockingQueue<Integer> queue = new ArrayBlockingQueue<>(10);

        // Create a producer thread
        Thread producer = new Thread(() -> {
            try {
                // Put elements into the queue
                for (int i = 0; i < 100; i++) {
                    queue.put(i);
                    System.out.println("Produced: " + i);
                }
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        });

        // Create a consumer thread
        Thread consumer = new Thread(() -> {
```

```
            try {
                // Take elements from the queue
                while (true) {
                    int value = queue.take();
                    System.out.println("Consumed: " + value);
                }
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        });

        // Start the producer and consumer threads
        producer.start();
        consumer.start();
    }
}
```

In this example, a blocking queue is created with a capacity of 10, and a producer thread is created that puts elements into the queue. A consumer thread is also created that takes elements from the queue. The producer and consumer threads communicate over the queue by putting and taking elements from it.
Blocking queues are a convenient way to create channels between threads in Java, as they provide a thread-safe and efficient way to communicate between threads.
https://www.baeldung.com/java-asynchronous-programming

n Java, you can use the java.util.concurrent.CompletableFuture class and the java.util.concurrent.Flow API to write asynchronous and reactive code similar to the way you would using the Tokio library in Rust.
The CompletableFuture class represents a deferred computation that can be completed, either normally or exceptionally, and provides methods for composing and combining asynchronous tasks. The Flow API is a set of interfaces and classes for building reactive streams-based applications, and provides a way to process a potentially unbounded sequence of elements asynchronously and backpressure.
Here is an example of how you can use the CompletableFuture class to write asynchronous code in Java:

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

public class Main {
    public static void main(String[] args) throws ExecutionException, InterruptedException {
        // Create a completable future
        CompletableFuture<String> future = CompletableFuture.supplyAsync(() -> {
            // Perform some asynchronous computation
            try {
```

```
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        return "Hello, World!";
    });

    // Get the result of the future
    String result = future.get();
    System.out.println(result);
  }
}
```

Community/Ecosystem

Java has a large and established ecosystem, with a wide range of libraries, frameworks, and tools available. Rust is a relatively newer language, but it has a growing community and ecosystem, with a focus on systems programming and performance.

Backup

The original goal of Java was to create a simple, portable, and reliable programming language that could be used to build a wide range of applications for consumer devices, such as home appliances, cable boxes, and mobile phones.
Java was developed by Sun Microsystems in the mid-1990s as a successor to the C++ programming language. The creators of Java, James Gosling and his team at Sun, set out to design a language that would be easy to learn and use, yet powerful enough to support the development of complex systems and applications.
To achieve this goal, the Java language was designed to be simple, expressive, and object-oriented, with a syntax that is similar to C and C++. Java also includes a number of features that make it easier to write and maintain code, such as automatic memory management, exception handling, and a rich set of libraries and frameworks.
However, it is worth noting that Java's "write once, run anywhere" capability is not perfect, and there are some limitations and considerations to keep in mind. For example, Java programs may not always perform as well as natively compiled programs, and may require additional resources, such as memory and CPU, to run. Additionally, Java programs may not have access

to all of the platform-specific features and APIs of the host platform, and may need to be designed and implemented differently to take advantage of these features.
Overall, Java has been successful in achieving its goal of being a "write once, run anywhere" programming language, and has become a cornerstone of modern computing and a language that is widely used and supported on a wide range of platforms and devices.

The original goal of Rust was to create a programming language that offers the performance and safety of a systems programming language, while also being expressive and easy to use. Rust was designed to be a safe and concurrent language that is well suited for building low-level systems and libraries, as well as for developing high-performance and concurrent applications.
Rust was developed by Mozilla Research in the mid-2000s as a research project to explore new programming language design and implementation techniques. The goal of Rust was to address some of the problems that developers commonly face when working with systems programming languages, such as C and C++, such as security vulnerabilities, performance issues, and complexity.
To achieve this goal, the Rust language was designed to be expressive and easy to use, with a syntax that is similar to C and C++, but with more modern and convenient features, such as pattern matching, trait-based generics, and algebraic data types. Rust also includes a number of features that make it safer and more reliable, such as a static type system, a borrow checker, and a powerful macro system.
In addition to its expressiveness and safety, Rust was designed to be fast and efficient, with a focus on low-level systems programming and performance. Rust uses a lightweight runtime and a statically linked, ahead-of-time compiled model, which allows it to achieve performance that is similar to C and C++, while also providing a higher level of safety and concurrency.

Learning resources

Just run rustup doc and read a local version of the rust book.

https://doc.rust-lang.org/book/

Java vs Rust Cheat Sheet

Dimension	Java	Rust

Language Goals	1. Memory Safety 2. Portability (compile once run anywhere via the JVM) 3. Ease of use (coming from C)	1. Memory Safety 2. Portability (compile for each platform via LLVM) 3. Efficiency 4. Replace C and C++.
Safety and Memory Management	Garbage collector. Possible to suffer from memory-related issues such as memory leaks, garbage collection pauses, or poor performance due to poor memory usage patterns. May require careful tuning and optimization of the program and the garbage collector to achieve good performance.	Mostly automatic, enforced by the borrow checker. The borrow checker is a static analysis tool that ensures safe and correct use of memory by checking that references to data are valid and that data is not borrowed mutably while it is also borrowed immutably. Developers still need to think carefully about the lifetime of values.
Compilation (look at tools for specific tools)	Generates bytecode. Bytecode is executed by the Java Virtual Machine (JVM), which is mostly a platform-independent execution environment.	Generates native machine code that can be run on the target platform. Uses LLVM as the compiler backend (just like Swift).
Syntax		
Variables mutability	Mutable by default, unless final is used. var javaNumber = 3;	Immutable by default, use mut to make it mutable. let mut rust_str = “yolo”;
Structure	Classes are templates for objects that define behavior (methods) and state (properties). Composition is possible via **interfaces.** Class inheritance is implemented using **extends.**	Structs are used mostly to store state (properties), traits and stand alone functions are used to define behavior. Composition is preferred and encouraged via traits on structs and enums. Struct inheritance is not possible, instead, use trait inheritance.
Generics	public class List<T extends Comparable<T>> {	struct Vec<T: Ord> { // ...

	// ... }	}
Concurrency		
Blocking		
Deadlocks possible?	Yes	Yes

Deadlock sample
 	
On both samples, what causes the deadlock is that locks are acquired in a different order.
 	
I recommend using non-blocking concurrency techniques to avoid this.
	class Deadlock {
	  static Object lock1 = new Object();
	  static Object lock2 = new Object();
	
	  public static void main(String[] args) {
	  Thread t1 = new Thread(new Runnable() {
	  public void run() {
	  synchronized(lock1) {
	  try {
	  Thread.sleep(100);
	  } catch (InterruptedException e) {
	  e.printStackTrace();
	  }
	  synchronized(lock2) {
	  System.out.println("Thread 1");
	  }
	  }
	  }
	  });
	
	  Thread t2 = new Thread(new Runnable() {
	  public void run() {
	  synchronized(lock2) {
	  try {
	  Thread.sleep(100);
	  } catch (InterruptedException e) {
	  e.printStackTrace();
	  }
	  synchronized(lock1) {
	  System.out.println("Thread 2");
	  }
	  }
	  }
	  });
	
	  t1.start();
	  t2.start();
	  }
	}
		fn main() {
		  let lock1 = Arc::new(Mutex::new(1));
		  let lock2 = Arc::new(Mutex::new(2));
		
		  let lock1_clone = lock1.clone();
		  let lock2_clone = lock2.clone();
		
		  let t1 = thread::spawn(move || {
		  let _guard1 = lock1_clone.lock().unwrap();
		  println!("Thread 1 acquired lock 1");
		
		thread::sleep(std::time::Duration::from_secs(1));
		  println!("Thread 1 trying to acquire lock 2");
		  let _guard2 = lock2_clone.lock().unwrap();
		  println!("Thread 1 acquired lock 2");
		  });
		
		  let t2 = thread::spawn(move || {
		  let _guard2 = lock2.lock().unwrap();
		  println!("Thread 2 acquired lock 2");
		
		thread::sleep(std::time::Duration::from_secs(1));
		  println!("Thread 2 trying to acquire lock 1");
		  let _guard1 = lock1.lock().unwrap();
		  println!("Thread 2 acquired lock 1");
		  });
		
		  t1.join().unwrap();
		  t2.join().unwrap();
		}

Non blocking		
Parallel collections	Streams API https://docs.oracle.com/javase/tutorial/collections/streams/parallelism.html	Rayon: https://crates.io/crates/rayon
Atomic objects	The specifications of these methods enable implementations to employ efficient machine-level atomic instructions that are available on contemporary	All atomic types in this module are guaranteed to be lock-free if they're available. This means they don't internally acquire a global mutex. Atomic types and

	processors. However on some platforms, support may entail some form of internal locking. Thus the methods are not strictly guaranteed to be non-blocking -- a thread may block transiently before performing the operation https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/atomic/package-summary.html	operations are not guaranteed to be wait-free. https://doc.rust-lang.org/std/sync/atomic/
MPMC: Multi-producer multi-consumer	https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/ConcurrentLinkedQueue.html	https://doc.rust-lang.org/rust-by-example/std_misc/channels.html
MPSC: Multi-producer single consumer	I did not find a specialized class to do this but you can use MPMC.	https://crates.io/crates/crossbeam-channel
Actor model and reactive programming	Akka https://akka.io/ Akka is no longer free for commercial use: https://www.lightbend.com/akka/license-faq	Actix https://github.com/actix/actix MIT License: https://github.com/actix/actix/blob/master/LICENSE-MIT
async/await	NONE	Tokio, async-std, actix
Tooling *		
Compiler	javac	rustc (backend: llvm)
Toolchain manager	sdkman https://sdkman.io/	cargo https://doc.rust-lang.org/cargo/getting-started/installation.html
Dependency Manager	Gradle, Maven (old)	cargo
**Resources needed to run a web server hello world (dev mode) ** **	Spring Boot 3.0.0 2 processes: MEM: 134MB | 80 MB CPU: 1.28 | 1.40	Actix-web 4.0 1 process: MEM: 7 MB CPU: 0.37

Licensing	OracleJDK (stay away from this garbage) https://www.oracle.com/downloads/licenses/no-fee-license.html OpenJDK (preferred) https://openjdk.org/legal/gplv2+ce.html	The Rust Programming Language and all other official projects are dual-licensed: • Apache License, Version 2.0 • MIT license
Cost	Oracle announced that beginning from Java JDK 17 and onwards Java is free for commercial usage. "If you are running older versions of Java 1-16, you are not affected by the new licensing agreement." **Never believe Oracle.**	Free Company donations are appreciated: https://www.rust-lang.org/sponsors

* These are my personal preferences.
** Tested with MacBook Pro (16-inch, 2021) M1

Disclosure: Oracle is infamous for breaking their licenses and suing people, so take this information with a grain of salt.