

# **Что такое Kotlin?**

**И причем тут Java.**

**Матвей Попов**

# **План на урок**

- Что такое Kotlin и как он появился?
- Как и где исполняется Kotlin. Причем тут Java?
- Как устроена JVM
- Какой путь проходит Kotlin-code перед тем как стать приложением на JVM?
- Какие перспективы и возможности у Kotlin?



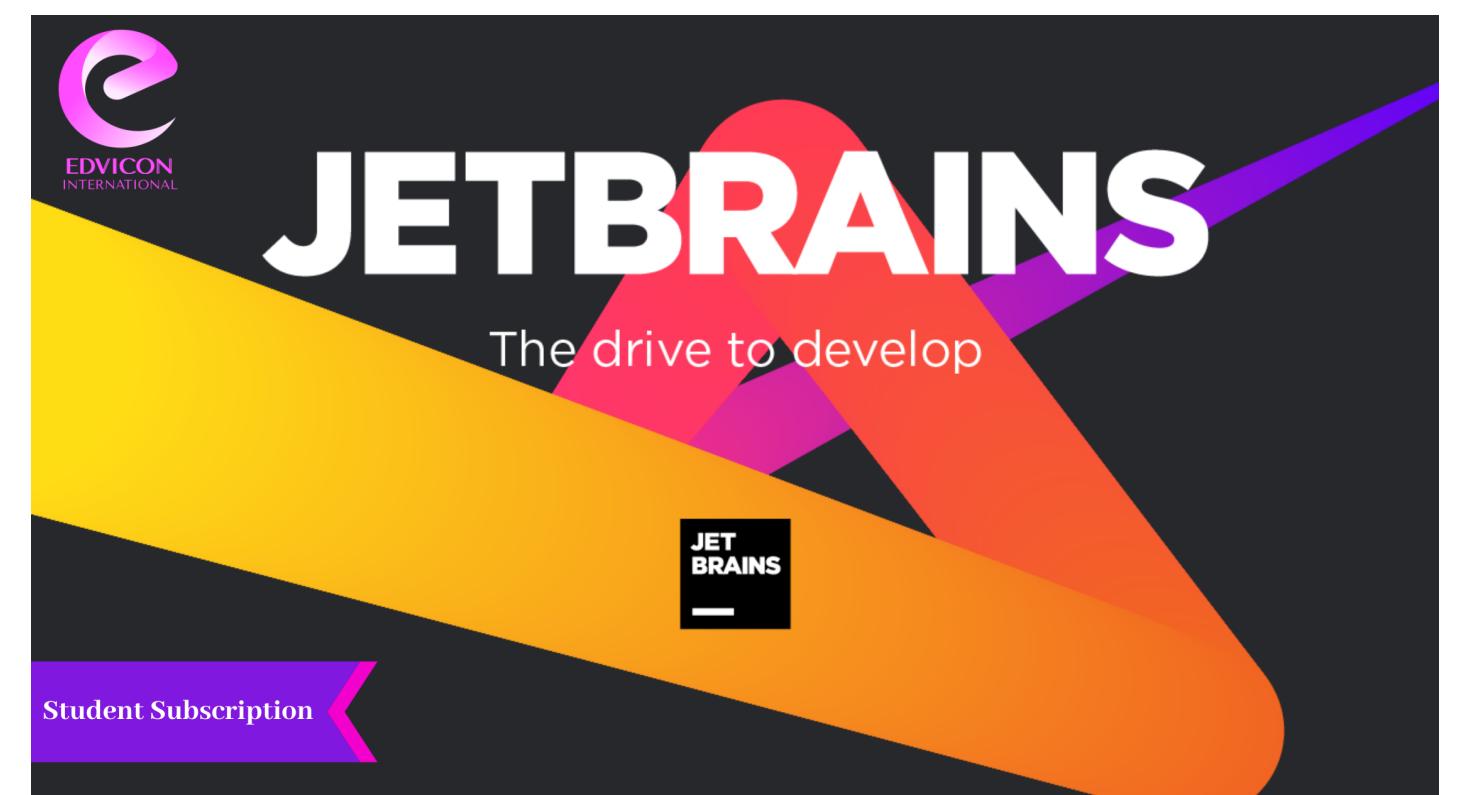
# Kotlin

```
    if (post.has_post_thumbnail()) {  
        val postThumbnail = post.get_post_thumbnail()  
        val postThumbnailUrl = postThumbnail?.get("url")  
        if (postThumbnailUrl != null) {  
            val thumbnailImage = Image(postThumbnailUrl)  
            return thumbnailImage  
        }  
    }  
    return null
```

# Историческая справка



- Статически типизированный, объектно-ориентированный язык программирования
- Первые обсуждения в 2010, первый релиз в 2016
- Создатель: Андрей Бреслав
- Компания: JetBrains
- Назван в честь острова Котлин в Финском заливе



# Статическая типизация

Приём, широко используемый в языках программирования, при котором переменная, параметр подпрограммы, возвращаемое значение функции связывается с типом в момент объявления и тип не может быть изменён позже



# Как и зачем решили создать Kotlin?



# Кто такие JetBrains?





FOR COMMUNITY

**JetBrains - компания, которая создает  
инструменты для разработки**



Если большое количество людей, которые разбираются в языке программирования, собираются вместе, то получается новый язык программирования.



- JetBrains долго развивался и копил знания
- JetBrains писали больше 10 лет на Java
- Мир изменился, а Java - нет
- Хотелось писать код продуктивнее, безопаснее и веселее



# Какие были требования?



- Статическая типизация
- Хороший инструментарий
- Взаимодействие, совместимость с существующими технологиями
- Доступность

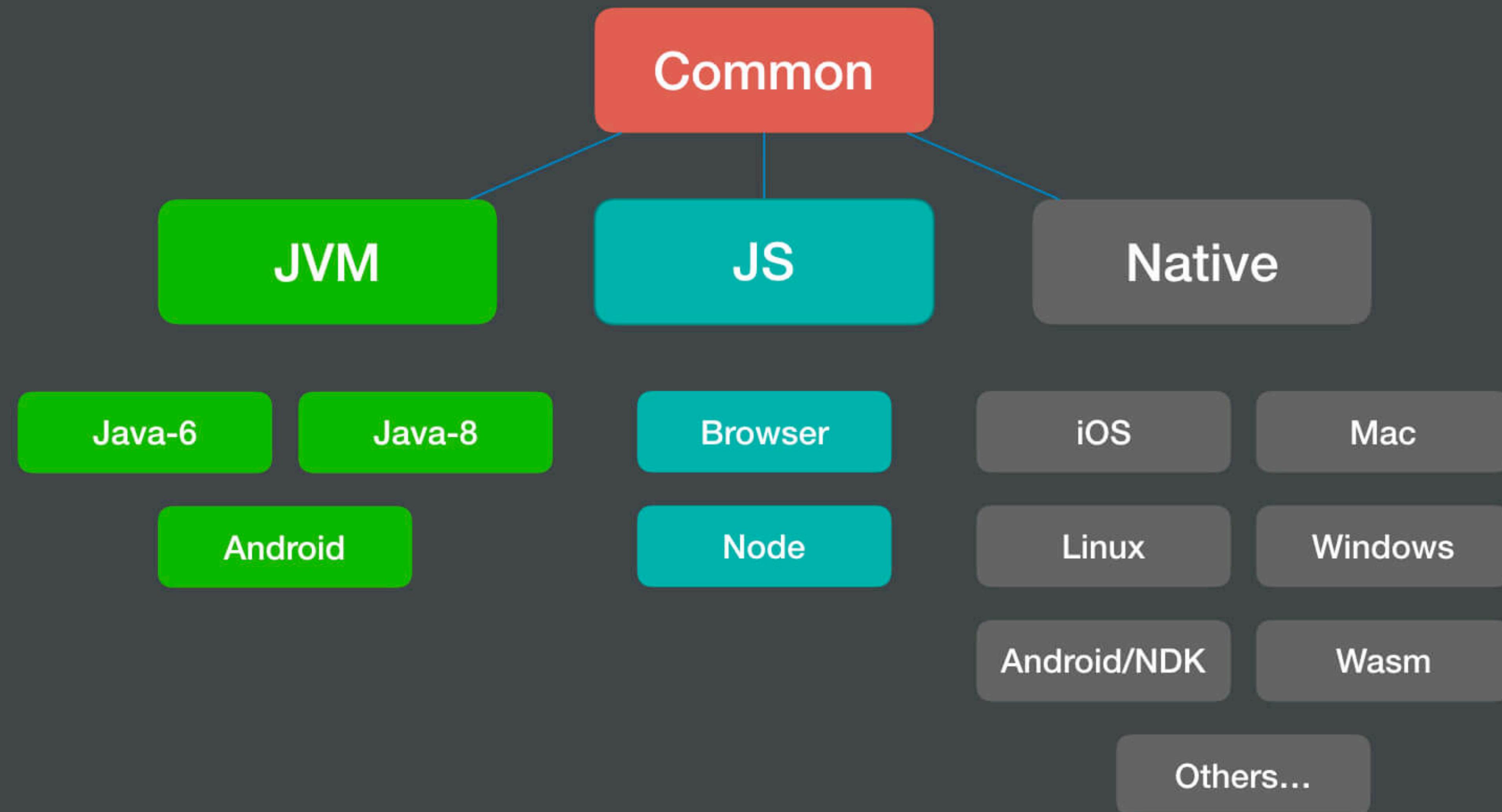


**И что мы в итоге получили?**



**Всё! И даже больше!**





# Что это значит?

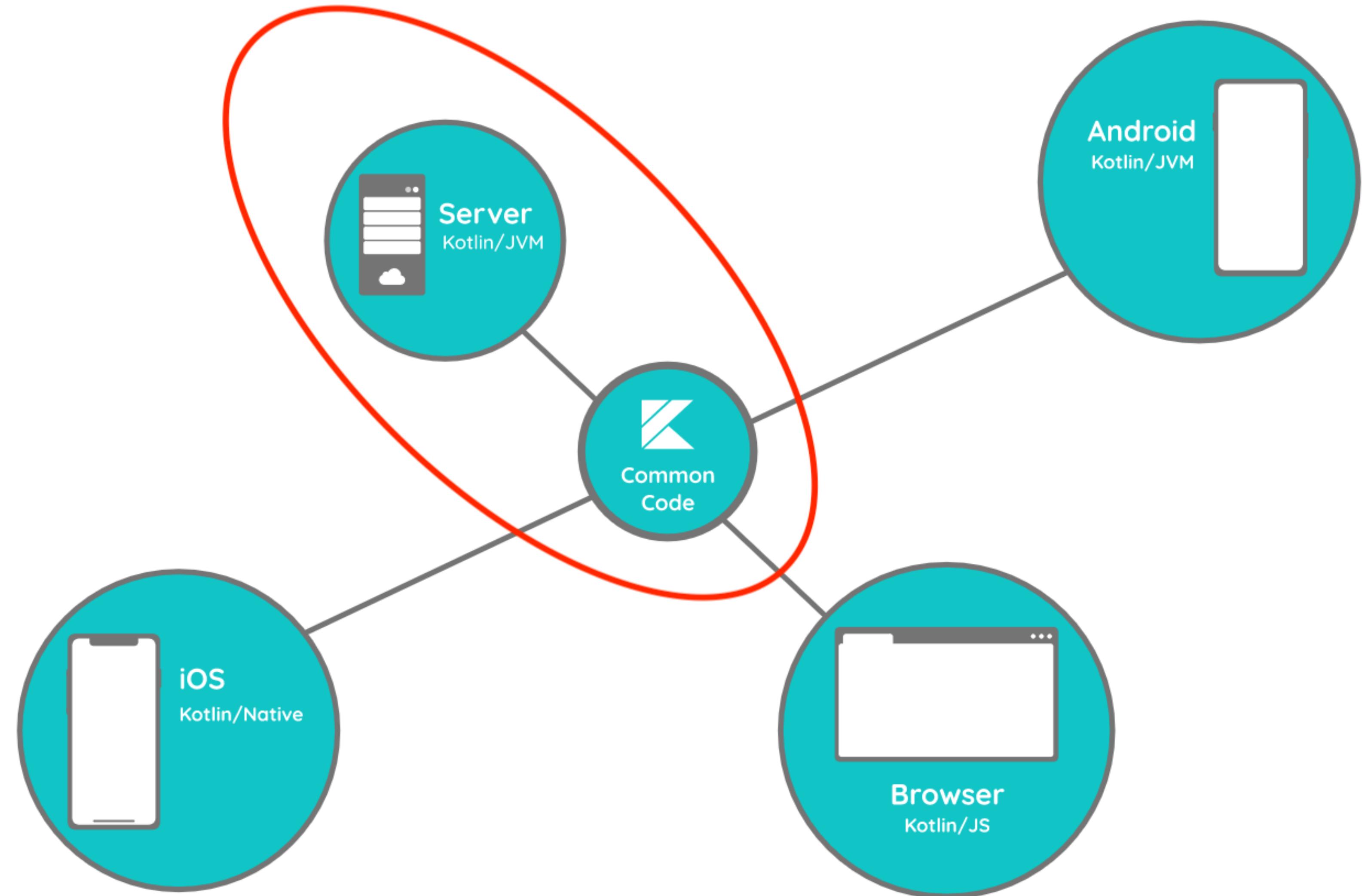


**Kotlin - это интерфейс**



# Что будем изучать мы?





# **JVM и Java - что это?**



# Java

- Многоплатформенный объектно-ориентированный язык
- Релиз в 1996 году
- Назван в честь марки кофе
- Разработан компанией Sun Microsystems



# Как работает Java?



# **Java - многоплатформенный язык**

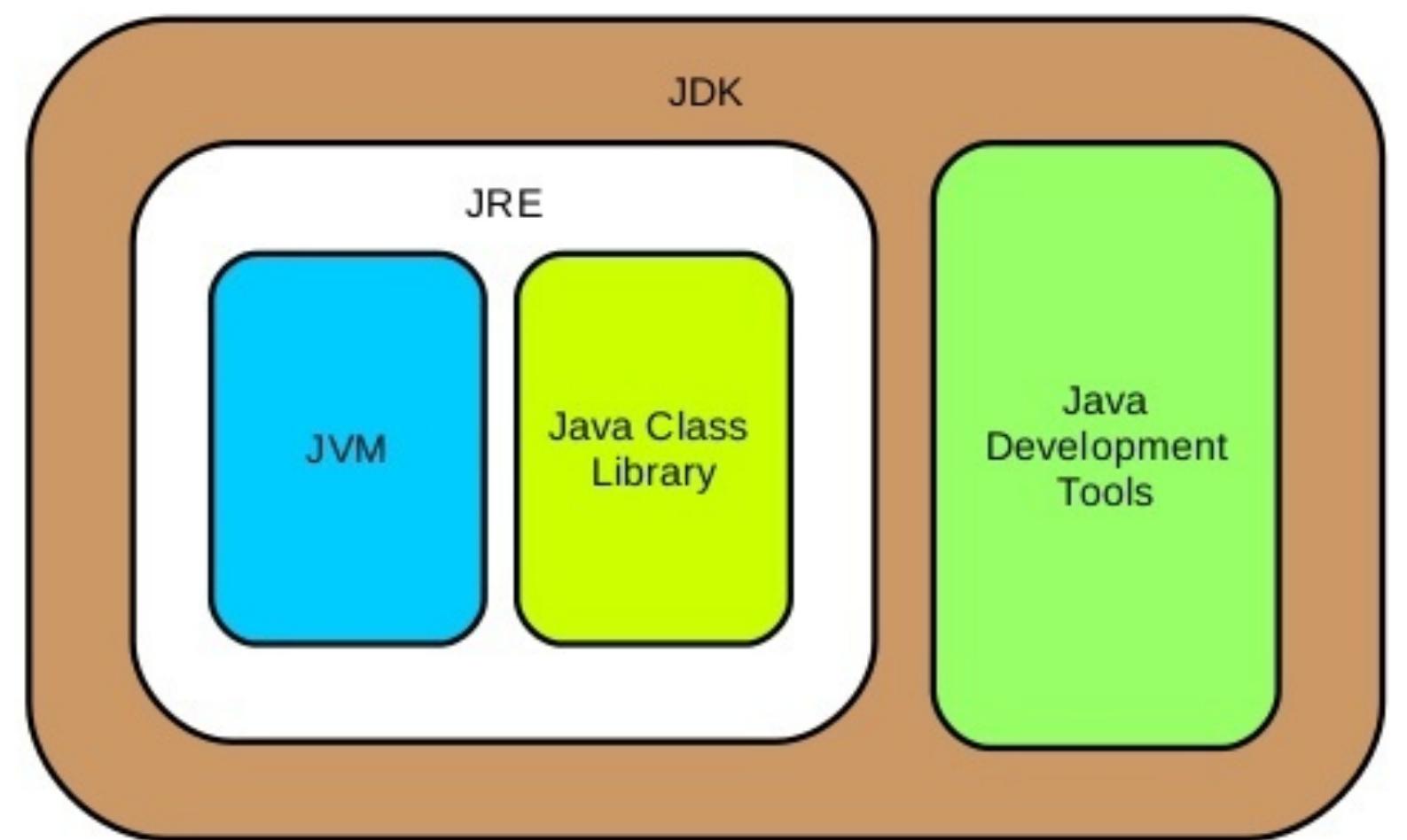


# Как такое возможно?

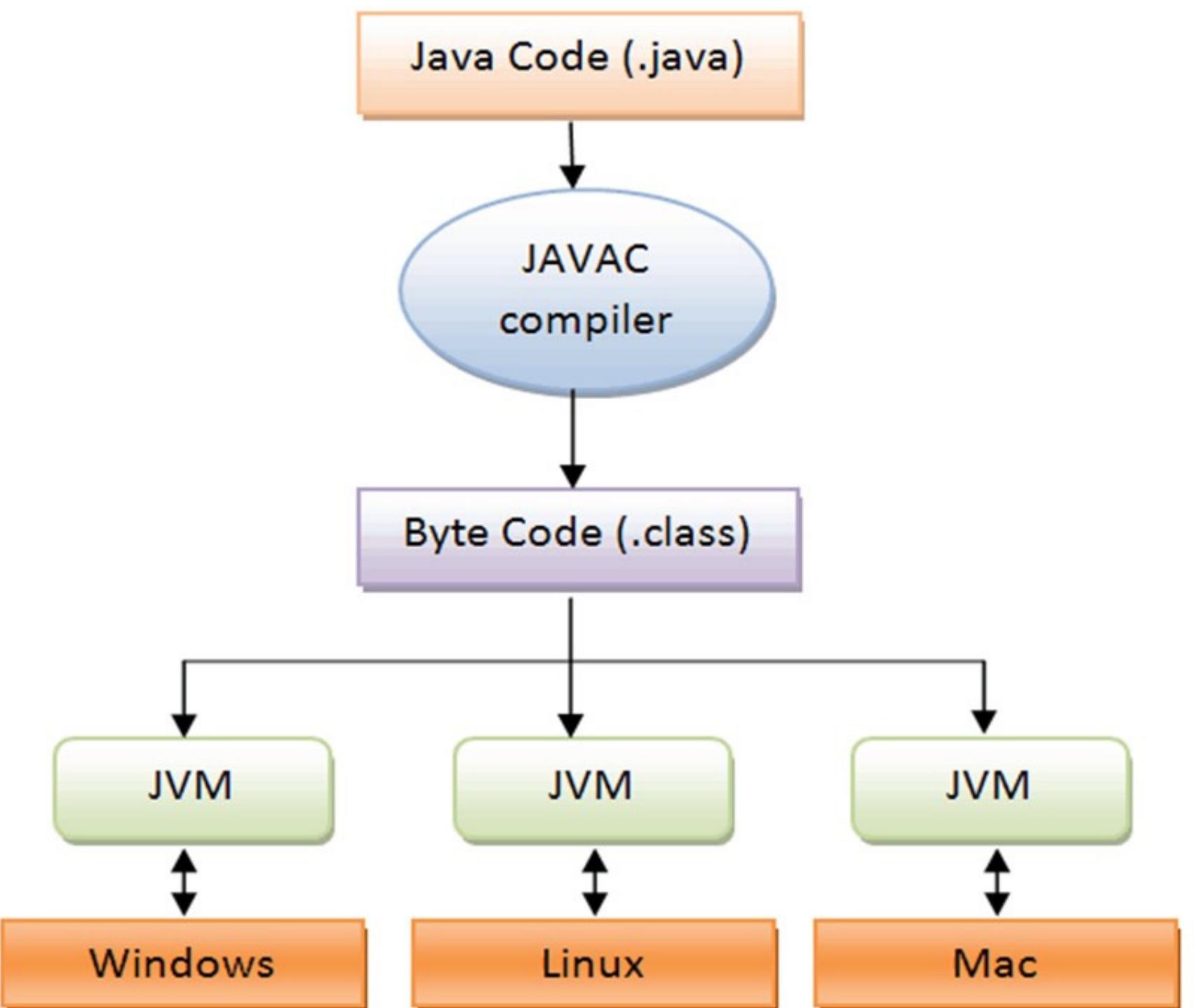


- Java находится внутри JDK
- JRE - Java runtime environment
- JVM - Java virtual machine
- Внутри JDK также есть множество dev tools

## Java Development Kit (JDK)



- Компилятор валидирует/оптимизирует код
- Компилятор преобразует исходный код в байт-код
- JVM интерпретирует байт-код в машинный код платформы
- Т.е. JVM исполняет код



**Внутри JVM исполняется ваш  
код**

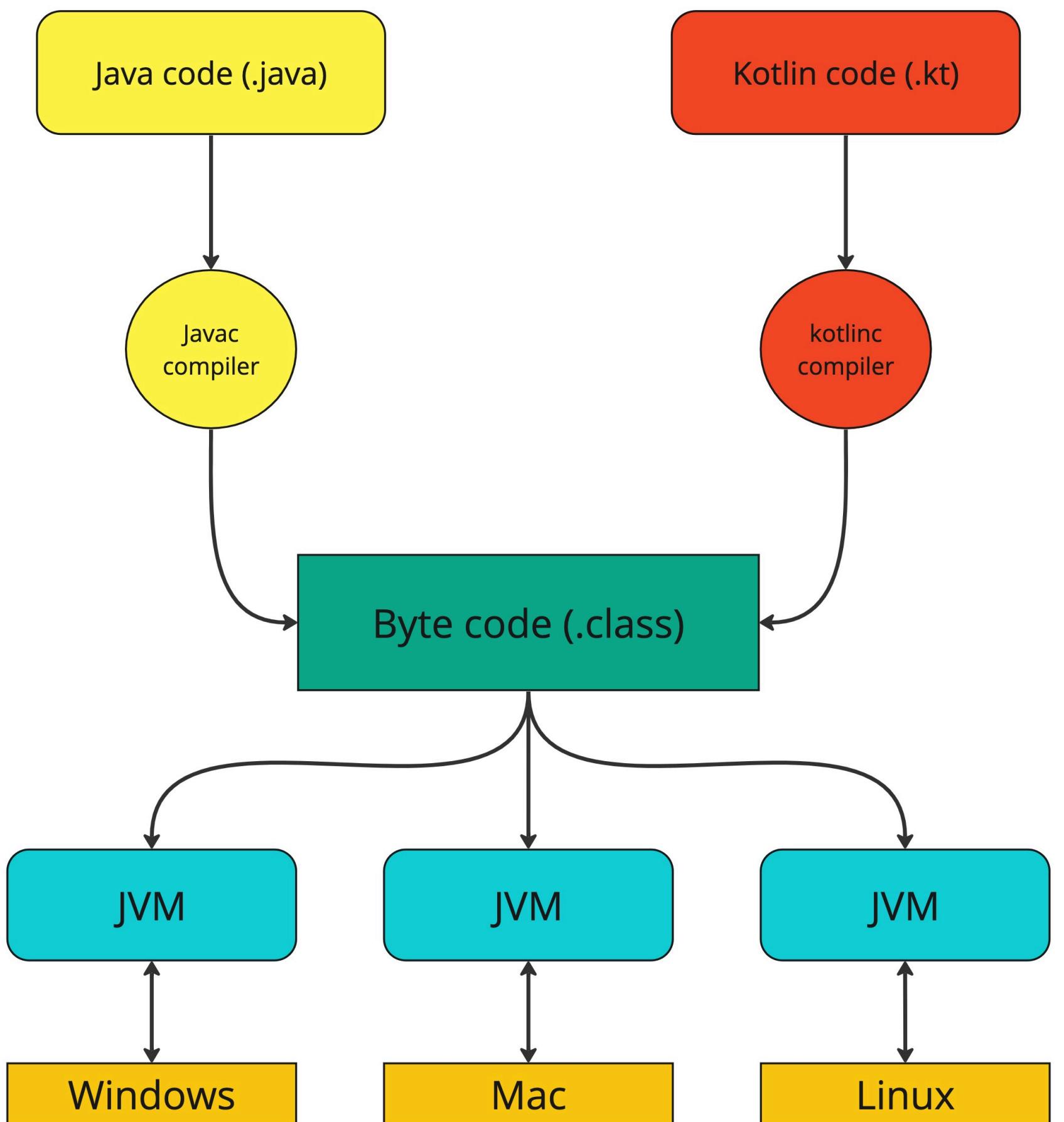


и Kotlin код тоже?



**Да!**





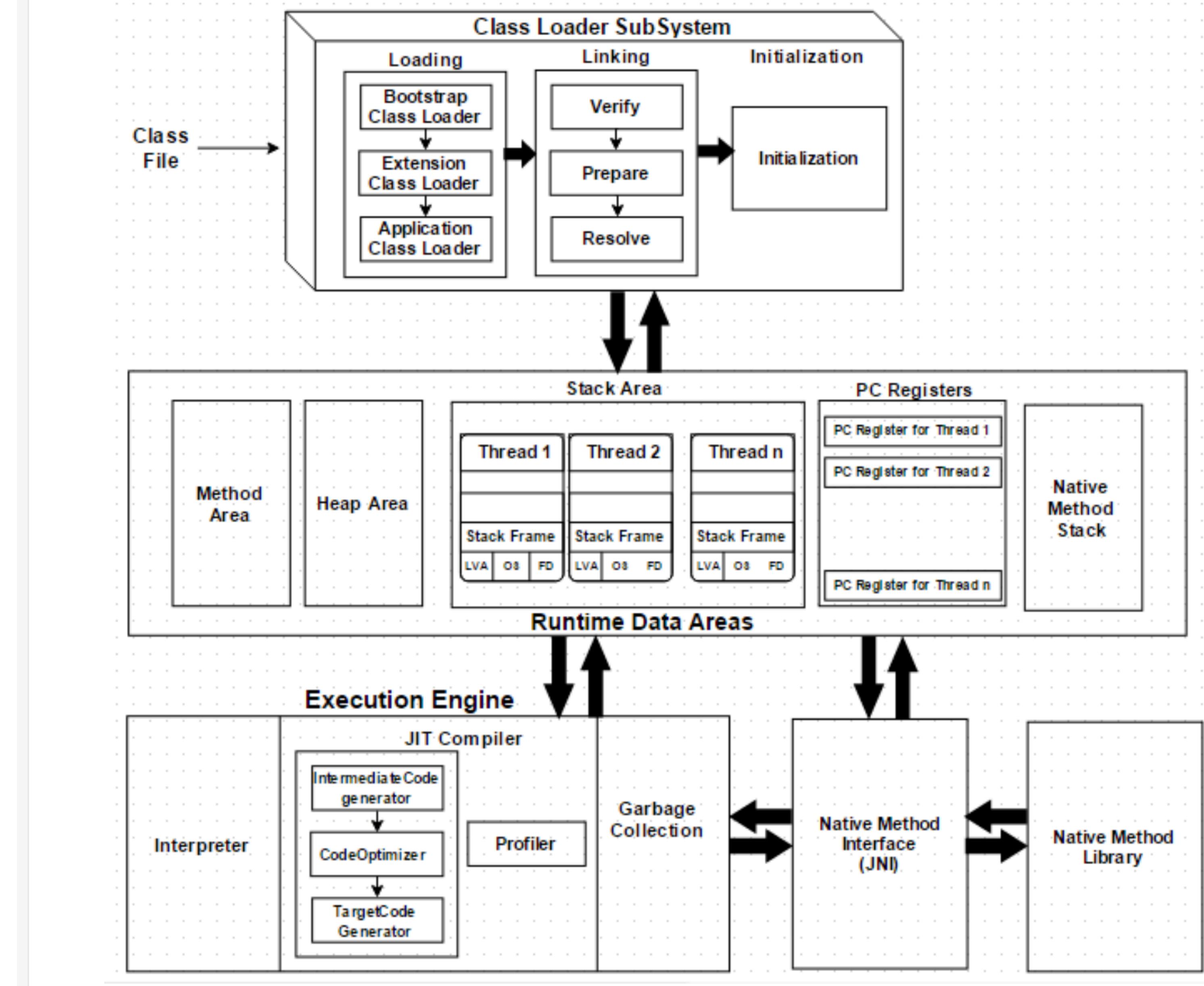
# Понимание работы JVM - успешная работа приложения



# **Из чего состоит JVM?**



# JVM Architecture Diagram





**Как-то сложновато...**



# **Мы начнем с простого**



**JVM - это в первую очередь  
программа**



# Задачи JVM

- Запускать программы компилируемые в Java byte-code
- Управлять и оптимизировать память, которую использует программа
- Предоставлять возможность гибко настраивать окружение
- Быть гибкой и расширяемой



# Что значит запускать программы?



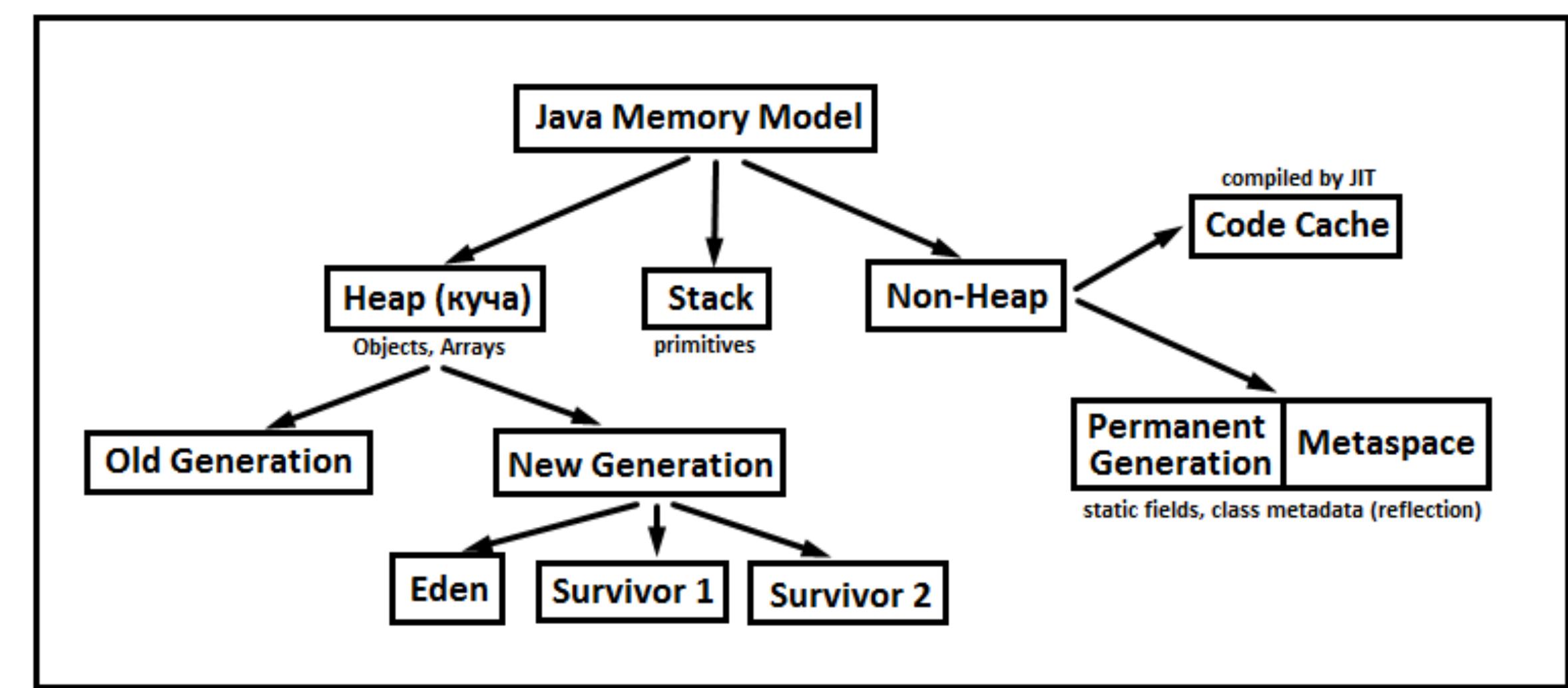
1. На вход поступают .class файлы
2. Classloader загружает эти классы в память
3. JVM выполняет код каждого класса
4. JVM предоставляет системный функционал



**Что значит управлять  
памятью?**



- Очистка неиспользуемой памяти
- Распределение и поддержание ссылочной структуры
- Оптимизация памяти
- Использование платформенного функционала



# **GC - garbage collector**



**GC - программа, работающая на виртуальной машине Java, которая избавляется от объектов, к которым невозможно получить доступ**



# Garbage collector

- Разные JVM могут иметь разные алгоритмы сборки мусора
- Также существуют разные GC
- Разные GC необходимы для разных ситуаций
- GC с помощью разных алгоритмов находит мусор и очищает его
- GC может работать как параллельно с основной программой, так и в режиме Stop the world



# Возможности конфигурации JVM



# JVM options

- Имеет более 600 различных опций JVM
- Некоторые JVM могут иметь свои собственные опции
- Можно настраивать размеры памяти, CPU, сбор дополнительных метрик
- Включение экспериментальных возможностей

HotSpot JVM options cheatsheet 

Available combinations of garbage collection algorithms in HotSpot JVM		
Young collector/Old collector/VM Flags	Serial (DefNew)	Serial Mark Sweep Compact -XX:+useSerialGC
	Parallel scavenger (PSYoungGen)	Serial Mark Sweep Compact (PSOldGen) -XX:+useParallelGC
	Parallel scavenger (PSYoungGen)	Parallel Mark Sweep Compact (ParOldGen) -XX:+useParallelOldGC
	Parallel (ParNew)	Serial Mark Sweep Compact -XX:+useParallelGC
	Serial (DefNew)	Concurrent Mark Sweep -XX:-useParNewGC -XX:+useConcMarkSweepGC
	Parallel (ParNew)	Concurrent Mark Sweep -XX:+useParNewGC -XX:+useConcMarkSweepGC
		Garbage First (G1) -XX:+useG1GC

All concrete numbers in JVM options in this card are for illustrational purposes only.

Java Process Memory

JVM Memory	Java Heap	Non-Heap
Young Gen	Eden	Metaspaces
Survivor 0	Survivor 1	Compressed Class Space
Old Gen	Code Cache	NIO Direct Buffers
	Thread Stacks	Native Memory (native libraries)

- Xmx / - Xmx -

Java 8 Alexey Ragozin -

GC log detail options

- verbose:gc or -XX:+PrintGC Print basic GC info
- XX:+PrintGCDetails Print more details GC info
- XX:+PrintGCTimestamps Print timestamps for each GC event (seconds count from start of JVM)
- XX:+PrintGCDateStamps Print date stamp at garbage collection events: 2011-09-08T14:20:29.517+0400 [
- XX:+PrintTenuringDistribution Print detailed demography of young space after each collection
- XX:+PrintTTLAB Print TLAB allocation statistics
- XX:+PrintTTLAB Print survivor TLAB details
- XX:+PrintOldPLAB Print old space PLAB details
- XX:+PrintGCTaskTimestamps Print timestamps for individual GC worker thread tasks (very verbose)
- XX:+PrintPromotionFailure Print additional information for promotion failure
- XX:+PrintHeapAtGC Print heap details on GC
- XX:+PrintHeapAtSIGBREAK Print heap details on signal
- XX:+PrintGCApplicationStoppedTime Print summary after each JVM safepoint (including non-GC)
- XX:+PrintGCApplicationConcurrentTime Print time for each concurrent phase of GC
- XX:+PrintClassHistogramsAfterFullGC Prints class histogram after full GC
- XX:+PrintClassHistogramsBeforeFullGC Prints class histogram before full GC

GC Log rotation

- Xloggc:<file> Redirects GC output to a file instead of console
- XX:+useGCLogFileRotation Enable GC log rotation
- XX:GCLogFileSize=512m Size threshold for GC log file
- XX:NumberOfGCFLogs=5 Number GC log files

More logging options

- XX:+NewRatio=3 Alternative way to specify size of young space. Sets ratio of young vs old space (-XX:NewSize=64m -XX:SurvivorRatio=3 means that each survivor space will be 64m and old space 192m).
- XX:SurvivorRatio=15 Sets size of single survivor space relative to Eden space size. Survivor space will be 15 times smaller than old space.
- XX:Metaspacesize=512m Initial and max size of VM's metaspaces
- XX:MaxMetaspacesize=1g Maximum metaspacesize (in bytes)
- XX:ThreadStackSize=16 Thread stack size
- XX:CompressedClassSpaceSize=256 Memory reserved for compressed class space (64bit only)
- XX:InitialCodeCacheSize=256 Initial size and max size of reserved code cache area
- XX:MaxDirectMemorySize=2g Maximum amount of memory available for NIO off-heap byte buffers

Memory sizing options

- Xms256m or -XX:InitialHeapSize=256m Initial size of JVM heap (young + old)
- Xmx2g or -XX:MaxHeapSize=2g Max size of JVM heap (young + old)
- XX:NewSize=64m Absolute (initial and max) size of young space (Eden + 2 survivors)
- XX:MaxNewSize=64m young space (Eden + 2 survivors)

Young space tenuring

- XX:InitialTenuringThreshold=8 Initial value for tenuring threshold (number of collections before object will be promoted to old space)
- XX:MaxTenuringThreshold=15 Max value for tenuring threshold
- XX:PretenureSizeThreshold=2m Max object size allowed to be allocated in young space (large objects will be allocated directly in old space). Thread local allocation bypasses this check, so if TLAB is large enough object exceeding size threshold still may be allocated in young space.
- XX:+AlwaysTenure Promote all objects surviving young collection immediately to tenured space (equivalent of -XX:MaxTenuringThreshold=0)
- XX:+NeverTenure Objects from young space will never get promoted to tenured space unless survivor space is not enough to keep them

Concurrent Mark Sweep (CMS)

CMS initiating options

- XX:+UseCMSInitiatingOccupancyOnly Only use predefined occupancy as only criterion for starting a CMS collection (disable adaptive behaviour)
- XX:CMSInitiatingOccupancyFraction=70 Percentage CMS generation occupancy to start a CMS cycle. A negative value means that CMSTriggerRatio is used.
- XX:CMSBootstrapOccupancy=50 Percentage CMS generation occupancy at which to initiate CMS collection for bootstrapping collection stats
- XX:CMSTriggerRatio=70 Percentage of MinHeapFreeRatio in CMS generation that is allocated before a CMS collection cycle commences.
- XX:CMSTriggerInterval=60000 Periodically triggers CMS collection. Useful for deterministic object finalization.
- XX:G1HeapWastePercent=10 If garbage level is below threshold, G1 will not attempt to reclaim memory further
- XX:G1ConfidencePercent=50 Confidence level for MMU/pause prediction
- XX:MaxGCPauseMillis=500 Target GC pause duration. G1 is not deterministic, so no guarantees for GC pause to satisfy this limit.

CMS diagnostic options

- XX:+PrintCMSStatistics Print additional CMS statistics. Very verbose if n=2.
- XX:+PrintCMSInitiationStatistics Print CMS initiation details
- XX:+PrintCMSDumpAtPromotionFailure Dump useful information about the state of the CMS old generation upon a promotion failure
- XX:+PrintCMSScavengeChunksInProgress (with option above) Add more detailed information about the free chunks
- XX:+PrintG1ObjectsInFlight (with option above) Add more detailed information about the allocated objects
- XX:+PrintG1SamplingRatio Start sampling Eden top at least before young generation occupancy reaches 1/ of the size at which we plan to schedule remark

CMS Stop-the-World pause tuning

- XX:CMSInitTime=30000 Once CMS collection is triggered, it will wait for next young collection to perform initial mark right after. This parameter specifies how long CMS can wait for young collection
- XX:+CMSScavengeBeforeRemark Force young collection before remark phase
- XX:+CMSScheduleRemarkDedicatedThreshold If Eden used is below this value, don't try to schedule remark
- XX:CMSScheduleRemarkEdenPenetration=20 Eden occupancy % at which to trigger schedule remark
- XX:+CMScheduleRemarkSamplingRatio=4 Start sampling Eden top at least before young generation occupancy reaches 1/ of the size at which we plan to schedule remark

CMS Concurrency options

- XX:+CMSConcurrentLockingEnabled If not enabled, CMS will not clean permanent space. You may need to enable it for containers such as JEE or OSGi.
- XX:+ExplicitGCInvokesConcurrent Let C trigger concurrent collection instead of full GC
- XX:+ExplicitGCInvokesConcurrentAndUnloadsClasses Same as above but also triggers permanent space collection.
- XX:SoftRefLRUPolicyPerMRB=1000 Factor for calculating soft reference TTL based on free heap size
- XX:OnOutOfMemoryError= Command to be executed in case of out of memory. "kill -9 \$pid" on Unix or "taskkill /F /pid \$pid" on Windows
- XX:+CMSConcurrentMTEnabled Use multiple threads for concurrent phases.

CMS Options for "incremental" CMS, they disable some heuristics and require careful validation



# Гибкость и расширяемость JVM



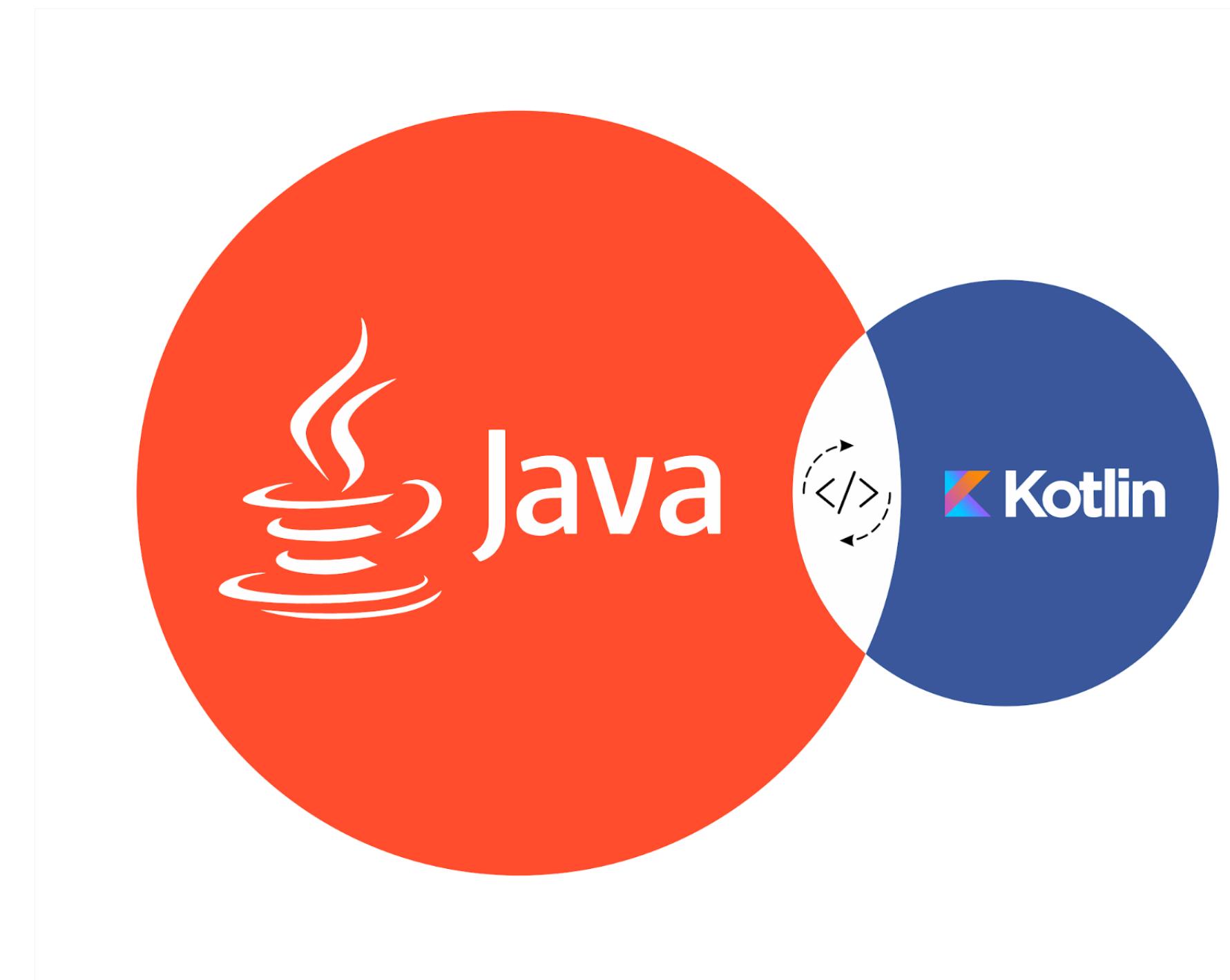
- JVM - просто спецификация
- Самая популярная реализации HotSpot в OpenSource
- Практически неограниченно портируемая
- Гибкая и продуманная архитектура



# Выводы



- Kotlin создан разработчиками для разработчиков
- Kotlin может компилироваться в разные платформы
- Kotlin JVM призван избавиться от старой и не удобной Java
- Kotlin JVM так же надежен и стабилен как и Java
- Переиспользуется опыт Java



Что есть еще интересного у  
Kotlin?



# Kotlin for DataScience!



- Kotlin Notebook
- Jupyter Kotlin kernel
- Kotlin Notebooks in Datalore
- Zeppelin Kotlin interpreter

The screenshot shows the IntelliJ IDEA IDE interface with a Kotlin Notebook session open. The code editor displays the following code:

```

plot(dataset) { this: DataFramePlotContext<*>
    points { this: PointsContext |
        // maps values from "time, ms" to X
        x(timeMs)
        // maps values from "relativeHumidity" to Y
        y(humidity)
    }
}

```

The output panel shows the execution results and a scatter plot. The plot has 'time, ms' on the x-axis (ranging from 50 to 250) and 'relativeHumidity' on the y-axis (ranging from 0.2 to 0.8). The data points are:

time, ms	relativeHumidity
87	0.300
100	0.450
150	0.200
200	0.220
250	0.800
250	0.350



# KotlinDL!



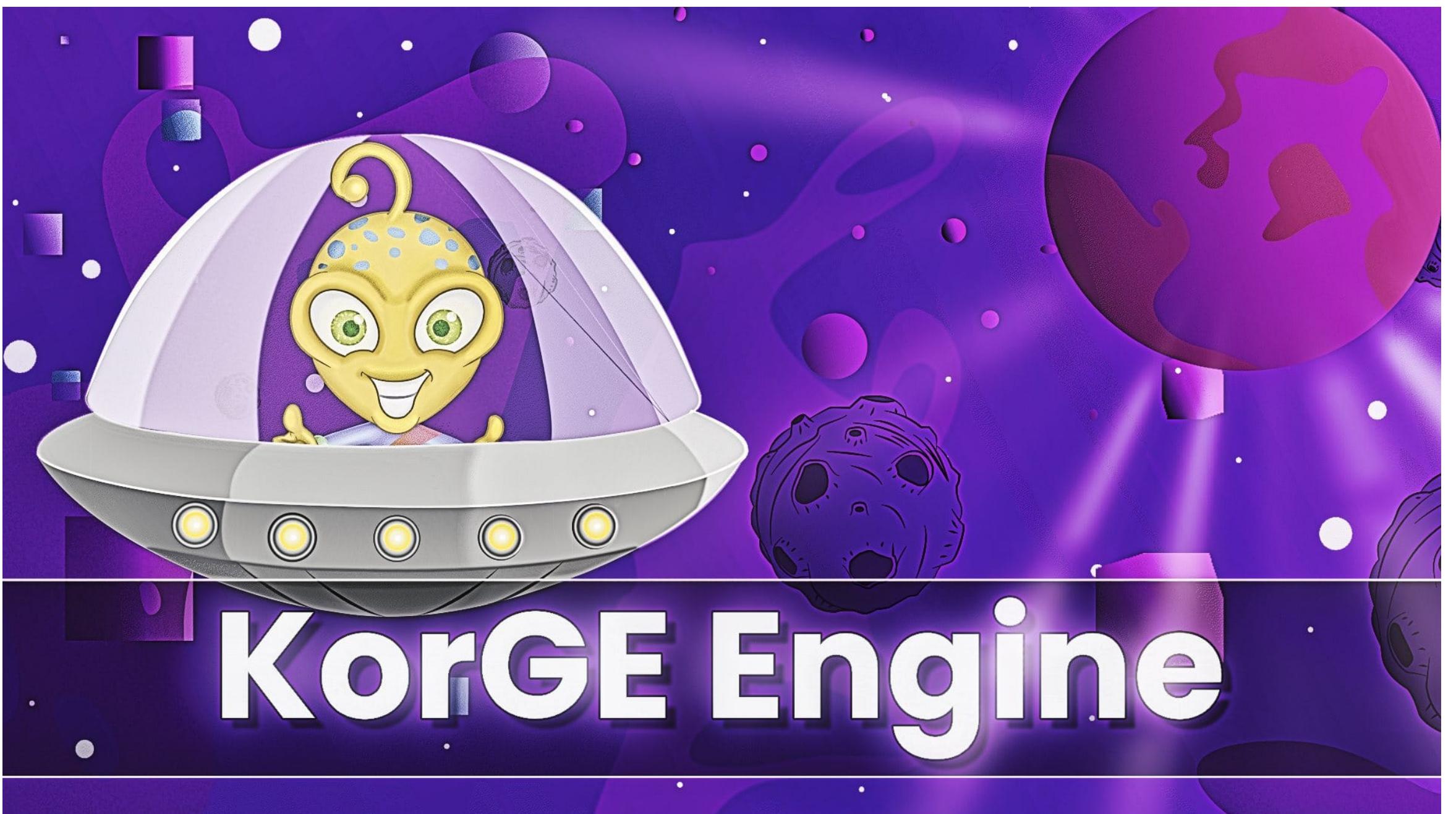
- Активно развивающиеся направление
- Имеются production ready библиотеки
- Лаконичный и безопасный код



# GameDev with Kotlin!



- Благодаря Kotlin Native можно использовать OpenGL
- Есть движок для 2D игр KorGE
- Возможность использования огромного количества Java библиотек



# Материалы



- Официальная документация по JVM от Oracle: <https://clck.ru/35PLSR>
- Документация по Java: <https://clck.ru/35PLTh>
- Официальный сайт Kotlin: <https://kotlinlang.org/>



# Вопросы?



**Поговорим об интересном**

**для чего Java?**

**Конечно для Minecraft**

**А если серьезно?**

**Write once,  
Run everywhere**

**Спасибо за уделенное время!**

