Writing Your First Kotlin Compiler Plugin

Kevin Most

A brief intro

Are these basically annotation processors?

- Annotation Processors:
  - **Your code runs at compile-time**
  - Public, documented API
  - Emit Java source code
  - Works on Kotlin/Java source code
  - Multiplatform not supported
- Compiler Plugins:
  - **Your code runs at compile-time**
  - Private, undocumented API
  - Emit Java bytecode (or LLVM IR)
  - Works on Kotlin source code only
  - Multiplatform supported

Why write compiler plugins?

- Incredibly powerful API; you can modify function/class internals
- Enables you to solve new classes of metaprogramming problems
- Annotation processors are JVM-only, while compiler plugins aren't

Why NOT write compiler plugins?

- Annotation processors are much easier to write (if you only care about JVM)
- Compiler plugins are a lot of work. You need to write:
  - An IntelliJ plugin (if creating synthetic members)
  - A Gradle (or Maven, or other build tool) plugin
  - Slightly different extensions for JVM, JS, and Native targets

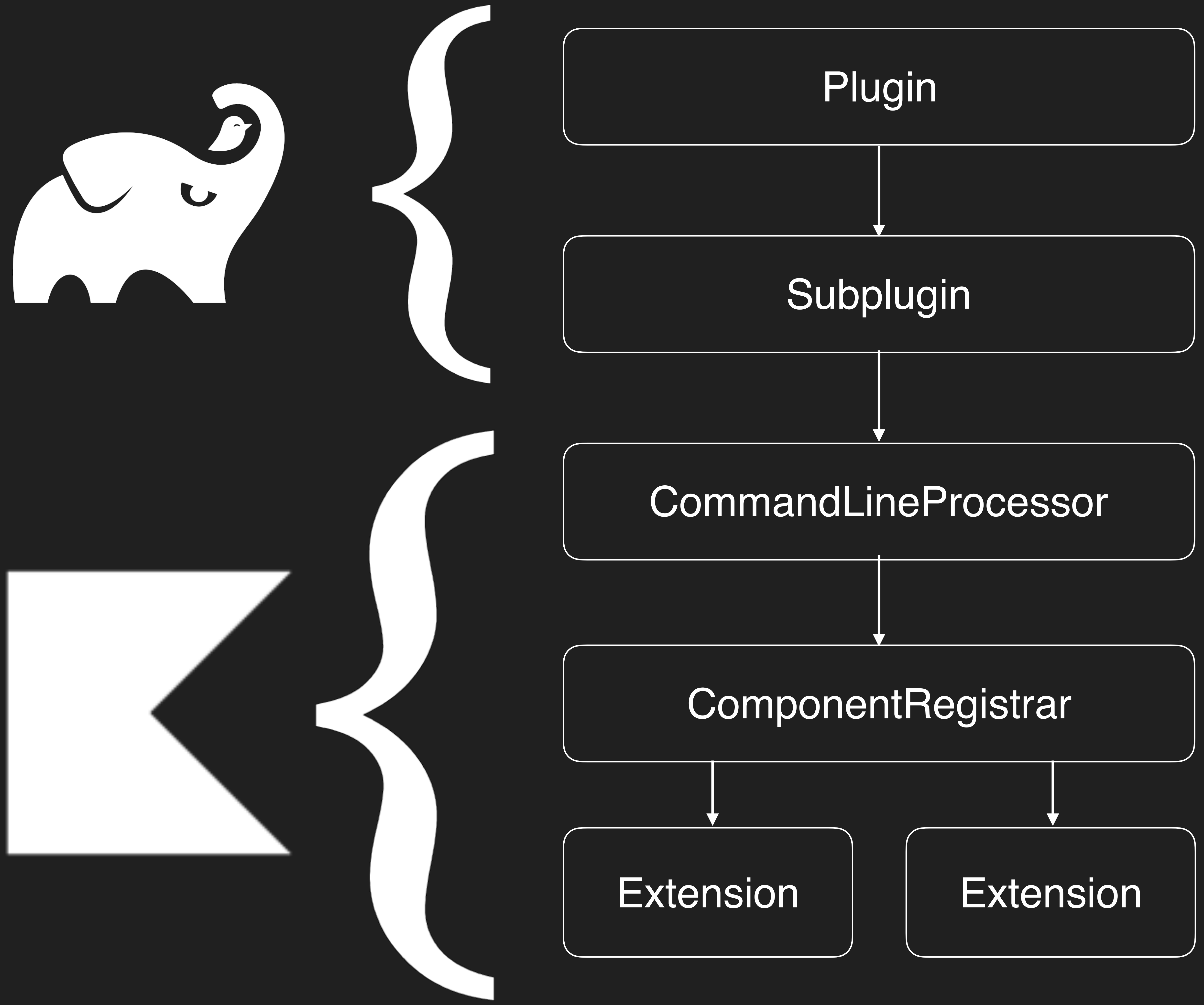
Examples of compiler plugins

- allopen: Modifies annotated class to be open
- noarg: Modifies annotated class to have a zero-argument constructor
- android-extensions: findViewById(R.id.foo) aliased, and automatic Parcelable impl generation via @Parcelize
- kotlin-serialization: Automatic generation of Serializable impl
  - First multiplatform-ready plugin (generates LLVM IR for native too)

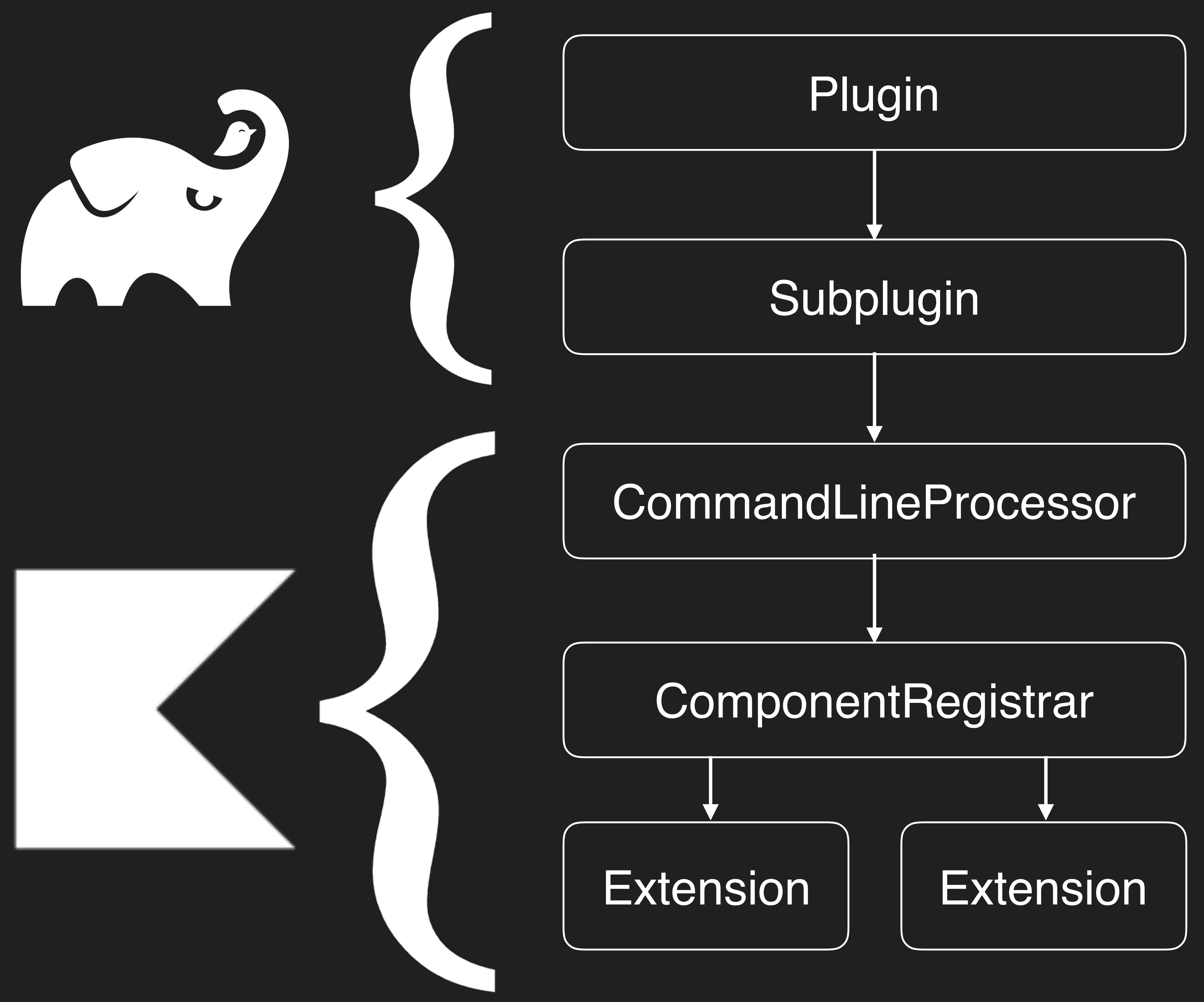
Examples of compiler plugins

- All existing compiler plugins are 1st party (github.com/JetBrains/kotlin)
- plugins/{name}/... for the actual plugin business logic
- libraries/tools/kotlin-{name}/... for the Gradle wrappers
- libraries/tools/kotlin-maven-{name}/... for the Maven wrappers

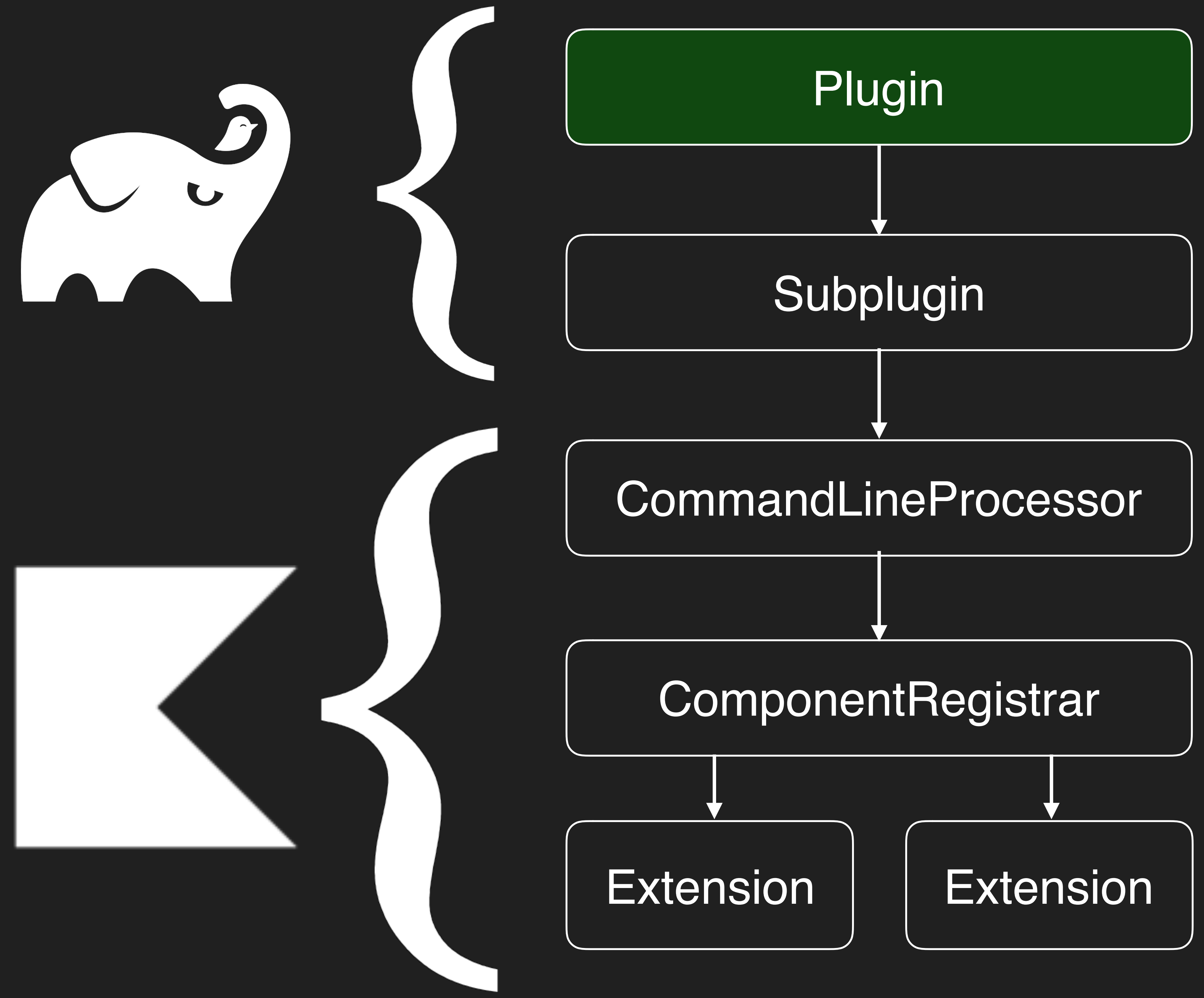
Plugin Architecture



Plugin Architecture

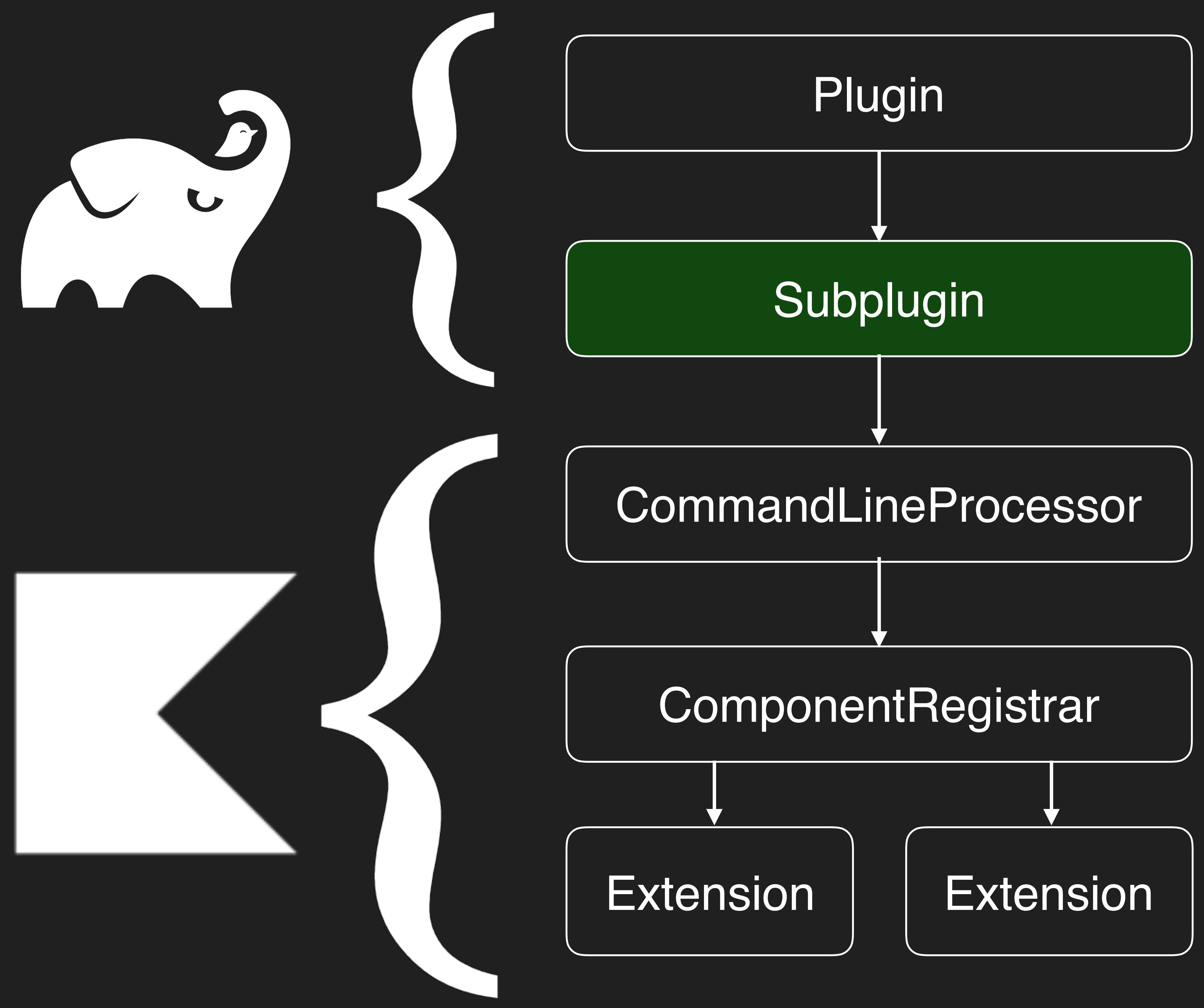


Plugin Architecture



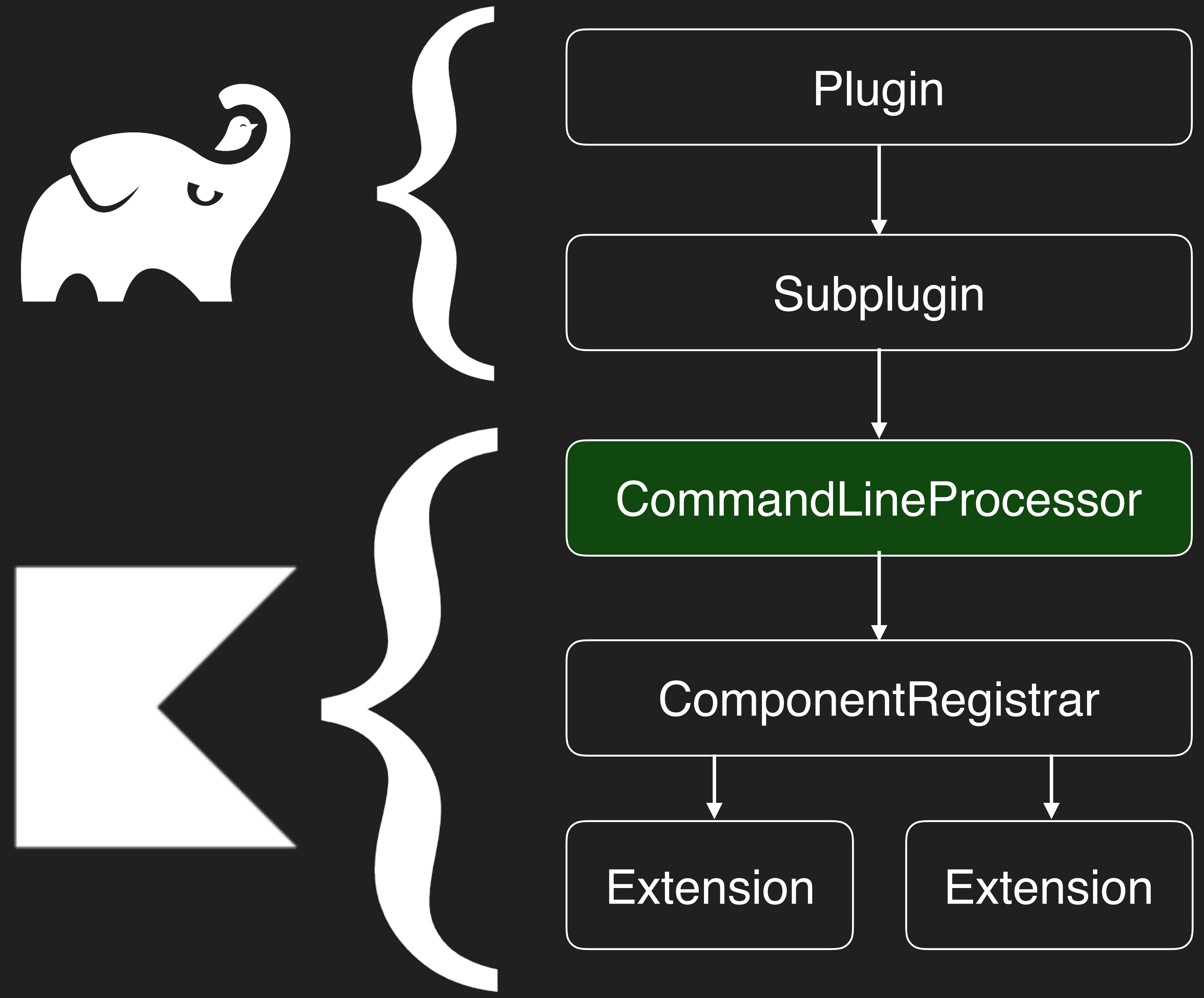
- Gradle API (totally unrelated to Kotlin)
- Provides an entry point from a build.gradle script
- Allows configuration via Gradle extensions

Plugin Architecture



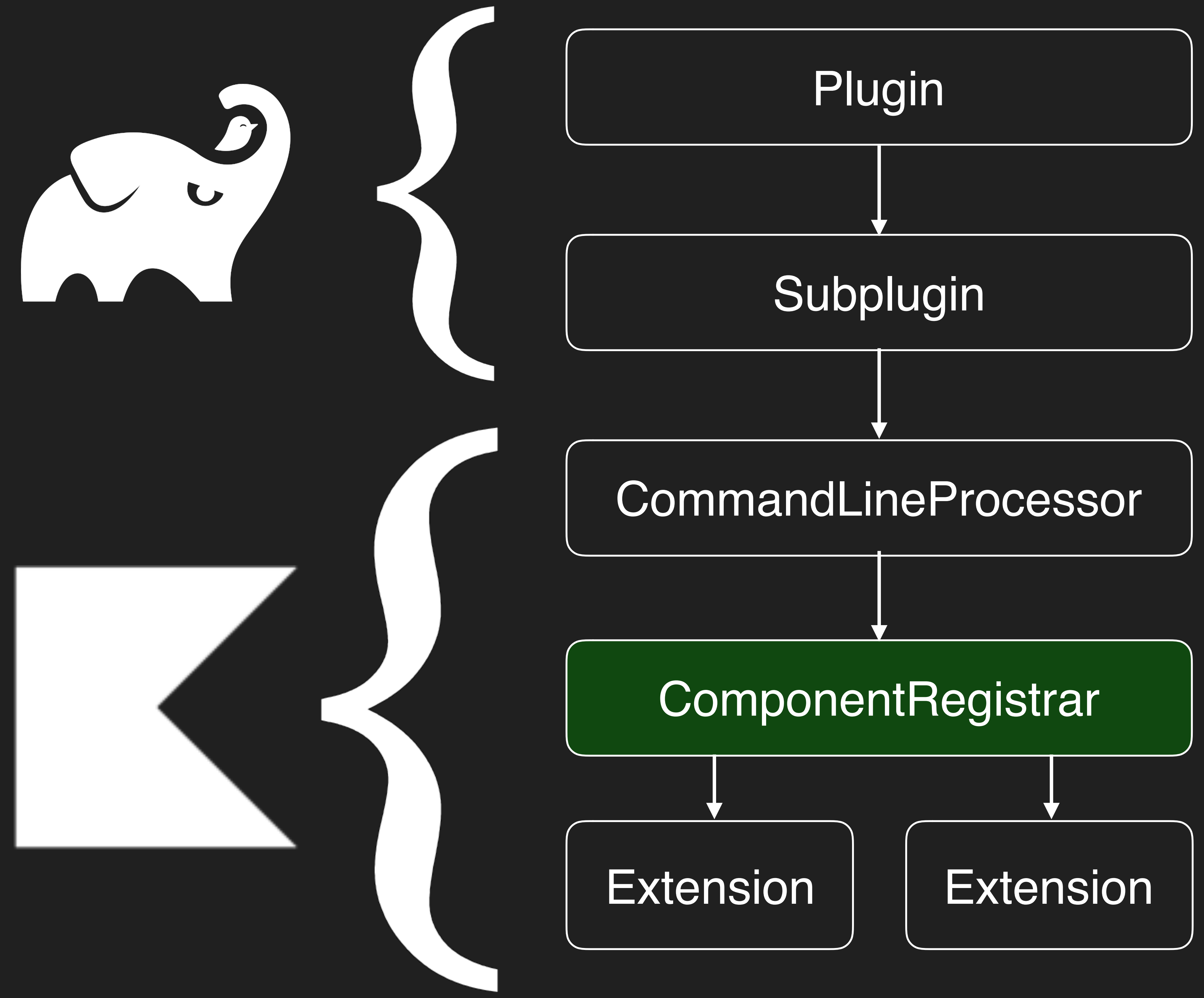
- The interface between the Gradle and Kotlin APIs
- Read Gradle extension options
- Write out Kotlin SubpluginOptions
- Define the compiler plugin's ID (internal unique key)
- Define the Kotlin plugin's Maven coordinates so the compiler can download it

Plugin Architecture



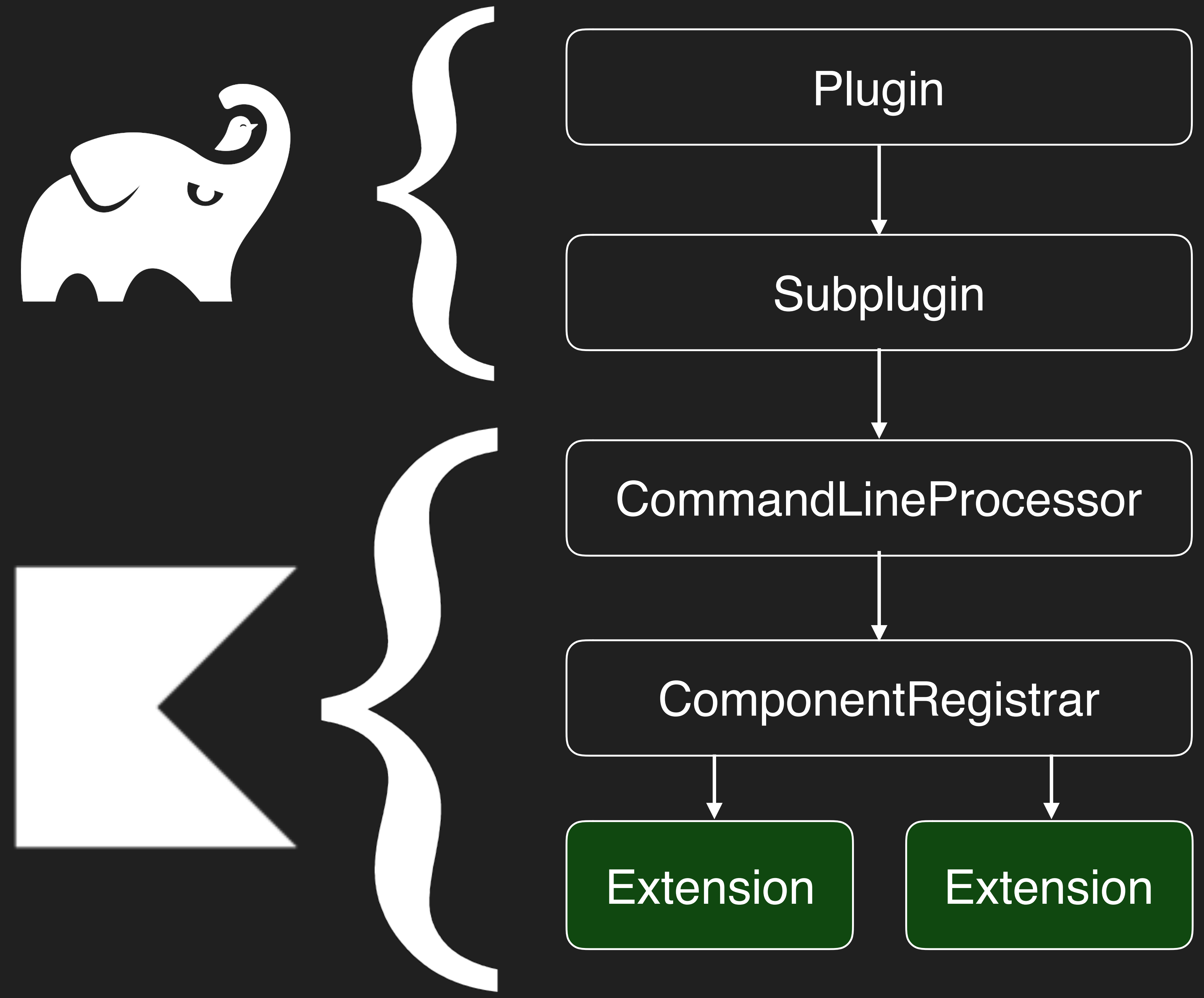
- Reads kotlinc -Xplugin args
- Subplugin options actually get passed through this pipeline
- Write CompilerConfigurationKeys

Plugin Architecture



- Read CompilerConfigurationKeys
- Register Extensions

Plugin Architecture



- Generates code (finally!)
- Multiple types of extensions, such as:
  - ExpressionCodegenExtension
  - ClassBuilderInterceptorExtension
  - StorageComponentContainerContributor
  - IrGenerationExtension (!!)
- Write bytecode (or LLVM IR!!)

Let's build our own!

Let's build our own!

- We'll build a compiler plugin that traces method calls
- A method annotated with a debug-log annotation will have its method body modified to include logging
- Could not be an annotation processor; modifies the function body
- Prior art:
  - Hugo: github.com/jakewharton/hugo
  - Firebase Performance Monitoring: firebase.google.com/docs/perf-mon
  - Both use AspectJ bytecode weaving + Android Gradle Transform API

The goal

```
fun prime(n: Int): Long {
  println("⇢ prime(n=$n)")
  val startTime = System.currentTimeMillis()
  val result = primeNumberSequence.take(n).last()
  val timeToRun = System.currentTimeMillis() - startTime
  println("⇠ prime [ran in $timeToRun ms]")
  return result
}
```

The goal

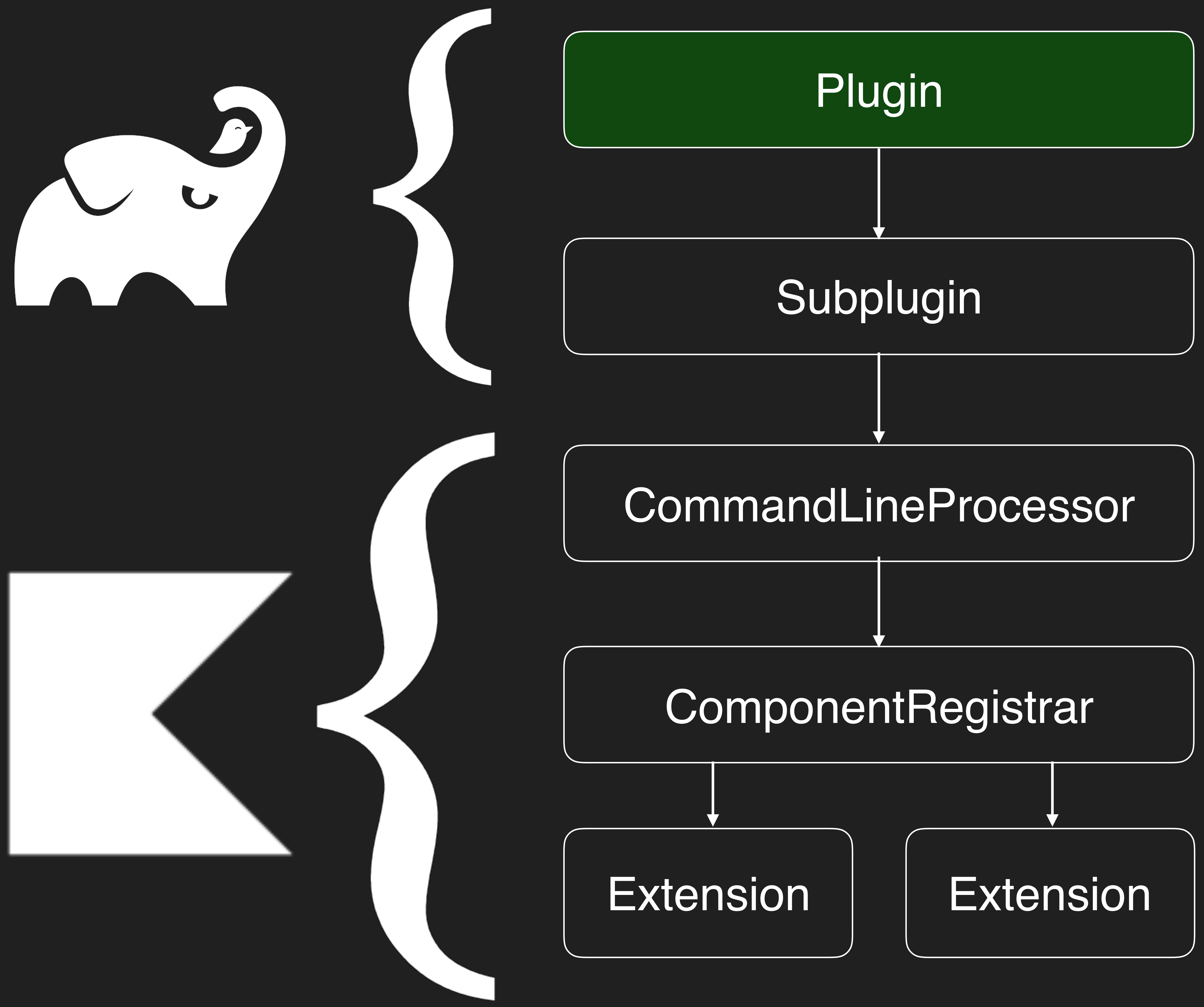
```
fun prime(n: Int): Long {
  println("⇢ prime(n=$n)")
  val startTime = System.currentTimeMillis()
  val result = primeNumberSequence.take(n).last()
  val timeToRun = System.currentTimeMillis() - startTime
  println("⇠ prime [ran in $timeToRun ms]")
  return result
}
```

The goal

```
fun prime(n: Int): Long {
  println("⇢ prime(n=$n)")
  val startTime = System.currentTimeMillis()
  val result = primeNumberSequence.take(n).last()
  val timeToRun = System.currentTimeMillis() - startTime
  println("⇠ prime [ran in $timeToRun ms]")
  return result
}
```

```
@DebugLog fun prime(n: Int): Long = primeNumberSequence.take(n).last()
```

Let's build our own!


Plugin
Subplugin
CommandLineProcessor
ComponentRegistrar
Extension
Extension

gradle-plugin/build.gradle

```
apply plugin: "java-gradle-plugin"
apply plugin: "org.jetbrains.kotlin.jvm"
apply plugin: "kotlin-kapt"

gradlePlugin {
  plugins {
    simplePlugin {
      id = "debuglog.plugin"
      implementationClass = "debuglog.DebugLogGradlePlugin"
    }
  }
}

dependencies {
  implementation "org.jetbrains.kotlin:kotlin-stdlib:$ktVersion"
  implementation "org.jetbrains.kotlin:kotlin-gradle-plugin-api:$ktVersion"

  compileOnly "com.google.auto.service:auto-service:1.0-rc4"
  kapt "com.google.auto.service:auto-service:1.0-rc4"
}
```

gradle-plugin/build.gradle

```
apply plugin: "java-gradle-plugin"
apply plugin: "org.jetbrains.kotlin.jvm"
apply plugin: "kotlin-kapt"

gradlePlugin {
  plugins {
    simplePlugin {
      id = "debuglog.plugin"
      implementationClass = "debuglog.DebugLogGradlePlugin"
    }
  }
}

dependencies {
  implementation "org.jetbrains.kotlin:kotlin-stdlib:$ktVersion"
  implementation "org.jetbrains.kotlin:kotlin-gradle-plugin-api:$ktVersion"

  compileOnly "com.google.auto.service:auto-service:1.0-rc4"
  kapt "com.google.auto.service:auto-service:1.0-rc4"
}
```

gradle-plugin/build.gradle

```
apply plugin: "java-gradle-plugin"
apply plugin: "org.jetbrains.kotlin.jvm"
apply plugin: "kotlin-kapt"

gradlePlugin {
  plugins {
    simplePlugin {
      id = "debuglog.plugin" // `apply plugin: "debuglog.plugin"`
      implementationClass = "debuglog.DebugLogGradlePlugin"
    }
  }
}

dependencies {
  implementation "org.jetbrains.kotlin:kotlin-stdlib:$ktVersion"
  implementation "org.jetbrains.kotlin:kotlin-gradle-plugin-api:$ktVersion"

  compileOnly "com.google.auto.service:auto-service:1.0-rc4"
  kapt "com.google.auto.service:auto-service:1.0-rc4"
}
```

gradle-plugin/build.gradle

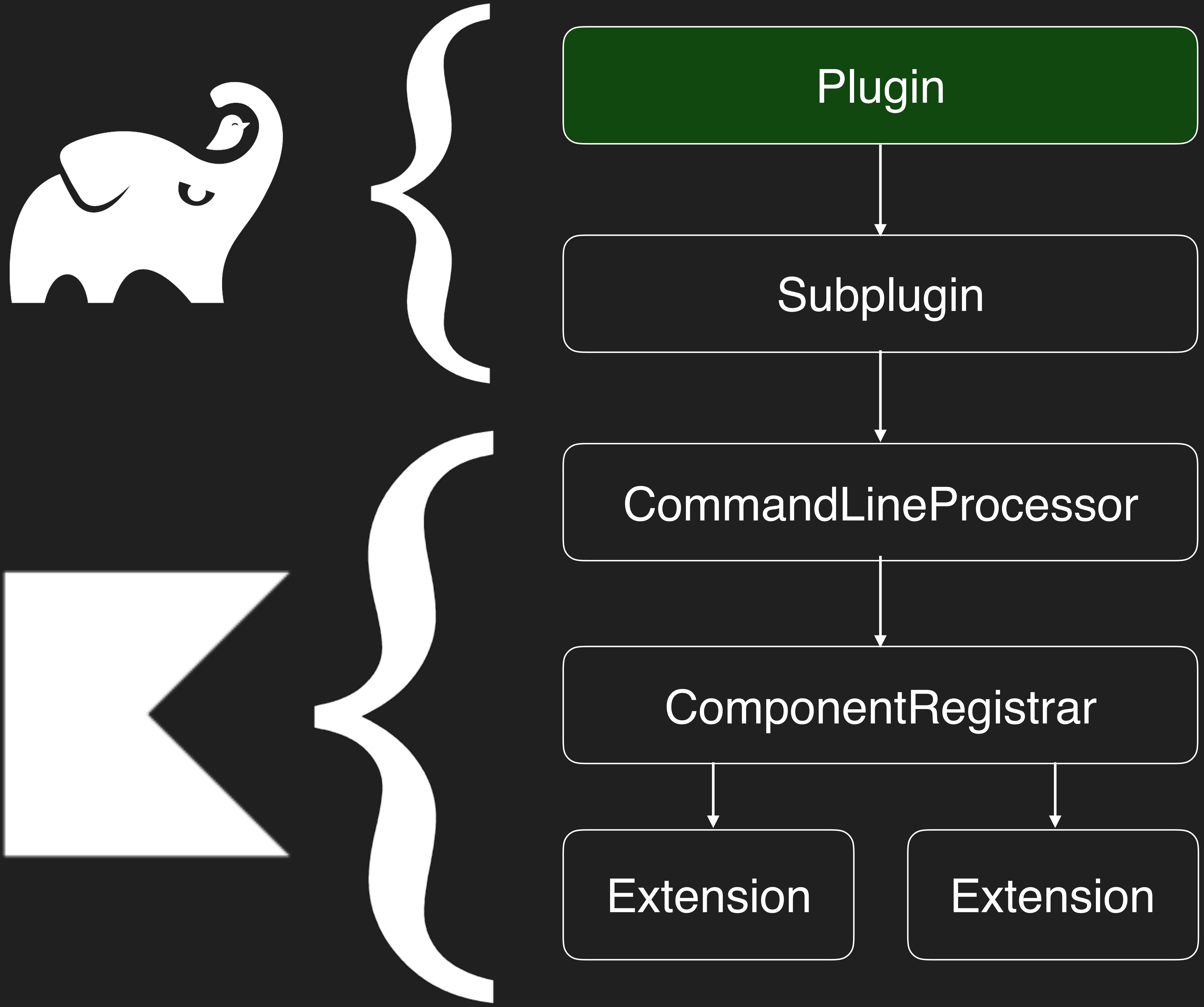
```
apply plugin: "java-gradle-plugin"
apply plugin: "org.jetbrains.kotlin.jvm"
apply plugin: "kotlin-kapt"

gradlePlugin {
  plugins {
    simplePlugin {
      id = "debuglog.plugin" // `apply plugin: "debuglog.plugin"`
      implementationClass = "debuglog.DebugLogGradlePlugin" // entry-point class
    }
  }
}

dependencies {
  implementation "org.jetbrains.kotlin:kotlin-stdlib:$ktVersion"
  implementation "org.jetbrains.kotlin:kotlin-gradle-plugin-api:$ktVersion"

  compileOnly "com.google.auto.service:auto-service:1.0-rc4"
  kapt "com.google.auto.service:auto-service:1.0-rc4"
}
```

gradle-plugin/build.gradle

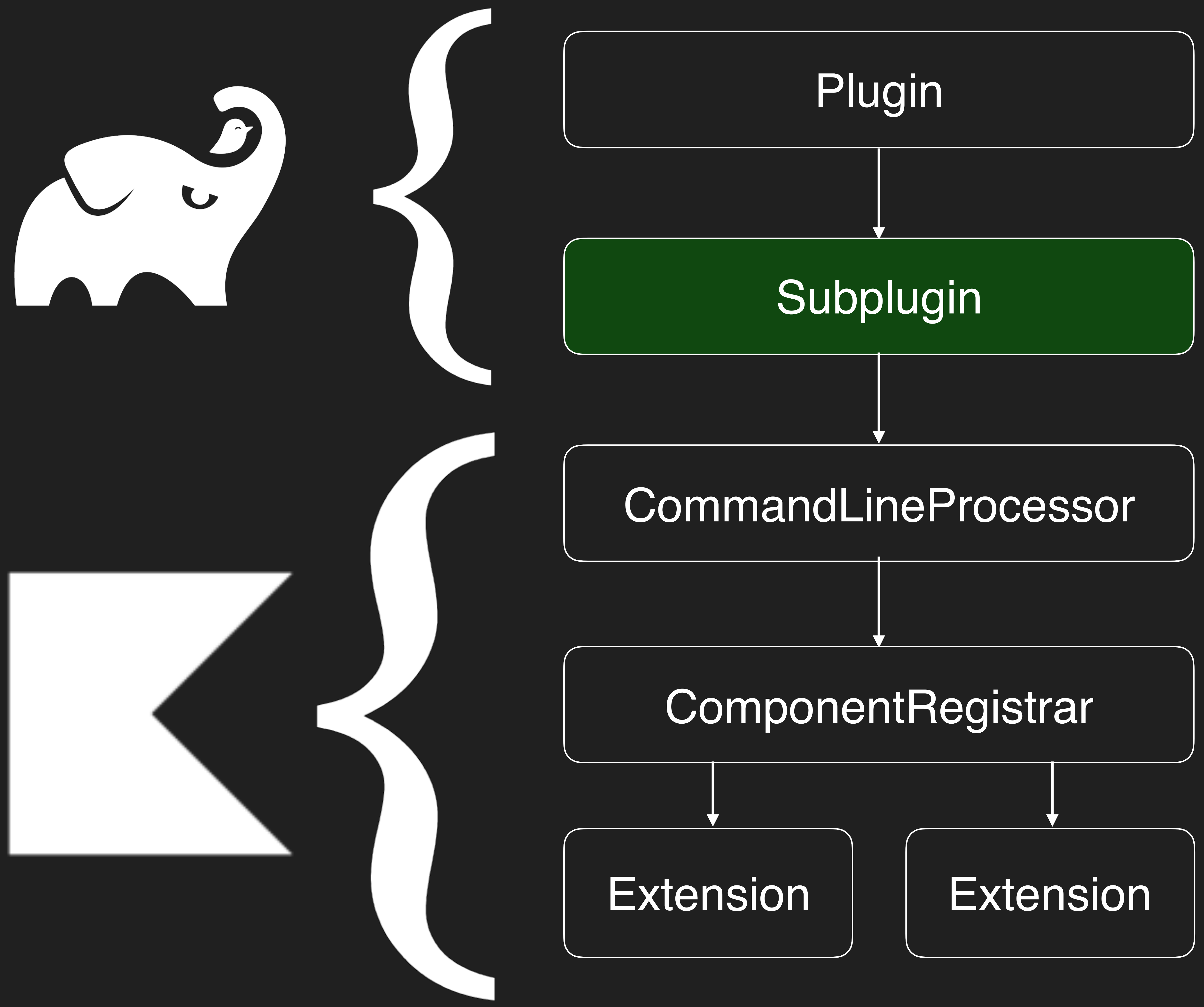
```
apply plugin: "java-gradle-plugin"
apply plugin: "org.jetbrains.kotlin.jvm"
apply plugin: "kotlin-kapt"

gradlePlugin {
  plugins {
    simplePlugin {
      id = "debuglog.plugin" // `apply plugin: "debuglog.plugin"`
      implementationClass = "debuglog.DebugLogGradlePlugin" // entry-point class
    }
  }
}

dependencies {
  implementation "org.jetbrains.kotlin:kotlin-stdlib:$ktVersion"
  implementation "org.jetbrains.kotlin:kotlin-gradle-plugin-api:$ktVersion"

  compileOnly "com.google.auto.service:auto-service:1.0-rc4"
  kapt "com.google.auto.service:auto-service:1.0-rc4"
}
```

gradle-plugin/build.gradle

```
apply plugin: "java-gradle-plugin"
apply plugin: "org.jetbrains.kotlin.jvm"
apply plugin: "kotlin-kapt"

gradlePlugin {
  plugins {
    simplePlugin {
      id = "debuglog.plugin" // `apply plugin: "debuglog.plugin"`
      implementationClass = "debuglog.DebugLogGradlePlugin" // entry-point class
    }
  }
}

dependencies {
  implementation "org.jetbrains.kotlin:kotlin-stdlib:$ktVersion"
  implementation "org.jetbrains.kotlin:kotlin-gradle-plugin-api:$ktVersion"

  compileOnly "com.google.auto.service:auto-service:1.0-rc4"
  kapt "com.google.auto.service:auto-service:1.0-rc4"
}
```


Plugin
Subplugin
CommandLineProcessor
ComponentRegistrar
Extension
Extension

```kotlin
class DebugLogGradlePlugin : org.gradle.api.Plugin<Project> {
  override fun apply(project: Project) {
    project.extensions.create(
        "debugLog",
        DebugLogGradleExtension::class.java
    )
  }
}

open class DebugLogGradleExtension {
  var enabled: Boolean = true
  var annotations: List<String> = emptyList()
}
```


Plugin
Subplugin
CommandLineProcessor
ComponentRegistrar
Extension
Extension

```kotlin
@AutoService(KotlinGradleSubplugin::class) // don't forget!
class DebugLogGradleSubplugin : KotlinGradleSubplugin<AbstractCompile> {

  override fun isApplicable(
      project: Project,
      task: AbstractCompile
  ): Boolean = TODO()

  override fun getCompilerPluginId(): String = TODO()

  override fun getPluginArtifact(): SubpluginArtifact = TODO()

  override fun apply(project: Project, /*...*/): List<SubpluginOption> {
    TODO()
  }
}
```

```kotlin
@AutoService(KotlinGradleSubplugin::class) // don't forget!
class DebugLogGradleSubplugin : KotlinGradleSubplugin<AbstractCompile> {

  override fun isApplicable(
      project: Project,
      task: AbstractCompile
  ): Boolean = TODO()

  override fun getCompilerPluginId(): String = TODO()

  override fun getPluginArtifact(): SubpluginArtifact = TODO()

  override fun apply(project: Project, /*...*/): List<SubpluginOption> {
    TODO()
  }
}
```

```kotlin
@AutoService(KotlinGradleSubplugin::class) // don't forget!
class DebugLogGradleSubplugin : KotlinGradleSubplugin<AbstractCompile> {

  override fun isApplicable(
      project: Project,
      task: AbstractCompile
  ): Boolean = project.plugins.hasPlugin(DebugLogGradlePlugin::class.java)

  override fun getCompilerPluginId(): String = TODO()

  override fun getPluginArtifact(): SubpluginArtifact = TODO()

  override fun apply(project: Project, /*...*/): List<SubpluginOption> {
    TODO()
  }
}
```

```kotlin
@AutoService(KotlinGradleSubplugin::class) // don't forget!
class DebugLogGradleSubplugin : KotlinGradleSubplugin<AbstractCompile> {

  override fun isApplicable(
      project: Project,
      task: AbstractCompile
  ): Boolean = project.plugins.hasPlugin(DebugLogGradlePlugin::class.java)

  override fun getCompilerPluginId(): String = TODO()

  override fun getPluginArtifact(): SubpluginArtifact = TODO()

  override fun apply(project: Project, /*...*/): List<SubpluginOption> {
    TODO()
  }
}
```

```kotlin
@AutoService(KotlinGradleSubplugin::class) // don't forget!
class DebugLogGradleSubplugin : KotlinGradleSubplugin<AbstractCompile> {

  override fun isApplicable(
      project: Project,
      task: AbstractCompile
  ): Boolean = project.plugins.hasPlugin(DebugLogGradlePlugin::class.java)

  override fun getCompilerPluginId(): String = TODO()

  override fun getPluginArtifact(): SubpluginArtifact = TODO()

  override fun apply(project: Project, /*...*/): List<SubpluginOption> {
    TODO()
  }
}
```

```kotlin
@AutoService(KotlinGradleSubplugin::class) // don't forget!
class DebugLogGradleSubplugin : KotlinGradleSubplugin<AbstractCompile> {

  override fun isApplicable(
      project: Project,
      task: AbstractCompile
  ): Boolean = project.plugins.hasPlugin(DebugLogGradlePlugin::class.java)

  override fun getCompilerPluginId(): String = "debuglog"

  override fun getPluginArtifact(): SubpluginArtifact = TODO()

  override fun apply(project: Project, /*...*/): List<SubpluginOption> {
    TODO()
  }
}
```

```kotlin
@AutoService(KotlinGradleSubplugin::class) // don't forget!
class DebugLogGradleSubplugin : KotlinGradleSubplugin<AbstractCompile> {

  override fun isApplicable(
      project: Project,
      task: AbstractCompile
  ): Boolean = project.plugins.hasPlugin(DebugLogGradlePlugin::class.java)

  override fun getCompilerPluginId(): String = "debuglog"

  override fun getPluginArtifact(): SubpluginArtifact = TODO()

  override fun apply(project: Project, /*...*/): List<SubpluginOption> {
    TODO()
  }
}
```

```kotlin
@AutoService(KotlinGradleSubplugin::class) // don't forget!
class DebugLogGradleSubplugin : KotlinGradleSubplugin<AbstractCompile> {

  override fun isApplicable(
      project: Project,
      task: AbstractCompile
  ): Boolean = project.plugins.hasPlugin(DebugLogGradlePlugin::class.java)

  override fun getCompilerPluginId(): String = "debuglog"

  override fun getPluginArtifact(): SubpluginArtifact = TODO()

  override fun apply(project: Project, /*...*/): List<SubpluginOption> {
    TODO()
  }
}
```

```kotlin
@AutoService(KotlinGradleSubplugin::class) // don't forget!
class DebugLogGradleSubplugin : KotlinGradleSubplugin<AbstractCompile> {

  override fun isApplicable(
      project: Project,
      task: AbstractCompile
  ): Boolean = project.plugins.hasPlugin(DebugLogGradlePlugin::class.java)

  override fun getCompilerPluginId(): String = "debuglog"

  override fun getPluginArtifact(): SubpluginArtifact = SubpluginArtifact(
      groupId = "debuglog", artifactId = "kotlin-plugin", version = "0.0.1"
  )

  override fun apply(project: Project, /*...*/): List<SubpluginOption> {
    TODO()
  }
}
```

```kotlin
@AutoService(KotlinGradleSubplugin::class) // don't forget!
class DebugLogGradleSubplugin : KotlinGradleSubplugin<AbstractCompile> {

  override fun isApplicable(
      project: Project,
      task: AbstractCompile
  ): Boolean = project.plugins.hasPlugin(DebugLogGradlePlugin::class.java)

  override fun getCompilerPluginId(): String = "debuglog"

  override fun getPluginArtifact(): SubpluginArtifact = SubpluginArtifact(
      groupId = "debuglog", artifactId = "kotlin-plugin", version = "0.0.1"
  )

  override fun apply(project: Project, /*...*/): List<SubpluginOption> {
    TODO()
  }
}
```

```kotlin
@AutoService(KotlinGradleSubplugin::class) // don't forget!
class DebugLogGradleSubplugin  :  KotlinGradleSubplugin<AbstractCompile>  {

  override fun isApplicable(
      project: Project,
      task: AbstractCompile
  ): Boolean = project.plugins.hasPlugin(DebugLogGradlePlugin::class.java)

  override fun getCompilerPluginId(): String = "debuglog"

  override fun getPluginArtifact(): SubpluginArtifact = SubpluginArtifact(
      groupId = "debuglog", artifactId = "kotlin-plugin", version = "0.0.1"
  )

  override fun apply(project: Project, /*...*/): List<SubpluginOption>  {
    TODO()
  }
}
```

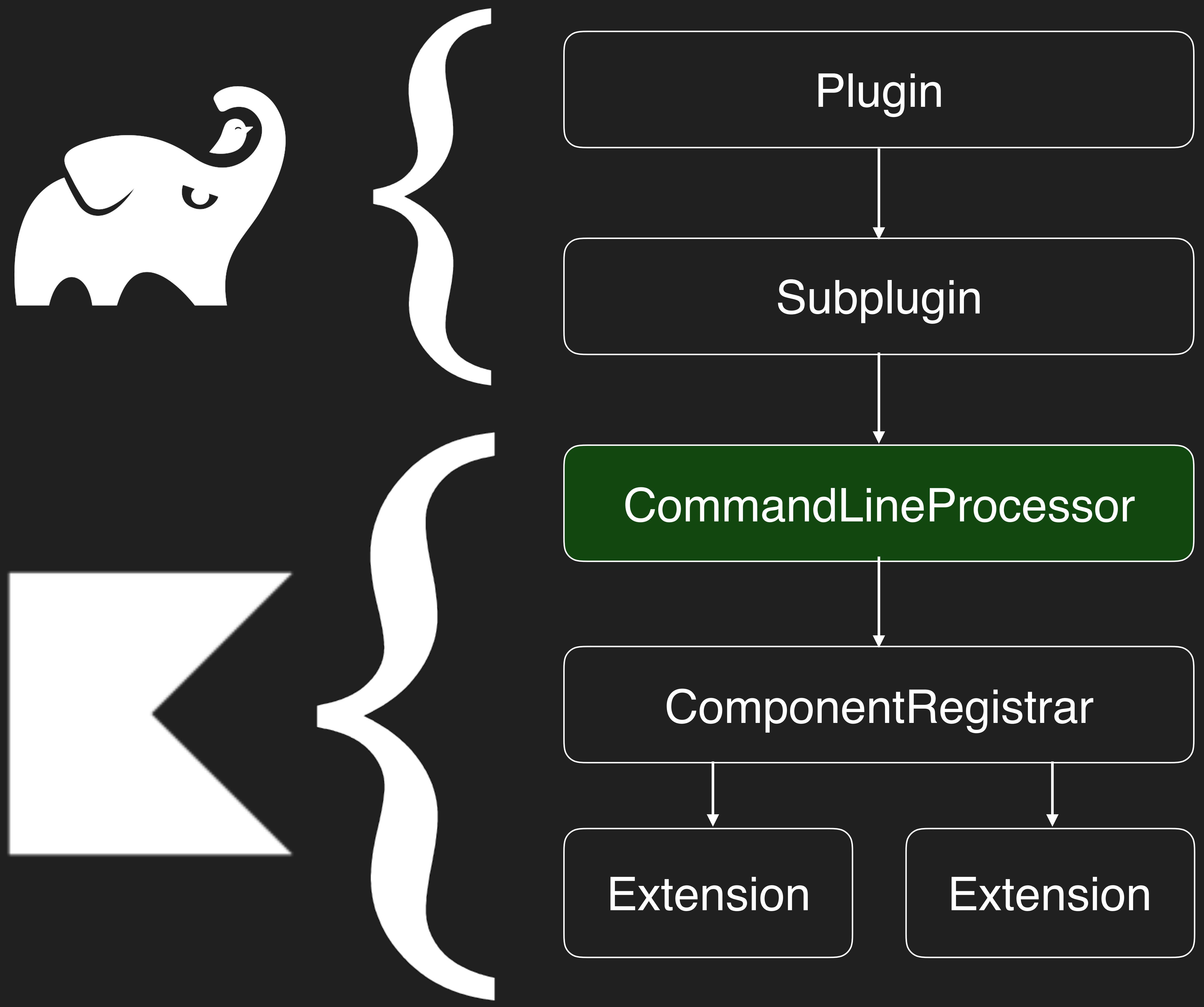
```kotlin
@AutoService(KotlinGradleSubplugin::class) // don't forget!
class DebugLogGradleSubplugin  :  KotlinGradleSubplugin<AbstractCompile>  {

  // other method impls

  override fun apply(project: Project, /*...*/): List<SubpluginOption>  {
    TODO()
  }
}
```

```kotlin
@AutoService(KotlinGradleSubplugin::class) // don't forget!
class DebugLogGradleSubplugin  :  KotlinGradleSubplugin<AbstractCompile>  {

  // other method impls

  override fun apply(project: Project, /*...*/): List<SubpluginOption>  {
    val extension = project.extensions.findByType<DebugLogGradleExtension>()
        ?: DebugLogGradleExtension()
    if (extension.enabled && extension.annotations.isEmpty())
      error("DebugLog is enabled, but no annotations were set")

    val annotationOptions = extension.annotations
        .map { SubpluginOption(key = "debugLogAnnotation", value = it) }
    val enabledOption = SubpluginOption(
        key = "enabled", value = extension.enabled.toString())
    return annotationOptions + enabledOption
  }
}
```

```kotlin
@AutoService(KotlinGradleSubplugin::class) // don't forget!
class DebugLogGradleSubplugin  :  KotlinGradleSubplugin<AbstractCompile>  {

  // other method impls

  override fun apply(project: Project, /*...*/): List<SubpluginOption>  {
    val extension = project.extensions.findByType<DebugLogGradleExtension>()
        ?: DebugLogGradleExtension()
    if (extension.enabled && extension.annotations.isEmpty())
      error("DebugLog is enabled, but no annotations were set")

    val annotationOptions = extension.annotations
        .map { SubpluginOption(key = "debugLogAnnotation", value = it) }
    val enabledOption = SubpluginOption(
        key = "enabled", value = extension.enabled.toString())
    return annotationOptions + enabledOption
  }
}
```

```kotlin
@AutoService(KotlinGradleSubplugin::class) // don't forget!
class DebugLogGradleSubplugin  :  KotlinGradleSubplugin<AbstractCompile>  {

  // other method impls

  override fun apply(project: Project, /*...*/): List<SubpluginOption>  {
    val extension = project.extensions.findByType<DebugLogGradleExtension>()
        ?: DebugLogGradleExtension()
    if (extension.enabled && extension.annotations.isEmpty())
      error("DebugLog is enabled, but no annotations were set")

    val annotationOptions = extension.annotations
        .map { SubpluginOption(key = "debugLogAnnotation", value = it) }
    val enabledOption = SubpluginOption(
        key = "enabled", value = extension.enabled.toString())
    return annotationOptions + enabledOption
  }
}
```

```kotlin
@AutoService(KotlinGradleSubplugin::class) // don't forget!
class DebugLogGradleSubplugin : KotlinGradleSubplugin<AbstractCompile> {

  // other method impls

  override fun apply(project: Project, /*...*/): List<SubpluginOption> {
    val extension = project.extensions.findByType<DebugLogGradleExtension>()
        ?: DebugLogGradleExtension()
    if (extension.enabled && extension.annotations.isEmpty())
      error("DebugLog is enabled, but no annotations were set")

    val annotationOptions = extension.annotations
        .map { SubpluginOption(key = "debugLogAnnotation", value = it) }
    val enabledOption = SubpluginOption(
        key = "enabled", value = extension.enabled.toString())
    return annotationOptions + enabledOption
  }
}
```


Plugin
Subplugin
CommandLineProcessor
ComponentRegistrar
Extension
Extension

kotlin-plugin/build.gradle

```
apply plugin: "org.jetbrains.kotlin.jvm"
apply plugin: "kotlin-kapt"

dependencies {
  implementation "org.jetbrains.kotlin:kotlin-stdlib:$ktVersion"
  compileOnly "org.jetbrains.kotlin:kotlin-compiler-embeddable:$ktVersion"

  compileOnly "com.google.auto.service:auto-service:1.0-rc4"
  kapt "com.google.auto.service:auto-service:1.0-rc4"
}
```

kotlin-plugin/build.gradle

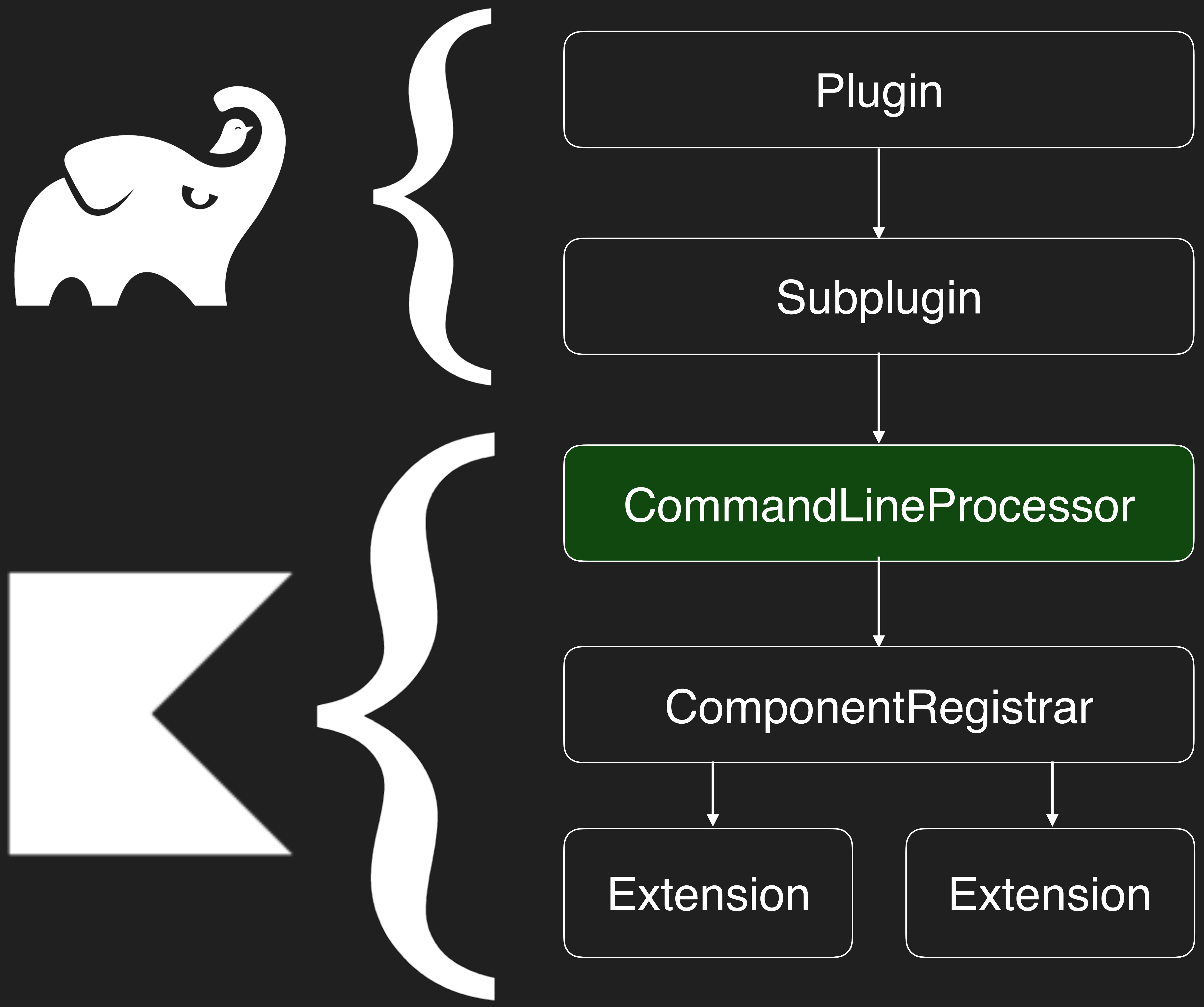
```
apply plugin: "org.jetbrains.kotlin.jvm"
apply plugin: "kotlin-kapt"

dependencies {
  implementation "org.jetbrains.kotlin:kotlin-stdlib:$ktVersion"
  compileOnly "org.jetbrains.kotlin:kotlin-compiler-embeddable:$ktVersion"

  compileOnly "com.google.auto.service:auto-service:1.0-rc4"
  kapt "com.google.auto.service:auto-service:1.0-rc4"
}
```

kotlin-plugin/build.gradle

```
apply plugin: "org.jetbrains.kotlin.jvm"
apply plugin: "kotlin-kapt"

dependencies {
  implementation "org.jetbrains.kotlin:kotlin-stdlib:$ktVersion"
  compileOnly "org.jetbrains.kotlin:kotlin-compiler-embeddable:$ktVersion"

  compileOnly "com.google.auto.service:auto-service:1.0-rc4"
  kapt "com.google.auto.service:auto-service:1.0-rc4"
}
```

kotlin-plugin/build.gradle

```
apply plugin: "org.jetbrains.kotlin.jvm"
apply plugin: "kotlin-kapt"

dependencies {
  implementation "org.jetbrains.kotlin:kotlin-stdlib:$ktVersion"
  compileOnly "org.jetbrains.kotlin:kotlin-compiler-embeddable:$ktVersion"

  compileOnly "com.google.auto.service:auto-service:1.0-rc4"
  kapt "com.google.auto.service:auto-service:1.0-rc4"
}
```


Plugin
Subplugin
CommandLineProcessor
ComponentRegistrar
Extension
Extension

```kotlin
@AutoService(CommandLineProcessor::class)
class DebugLogCommandLineProcessor : CommandLineProcessor {

  override val pluginId: String = TODO()

  override val pluginOptions: Collection<CliOption> = listOf(
  )

  override fun processOption(
      option: CliOption,
      value: String,
      configuration: CompilerConfiguration
  ) {
    TODO()
  }
}
```

```kotlin
@AutoService(CommandLineProcessor::class)
class DebugLogCommandLineProcessor : CommandLineProcessor {

  override val pluginId: String = TODO()

  override val pluginOptions: Collection<CliOption> = listOf(
  )

  override fun processOption(
      option: CliOption,
      value: String,
      configuration: CompilerConfiguration
  ) {
    TODO()
  }
}
```

```kotlin
@AutoService(CommandLineProcessor::class)
class DebugLogCommandLineProcessor : CommandLineProcessor {

  override val pluginId: String = "debuglog" // same as ID from subplugin

  override val pluginOptions: Collection<CliOption> = listOf(
  )

  override fun processOption(
      option: CliOption,
      value: String,
      configuration: CompilerConfiguration
  ) {
    TODO()
  }
}
```

```kotlin
@AutoService(CommandLineProcessor::class)
class DebugLogCommandLineProcessor : CommandLineProcessor {

  override val pluginId: String = "debuglog" // same as ID from subplugin

  override val pluginOptions: Collection<CliOption> = listOf(
  )

  override fun processOption(
      option: CliOption,
      value: String,
      configuration: CompilerConfiguration
  ) {
    TODO()
  }
}
```

```
@AutoService(CommandLineProcessor::class)
class DebugLogCommandLineProcessor : CommandLineProcessor {

  override val pluginId: String = "debuglog" // same as ID from subplugin

  override val pluginOptions: Collection<CliOption> = listOf(
  )

  override fun processOption(
      option: CliOption,
      value: String,
      configuration: CompilerConfiguration
  ) {
    TODO()
  }
}
```

```
@AutoService(CommandLineProcessor::class)
class DebugLogCommandLineProcessor : CommandLineProcessor {

  override val pluginId: String = "debuglog" // same as ID from subplugin

  override val pluginOptions: Collection<CliOption> = listOf(
      CliOption("enabled", "<true|false>", "whether plugin is enabled"),
      CliOption(
          "debugLogAnnotation", "<fqname>", "debug-log annotation names",
          required = true, allowMultipleOccurrences = true))

  override fun processOption(
      option: CliOption,
      value: String,
      configuration: CompilerConfiguration
  ) {
    TODO()
  }
}
```

```
@AutoService(CommandLineProcessor::class)
class DebugLogCommandLineProcessor : CommandLineProcessor {

  override val pluginId: String = "debuglog" // same as ID from subplugin

  override val pluginOptions: Collection<CliOption> = listOf(
      CliOption("enabled", "<true|false>", "whether plugin is enabled"),
      CliOption(
          "debugLogAnnotation", "<fqname>", "debug-log annotation names",
          required = true, allowMultipleOccurrences = true))

  override fun processOption(
      option: CliOption,
      value: String,
      configuration: CompilerConfiguration
  ) {
    TODO()
  }
}
```

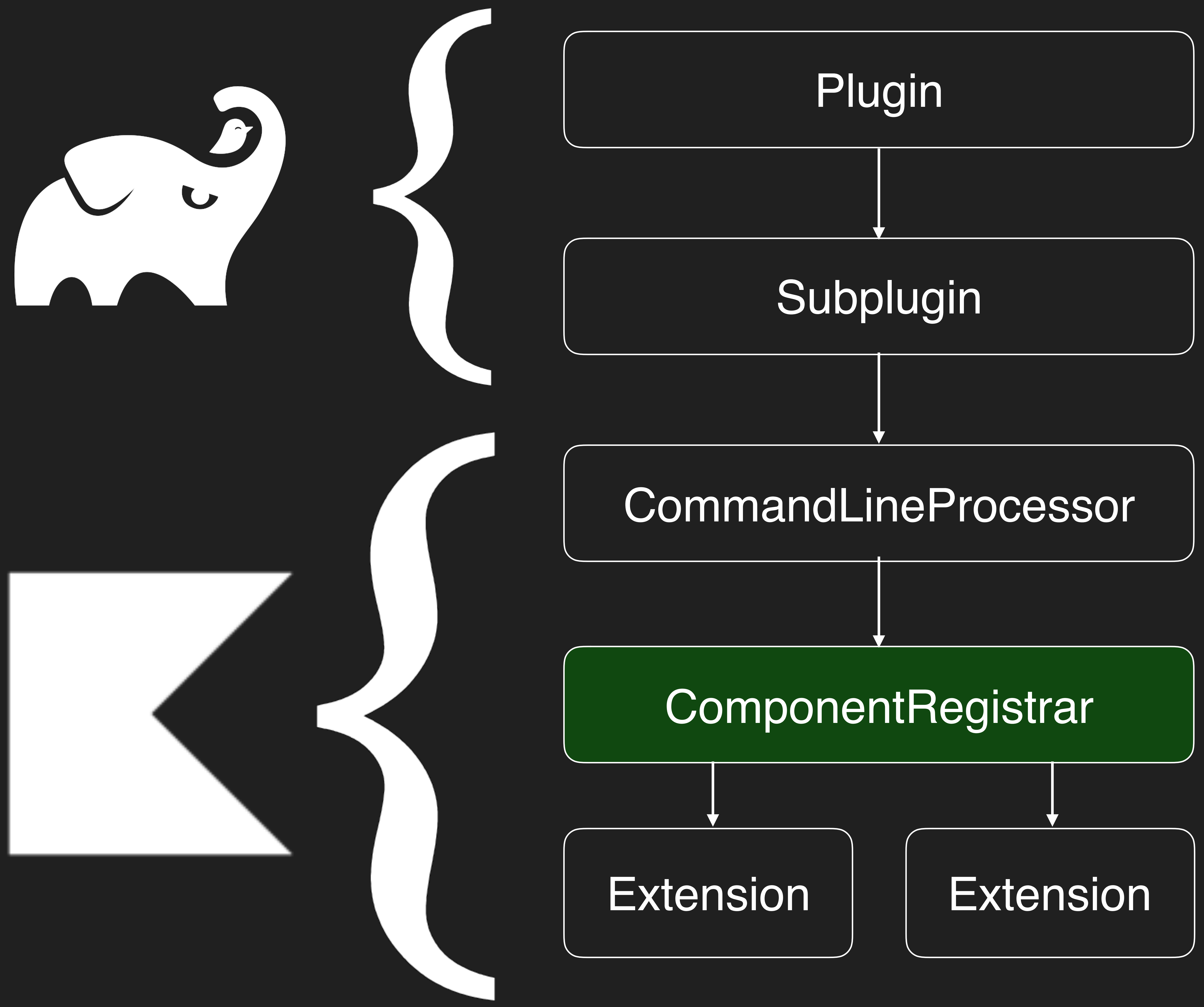
```
@AutoService(CommandLineProcessor::class)
class DebugLogCommandLineProcessor : CommandLineProcessor {

  override val pluginId: String = "debuglog" // same as ID from subplugin

  override val pluginOptions: Collection<CliOption> = listOf(
      CliOption("enabled", "<true|false>", "whether plugin is enabled"),
      CliOption(
          "debugLogAnnotation", "<fqname>", "debug-log annotation names",
          required = true, allowMultipleOccurrences = true))

  override fun processOption(
      option: CliOption,
      value: String,
      configuration: CompilerConfiguration
  ) {
    TODO()
  }
}
```

```kotlin
@AutoService(CommandLineProcessor::class)
class DebugLogCommandLineProcessor : CommandLineProcessor {

  override val pluginId: String = "debuglog" // same as ID from subplugin

  override val pluginOptions: Collection<CliOption> = listOf(
      CliOption("enabled", "<true|false>", "whether plugin is enabled"),
      CliOption(
          "debugLogAnnotation", "<fqname>", "debug-log annotation names",
          required = true, allowMultipleOccurrences = true))

  override fun processOption(
      option: CliOption,
      value: String,
      configuration: CompilerConfiguration
  ) = when (option.name) {

  }
}
```

```
@AutoService(CommandLineProcessor::class)
class DebugLogCommandLineProcessor : CommandLineProcessor {

  override val pluginId: String = "debuglog" // same as ID from subplugin

  override val pluginOptions: Collection<CliOption> = listOf(
      CliOption("enabled", "<true|false>", "whether plugin is enabled"),
      CliOption(
          "debugLogAnnotation", "<fqname>", "debug-log annotation names",
          required = true, allowMultipleOccurrences = true))

  override fun processOption(
      option: CliOption,
      value: String,
      configuration: CompilerConfiguration
  ) = when (option.name) {
    "enabled" -> configuration.put(KEY_ENABLED, value.toBoolean())
  }
}
```

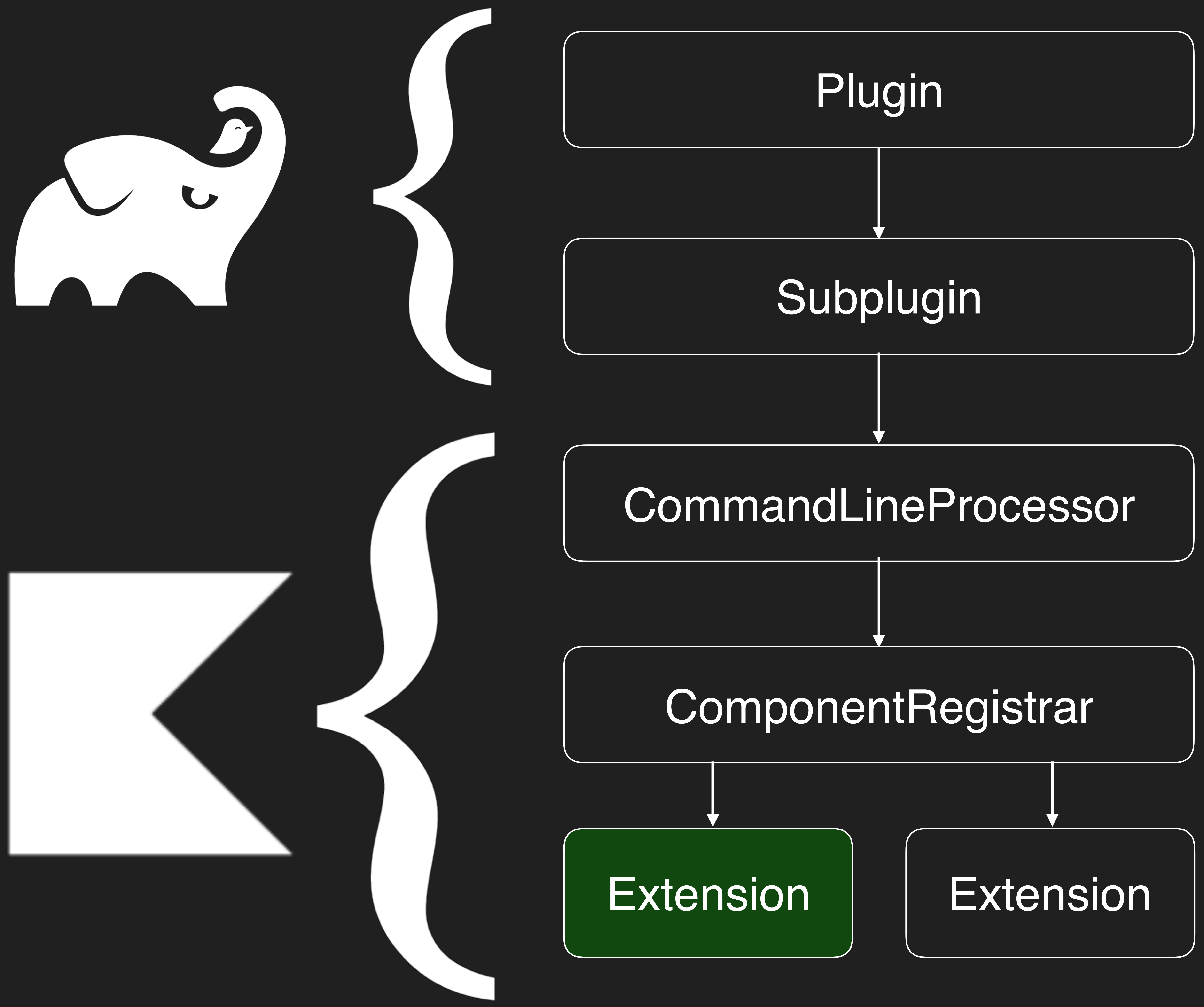
```kotlin
@AutoService(CommandLineProcessor::class)
class DebugLogCommandLineProcessor : CommandLineProcessor {

  override val pluginId: String = "debuglog" // same as ID from subplugin

  override val pluginOptions: Collection<CliOption> = listOf(
      CliOption("enabled", "<true|false>", "whether plugin is enabled"),
      CliOption(
          "debugLogAnnotation", "<fqname>", "debug-log annotation names",
          required = true, allowMultipleOccurrences = true))

  override fun processOption(
      option: CliOption,
      value: String,
      configuration: CompilerConfiguration
  ) = when (option.name) {
    "enabled" -> configuration.put(KEY_ENABLED, value.toBoolean())
    "debugLogAnnotation" -> configuration.appendList(KEY_ANNOTATIONS, value)
  }
}
```

```kotlin
@AutoService(CommandLineProcessor::class)
class DebugLogCommandLineProcessor : CommandLineProcessor {

  override val pluginId: String = "debuglog" // same as ID from subplugin

  override val pluginOptions: Collection<CliOption> = listOf(
      CliOption("enabled", "<true|false>", "whether plugin is enabled"),
      CliOption(
          "debugLogAnnotation", "<fqname>", "debug-log annotation names",
          required = true, allowMultipleOccurrences = true))

  override fun processOption(
      option: CliOption,
      value: String,
      configuration: CompilerConfiguration
  ) = when (option.name) {
    "enabled" -> configuration.put(KEY_ENABLED, value.toBoolean())
    "debugLogAnnotation" -> configuration.appendList(KEY_ANNOTATIONS, value)
    else -> error("Unexpected config option ${option.name}")
  }
}
```


Plugin
Subplugin
CommandLineProcessor
ComponentRegistrar
Extension
Extension

```kotlin
@AutoService(ComponentRegistrar::class)
class DebugLogComponentRegistrar : ComponentRegistrar {
 override fun registerProjectComponents(
     project: MockProject,
     configuration: CompilerConfiguration
 ) {
   if (configuration[KEY_ENABLED] == false) {
     return
   }
   ClassBuilderInterceptorExtension.registerExtension(
       project,
       DebugLogClassGenerationInterceptor(
           debugLogAnnotations = configuration[KEY_ANNOTATIONS]
               ?: error("debuglog plugin requires at least one annotation class option passed to it")
       )
   )
 }
}
```

```kotlin
@AutoService(ComponentRegistrar::class)
class DebugLogComponentRegistrar : ComponentRegistrar {
  override fun registerProjectComponents(
      project: MockProject,
      configuration: CompilerConfiguration
  ) {
    if (configuration[KEY_ENABLED] == false) {
      return
    }
    ClassBuilderInterceptorExtension.registerExtension(
        project,
        DebugLogClassGenerationInterceptor(
            debugLogAnnotations = configuration[KEY_ANNOTATIONS]
                ?: error("debuglog plugin requires at least one annotation class option passed to it")
        )
    )
  }
}
```


Plugin
Subplugin
CommandLineProcessor
ComponentRegistrar
Extension
Extension

kotlin-plugin/src/main/kotlin/debuglog/DebugLogClassGenerationInterceptor.kt

```kotlin
class DebugLogClassGenerationInterceptor(
    val debugLogAnnotations: List<String>
) : ClassBuilderInterceptorExtension {
  override fun interceptClassBuilderFactory(
      interceptedFactory: ClassBuilderFactory,
      bindingContext: BindingContext,
      diagnostics: DiagnosticSink
  ): ClassBuilderFactory = TODO()
}
```

kotlin-plugin/src/main/kotlin/debuglog/DebugLogClassGenerationInterceptor.kt

```
class DebugLogClassGenerationInterceptor(
    val debugLogAnnotations: List<String>
) : ClassBuilderInterceptorExtension {
  override fun interceptClassBuilderFactory(
      interceptedFactory: ClassBuilderFactory,
      bindingContext: BindingContext,
      diagnostics: DiagnosticSink
  ): ClassBuilderFactory = object: ClassBuilderFactory by interceptedFactory
}
```

kotlin-plugin/src/main/kotlin/debuglog/DebugLogClassGenerationInterceptor.kt

```kotlin
class DebugLogClassGenerationInterceptor(
    val debugLogAnnotations: List<String>
) : ClassBuilderInterceptorExtension {
  override fun interceptClassBuilderFactory(
      interceptedFactory: ClassBuilderFactory,
      bindingContext: BindingContext,
      diagnostics: DiagnosticSink
  ): ClassBuilderFactory = object: ClassBuilderFactory by interceptedFactory {
    override fun newClassBuilder(origin: JvmDeclarationOrigin) =
        DebugLogClassBuilder(
            annotations = debugLogAnnotations,
            delegateBuilder =  interceptedFactory.newClassBuilder(origin))
  }
}
```

kotlin-plugin/src/main/kotlin/debuglog/DebugLogClassBuilder.kt

```kotlin
private class DebugLogClassBuilder(
    val annotations: List<String>,
    delegateBuilder: ClassBuilder
) : DelegatingClassBuilder(delegateBuilder) {
  override fun newMethod(
      origin: JvmDeclarationOrigin, access: Int,
      name: String, desc: String,
      signature: String?, exceptions: Array<out String>?
  ): MethodVisitor {
  }
}
```

kotlin-plugin/src/main/kotlin/debuglog/DebugLogClassBuilder.kt

```
private class DebugLogClassBuilder(
    val annotations: List<String>,
    delegateBuilder: ClassBuilder
) : DelegatingClassBuilder(delegateBuilder) {
  override fun newMethod(
      origin: JvmDeclarationOrigin,...
  ): MethodVisitor {
  }
}
```

kotlin-plugin/src/main/kotlin/debuglog/DebugLogClassBuilder.kt

```kotlin
private class DebugLogClassBuilder(
    val annotations: List<String>,
    delegateBuilder: ClassBuilder
) : DelegatingClassBuilder(delegateBuilder) {
  override fun newMethod(
      origin: JvmDeclarationOrigin,...
  ): MethodVisitor {
    val original = super.newMethod(origin, ...)
    val function = origin.descriptor as? FunctionDescriptor ?: return original
    if (annotations.none { descriptor.annotations.hasAnnotation(it) }) {
      return original
    }
  }
}
```

kotlin-plugin/src/main/kotlin/debuglog/DebugLogClassBuilder.kt

```
private class DebugLogClassBuilder(
    val annotations: List<String>,
    delegateBuilder: ClassBuilder
) : DelegatingClassBuilder(delegateBuilder) {
  override fun newMethod(
      origin: JvmDeclarationOrigin,...
  ): MethodVisitor {
    val original = super.newMethod(origin, ...)
    val function = origin.descriptor as? FunctionDescriptor ?: return original
    if (annotations.none { descriptor.annotations.hasAnnotation(it) }) {
      return original
    }
    return object : MethodVisitor(Opcodes.ASM5, original) {
    }
  }
}
```

kotlin-plugin/src/main/kotlin/debuglog/DebugLogClassBuilder.kt

```
return object : MethodVisitor(Opcodes.ASM5, original) {
}
```

kotlin-plugin/src/main/kotlin/debuglog/DebugLogClassBuilder.kt

```kotlin
return object : MethodVisitor(Opcodes.ASM5, original) {
  override fun visitCode() {
    super.visitCode()
    InstructionAdapter(this).apply { TODO("on method entry") }
  }
}
```

kotlin-plugin/src/main/kotlin/debuglog/DebugLogClassBuilder.kt

```kotlin
return object : MethodVisitor(Opcodes.ASM5, original) {
  override fun visitCode() {
    super.visitCode()
    InstructionAdapter(this).apply { TODO("on method entry") }
  }
  override fun visitInsn(opcode: Int) {
    when (opcode) {
      RETURN /* void */, ARETURN /* object */, IRETURN /* int */ -> {
        InstructionAdapter(this).apply { TODO("on method exit") }
      }
    }
    super.visitInsn(opcode)
  }
}
```

What now?

- You write bytecode
- Uses the ObjectWeb ASM API
  - Neither related to ASM (assembly) or Web ASM (wasm) in any way
  - An API for modifying JVM bytecode
- The JVM is a *stack machine*
  - One stack that methods operate upon
  - You can also read arbitrary variables from the Local Variable Array

What does bytecode look like?

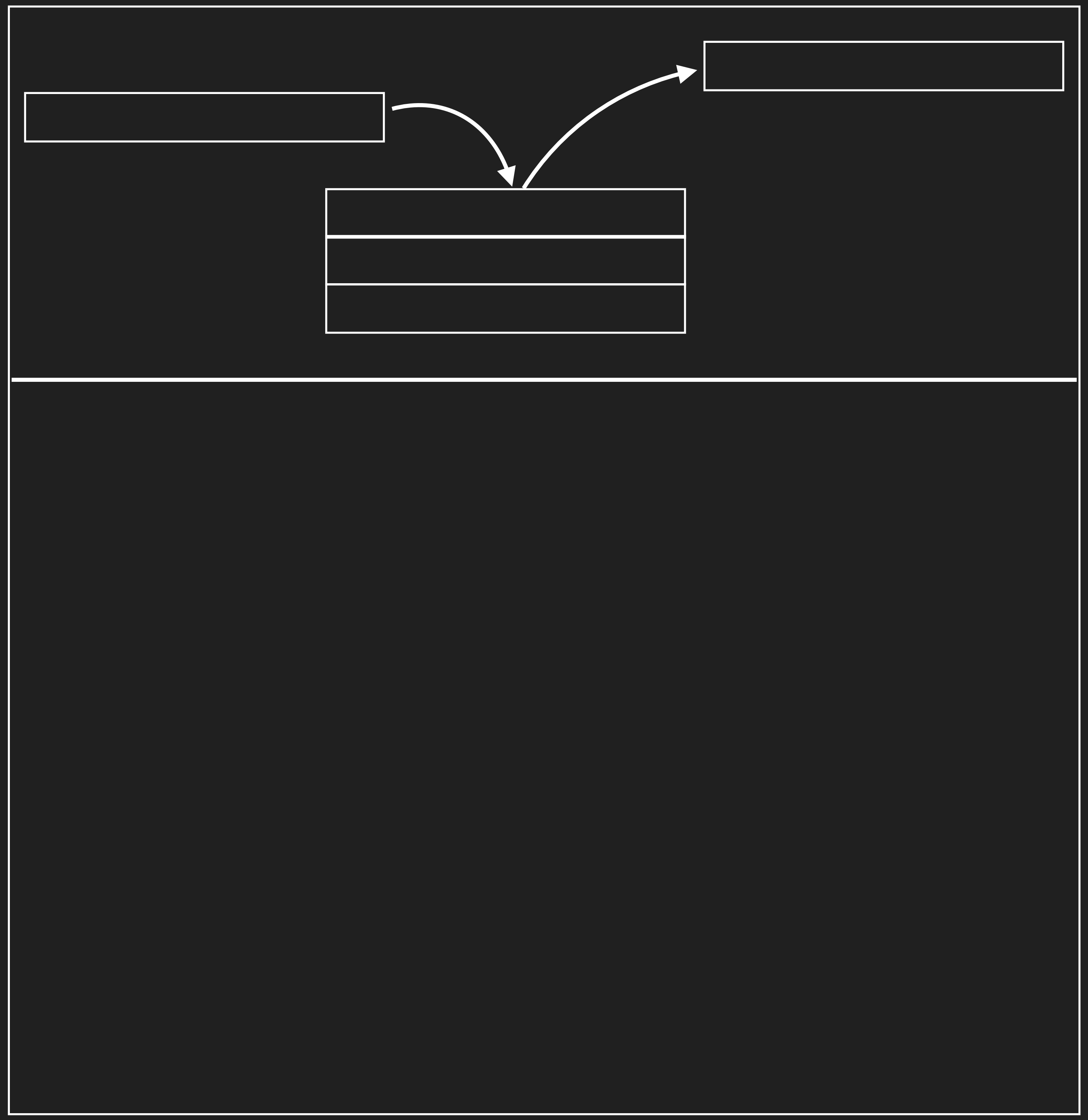
```kotlin
fun printSimpleSum() {
  val sum = v1() + v2()
  println("sum of values was $sum")
}
```

```
INVOKESTATIC myapp/RunnerKt.v1 ()I
INVOKESTATIC myapp/RunnerKt.v2 ()I
IADD
ISTORE 1
GETSTATIC j/l/System.out : Lj/io/PrintStream;
NEW j/l/StringBuilder
DUP
INVOKESPECIAL j/l/StringBuilder.<init> ()V
LDC "sum of values was "
INVOKEVIRTUAL j/l/StringBuilder.append (Lj/l/String;)Lj/l/StringBuilder;
ILOAD 1
INVOKEVIRTUAL j/l/StringBuilder.append (I)Lj/l/StringBuilder;
INVOKEVIRTUAL j/l/StringBuilder.toString ()Lj/l/String;
INVOKEVIRTUAL j/io/PrintStream.println (Lj/l/String;)V
```

What does bytecode look like?

```
fun printSimpleSum() {
  val sum = v1() + v2()
  println("sum of values was $sum")
}
```

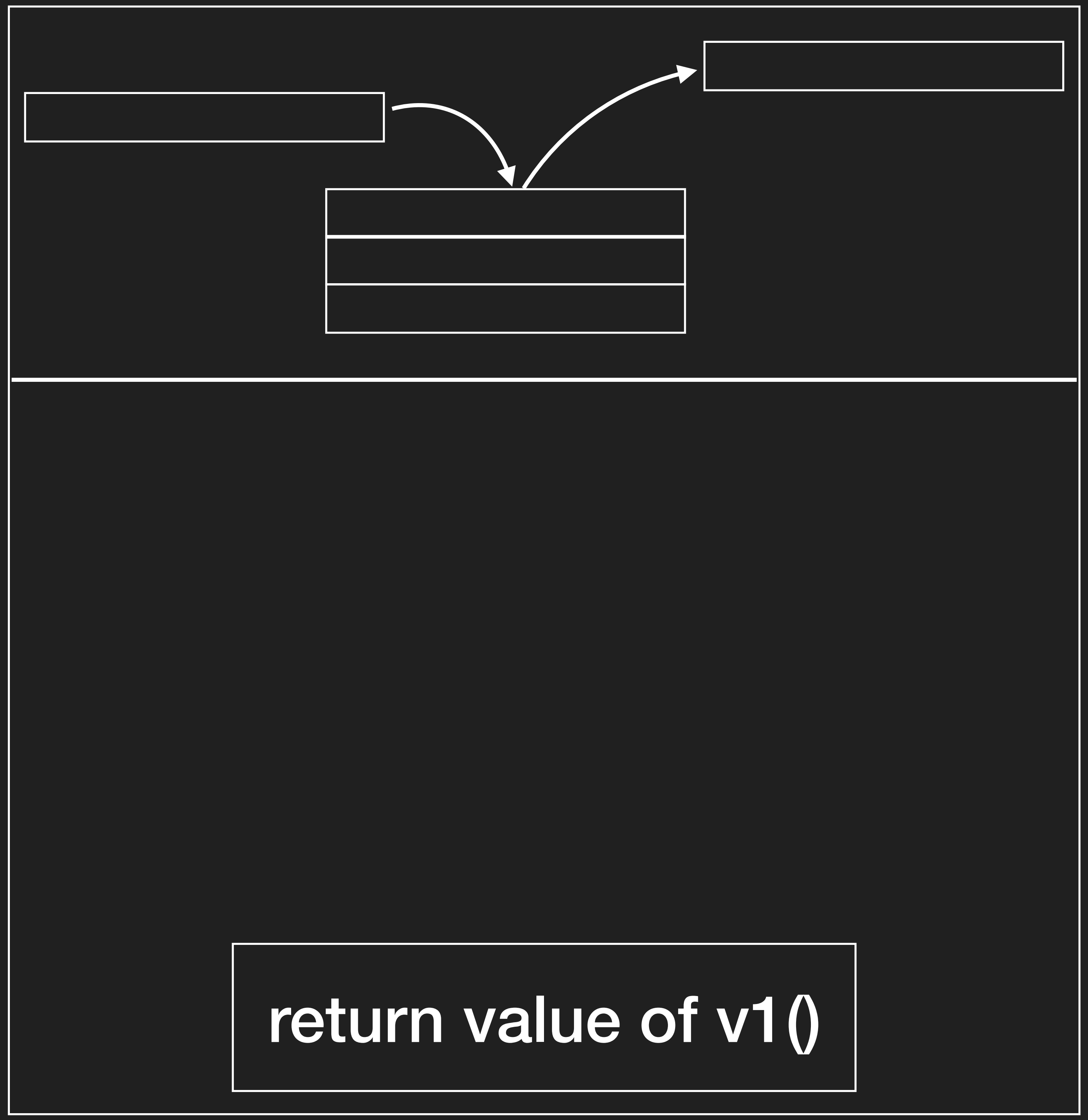
```
INVOKESTATIC myapp/RunnerKt.v1 ()I
INVOKESTATIC myapp/RunnerKt.v2 ()I
IADD
ISTORE 1
GETSTATIC j/l/System.out : Lj/io/PrintStream;
NEW j/l/StringBuilder
DUP
INVOKESPECIAL j/l/StringBuilder.<init> ()V
LDC "sum of values was "
INVOKEVIRTUAL j/l/StringBuilder.append (Lj/l/String;)Lj/l/StringBuilder;
ILOAD 1
INVOKEVIRTUAL j/l/StringBuilder.append (I)Lj/l/StringBuilder;
INVOKEVIRTUAL j/l/StringBuilder.toString ()Lj/l/String;
INVOKEVIRTUAL j/io/PrintStream.println (Lj/l/String;)V
```



What does bytecode look like?

```
fun printSimpleSum() {
  val sum = v1() + v2()
  println("sum of values was $sum")
}
```

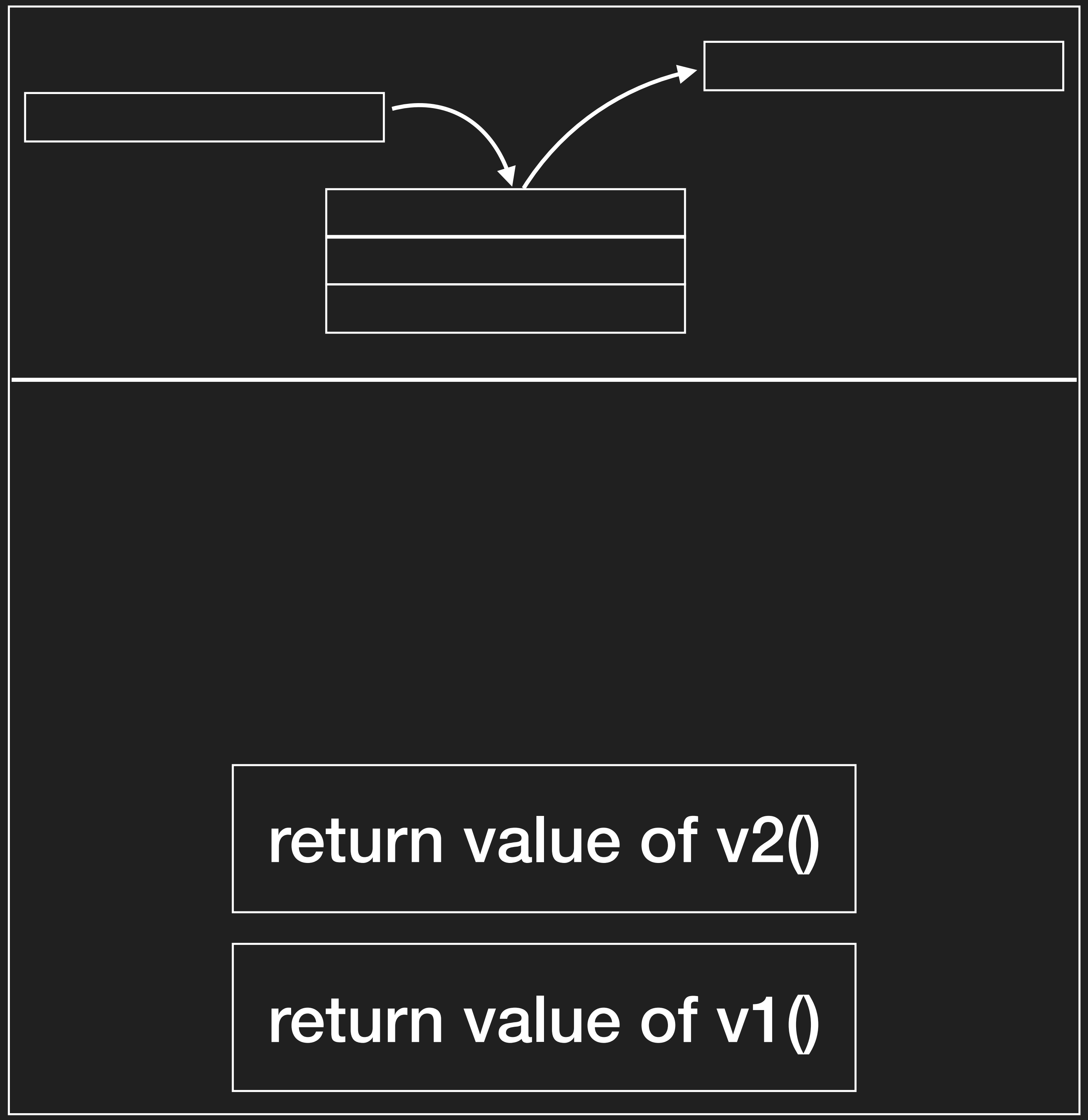
```
INVOKESTATIC myapp/RunnerKt.v1 ()I
INVOKESTATIC myapp/RunnerKt.v2 ()I
IADD
ISTORE 1
GETSTATIC j/l/System.out : Lj/io/PrintStream;
NEW j/l/StringBuilder
DUP
INVOKESPECIAL j/l/StringBuilder.<init> ()V
LDC "sum of values was "
INVOKEVIRTUAL j/l/StringBuilder.append (Lj/l/String;)Lj/l/StringBuilder;
ILOAD 1
INVOKEVIRTUAL j/l/StringBuilder.append (I)Lj/l/StringBuilder;
INVOKEVIRTUAL j/l/StringBuilder.toString ()Lj/l/String;
INVOKEVIRTUAL j/io/PrintStream.println (Lj/l/String;)V
```



What does bytecode look like?

```
fun printSimpleSum() {
  val sum = v1() + v2()
  println("sum of values was $sum")
}
```

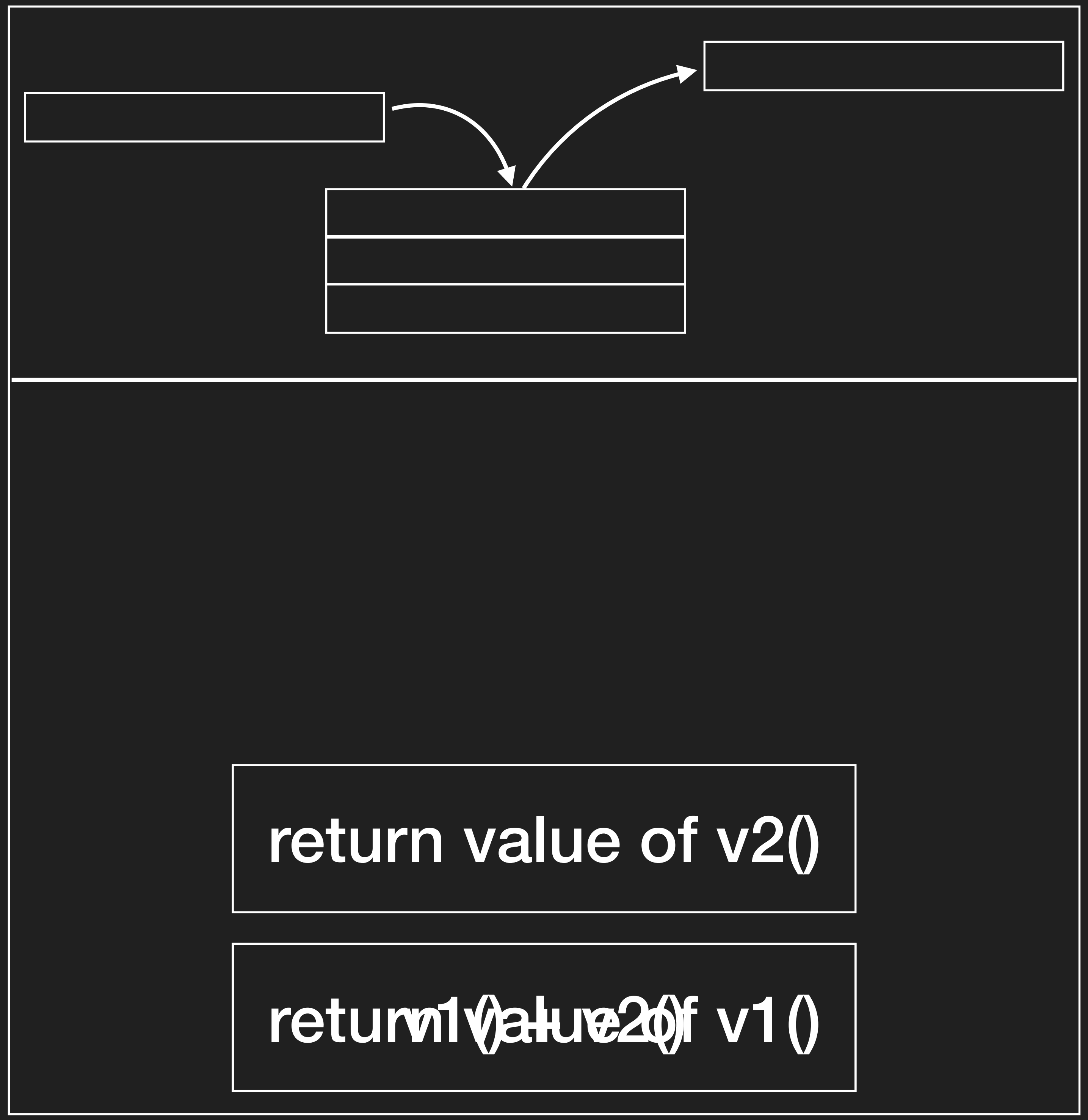
```
INVOKESTATIC myapp/RunnerKt.v1 ()I
INVOKESTATIC myapp/RunnerKt.v2 ()I
IADD
ISTORE 1
GETSTATIC j/l/System.out : Lj/io/PrintStream;
NEW j/l/StringBuilder
DUP
INVOKESPECIAL j/l/StringBuilder.<init> ()V
LDC "sum of values was "
INVOKEVIRTUAL j/l/StringBuilder.append (Lj/l/String;)Lj/l/StringBuilder;
ILOAD 1
INVOKEVIRTUAL j/l/StringBuilder.append (I)Lj/l/StringBuilder;
INVOKEVIRTUAL j/l/StringBuilder.toString ()Lj/l/String;
INVOKEVIRTUAL j/io/PrintStream.println (Lj/l/String;)V
```



What does bytecode look like?

```
fun printSimpleSum() {
  val sum = v1() + v2()
  println("sum of values was $sum")
}
```

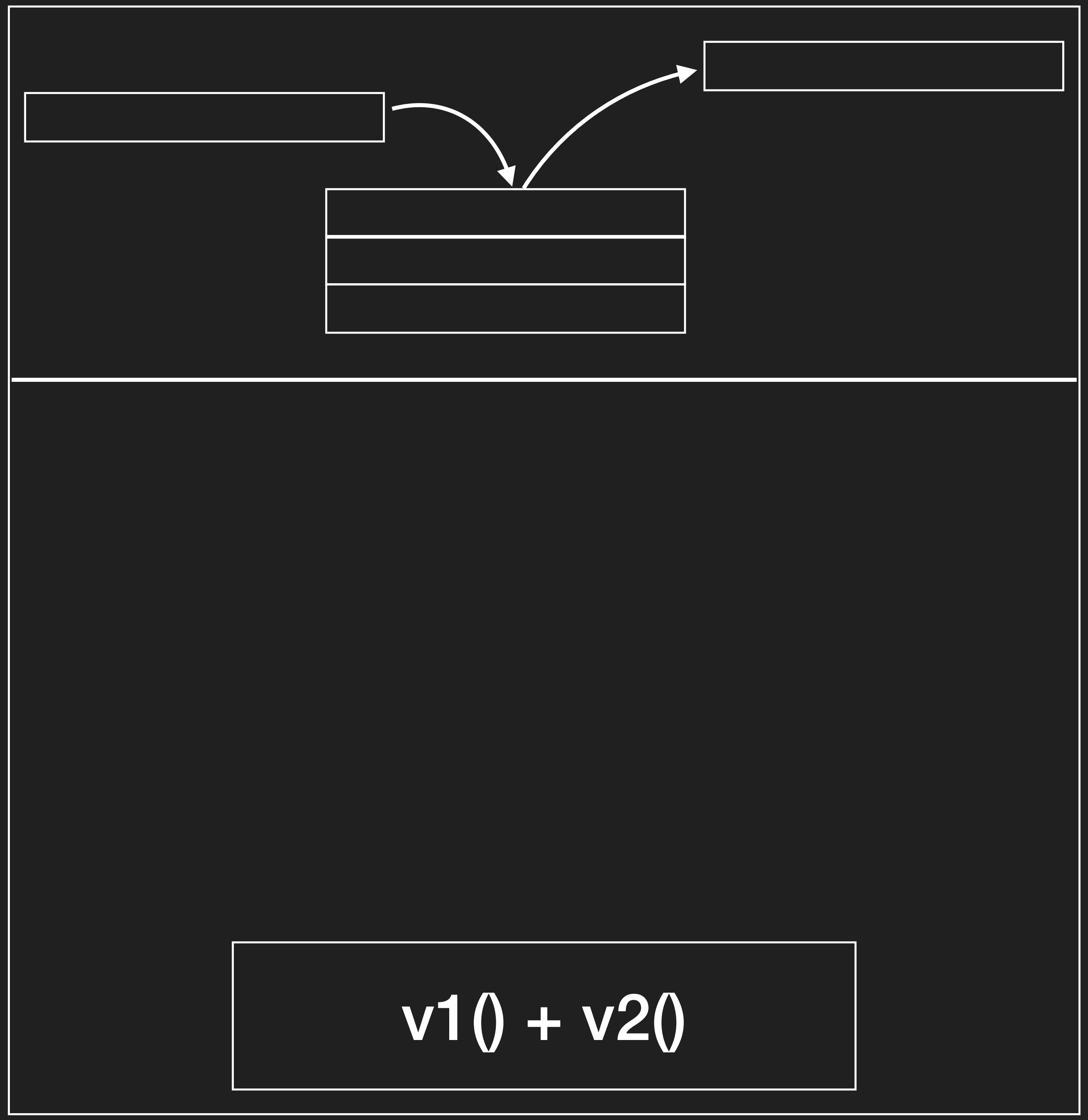
```
INVOKESTATIC myapp/RunnerKt.v1 ()I
INVOKESTATIC myapp/RunnerKt.v2 ()I
IADD
ISTORE 1
GETSTATIC j/l/System.out : Lj/io/PrintStream;
NEW j/l/StringBuilder
DUP
INVOKESPECIAL j/l/StringBuilder.<init> ()V
LDC "sum of values was "
INVOKEVIRTUAL j/l/StringBuilder.append (Lj/l/String;)Lj/l/StringBuilder;
ILOAD 1
INVOKEVIRTUAL j/l/StringBuilder.append (I)Lj/l/StringBuilder;
INVOKEVIRTUAL j/l/StringBuilder.toString ()Lj/l/String;
INVOKEVIRTUAL j/io/PrintStream.println (Lj/l/String;)V
```



What does bytecode look like?

```
fun printSimpleSum() {
  val sum = v1() + v2()
  println("sum of values was $sum")
}
```

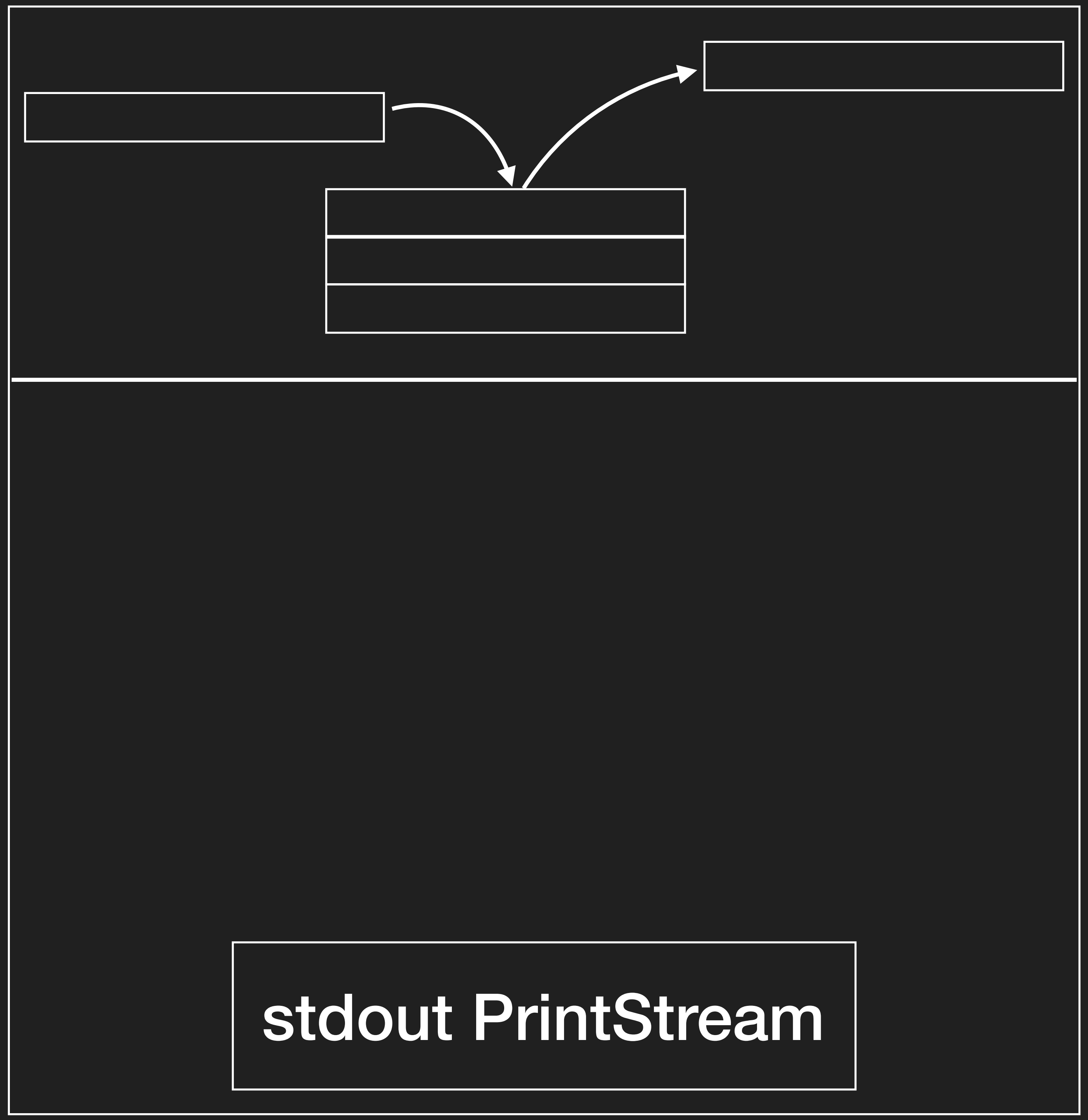
```
INVOKESTATIC myapp/RunnerKt.v1 ()I
INVOKESTATIC myapp/RunnerKt.v2 ()I
IADD
ISTORE 1
GETSTATIC j/l/System.out : Lj/io/PrintStream;
NEW j/l/StringBuilder
DUP
INVOKESPECIAL j/l/StringBuilder.<init> ()V
LDC "sum of values was "
INVOKEVIRTUAL j/l/StringBuilder.append (Lj/l/String;)Lj/l/StringBuilder;
ILOAD 1
INVOKEVIRTUAL j/l/StringBuilder.append (I)Lj/l/StringBuilder;
INVOKEVIRTUAL j/l/StringBuilder.toString ()Lj/l/String;
INVOKEVIRTUAL j/io/PrintStream.println (Lj/l/String;)V
```



What does bytecode look like?

```
fun printSimpleSum() {
  val sum = v1() + v2()
  println("sum of values was $sum")
}
```

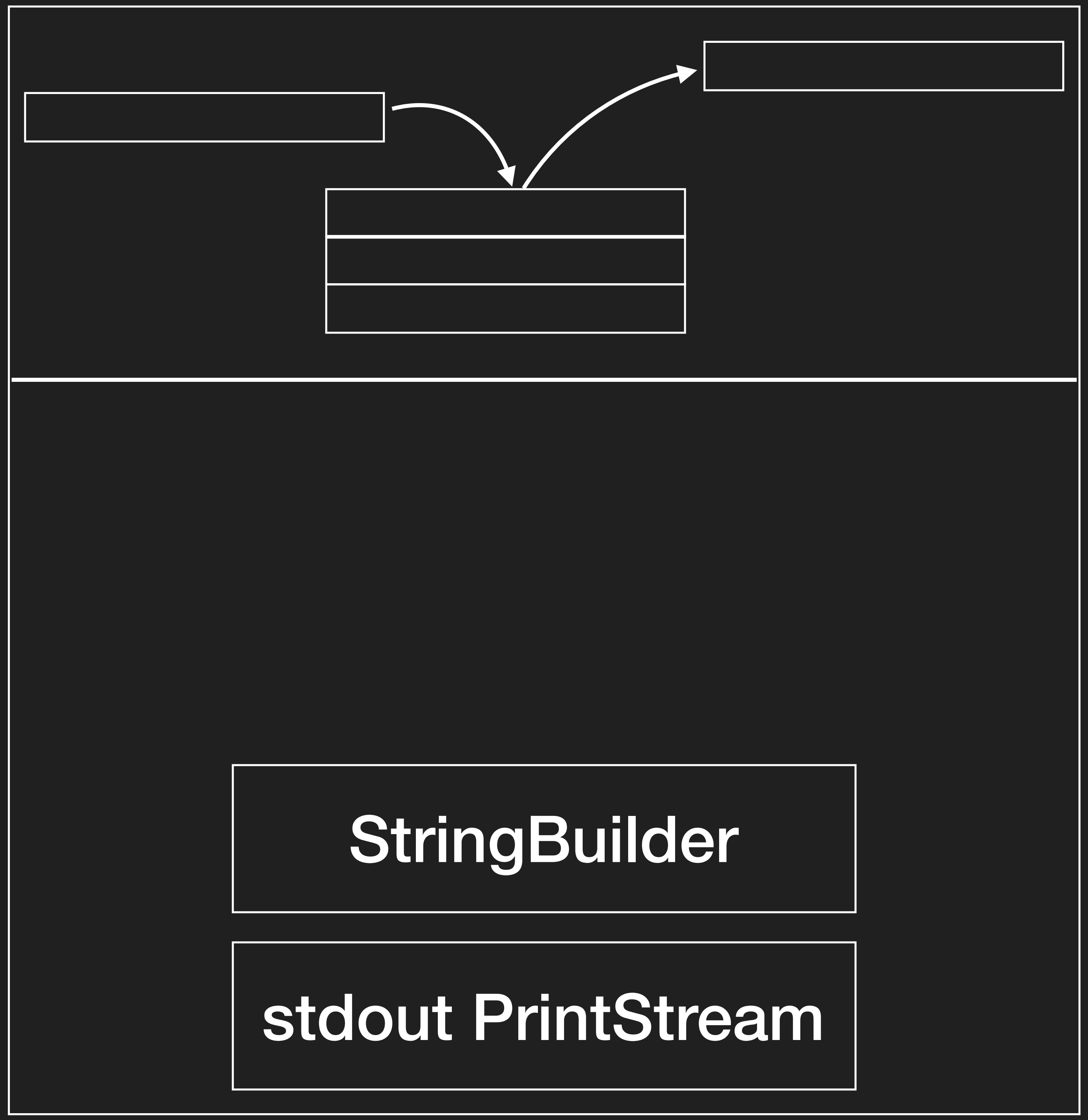
```
INVOKESTATIC myapp/RunnerKt.v1 ()I
INVOKESTATIC myapp/RunnerKt.v2 ()I
IADD
ISTORE 1
GETSTATIC j/l/System.out : Lj/io/PrintStream;
NEW j/l/StringBuilder
DUP
INVOKESPECIAL j/l/StringBuilder.<init> ()V
LDC "sum of values was "
INVOKEVIRTUAL j/l/StringBuilder.append (Lj/l/String;)Lj/l/StringBuilder;
ILOAD 1
INVOKEVIRTUAL j/l/StringBuilder.append (I)Lj/l/StringBuilder;
INVOKEVIRTUAL j/l/StringBuilder.toString ()Lj/l/String;
INVOKEVIRTUAL j/io/PrintStream.println (Lj/l/String;)V
```



What does bytecode look like?

```
fun printSimpleSum() {
  val sum = v1() + v2()
  println("sum of values was $sum")
}
```

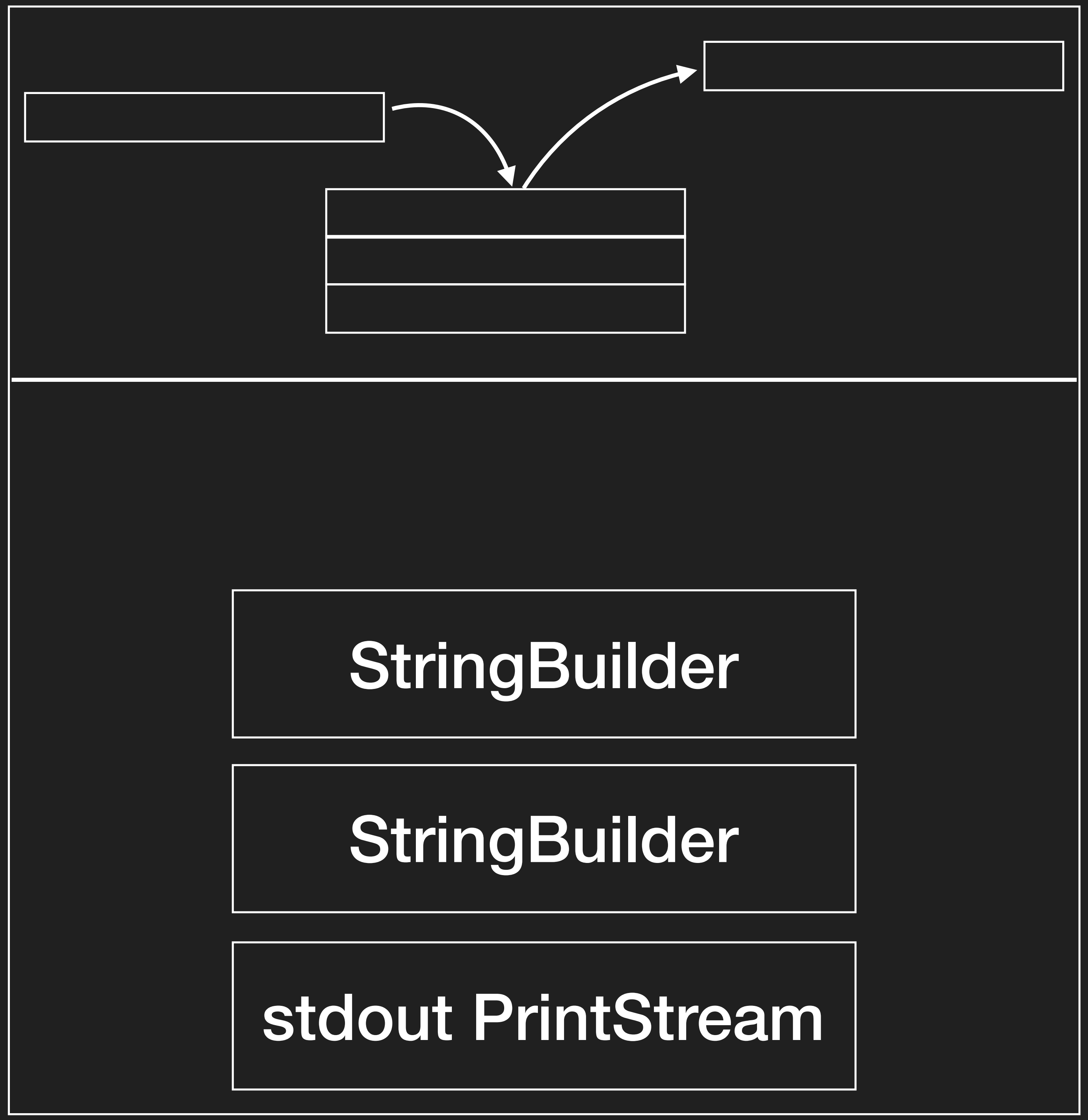
```
INVOKESTATIC myapp/RunnerKt.v1 ()I
INVOKESTATIC myapp/RunnerKt.v2 ()I
IADD
ISTORE 1
GETSTATIC j/l/System.out : Lj/io/PrintStream;
NEW j/l/StringBuilder
DUP
INVOKESPECIAL j/l/StringBuilder.<init> ()V
LDC "sum of values was "
INVOKEVIRTUAL j/l/StringBuilder.append (Lj/l/String;)Lj/l/StringBuilder;
ILOAD 1
INVOKEVIRTUAL j/l/StringBuilder.append (I)Lj/l/StringBuilder;
INVOKEVIRTUAL j/l/StringBuilder.toString ()Lj/l/String;
INVOKEVIRTUAL j/io/PrintStream.println (Lj/l/String;)V
```



What does bytecode look like?

```
fun printSimpleSum() {
  val sum = v1() + v2()
  println("sum of values was $sum")
}
```

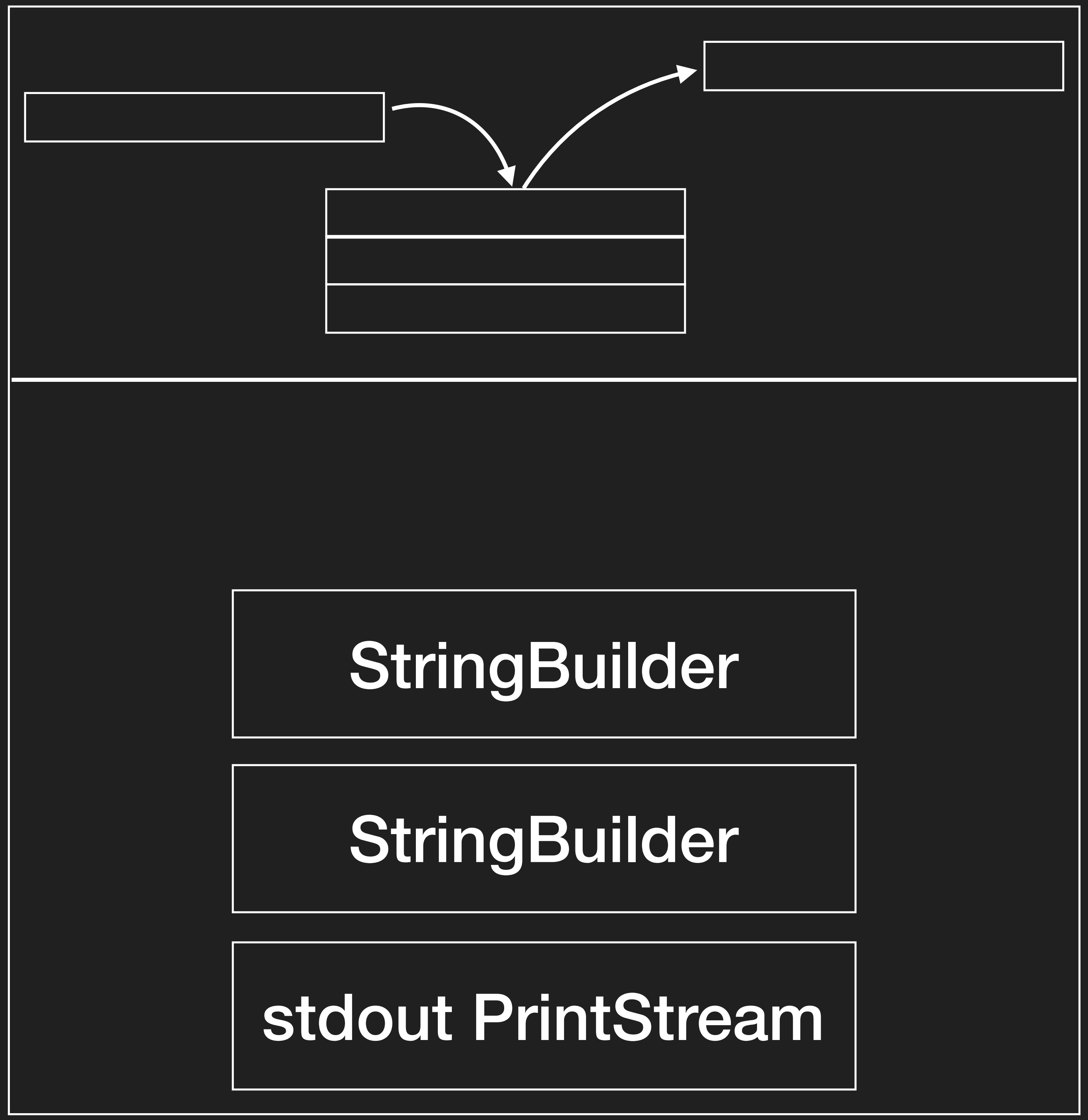
```
INVOKESTATIC myapp/RunnerKt.v1 ()I
INVOKESTATIC myapp/RunnerKt.v2 ()I
IADD
ISTORE 1
GETSTATIC j/l/System.out : Lj/io/PrintStream;
NEW j/l/StringBuilder
DUP
INVOKESPECIAL j/l/StringBuilder.<init> ()V
LDC "sum of values was "
INVOKEVIRTUAL j/l/StringBuilder.append (Lj/l/String;)Lj/l/StringBuilder;
ILOAD 1
INVOKEVIRTUAL j/l/StringBuilder.append (I)Lj/l/StringBuilder;
INVOKEVIRTUAL j/l/StringBuilder.toString ()Lj/l/String;
INVOKEVIRTUAL j/io/PrintStream.println (Lj/l/String;)V
```



What does bytecode look like?

```
fun printSimpleSum() {
  val sum = v1() + v2()
  println("sum of values was $sum")
}
```

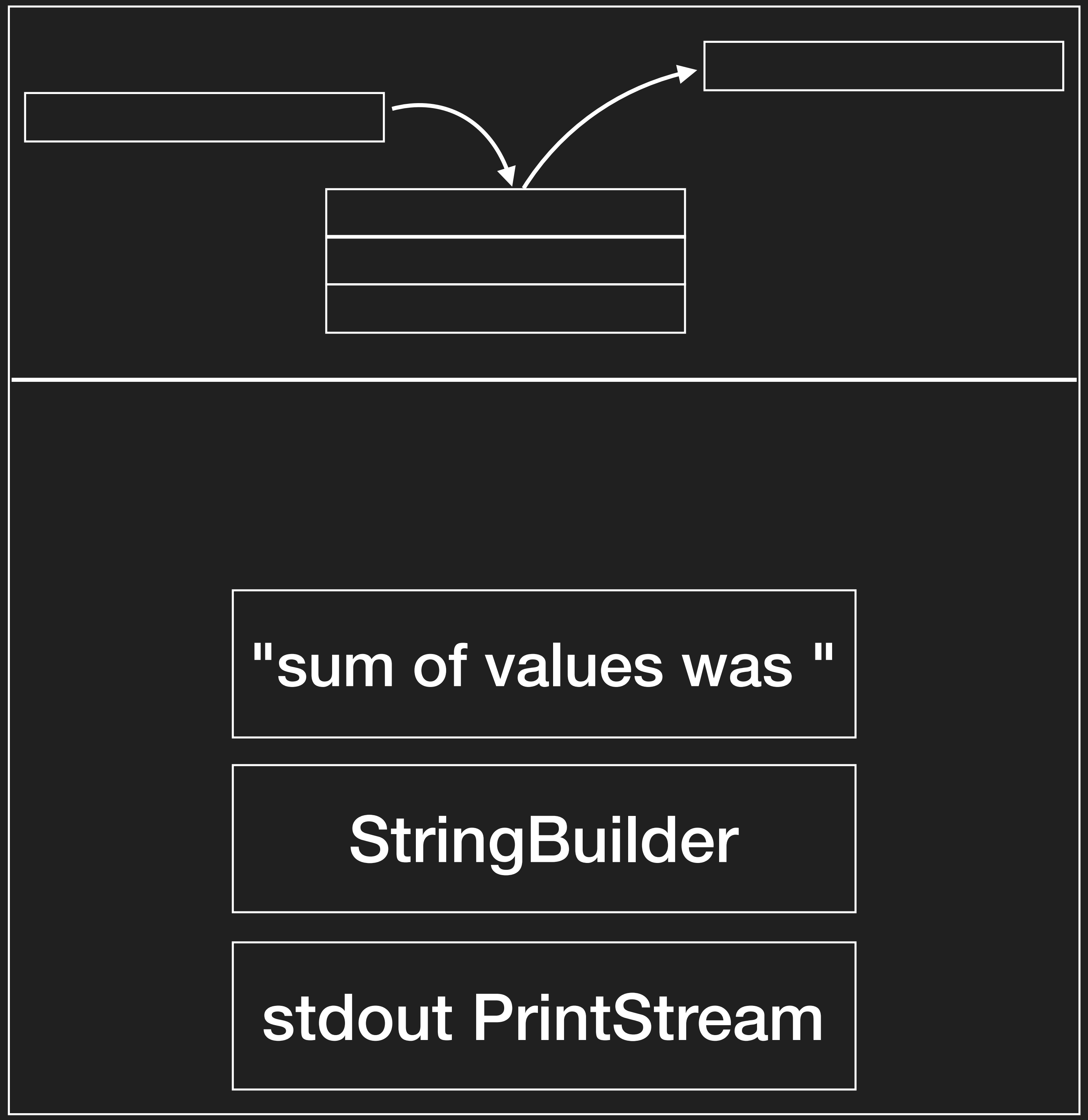
```
INVOKESTATIC myapp/RunnerKt.v1 ()I
INVOKESTATIC myapp/RunnerKt.v2 ()I
IADD
ISTORE 1
GETSTATIC j/l/System.out : Lj/io/PrintStream;
NEW j/l/StringBuilder
DUP
INVOKESPECIAL j/l/StringBuilder.<init> ()V
LDC "sum of values was "
INVOKEVIRTUAL j/l/StringBuilder.append (Lj/l/String;)Lj/l/StringBuilder;
ILOAD 1
INVOKEVIRTUAL j/l/StringBuilder.append (I)Lj/l/StringBuilder;
INVOKEVIRTUAL j/l/StringBuilder.toString ()Lj/l/String;
INVOKEVIRTUAL j/io/PrintStream.println (Lj/l/String;)V
```



What does bytecode look like?

```
fun printSimpleSum() {
  val sum = v1() + v2()
  println("sum of values was $sum")
}
```

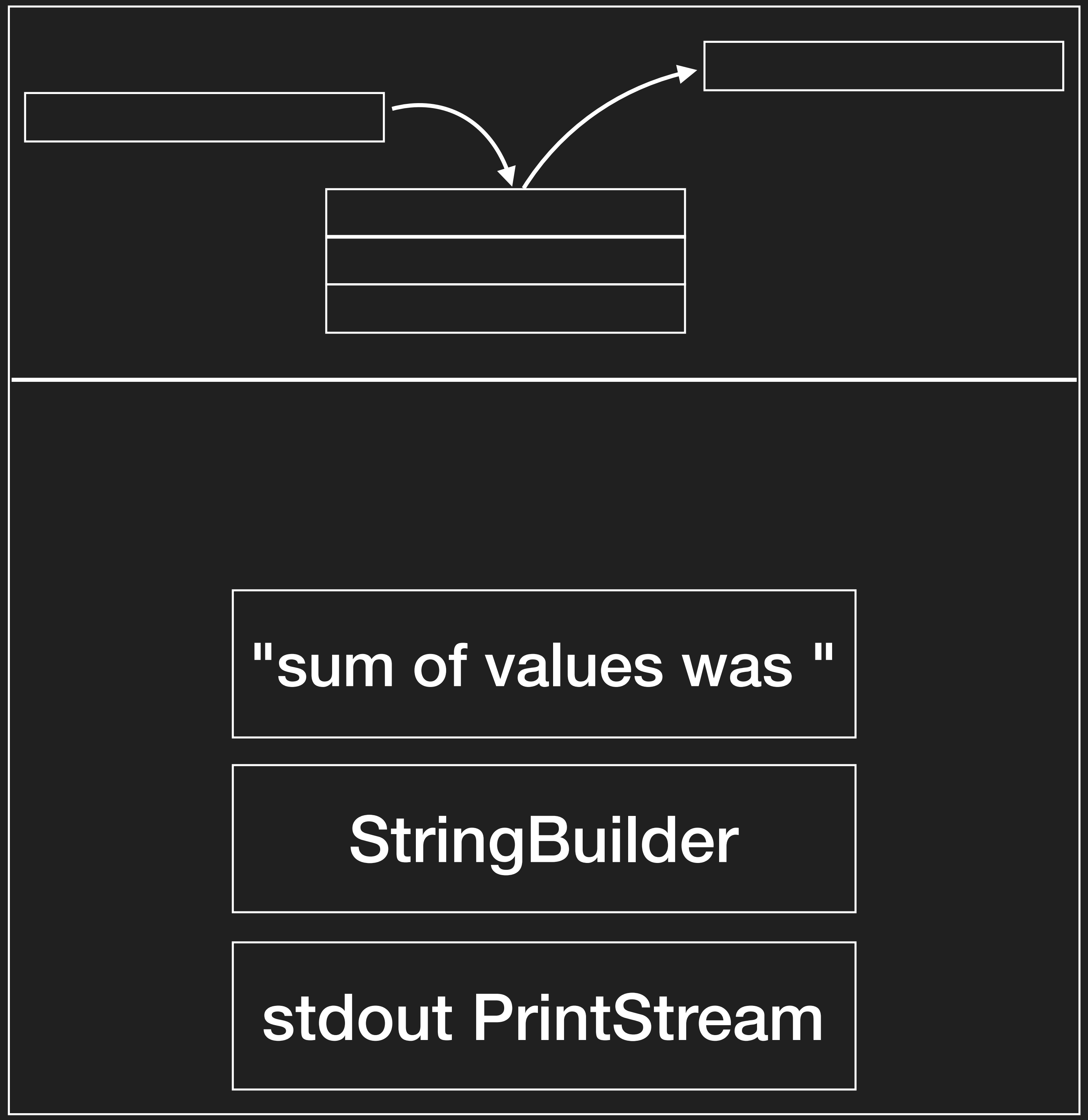
```
INVOKESTATIC myapp/RunnerKt.v1 ()I
INVOKESTATIC myapp/RunnerKt.v2 ()I
IADD
ISTORE 1
GETSTATIC j/l/System.out : Lj/io/PrintStream;
NEW j/l/StringBuilder
DUP
INVOKESPECIAL j/l/StringBuilder.<init> ()V
LDC "sum of values was "
INVOKEVIRTUAL j/l/StringBuilder.append (Lj/l/String;)Lj/l/StringBuilder;
ILOAD 1
INVOKEVIRTUAL j/l/StringBuilder.append (I)Lj/l/StringBuilder;
INVOKEVIRTUAL j/l/StringBuilder.toString ()Lj/l/String;
INVOKEVIRTUAL j/io/PrintStream.println (Lj/l/String;)V
```



What does bytecode look like?

```
fun printSimpleSum() {
  val sum = v1() + v2()
  println("sum of values was $sum")
}
```

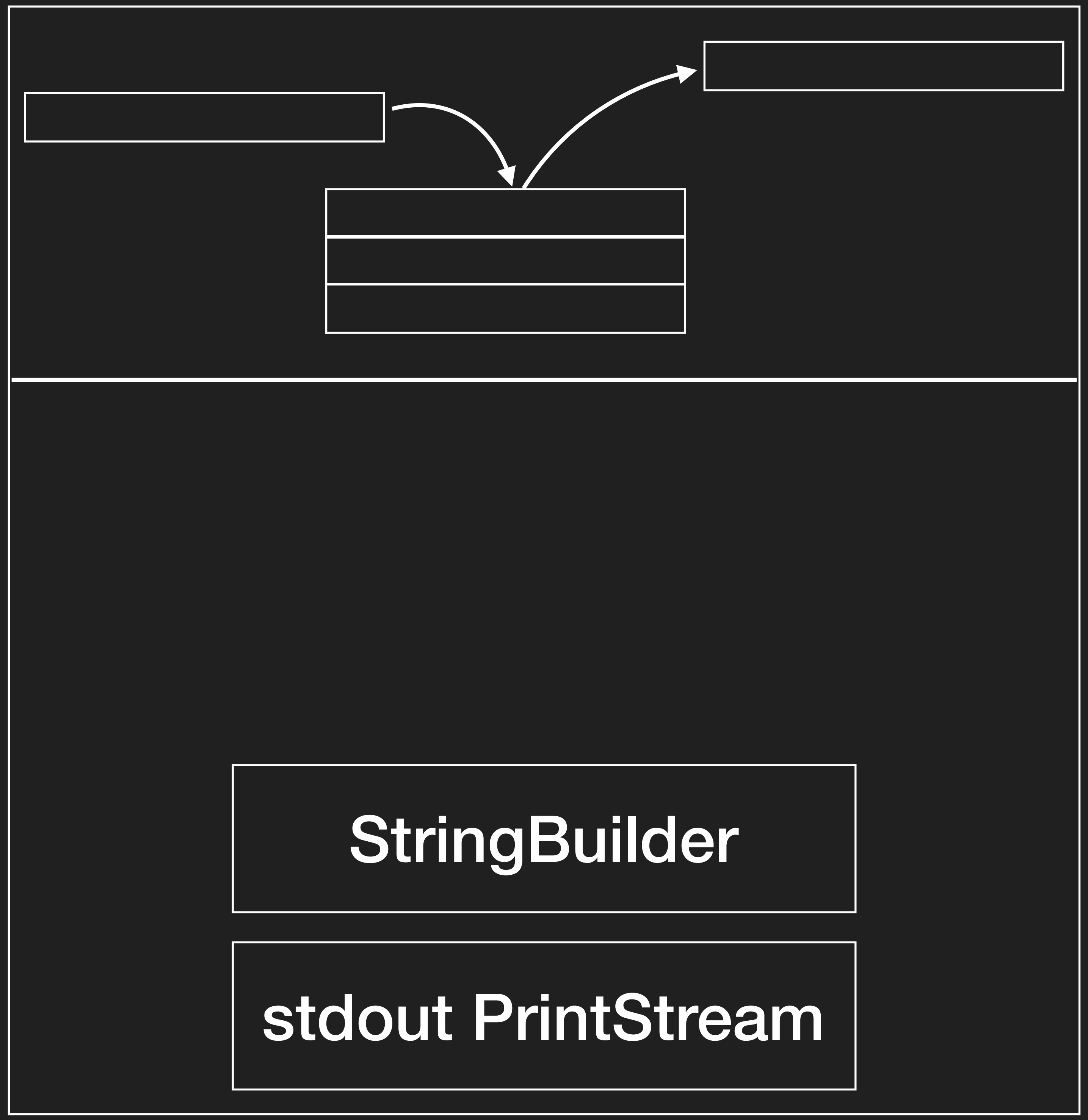
```
INVOKESTATIC myapp/RunnerKt.v1 ()I
INVOKESTATIC myapp/RunnerKt.v2 ()I
IADD
ISTORE 1
GETSTATIC j/l/System.out : Lj/io/PrintStream;
NEW j/l/StringBuilder
DUP
INVOKESPECIAL j/l/StringBuilder.<init> ()V
LDC "sum of values was "
INVOKEVIRTUAL j/l/StringBuilder.append (Lj/l/String;)Lj/l/StringBuilder;
ILOAD 1
INVOKEVIRTUAL j/l/StringBuilder.append (I)Lj/l/StringBuilder;
INVOKEVIRTUAL j/l/StringBuilder.toString ()Lj/l/String;
INVOKEVIRTUAL j/io/PrintStream.println (Lj/l/String;)V
```



What does bytecode look like?

```
fun printSimpleSum() {
  val sum = v1() + v2()
  println("sum of values was $sum")
}
```

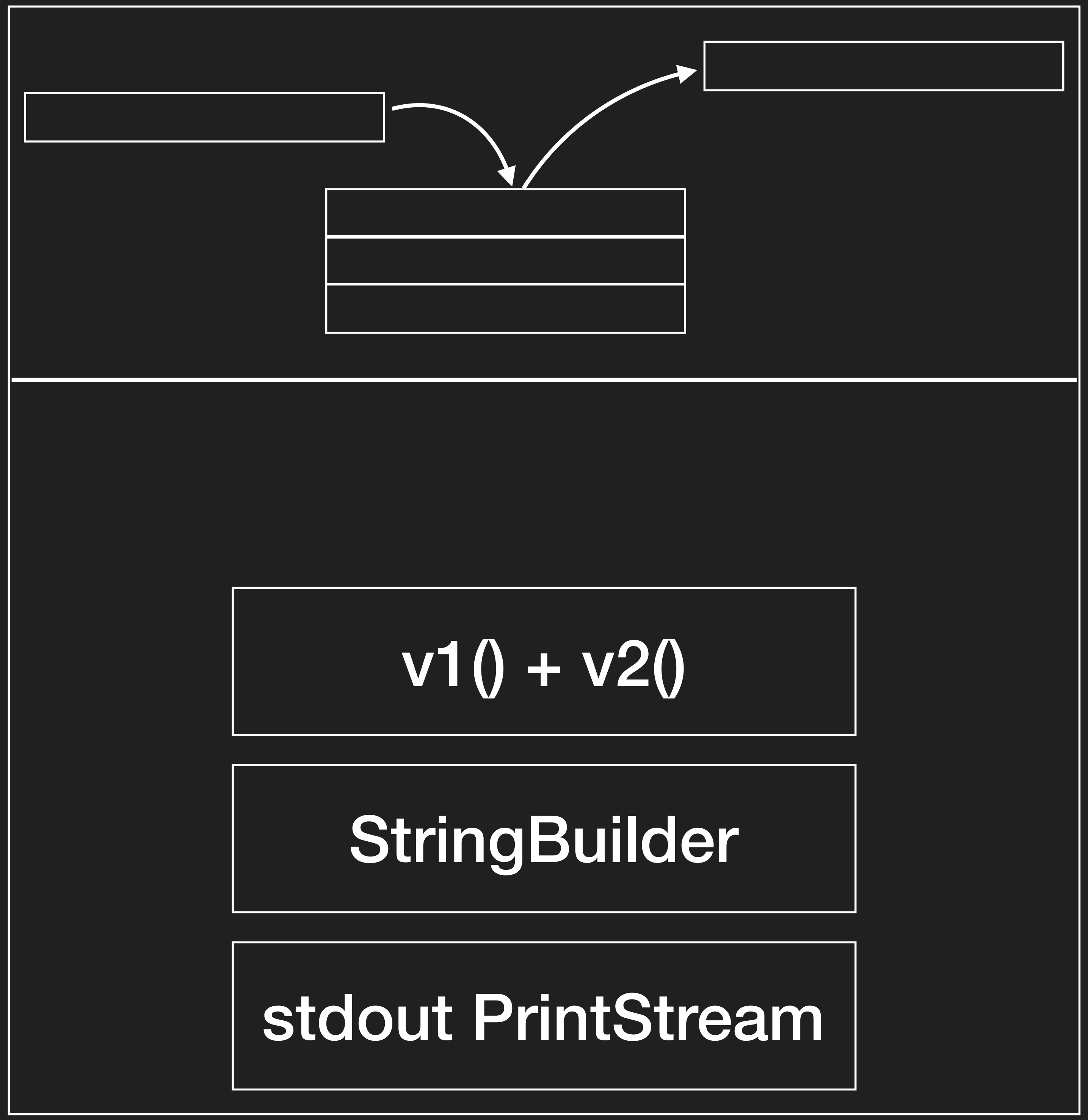
```
INVOKESTATIC myapp/RunnerKt.v1 ()I
INVOKESTATIC myapp/RunnerKt.v2 ()I
IADD
ISTORE 1
GETSTATIC j/l/System.out : Lj/io/PrintStream;
NEW j/l/StringBuilder
DUP
INVOKESPECIAL j/l/StringBuilder.<init> ()V
LDC "sum of values was "
INVOKEVIRTUAL j/l/StringBuilder.append (Lj/l/String;)Lj/l/StringBuilder;
ILOAD 1
INVOKEVIRTUAL j/l/StringBuilder.append (I)Lj/l/StringBuilder;
INVOKEVIRTUAL j/l/StringBuilder.toString ()Lj/l/String;
INVOKEVIRTUAL j/io/PrintStream.println (Lj/l/String;)V
```



What does bytecode look like?

```
fun printSimpleSum() {
  val sum = v1() + v2()
  println("sum of values was $sum")
}
```

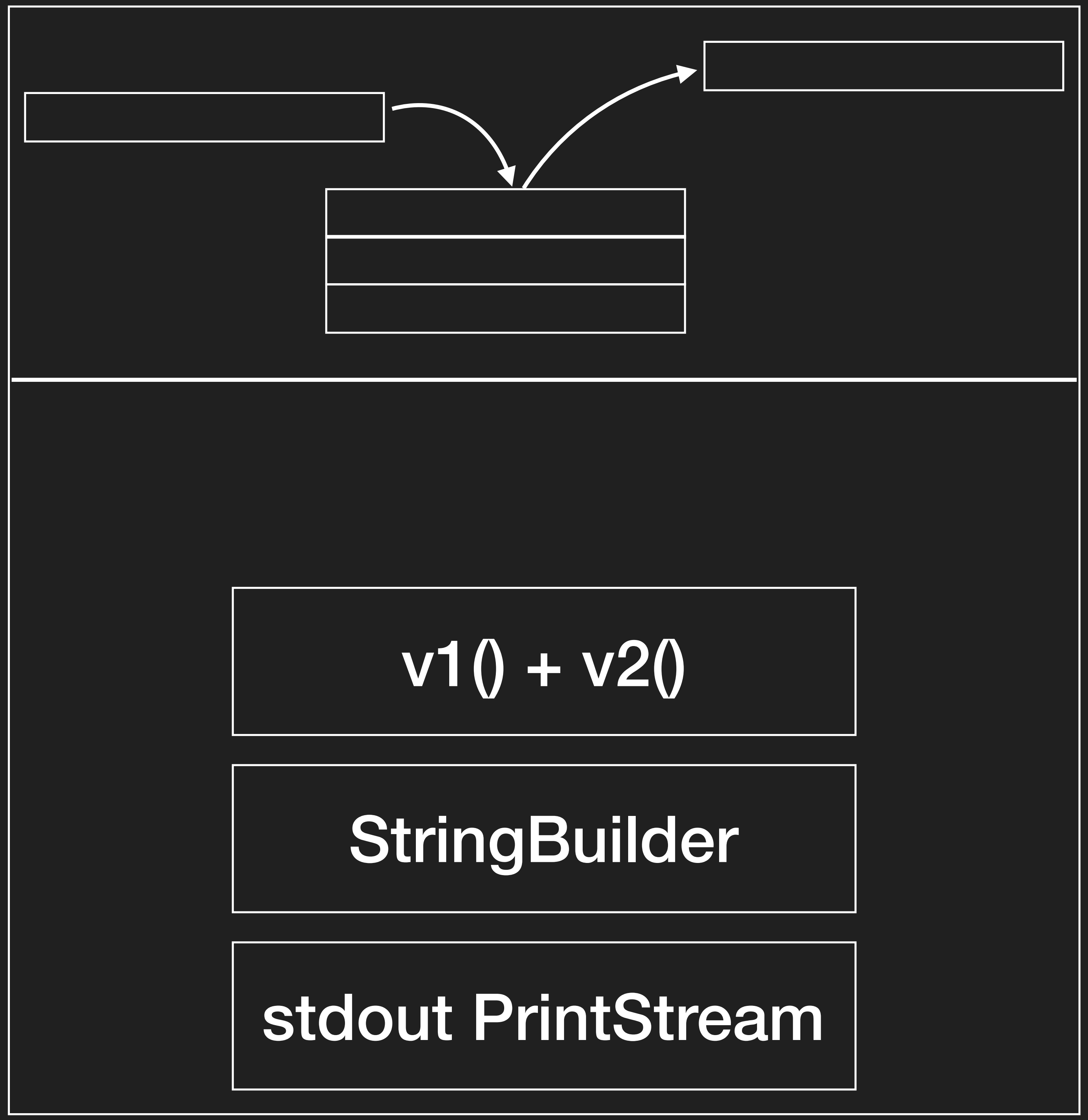
```
INVOKESTATIC myapp/RunnerKt.v1 ()I
INVOKESTATIC myapp/RunnerKt.v2 ()I
IADD
ISTORE 1
GETSTATIC j/l/System.out : Lj/io/PrintStream;
NEW j/l/StringBuilder
DUP
INVOKESPECIAL j/l/StringBuilder.<init> ()V
LDC "sum of values was "
INVOKEVIRTUAL j/l/StringBuilder.append (Lj/l/String;)Lj/l/StringBuilder;
ILOAD 1
INVOKEVIRTUAL j/l/StringBuilder.append (I)Lj/l/StringBuilder;
INVOKEVIRTUAL j/l/StringBuilder.toString ()Lj/l/String;
INVOKEVIRTUAL j/io/PrintStream.println (Lj/l/String;)V
```



What does bytecode look like?

```
fun printSimpleSum() {
  val sum = v1() + v2()
  println("sum of values was $sum")
}
```

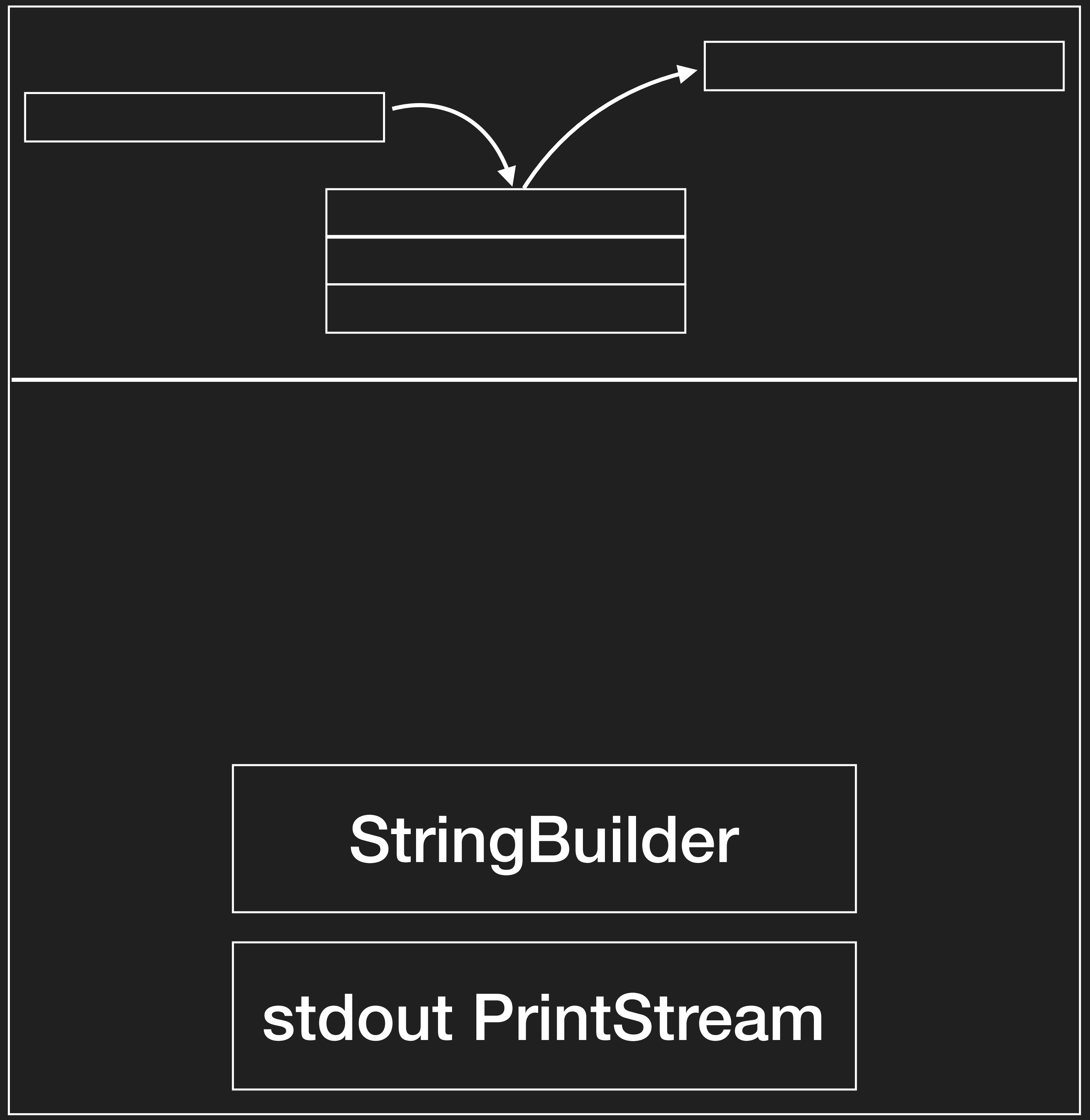
```
INVOKESTATIC myapp/RunnerKt.v1 ()I
INVOKESTATIC myapp/RunnerKt.v2 ()I
IADD
ISTORE 1
GETSTATIC j/l/System.out : Lj/io/PrintStream;
NEW j/l/StringBuilder
DUP
INVOKESPECIAL j/l/StringBuilder.<init> ()V
LDC "sum of values was "
INVOKEVIRTUAL j/l/StringBuilder.append (Lj/l/String;)Lj/l/StringBuilder;
ILOAD 1
INVOKEVIRTUAL j/l/StringBuilder.append (I)Lj/l/StringBuilder;
INVOKEVIRTUAL j/l/StringBuilder.toString ()Lj/l/String;
INVOKEVIRTUAL j/io/PrintStream.println (Lj/l/String;)V
```



What does bytecode look like?

```
fun printSimpleSum() {
  val sum = v1() + v2()
  println("sum of values was $sum")
}
```

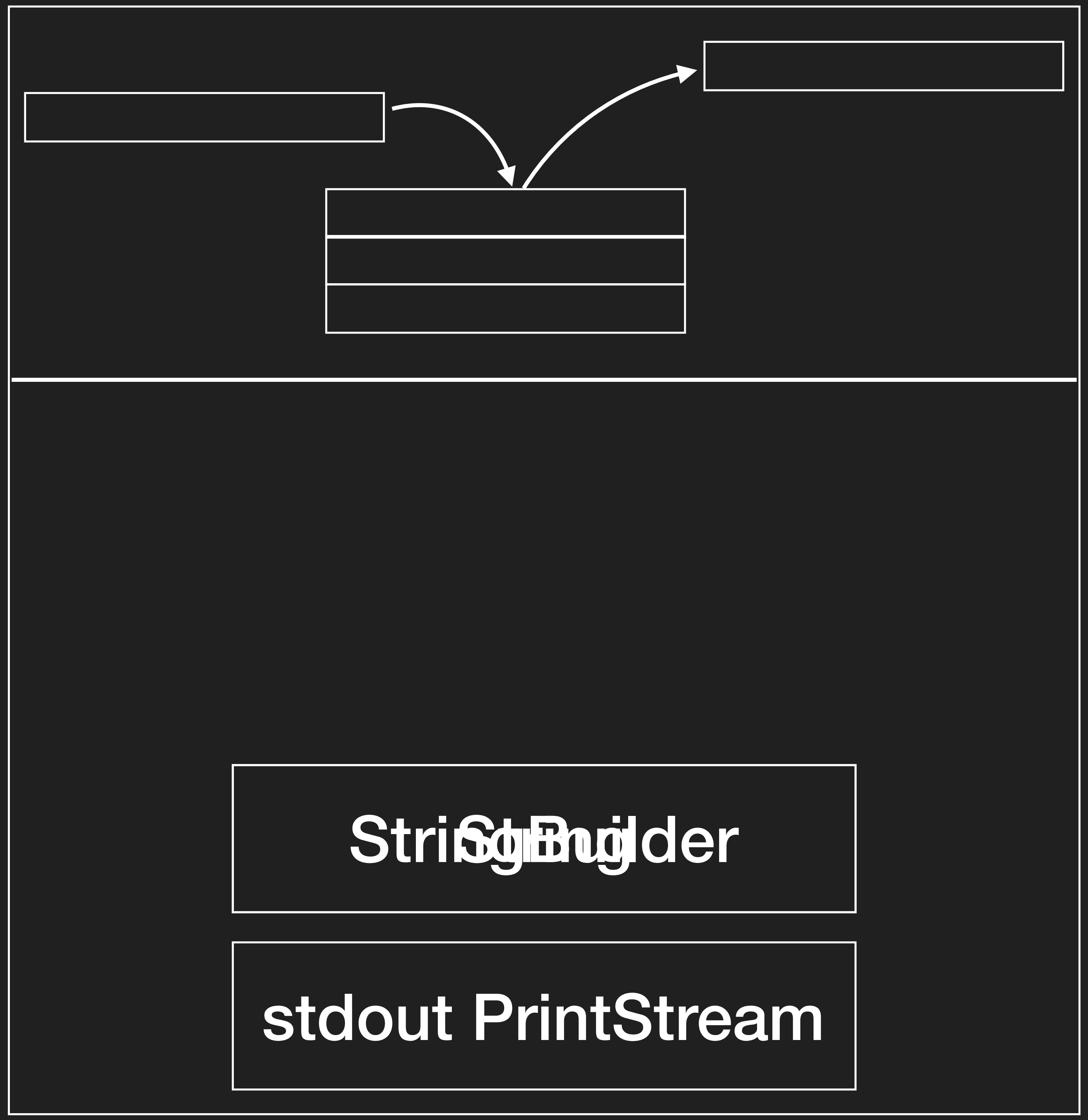
```
INVOKESTATIC myapp/RunnerKt.v1 ()I
INVOKESTATIC myapp/RunnerKt.v2 ()I
IADD
ISTORE 1
GETSTATIC j/l/System.out : Lj/io/PrintStream;
NEW j/l/StringBuilder
DUP
INVOKESPECIAL j/l/StringBuilder.<init> ()V
LDC "sum of values was "
INVOKEVIRTUAL j/l/StringBuilder.append (Lj/l/String;)Lj/l/StringBuilder;
ILOAD 1
INVOKEVIRTUAL j/l/StringBuilder.append (I)Lj/l/StringBuilder;
INVOKEVIRTUAL j/l/StringBuilder.toString ()Lj/l/String;
INVOKEVIRTUAL j/io/PrintStream.println (Lj/l/String;)V
```



What does bytecode look like?

```
fun printSimpleSum() {
  val sum = v1() + v2()
  println("sum of values was $sum")
}
```

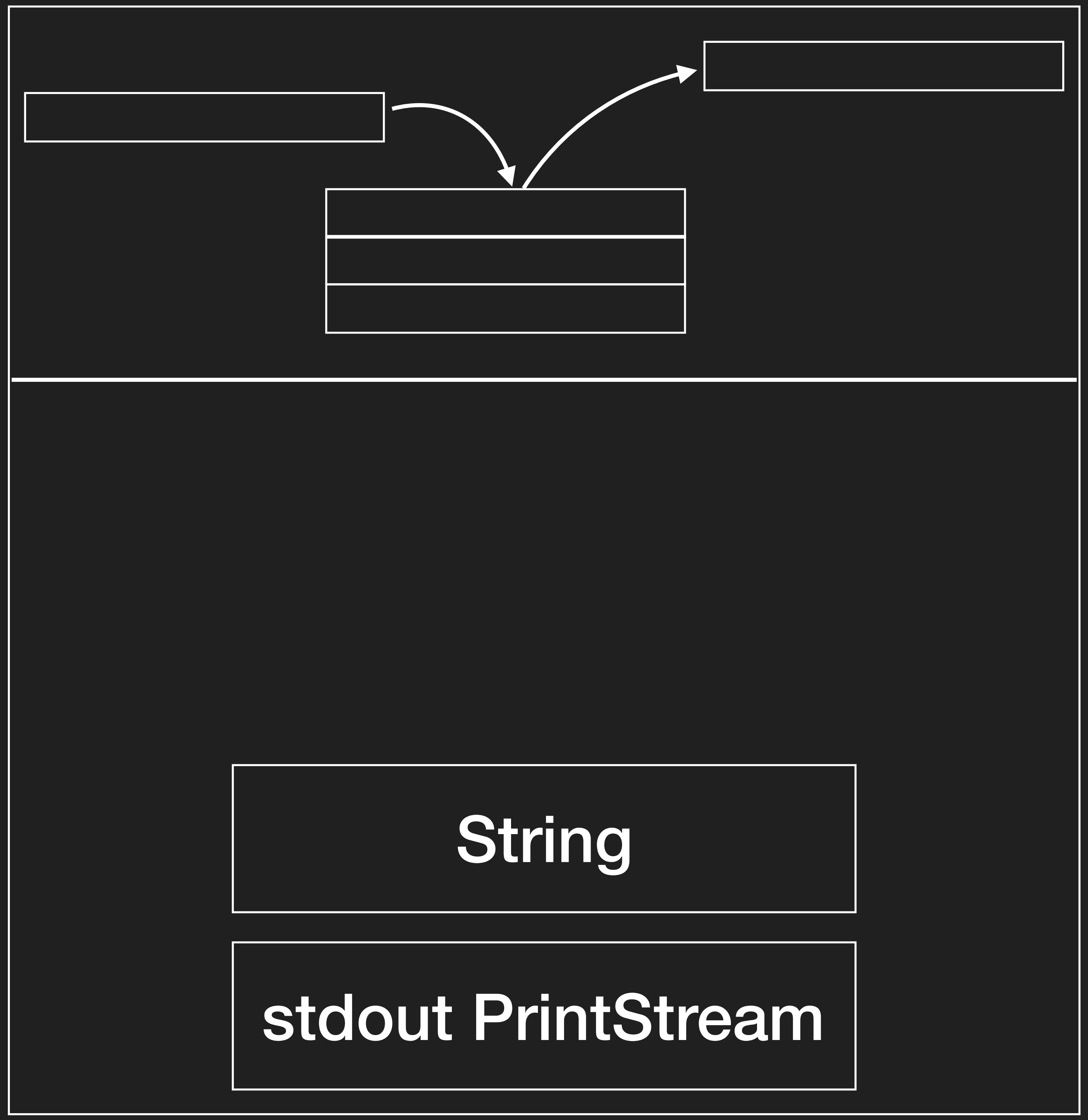
```
INVOKESTATIC myapp/RunnerKt.v1 ()I
INVOKESTATIC myapp/RunnerKt.v2 ()I
IADD
ISTORE 1
GETSTATIC j/l/System.out : Lj/io/PrintStream;
NEW j/l/StringBuilder
DUP
INVOKESPECIAL j/l/StringBuilder.<init> ()V
LDC "sum of values was "
INVOKEVIRTUAL j/l/StringBuilder.append (Lj/l/String;)Lj/l/StringBuilder;
ILOAD 1
INVOKEVIRTUAL j/l/StringBuilder.append (I)Lj/l/StringBuilder;
INVOKEVIRTUAL j/l/StringBuilder.toString ()Lj/l/String;
INVOKEVIRTUAL j/io/PrintStream.println (Lj/l/String;)V
```



What does bytecode look like?

```
fun printSimpleSum() {
  val sum = v1() + v2()
  println("sum of values was $sum")
}
```

```
INVOKESTATIC myapp/RunnerKt.v1 ()I
INVOKESTATIC myapp/RunnerKt.v2 ()I
IADD
ISTORE 1
GETSTATIC j/l/System.out : Lj/io/PrintStream;
NEW j/l/StringBuilder
DUP
INVOKESPECIAL j/l/StringBuilder.<init> ()V
LDC "sum of values was "
INVOKEVIRTUAL j/l/StringBuilder.append (Lj/l/String;)Lj/l/StringBuilder;
ILOAD 1
INVOKEVIRTUAL j/l/StringBuilder.append (I)Lj/l/StringBuilder;
INVOKEVIRTUAL j/l/StringBuilder.toString ()Lj/l/String;
INVOKEVIRTUAL j/io/PrintStream.println (Lj/l/String;)V
```



Remember the goal

```
fun prime(n: Int): Long {
  println("⇢ prime(n=$n)")
  val startTime = System.currentTimeMillis()
  val result = primeNumberSequence.take(n).last()
  val timeToRun = System.currentTimeMillis() - startTime
  println("⇠ prime [ran in $timeToRun ms]")
  return result
}
```

```
@DebugLog fun prime(n: Int): Long = primeNumberSequence.take(n).last()
```

Back to our MethodVisitor!

kotlin-plugin/src/main/kotlin/debuglog/DebugLogClassBuilder.kt

```kotlin
return object : MethodVisitor(Opcodes.ASM5, original) {
  override fun visitCode() {
    super.visitCode()
    InstructionAdapter(this).apply { TODO("on method entry") }
  }
  override fun visitInsn(opcode: Int) {
    when (opcode) {
      RETURN , ARETURN, IRETURN -> {
        InstructionAdapter(this).apply { TODO("on method exit") }
      }
    }
    super.visitInsn(opcode)
  }
}
```

kotlin-plugin/src/main/kotlin/debuglog/DebugLogClassBuilder.kt

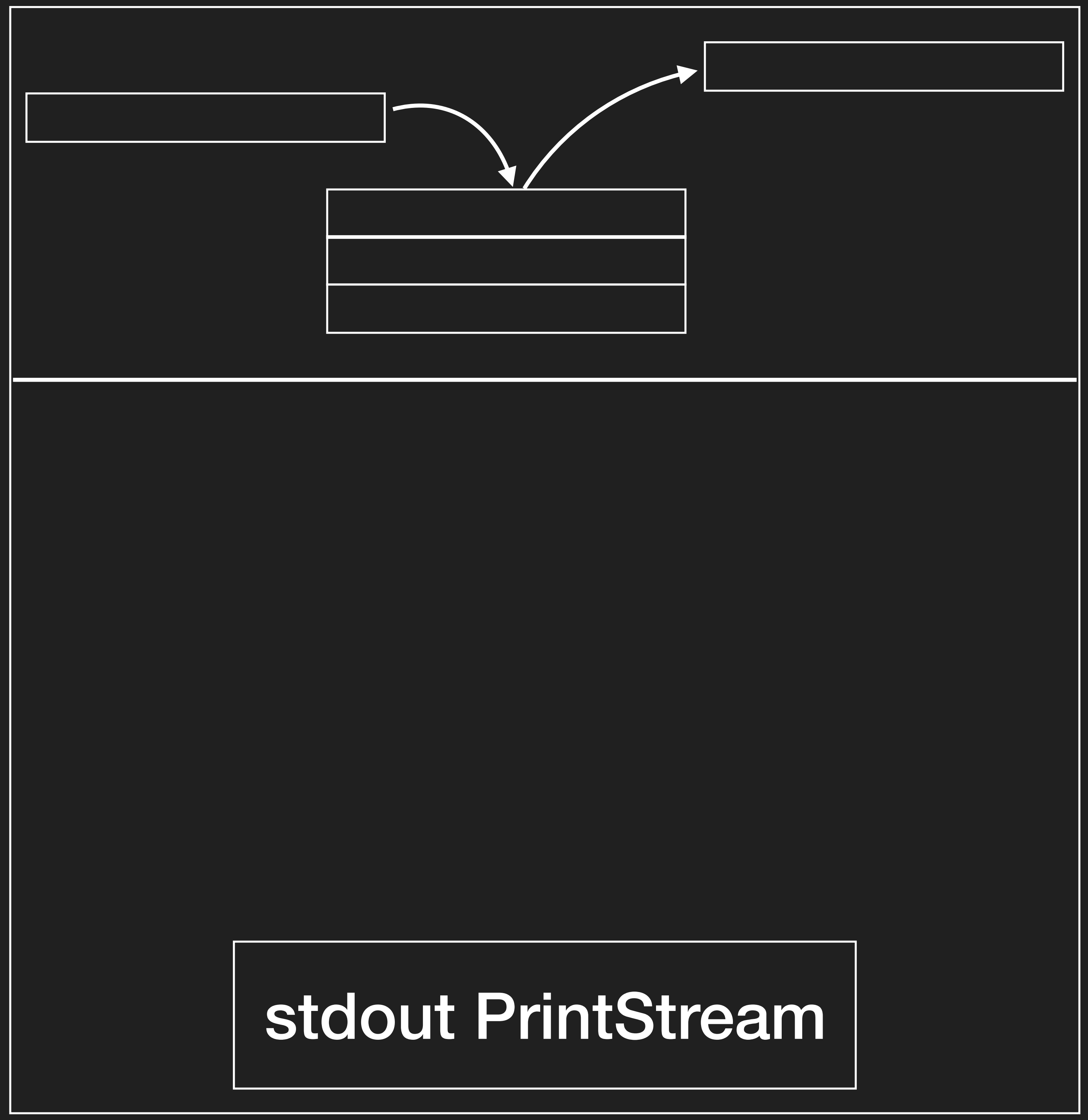
```
return object : MethodVisitor(Opcodes.ASM5, original) {
  override fun visitCode() {
    super.visitCode()
    InstructionAdapter(this).apply { TODO("on method entry") }
  }
  override fun visitInsn(opcode: Int) {
    when (opcode) {
      RETURN , ARETURN, IRETURN -> {
        InstructionAdapter(this).apply { TODO("on method exit") }
      }
    }
    super.visitInsn(opcode)
  }
}
```

kotlin-plugin/src/main/kotlin/debuglog/DebugLogClassBuilder.kt

```
InstructionAdapter(this).apply {
TODO("on method entry")
}
```

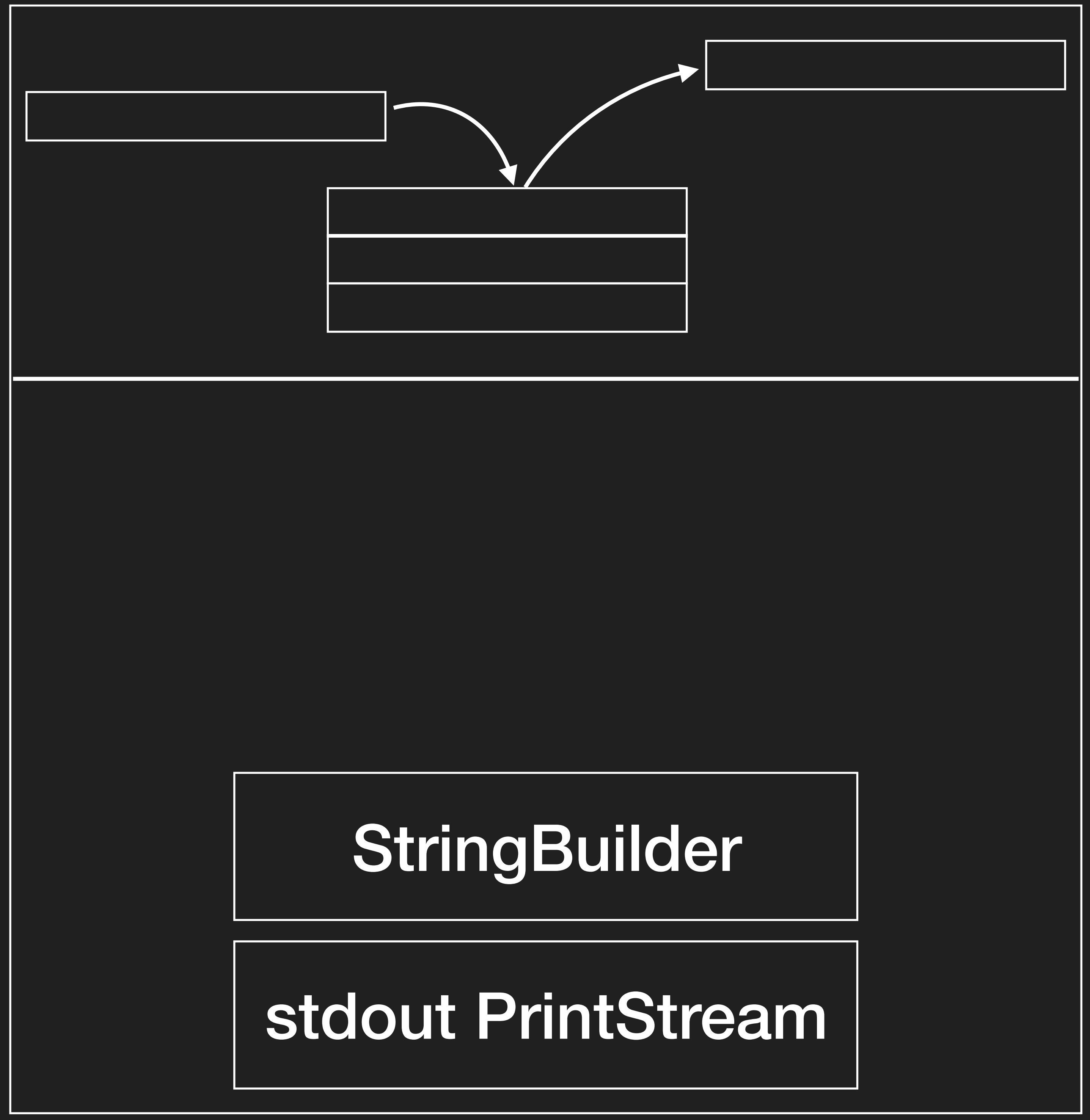
kotlin-plugin/src/main/kotlin/debuglog/DebugLogClassBuilder.kt

```
InstructionAdapter(this).apply {
getstatic("j/l/System", "out", "Ljava/io/PrintStream;")
}
```



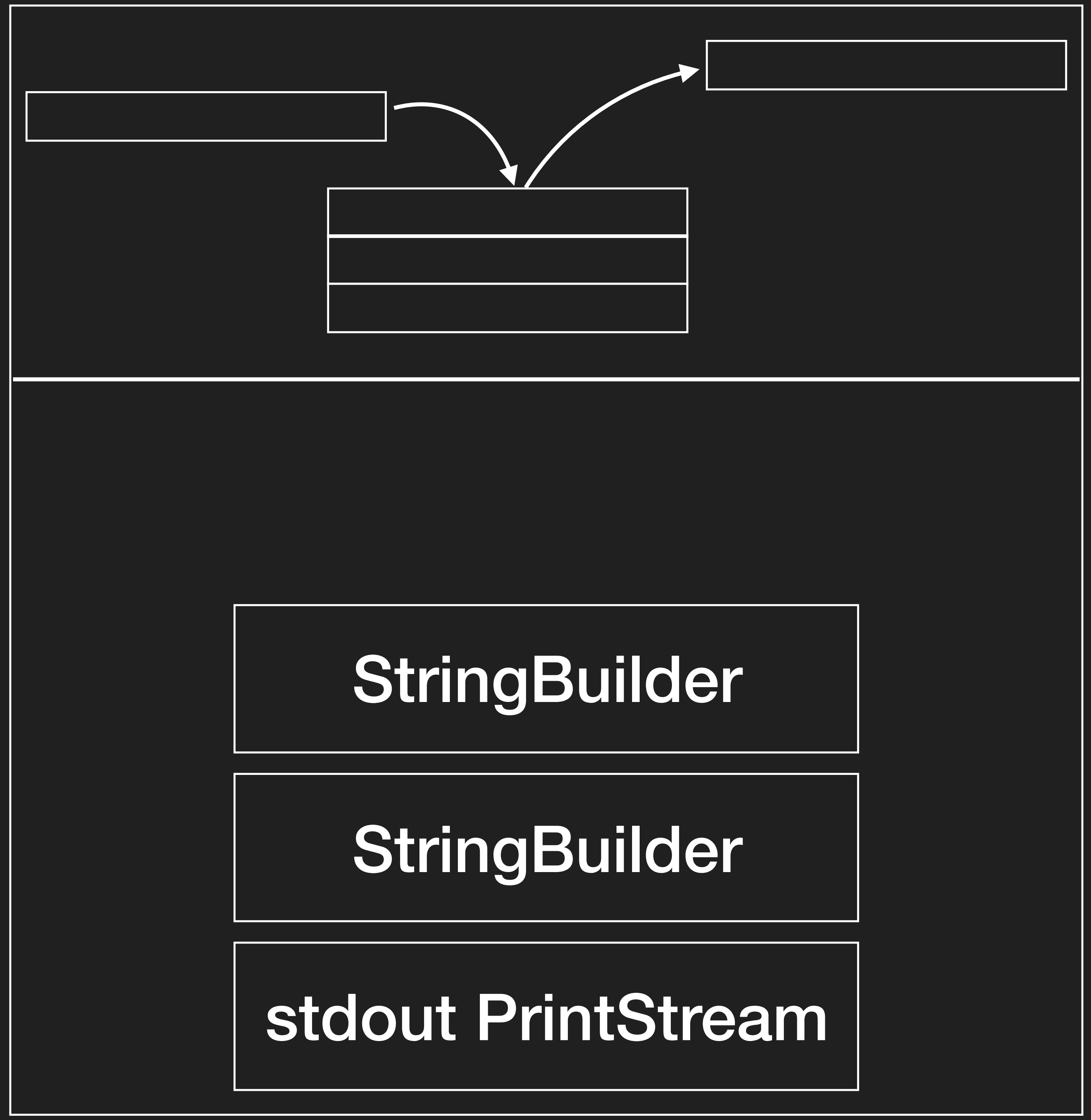
kotlin-plugin/src/main/kotlin/debuglog/DebugLogClassBuilder.kt

```
InstructionAdapter(this).apply {
getstatic("j/l/System", "out", "Ljava/io/PrintStream;")
anew("j/l/StringBuilder")
}
```



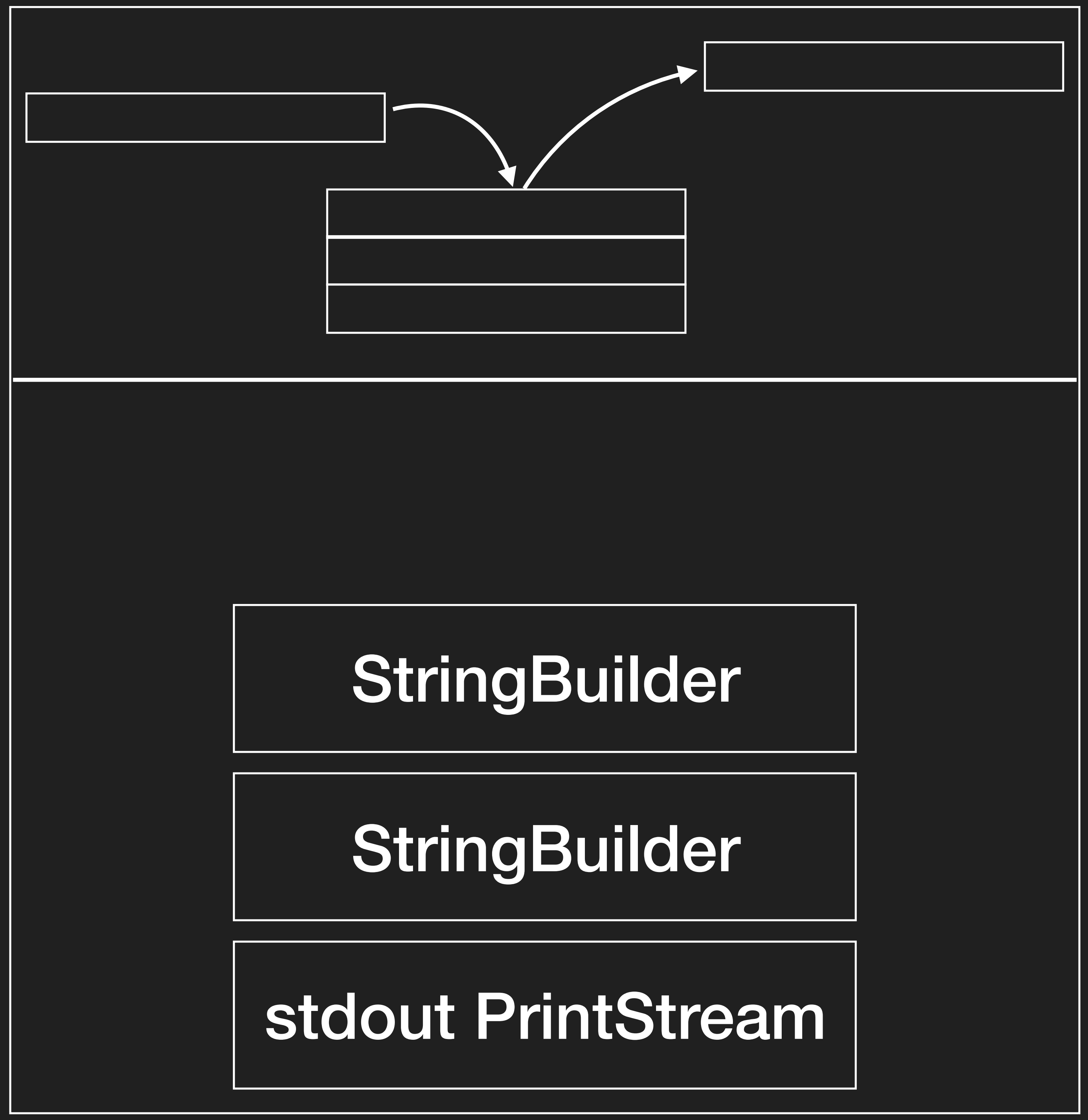
kotlin-plugin/src/main/kotlin/debuglog/DebugLogClassBuilder.kt

```
InstructionAdapter(this).apply {
getstatic("j/l/System", "out", "Ljava/io/PrintStream;")
anew("j/l/StringBuilder")
dup()
}
```



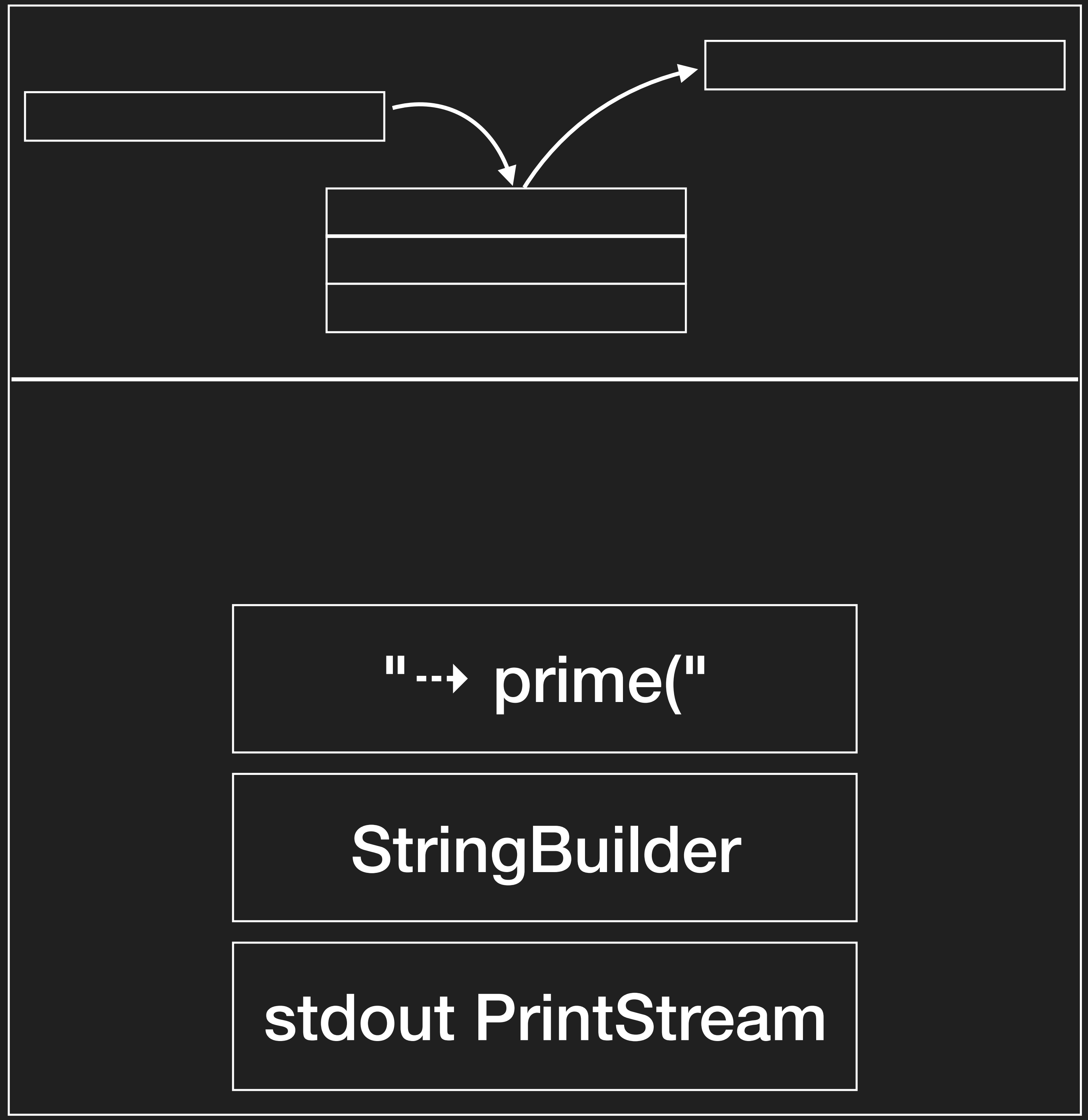
kotlin-plugin/src/main/kotlin/debuglog/DebugLogClassBuilder.kt

```
InstructionAdapter(this).apply {
getstatic("j/l/System", "out", "Ljava/io/PrintStream;")
anew("j/l/StringBuilder")
dup()
invokespecial("j/l/StringBuilder", "<init>", "()V")
}
```



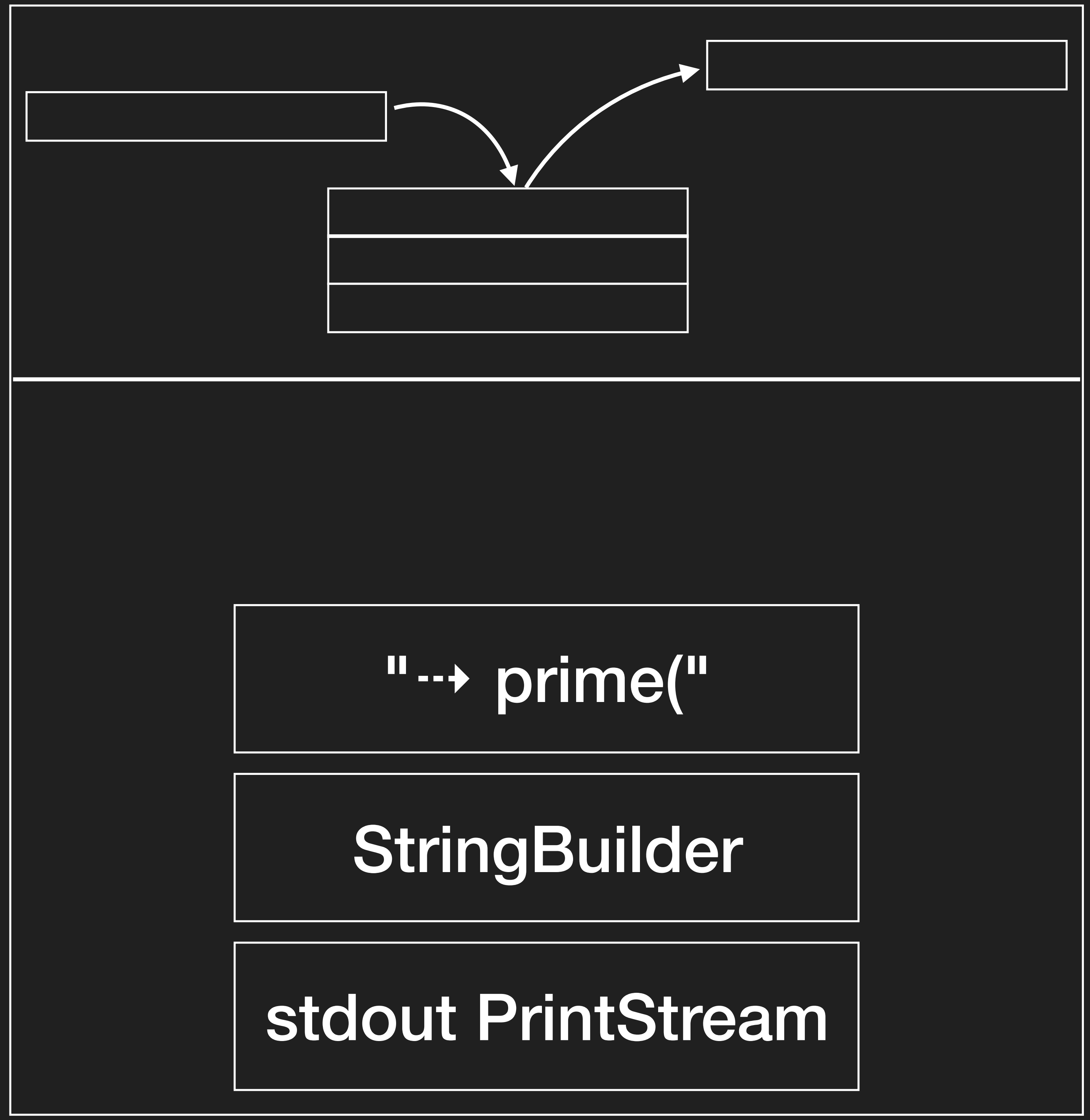
kotlin-plugin/src/main/kotlin/debuglog/DebugLogClassBuilder.kt

```
InstructionAdapter(this).apply {
getstatic("j/l/System", "out", "Ljava/io/PrintStream;")
anew("j/l/StringBuilder")
dup()
invokespecial("j/l/StringBuilder", "<init>", "()V")
visitLdcInsn("⇢ ${function.name}(")
}
```



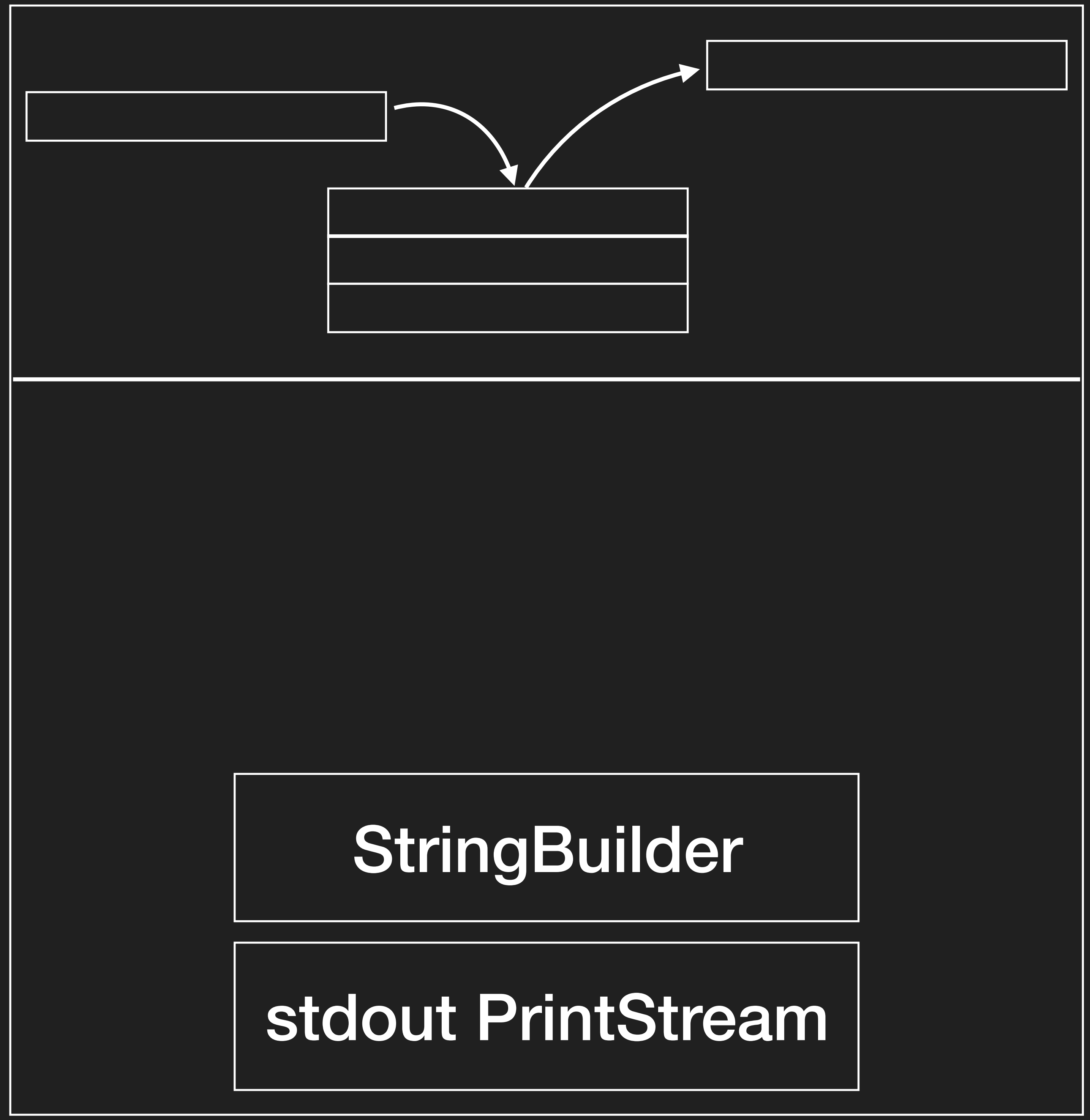
kotlin-plugin/src/main/kotlin/debuglog/DebugLogClassBuilder.kt

```
InstructionAdapter(this).apply {
getstatic("j/l/System", "out", "Ljava/io/PrintStream;")
anew("j/l/StringBuilder")
dup()
invokespecial("j/l/StringBuilder", "<init>", "()V")
visitLdcInsn("⇢ ${function.name}(")
invokevirtual("j/l/StringBuilder", "append",
    "(Lj/l/Object;)Lj/l/StringBuilder;")
}
```



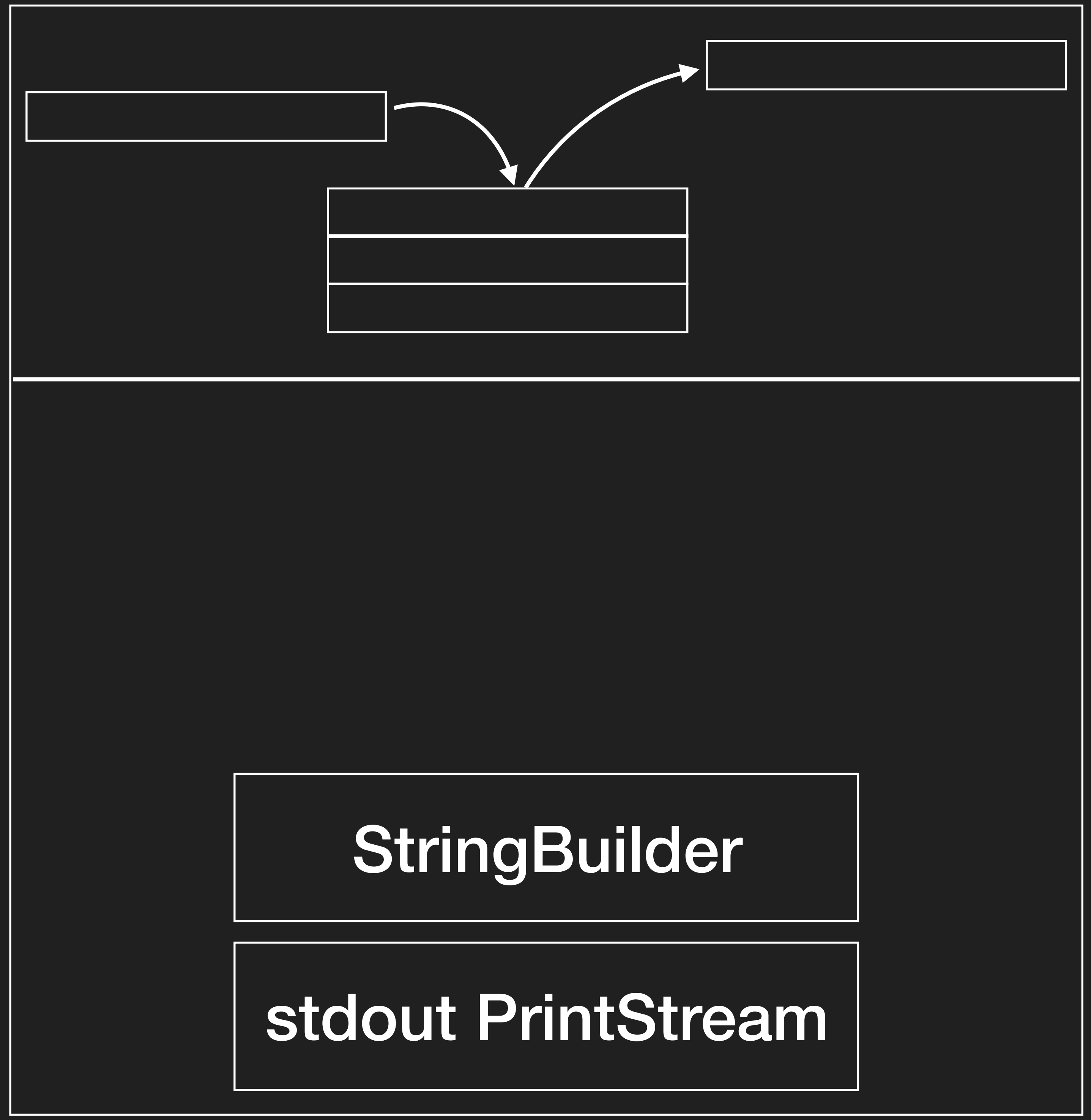
kotlin-plugin/src/main/kotlin/debuglog/DebugLogClassBuilder.kt

```
InstructionAdapter(this).apply {
getstatic("j/l/System", "out", "Ljava/io/PrintStream;")
anew("j/l/StringBuilder")
dup()
invokespecial("j/l/StringBuilder", "<init>", "()V")
visitLdcInsn("⇢ ${function.name}(")
invokevirtual("j/l/StringBuilder", "append",
    "(Lj/l/Object;)Lj/l/StringBuilder;")
}
```



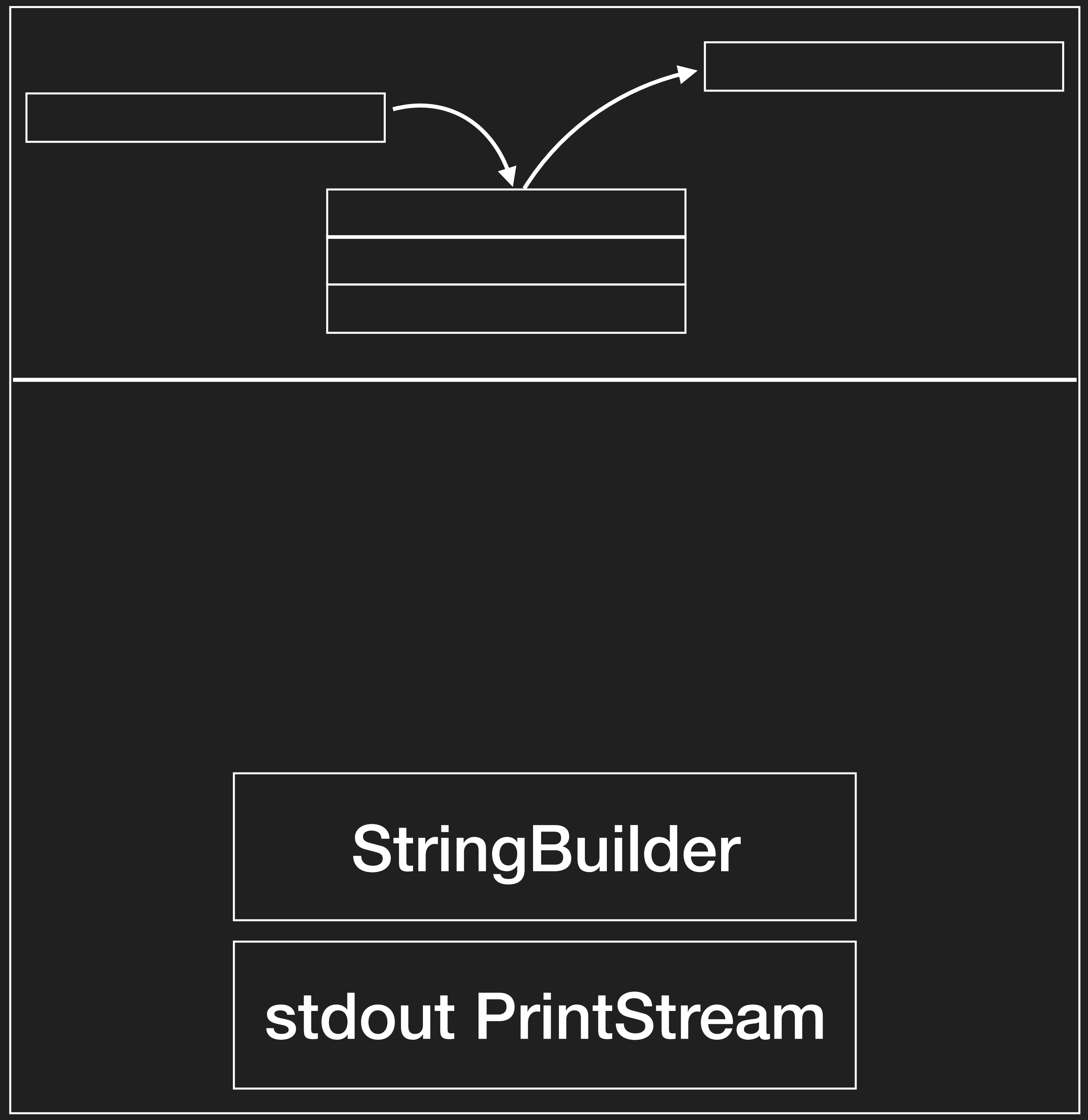
kotlin-plugin/src/main/kotlin/debuglog/DebugLogClassBuilder.kt

```
InstructionAdapter(this).apply {
getstatic("j/l/System", "out", "Ljava/io/PrintStream;")
anew("j/l/StringBuilder")
dup()
invokespecial("j/l/StringBuilder", "<init>", "()V")
visitLdcInsn("⇢ ${function.name}(")
invokevirtual("j/l/StringBuilder", "append",
    "(Lj/l/Object;)Lj/l/StringBuilder;")
}
```



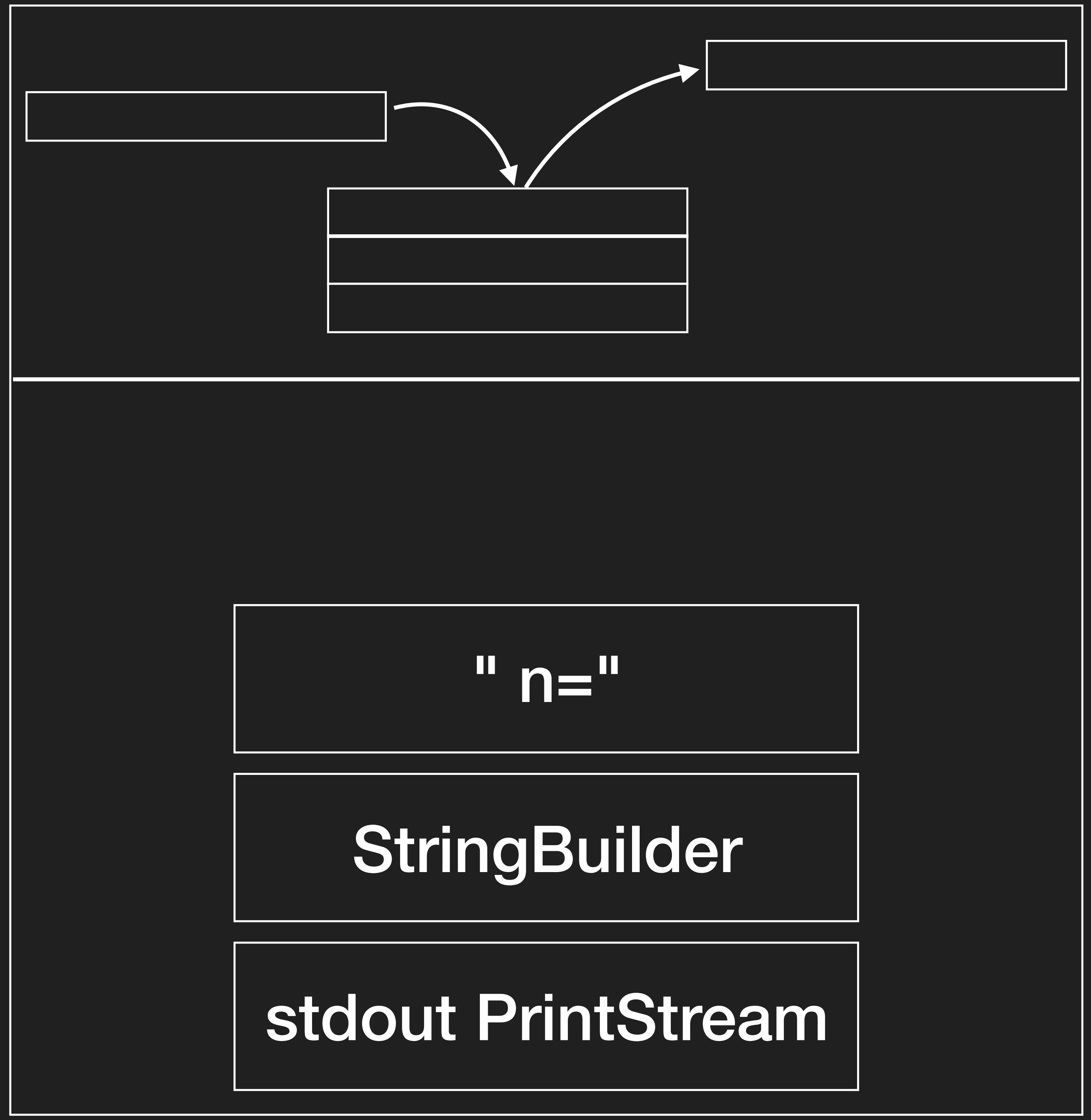
kotlin-plugin/src/main/kotlin/debuglog/DebugLogClassBuilder.kt

```
InstructionAdapter(this).apply {
getstatic("j/l/System", "out", "Ljava/io/PrintStream;")
anew("j/l/StringBuilder")
dup()
invokespecial("j/l/StringBuilder", "<init>", "()V")
visitLdcInsn("⇢ ${function.name}(")
invokevirtual("j/l/StringBuilder", "append",
   "(Lj/l/Object;)Lj/l/StringBuilder;")
function.valueParameters.forEachIndexed { i, param ->
  visitLdcInsn(" ${param.name}=")
  invokevirtual("j/l/StringBuilder", "append", "(Lj/l/String;)Lj/l/SB;")
  visitVarInsn(ALOAD, i + 1)
  invokevirtual("j/l/StringBuilder", "append", "(Lj/l/Object;)Lj/l/SB;")
}
}
```



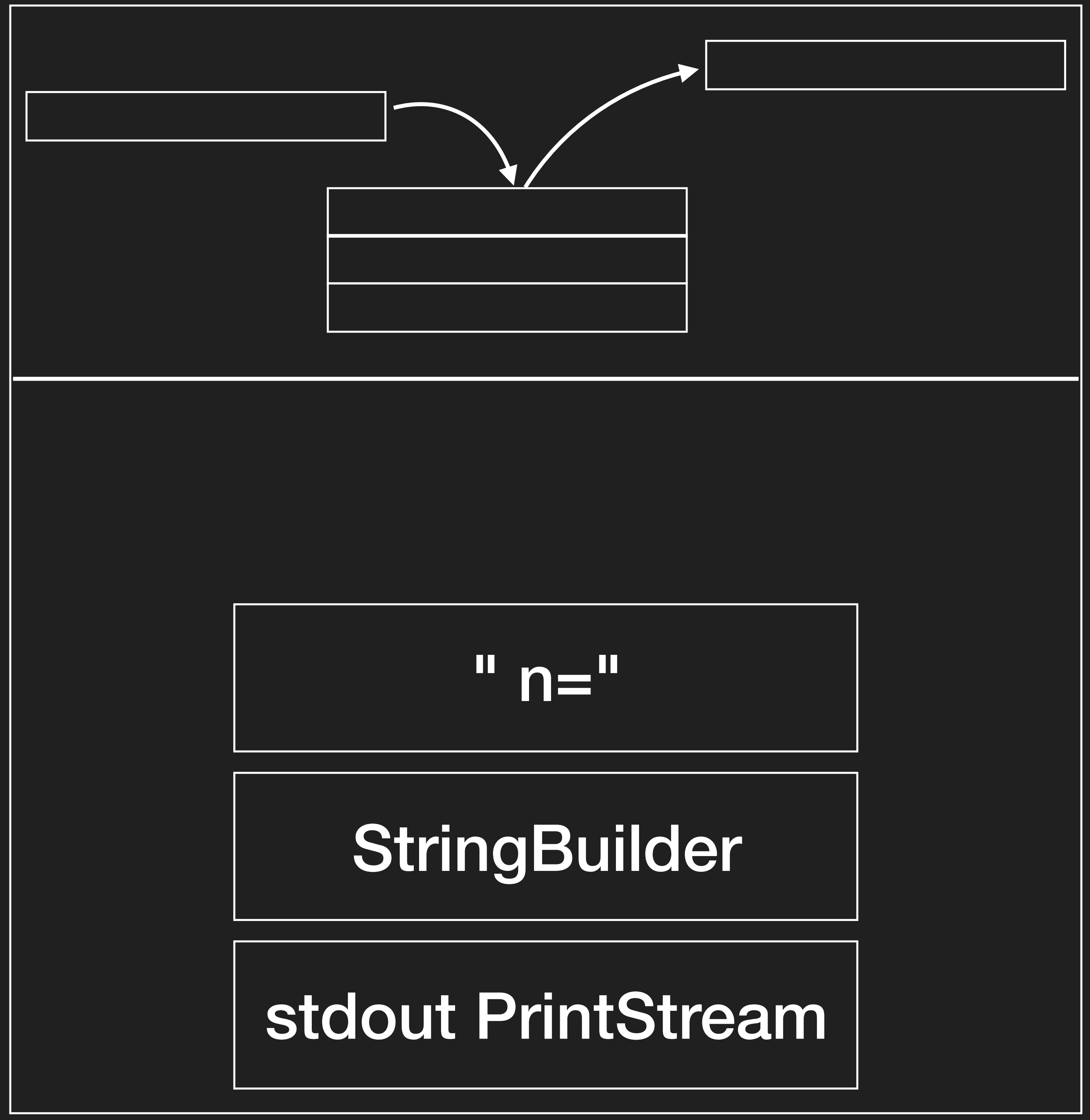
kotlin-plugin/src/main/kotlin/debuglog/DebugLogClassBuilder.kt

```
InstructionAdapter(this).apply {
getstatic("j/l/System", "out", "Ljava/io/PrintStream;")
anew("j/l/StringBuilder")
dup()
invokespecial("j/l/StringBuilder", "<init>", "()V")
visitLdcInsn("⇢ ${function.name}(")
invokevirtual("j/l/StringBuilder", "append",
    "(Lj/l/Object;)Lj/l/StringBuilder;")
function.valueParameters.forEachIndexed { i, param ->
  visitLdcInsn(" ${param.name}=")
  invokevirtual("j/l/StringBuilder", "append", "(Lj/l/String;)Lj/l/SB;")
  visitVarInsn(ALOAD, i + 1)
  invokevirtual("j/l/StringBuilder", "append", "(Lj/l/Object;)Lj/l/SB;")
}
}
```



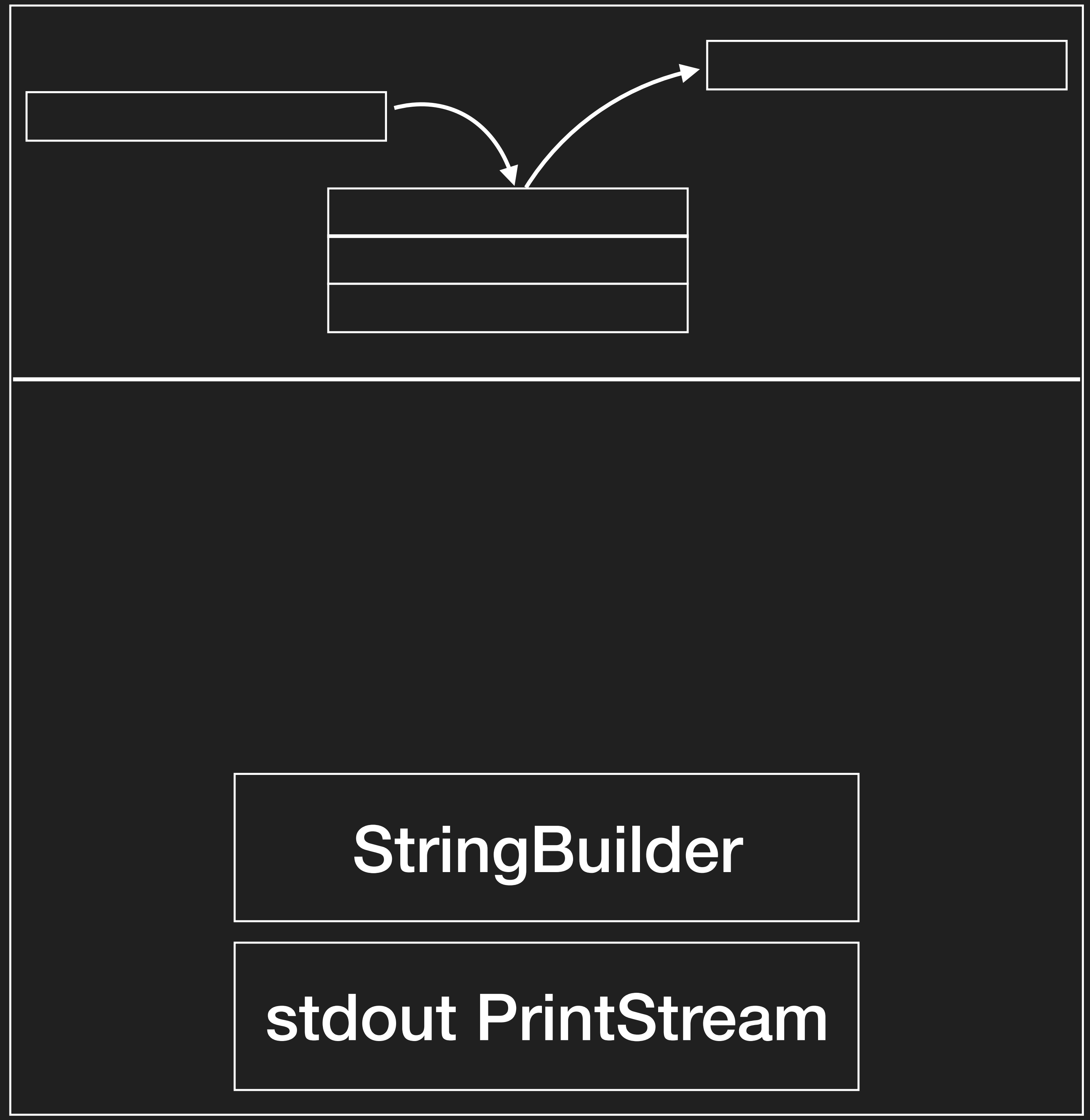
kotlin-plugin/src/main/kotlin/debuglog/DebugLogClassBuilder.kt

```
InstructionAdapter(this).apply {
getstatic("j/l/System", "out", "Ljava/io/PrintStream;")
anew("j/l/StringBuilder")
dup()
invokespecial("j/l/StringBuilder", "<init>", "()V")
visitLdcInsn("⇢ ${function.name}(")
invokevirtual("j/l/StringBuilder", "append",
   "(Lj/l/Object;)Lj/l/StringBuilder;")
function.valueParameters.forEachIndexed { i, param ->
  visitLdcInsn(" ${param.name}=")
  invokevirtual("j/l/StringBuilder", "append", "(Lj/l/String;)Lj/l/SB;")
  visitVarInsn(ALOAD, i + 1)
  invokevirtual("j/l/StringBuilder", "append", "(Lj/l/Object;)Lj/l/SB;")
}
}
```



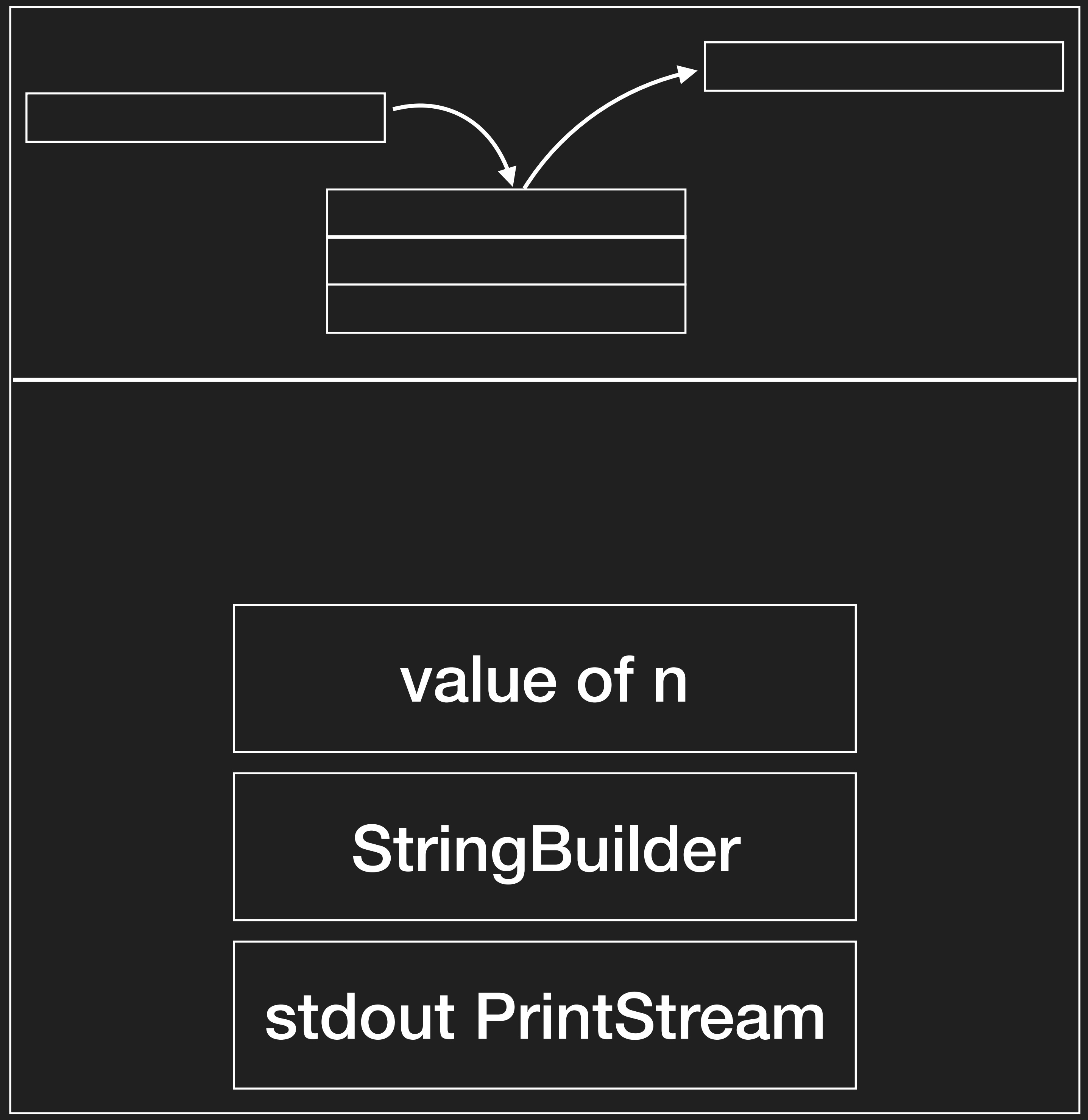
kotlin-plugin/src/main/kotlin/debuglog/DebugLogClassBuilder.kt

```
InstructionAdapter(this).apply {
getstatic("j/l/System", "out", "Ljava/io/PrintStream;")
anew("j/l/StringBuilder")
dup()
invokespecial("j/l/StringBuilder", "<init>", "()V")
visitLdcInsn("⇢ ${function.name}(")
invokevirtual("j/l/StringBuilder", "append",
    "(Lj/l/Object;)Lj/l/StringBuilder;")
function.valueParameters.forEachIndexed { i, param ->
  visitLdcInsn(" ${param.name}=")
  invokevirtual("j/l/StringBuilder", "append", "(Lj/l/String;)Lj/l/SB;")
  visitVarInsn(ALOAD, i + 1)
  invokevirtual("j/l/StringBuilder", "append", "(Lj/l/Object;)Lj/l/SB;")
}
}
```



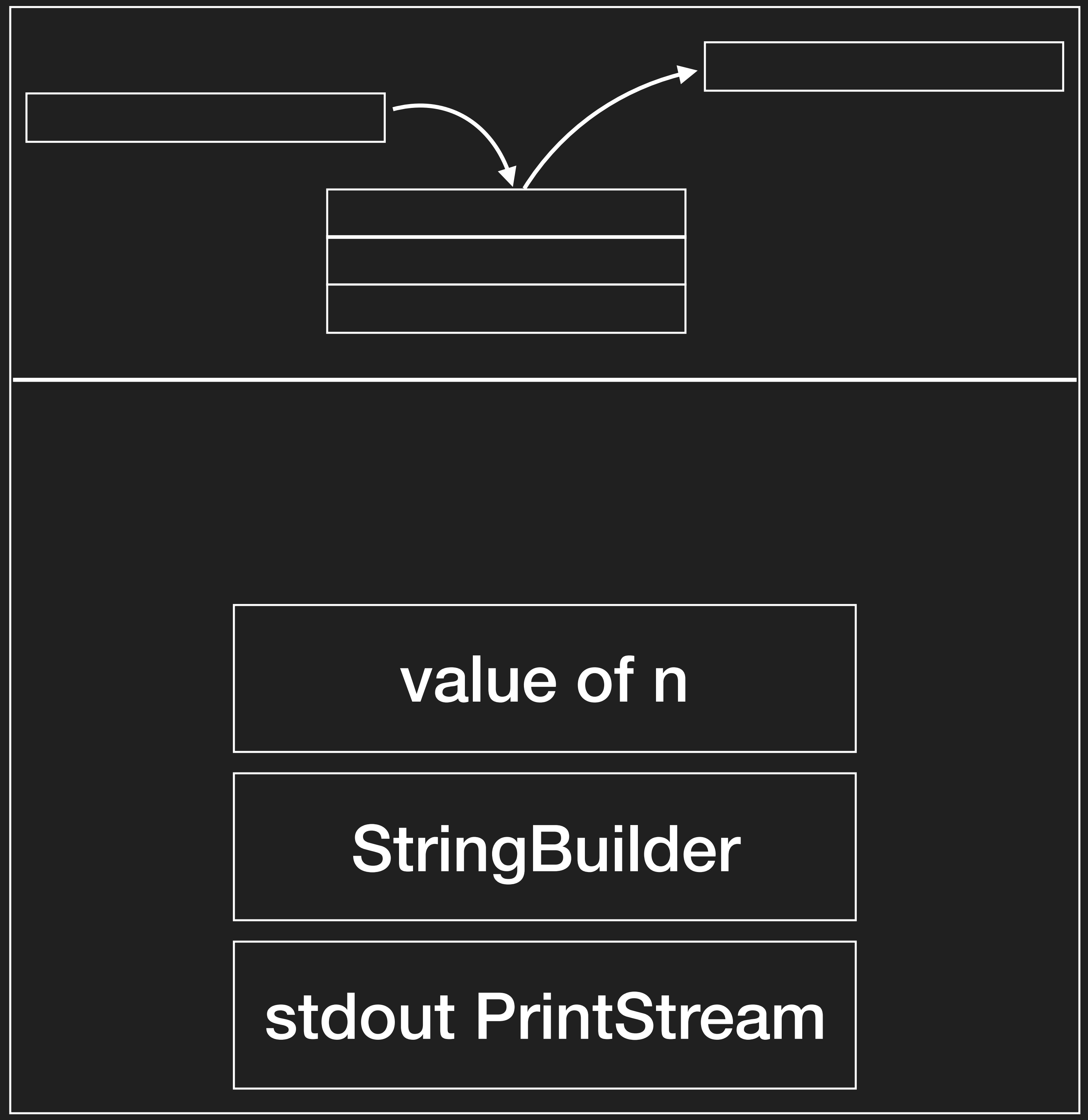
kotlin-plugin/src/main/kotlin/debuglog/DebugLogClassBuilder.kt

```
InstructionAdapter(this).apply {
getstatic("j/l/System", "out", "Ljava/io/PrintStream;")
anew("j/l/StringBuilder")
dup()
invokespecial("j/l/StringBuilder", "<init>", "()V")
visitLdcInsn("⇢ ${function.name}(")
invokevirtual("j/l/StringBuilder", "append",
    "(Lj/l/Object;)Lj/l/StringBuilder;")
function.valueParameters.forEachIndexed { i, param ->
  visitLdcInsn(" ${param.name}=")
  invokevirtual("j/l/StringBuilder", "append", "(Lj/l/String;)Lj/l/SB;")
  visitVarInsn(ALOAD, i + 1)
  invokevirtual("j/l/StringBuilder", "append", "(Lj/l/Object;)Lj/l/SB;")
}
}
```



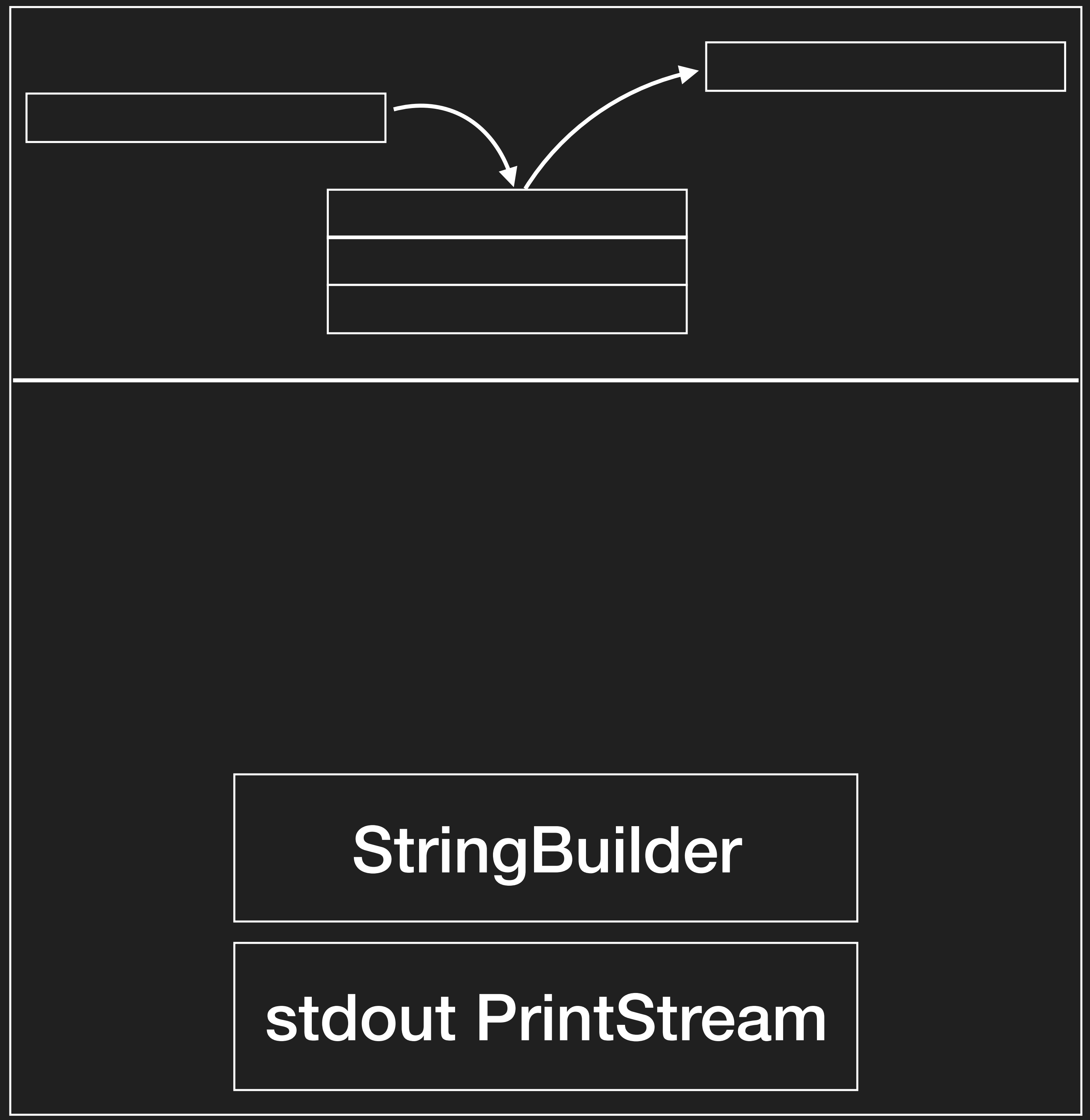
kotlin-plugin/src/main/kotlin/debuglog/DebugLogClassBuilder.kt

```
InstructionAdapter(this).apply {
getstatic("j/l/System", "out", "Ljava/io/PrintStream;")
anew("j/l/StringBuilder")
dup()
invokespecial("j/l/StringBuilder", "<init>", "()V")
visitLdcInsn("⇢ ${function.name}(")
invokevirtual("j/l/StringBuilder", "append",
    "(Lj/l/Object;)Lj/l/StringBuilder;")
function.valueParameters.forEachIndexed { i, param ->
  visitLdcInsn(" ${param.name}=")
  invokevirtual("j/l/StringBuilder", "append", "(Lj/l/String;)Lj/l/SB;")
  visitVarInsn(ALOAD, i + 1)
  invokevirtual("j/l/StringBuilder", "append", "(Lj/l/Object;)Lj/l/SB;")
}
}
```



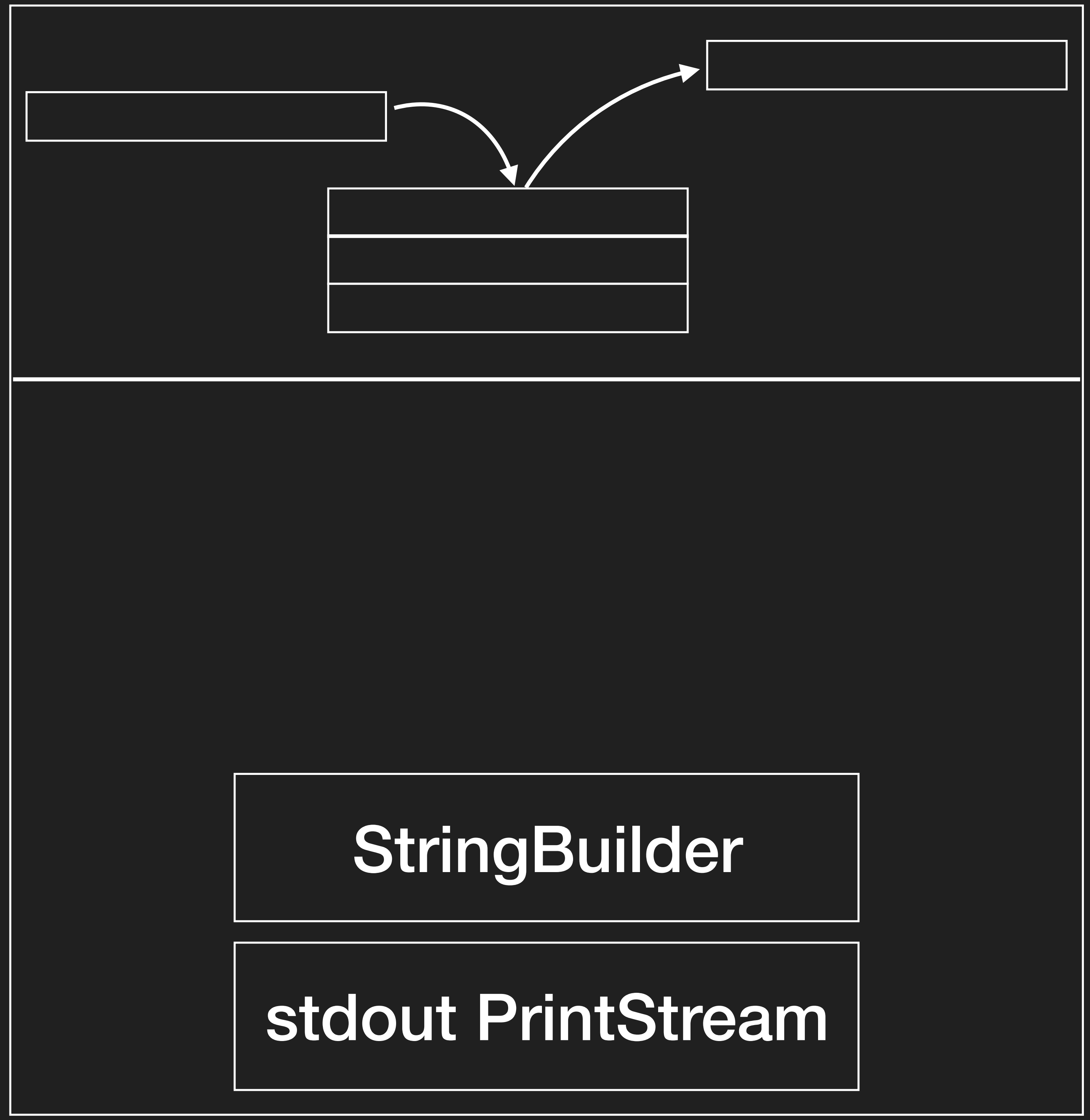
kotlin-plugin/src/main/kotlin/debuglog/DebugLogClassBuilder.kt

```
InstructionAdapter(this).apply {
getstatic("j/l/System", "out", "Ljava/io/PrintStream;")
anew("j/l/StringBuilder")
dup()
invokespecial("j/l/StringBuilder", "<init>", "()V")
visitLdcInsn("⇢ ${function.name}(")
invokevirtual("j/l/StringBuilder", "append",
    "(Lj/l/Object;)Lj/l/StringBuilder;")
function.valueParameters.forEachIndexed { i, param ->
  visitLdcInsn(" ${param.name}=")
  invokevirtual("j/l/StringBuilder", "append", "(Lj/l/String;)Lj/l/SB;")
  visitVarInsn(ALOAD, i + 1)
  invokevirtual("j/l/StringBuilder", "append", "(Lj/l/Object;)Lj/l/SB;")
}
}
```



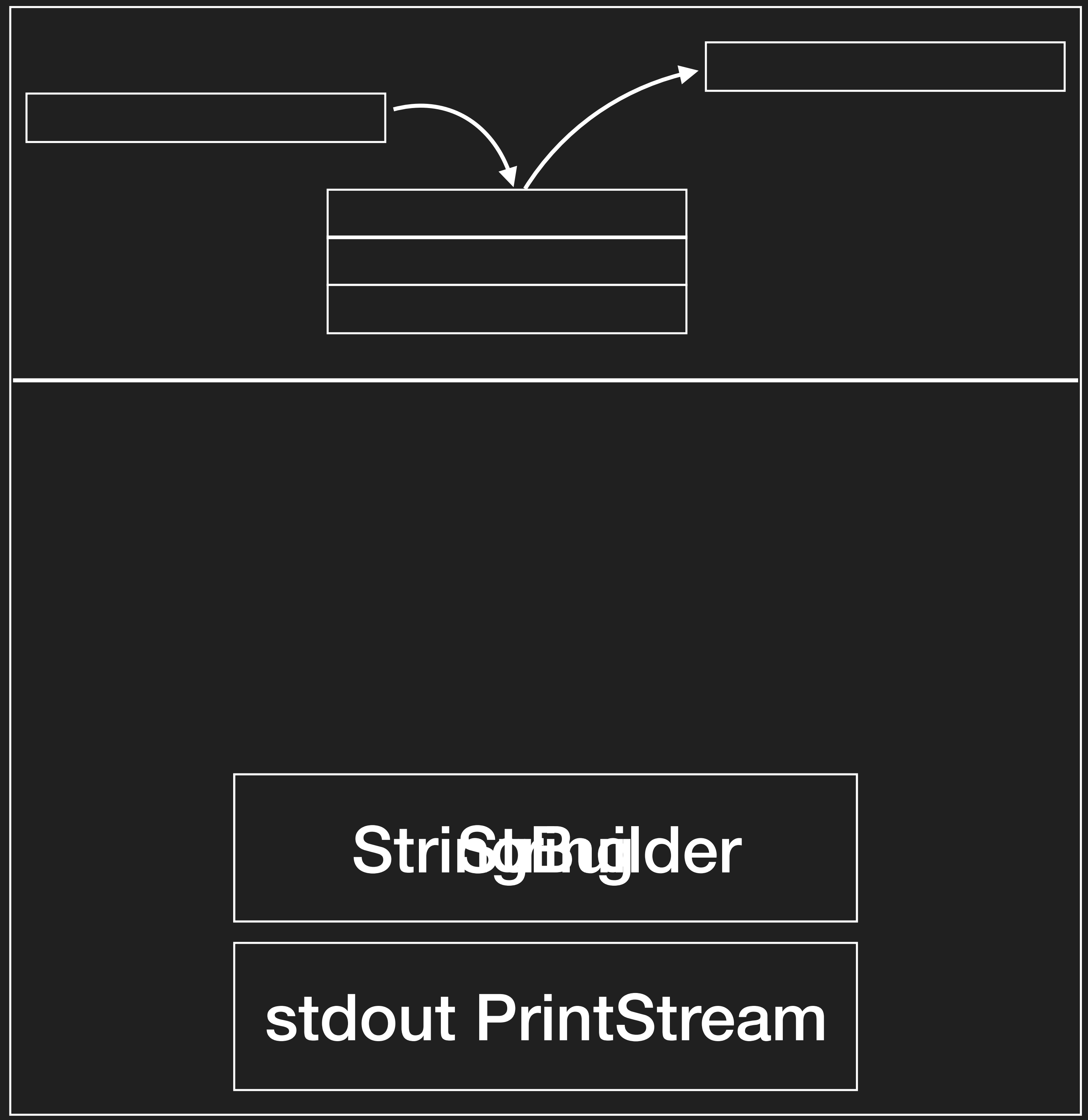
kotlin-plugin/src/main/kotlin/debuglog/DebugLogClassBuilder.kt

```
InstructionAdapter(this).apply {
getstatic("j/l/System", "out", "Ljava/io/PrintStream;")
anew("j/l/StringBuilder")
dup()
invokespecial("j/l/StringBuilder", "<init>", "()V")
visitLdcInsn("⇢ ${function.name}(")
invokevirtual("j/l/StringBuilder", "append",
   "(Lj/l/Object;)Lj/l/StringBuilder;")
function.valueParameters.forEachIndexed { i, param ->
  visitLdcInsn(" ${param.name}=")
  invokevirtual("j/l/StringBuilder", "append", "(Lj/l/String;)Lj/l/SB;")
  visitVarInsn(ALOAD, i + 1)
  invokevirtual("j/l/StringBuilder", "append", "(Lj/l/Object;)Lj/l/SB;")
}
}
```



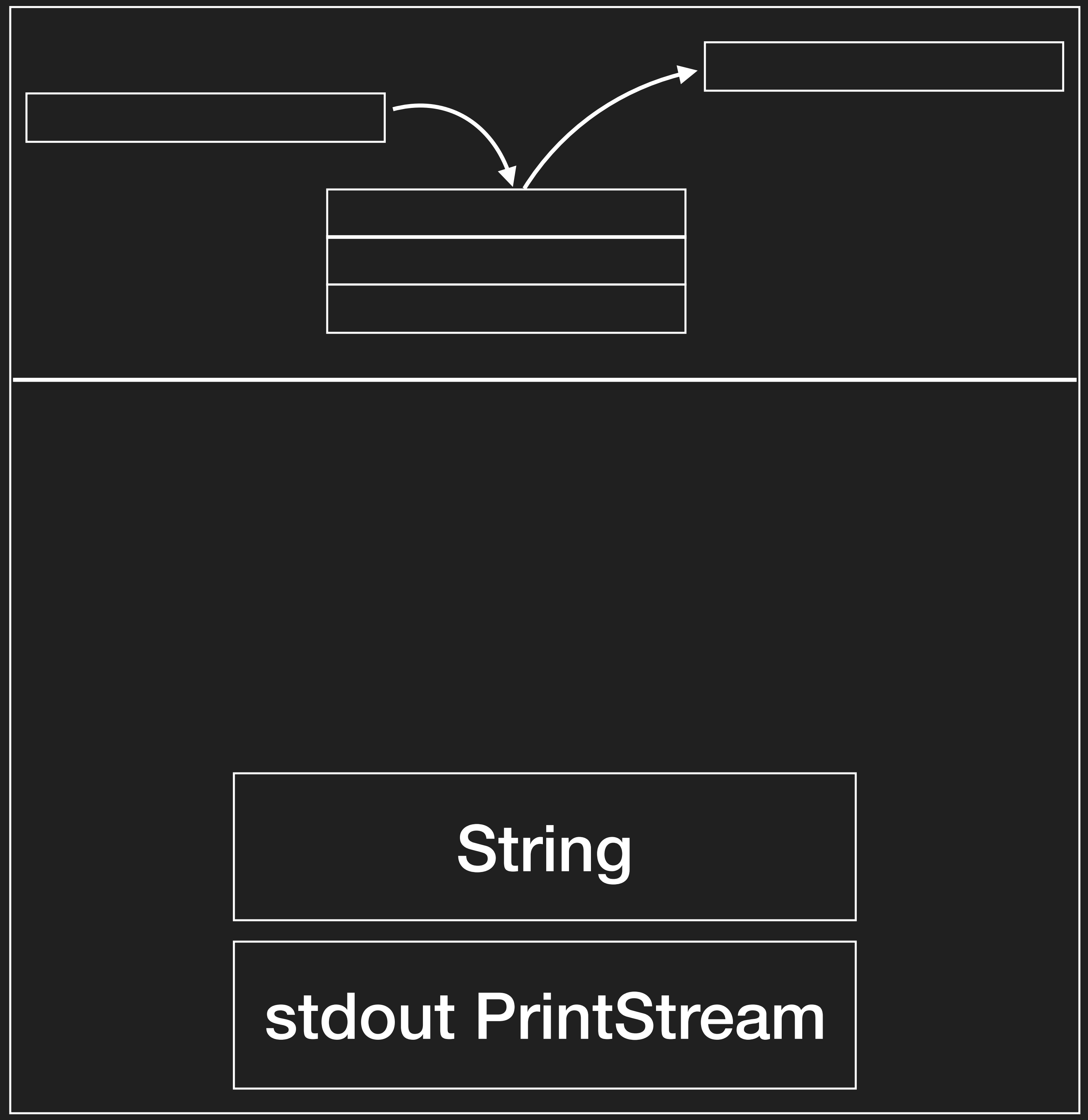
kotlin-plugin/src/main/kotlin/debuglog/DebugLogClassBuilder.kt

```
InstructionAdapter(this).apply {
getstatic("j/l/System", "out", "Ljava/io/PrintStream;")
anew("j/l/StringBuilder")
dup()
invokespecial("j/l/StringBuilder", "<init>", "()V")
visitLdcInsn("⇢ ${function.name}(")
invokevirtual("j/l/StringBuilder", "append",
    "(Lj/l/Object;)Lj/l/StringBuilder;")
function.valueParameters.forEachIndexed { i, param ->
  visitLdcInsn(" ${param.name}=")
  invokevirtual("j/l/StringBuilder", "append", "(Lj/l/String;)Lj/l/SB;")
  visitVarInsn(ALOAD, i + 1)
  invokevirtual("j/l/StringBuilder", "append", "(Lj/l/Object;)Lj/l/SB;")
}
invokevirtual("j/l/StringBuilder", "toString", "()Lj/l/String;")
}
```



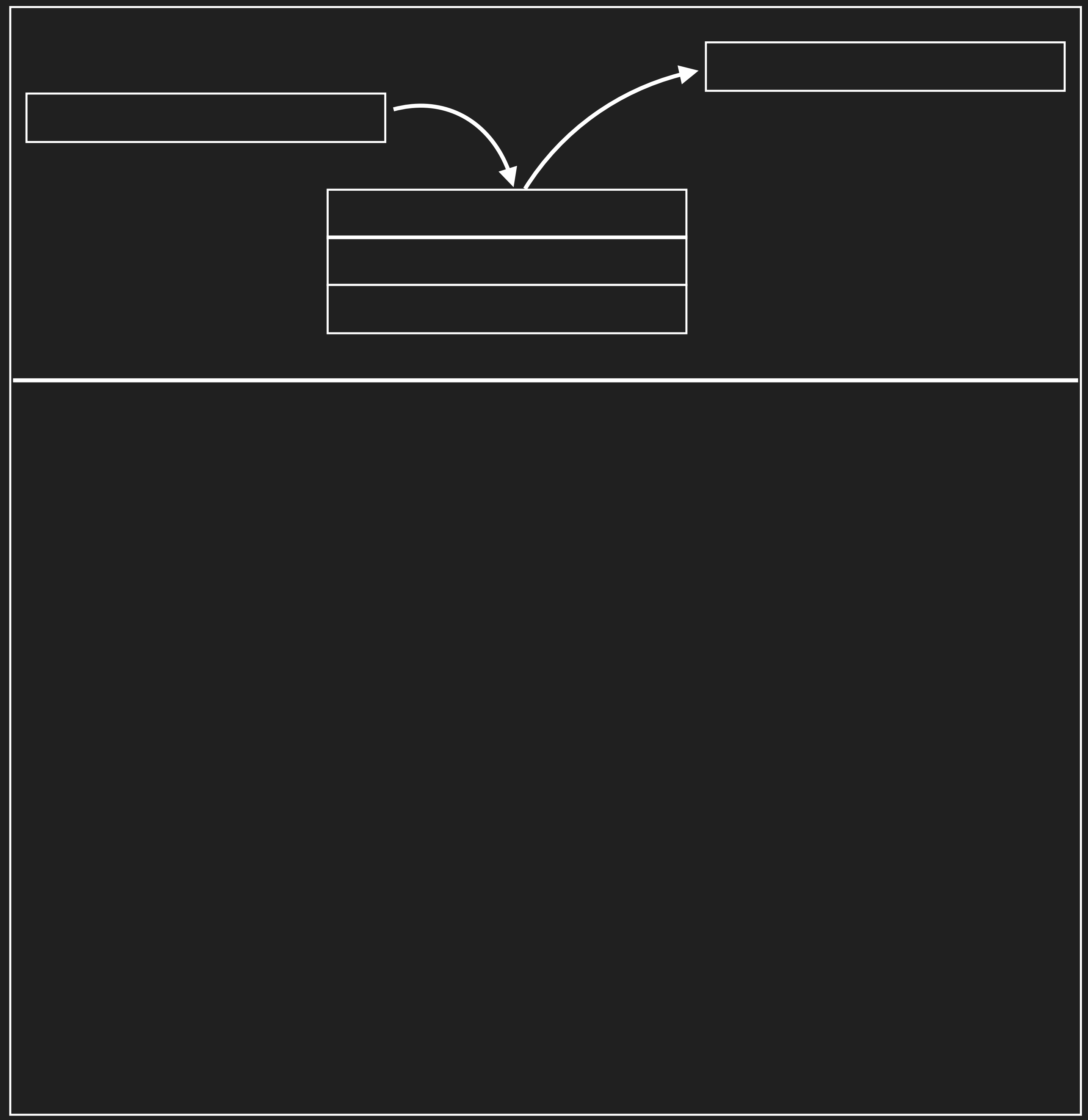
kotlin-plugin/src/main/kotlin/debuglog/DebugLogClassBuilder.kt

```
InstructionAdapter(this).apply {
getstatic("j/l/System", "out", "Ljava/io/PrintStream;")
anew("j/l/StringBuilder")
dup()
invokespecial("j/l/StringBuilder", "<init>", "()V")
visitLdcInsn("⇢ ${function.name}(")
invokevirtual("j/l/StringBuilder", "append",
    "(Lj/l/Object;)Lj/l/StringBuilder;")
function.valueParameters.forEachIndexed { i, param ->
  visitLdcInsn(" ${param.name}=")
  invokevirtual("j/l/StringBuilder", "append", "(Lj/l/String;)Lj/l/SB;")
  visitVarInsn(ALOAD, i + 1)
  invokevirtual("j/l/StringBuilder", "append", "(Lj/l/Object;)Lj/l/SB;")
}
invokevirtual("j/l/StringBuilder", "toString", "()Lj/l/String;")
invokevirtual("j/io/PrintStream", "println", "(Lj/l/String;)V")
}
```



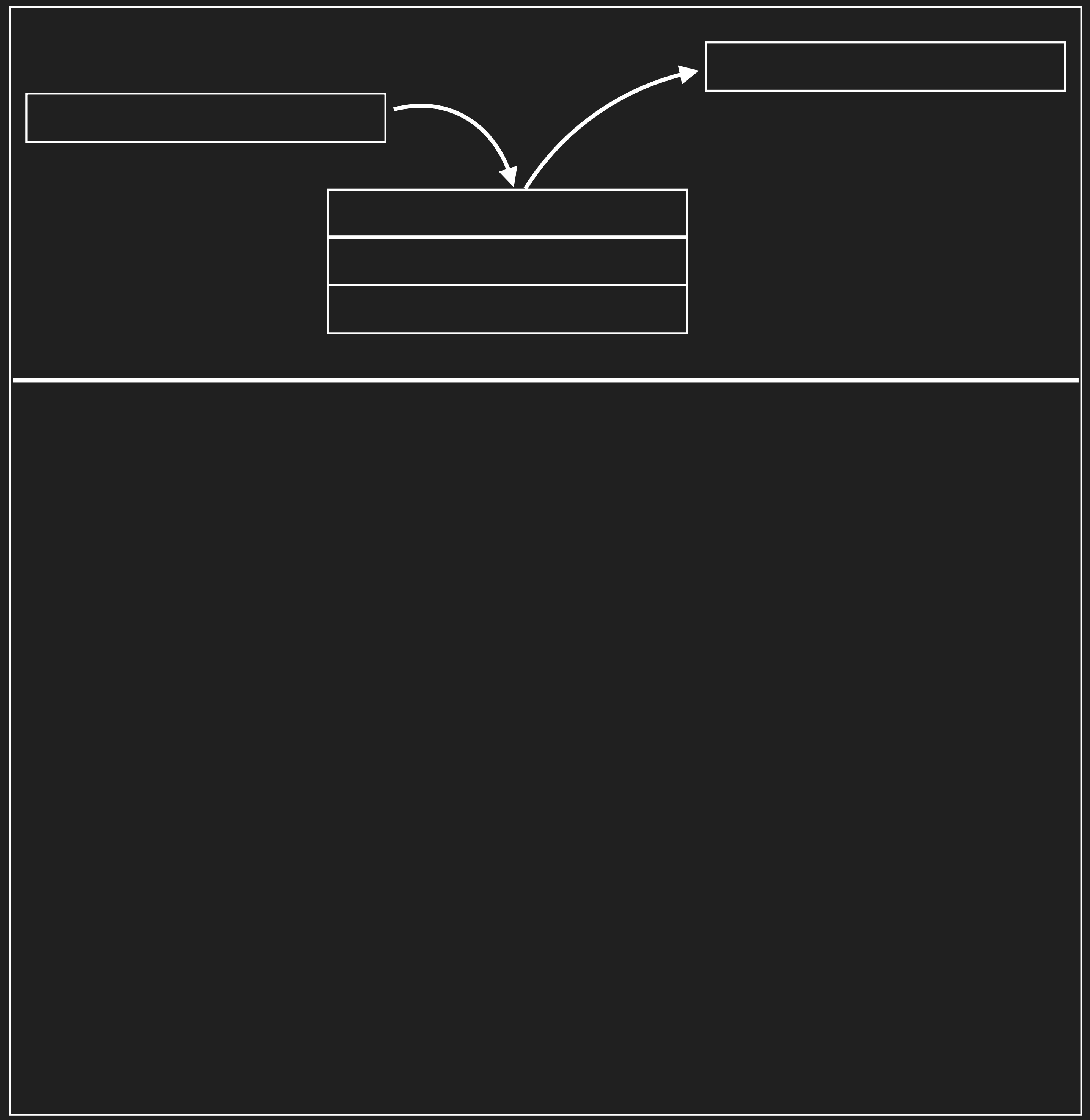
kotlin-plugin/src/main/kotlin/debuglog/DebugLogClassBuilder.kt

```
InstructionAdapter(this).apply {
getstatic("j/l/System", "out", "Ljava/io/PrintStream;")
anew("j/l/StringBuilder")
dup()
invokespecial("j/l/StringBuilder", "<init>", "()V")
visitLdcInsn("⇢ ${function.name}(")
invokevirtual("j/l/StringBuilder", "append",
    "(Lj/l/Object;)Lj/l/StringBuilder;")
function.valueParameters.forEachIndexed { i, param ->
  visitLdcInsn(" ${param.name}=")
  invokevirtual("j/l/StringBuilder", "append", "(Lj/l/String;)Lj/l/SB;")
  visitVarInsn(ALOAD, i + 1)
  invokevirtual("j/l/StringBuilder", "append", "(Lj/l/Object;)Lj/l/SB;")
}
invokevirtual("j/l/StringBuilder", "toString", "()Lj/l/String;")
invokevirtual("j/io/PrintStream", "println", "(Lj/l/String;)V")
}
```



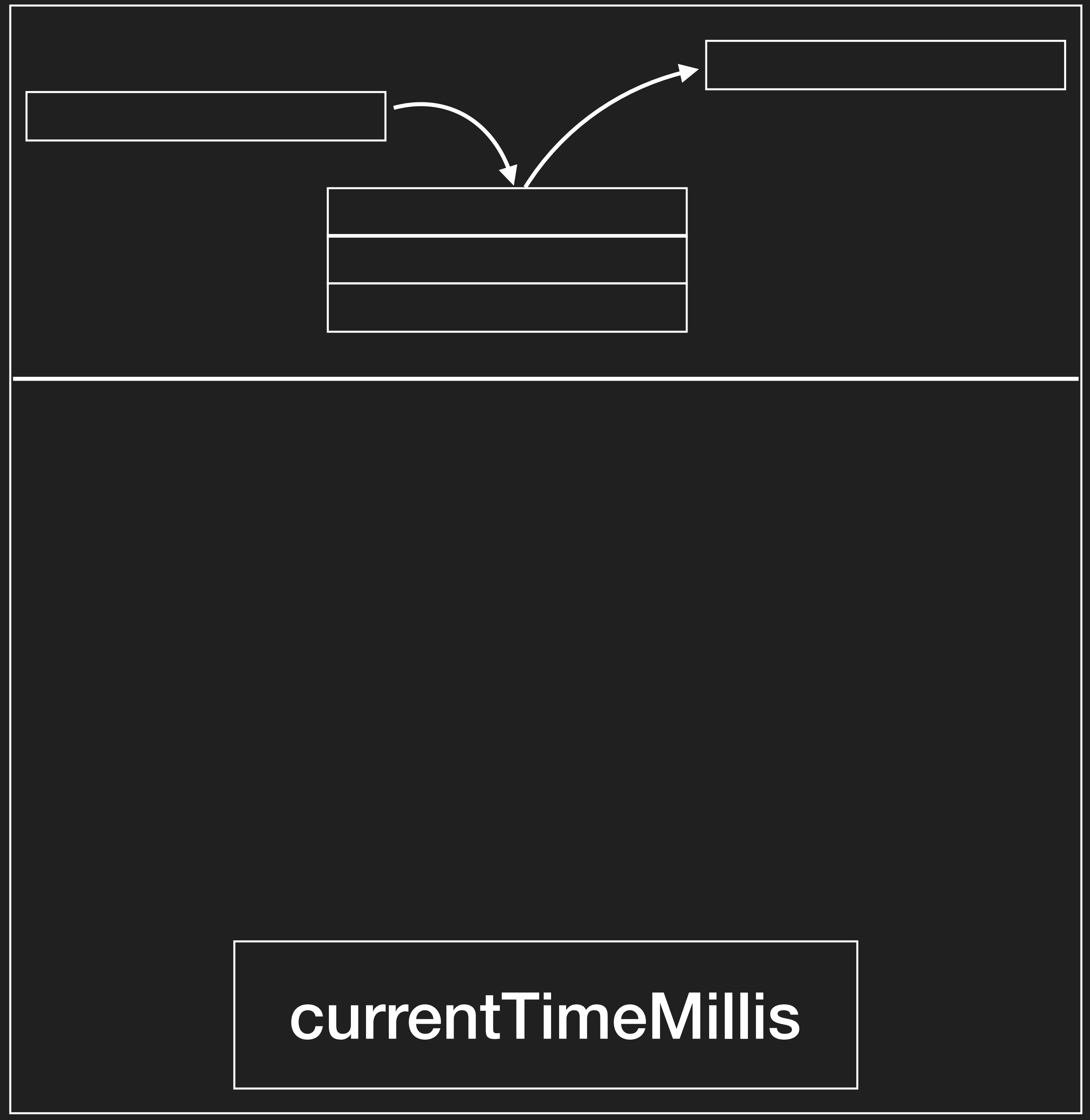
kotlin-plugin/src/main/kotlin/debuglog/DebugLogClassBuilder.kt

```
InstructionAdapter(this).apply {
// ... method-trace-printing code
}
```



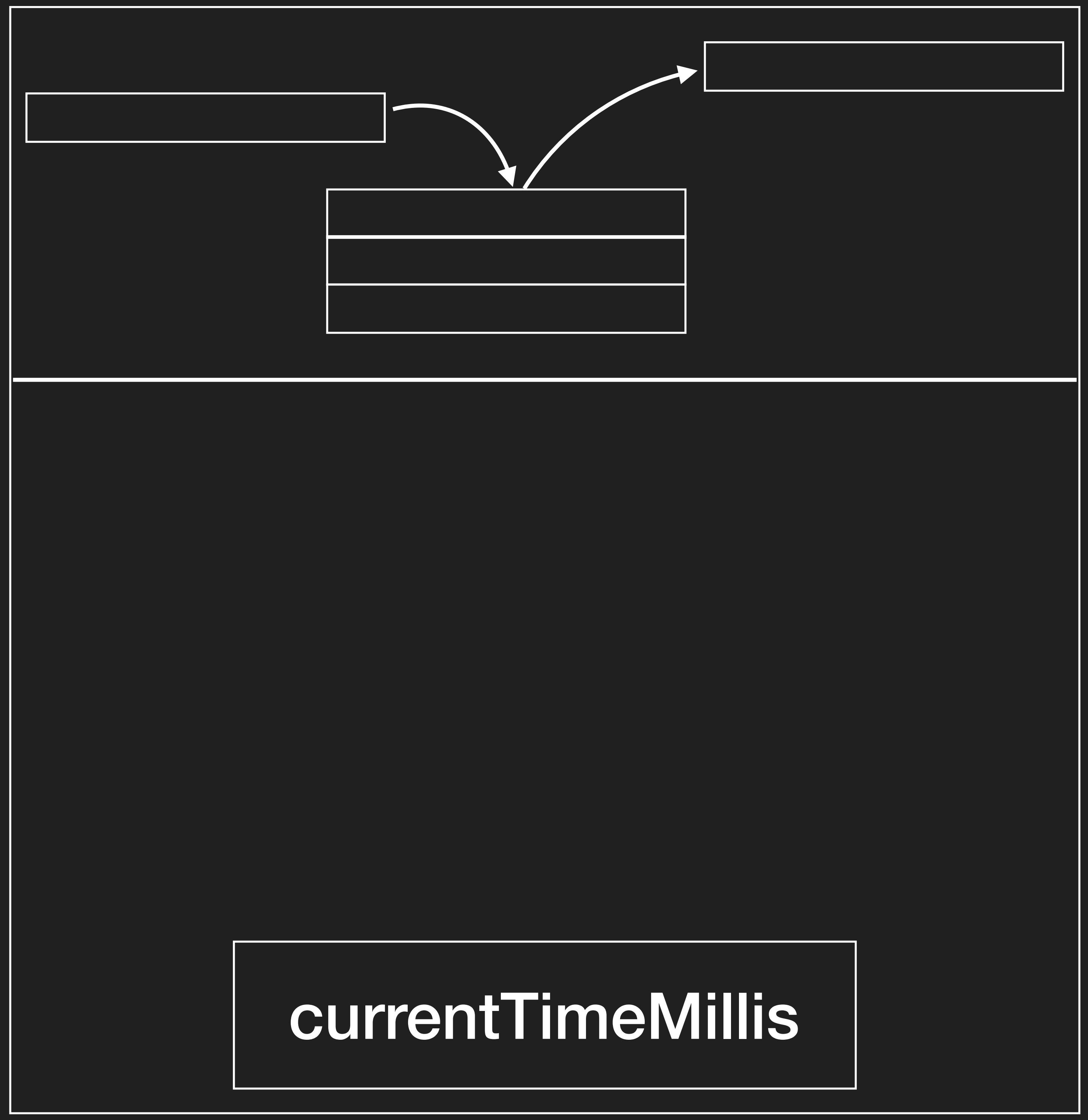
kotlin-plugin/src/main/kotlin/debuglog/DebugLogClassBuilder.kt

```
InstructionAdapter(this).apply {
// ... method-trace-printing code
invokestatic("j/l/System", "currentTimeMillis", "()J")
}
```



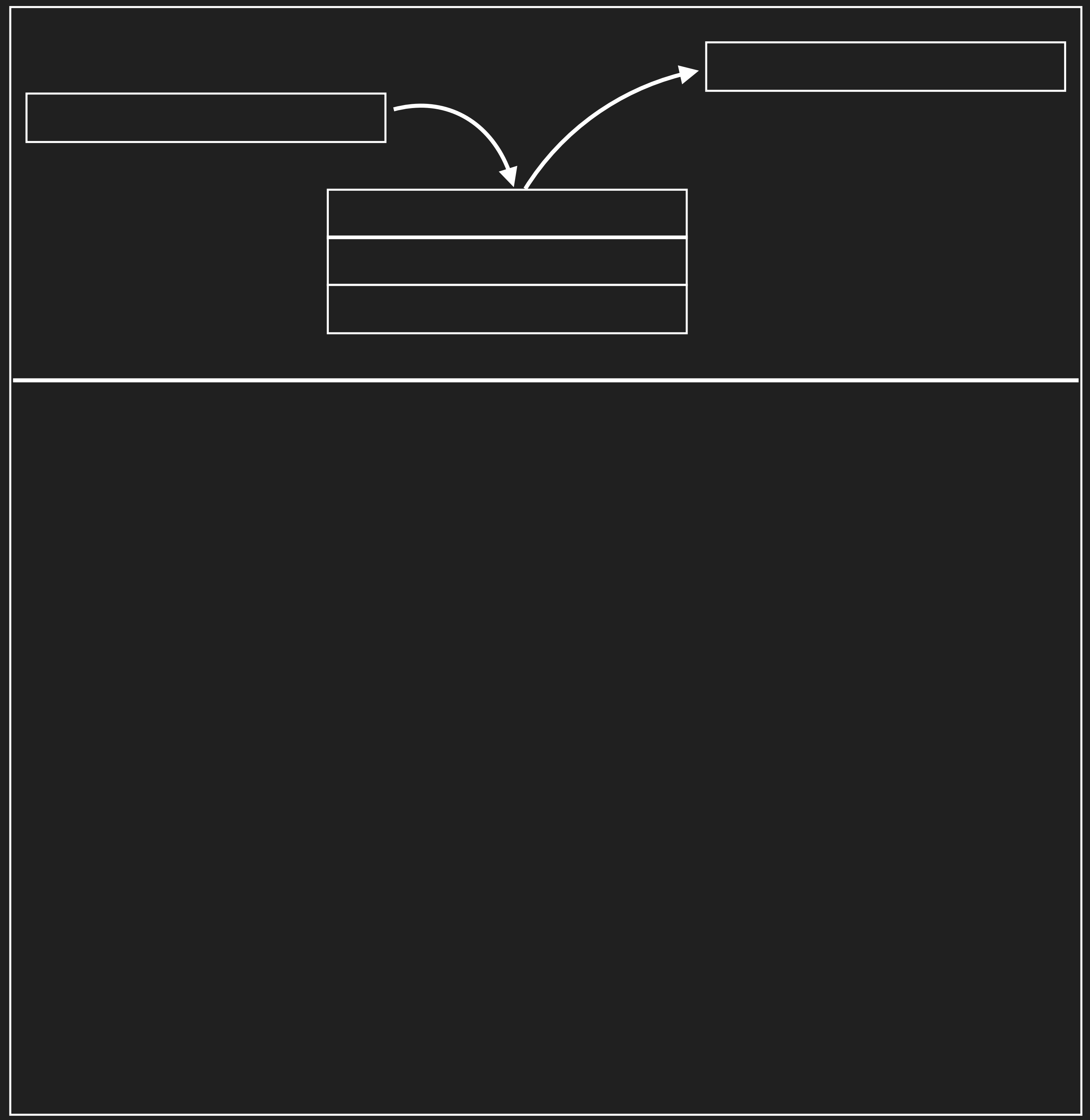
kotlin-plugin/src/main/kotlin/debuglog/DebugLogClassBuilder.kt

```
InstructionAdapter(this).apply {
// ... method-trace-printing code
invokestatic("j/l/System", "currentTimeMillis", "()J")
store(9001, LONG_TYPE)
}
```



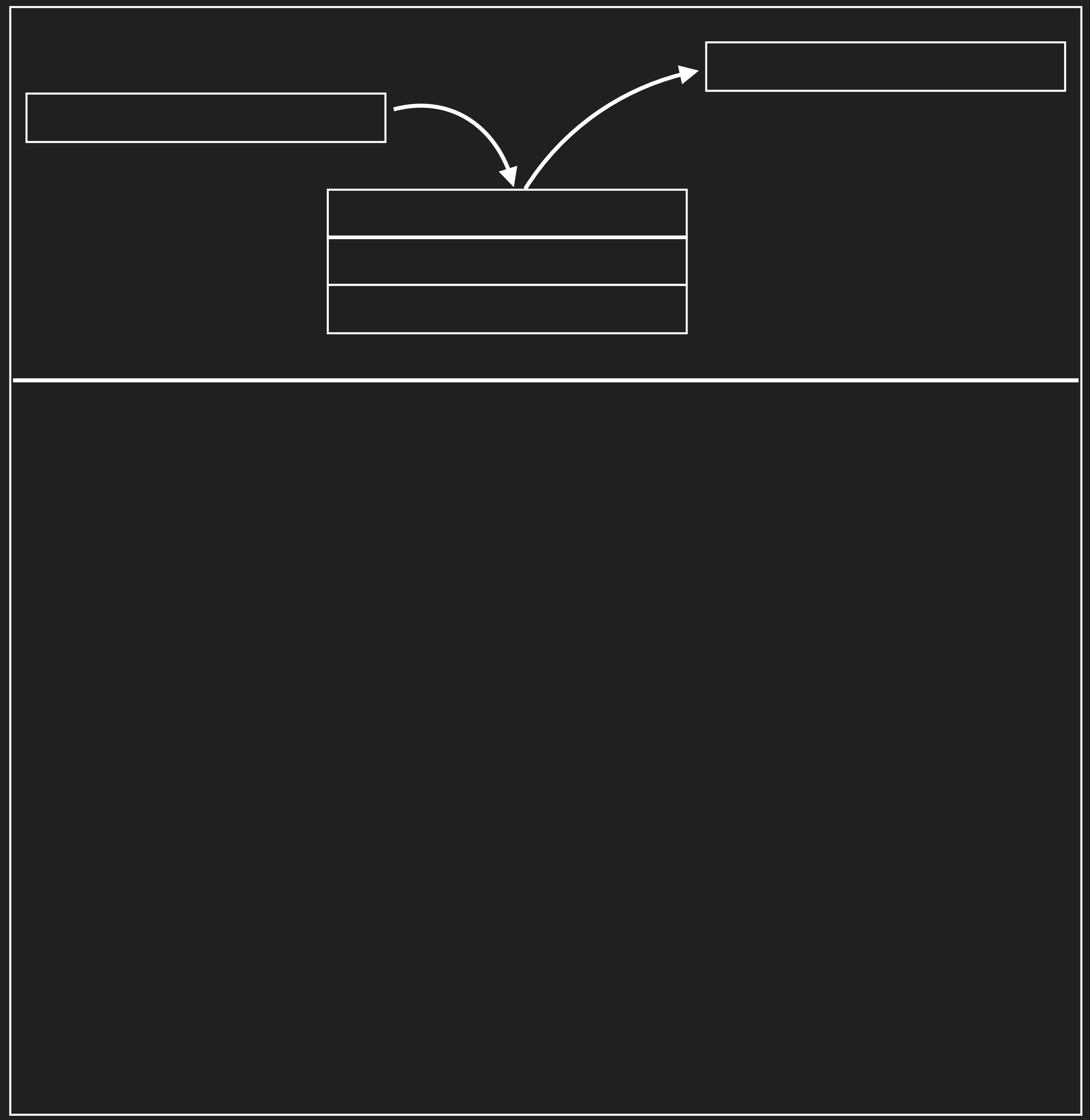
kotlin-plugin/src/main/kotlin/debuglog/DebugLogClassBuilder.kt

```
InstructionAdapter(this).apply  {
// ... method-trace-printing code
invokestatic("j/l/System", "currentTimeMillis", "()J")
store(9001, LONG_TYPE)
}
```



kotlin-plugin/src/main/kotlin/debuglog/DebugLogClassBuilder.kt

```
InstructionAdapter(this).apply {
// ... method-trace-printing code
// ... timestamp-storing code
}
```



kotlin-plugin/src/main/kotlin/debuglog/DebugLogClassBuilder.kt

```
return object : MethodVisitor(Opcodes.ASM5, original) {
  override fun visitCode() {
    super.visitCode()
    InstructionAdapter(this).apply  {
      // ... method-trace-printing code
      // ... timestamp-storing code
    }
  }
  override fun visitInsn(opcode: Int) {
    when (opcode) {
      RETURN , ARETURN, IRETURN -> {
        InstructionAdapter(this).apply { TODO("on method exit") }
      }
    }
    super.visitInsn(opcode)
  }
}
```

kotlin-plugin/src/main/kotlin/debuglog/DebugLogClassBuilder.kt

```kotlin
return object : MethodVisitor(Opcodes.ASM5, original) {
  override fun visitCode() {
    super.visitCode()
    InstructionAdapter(this).apply  {
      // ... method-trace-printing code
      // ... timestamp-storing code
    }
  }
  override fun visitInsn(opcode: Int) {
    when (opcode) {
      RETURN , ARETURN, IRETURN -> {
        InstructionAdapter(this).apply { TODO("on method exit") }
      }
    }
    super.visitInsn(opcode)
  }
}
```

kotlin-plugin/src/main/kotlin/debuglog/DebugLogClassBuilder.kt

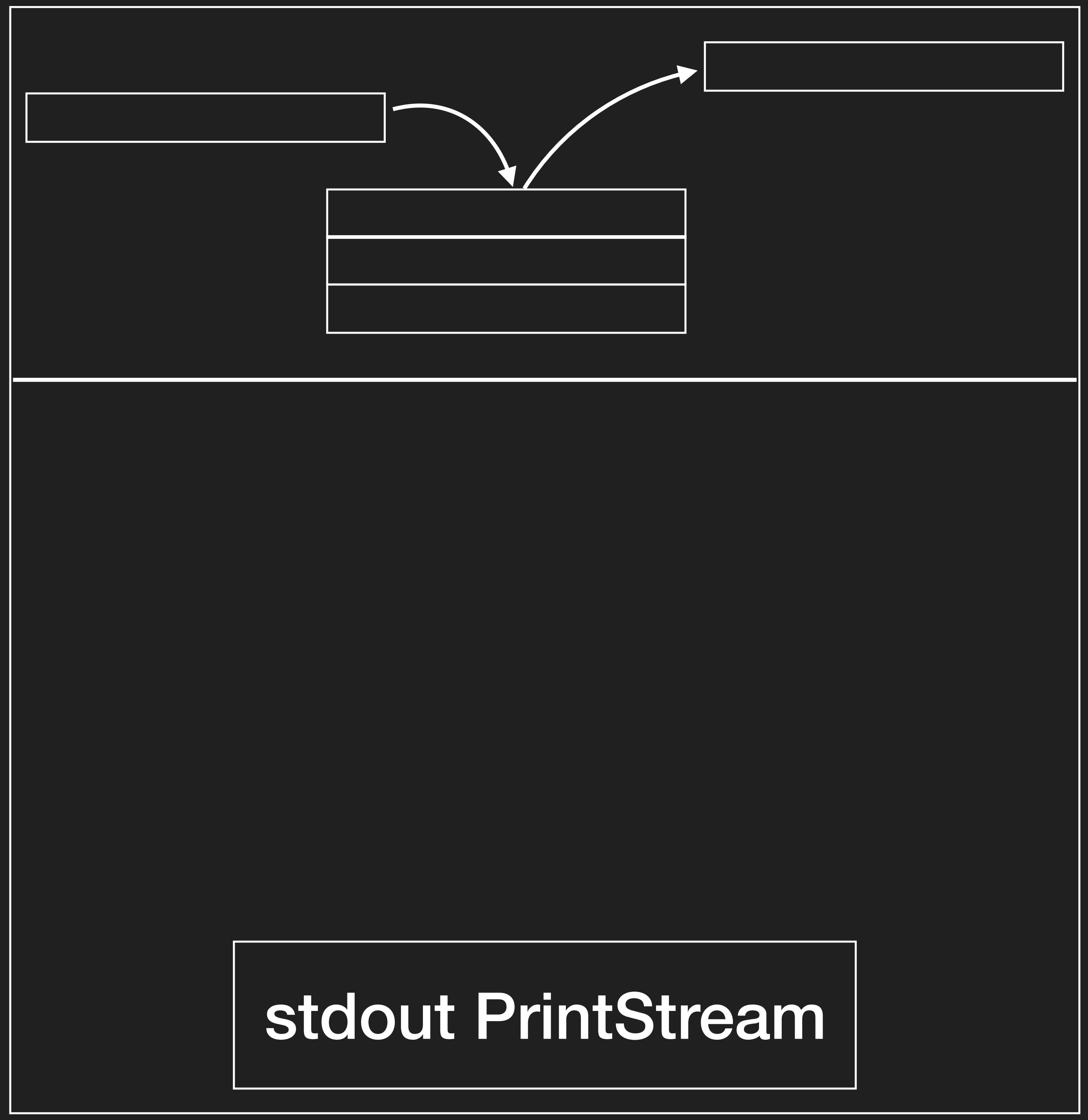
```
return object : MethodVisitor(Opcodes.ASM5, original) {
  override fun visitCode() {
    super.visitCode()
    InstructionAdapter(this).apply {
      // ... method-trace-printing code
      // ... timestamp-storing code
    }
  }
  override fun visitInsn(opcode: Int) {
    when (opcode) {
      RETURN , ARETURN, IRETURN -> {
        InstructionAdapter(this).apply   {   TODO("on method exit")   }
      }
    }
    super.visitInsn(opcode)
  }
}
```

kotlin-plugin/src/main/kotlin/debuglog/DebugLogClassBuilder.kt

```
InstructionAdapter(this).apply  {
TODO("on method exit")
}
```

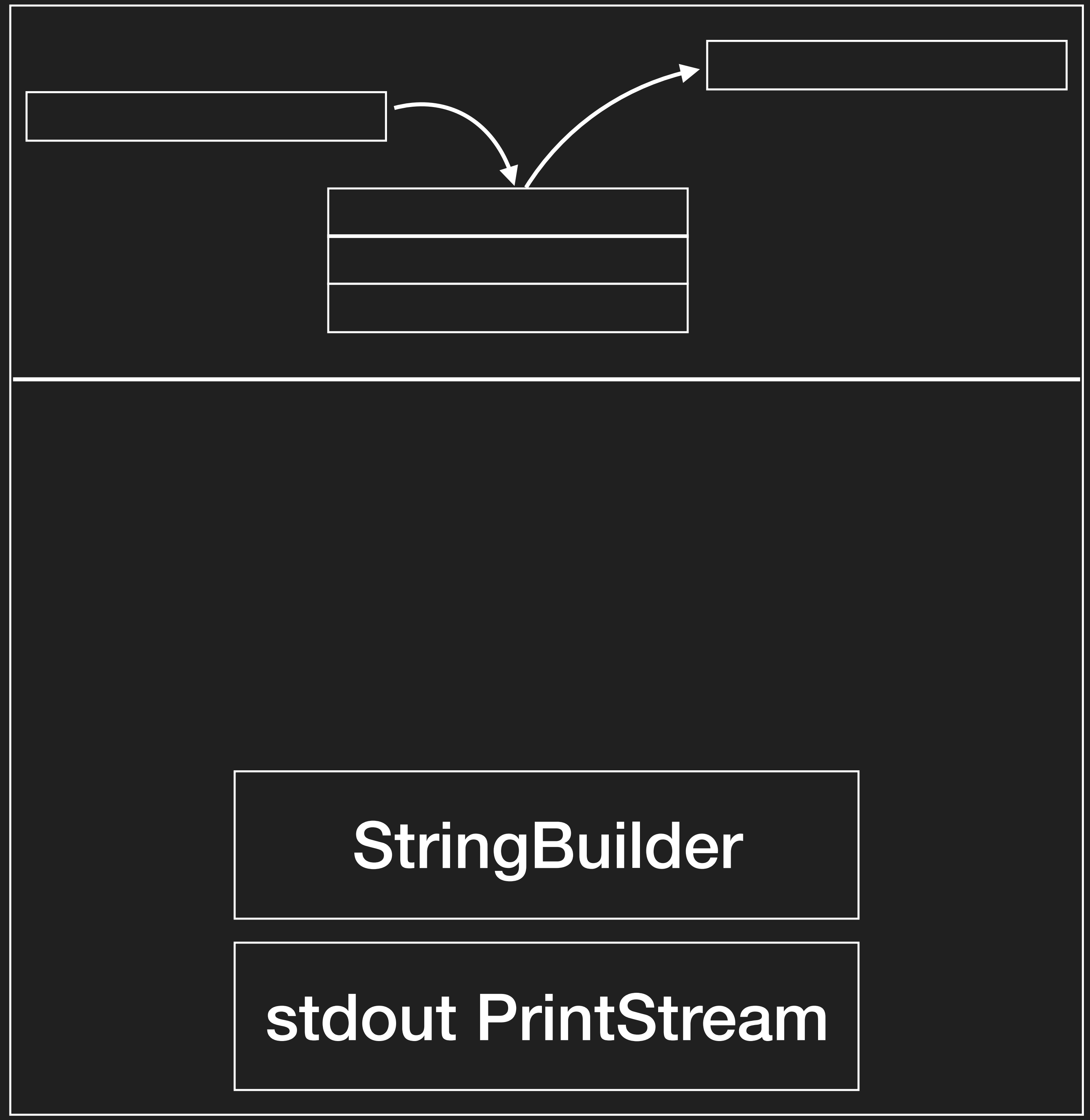
kotlin-plugin/src/main/kotlin/debuglog/DebugLogClassBuilder.kt

```
InstructionAdapter(this).apply  {
getstatic("j/l/System", "out", "Lj/io/PrintStream;")
}
```



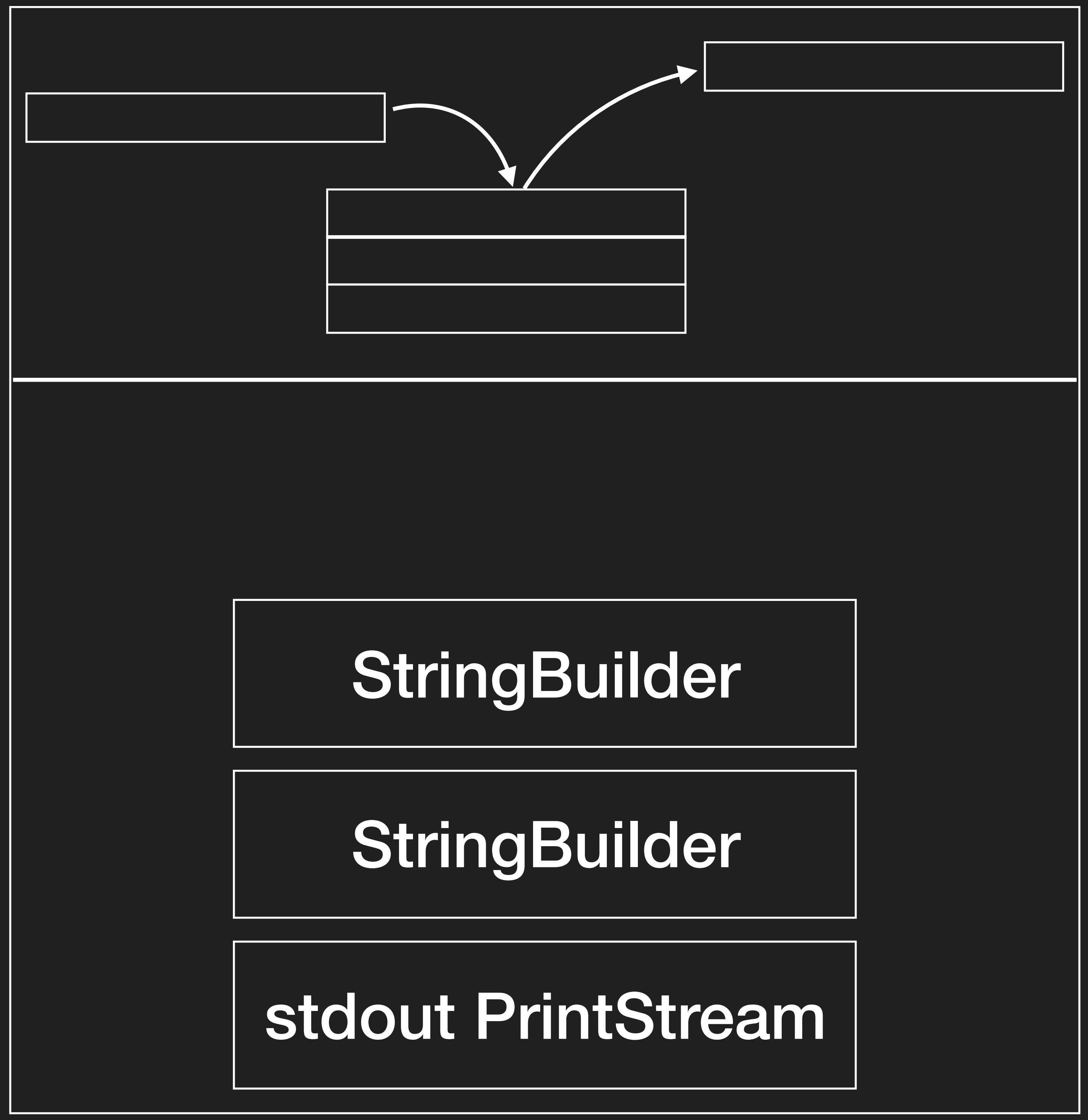
kotlin-plugin/src/main/kotlin/debuglog/DebugLogClassBuilder.kt

```
InstructionAdapter(this).apply  {
getstatic("j/l/System", "out", "Lj/io/PrintStream;")
anew("java/lang/StringBuilder")
}
```



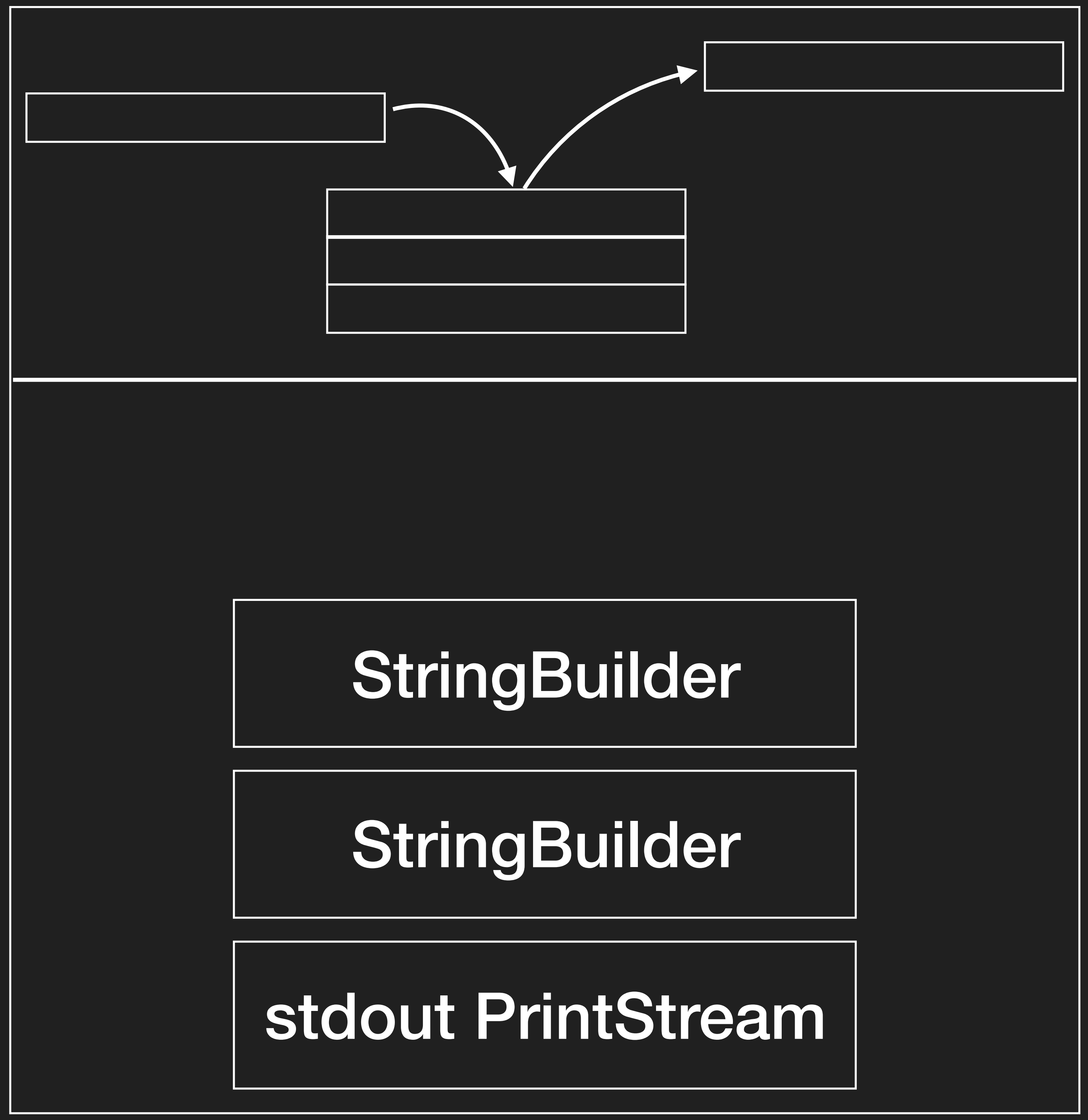
kotlin-plugin/src/main/kotlin/debuglog/DebugLogClassBuilder.kt

```
InstructionAdapter(this).apply  {
getstatic("j/l/System", "out", "Lj/io/PrintStream;")
anew("java/lang/StringBuilder")
dup()
}
```



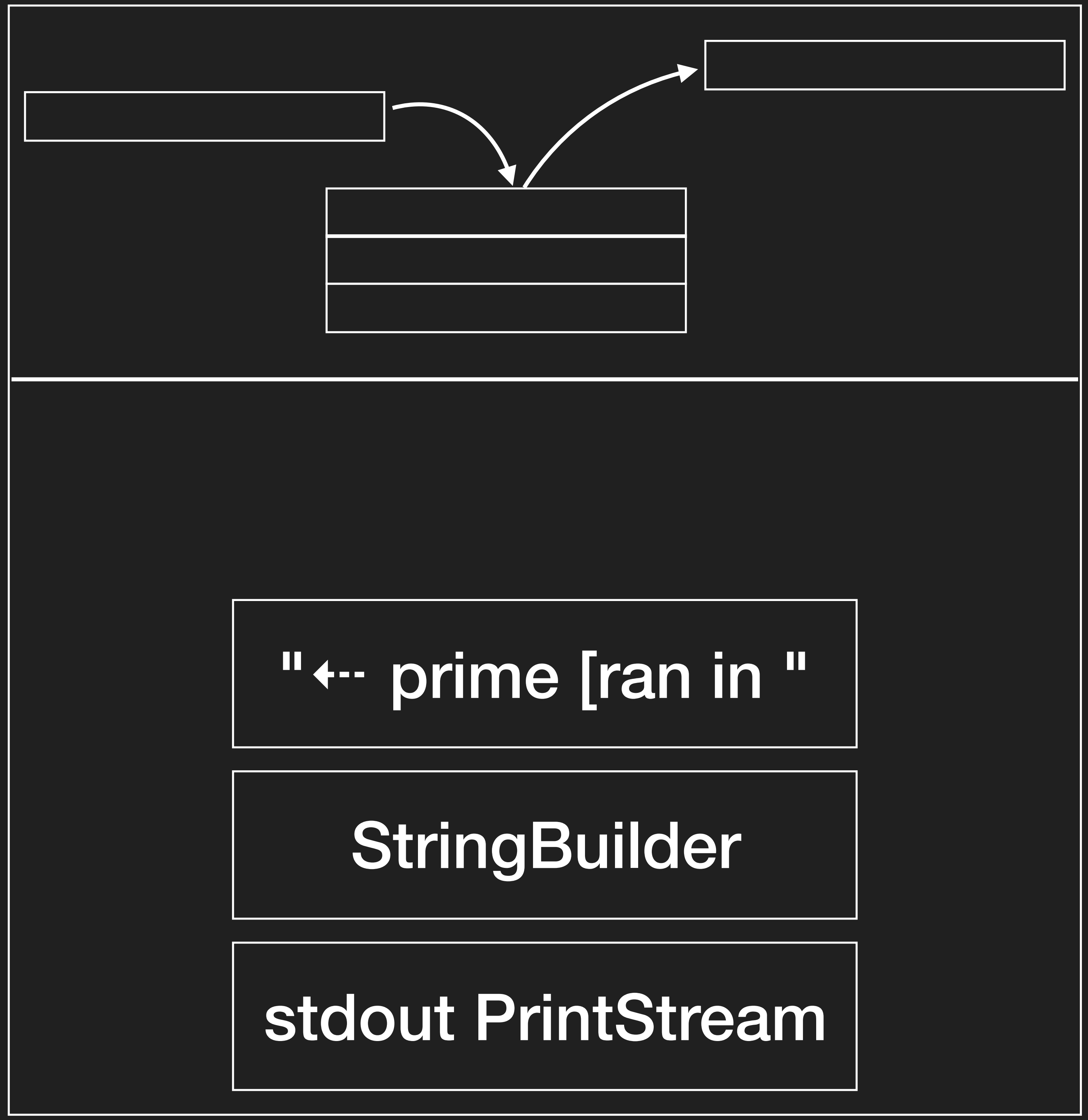
kotlin-plugin/src/main/kotlin/debuglog/DebugLogClassBuilder.kt

```
InstructionAdapter(this).apply  {
getstatic("j/l/System", "out", "Lj/io/PrintStream;")
anew("java/lang/StringBuilder")
dup()
invokespecial("j/l/StringBuilder", "<init>", "()V")
}
```



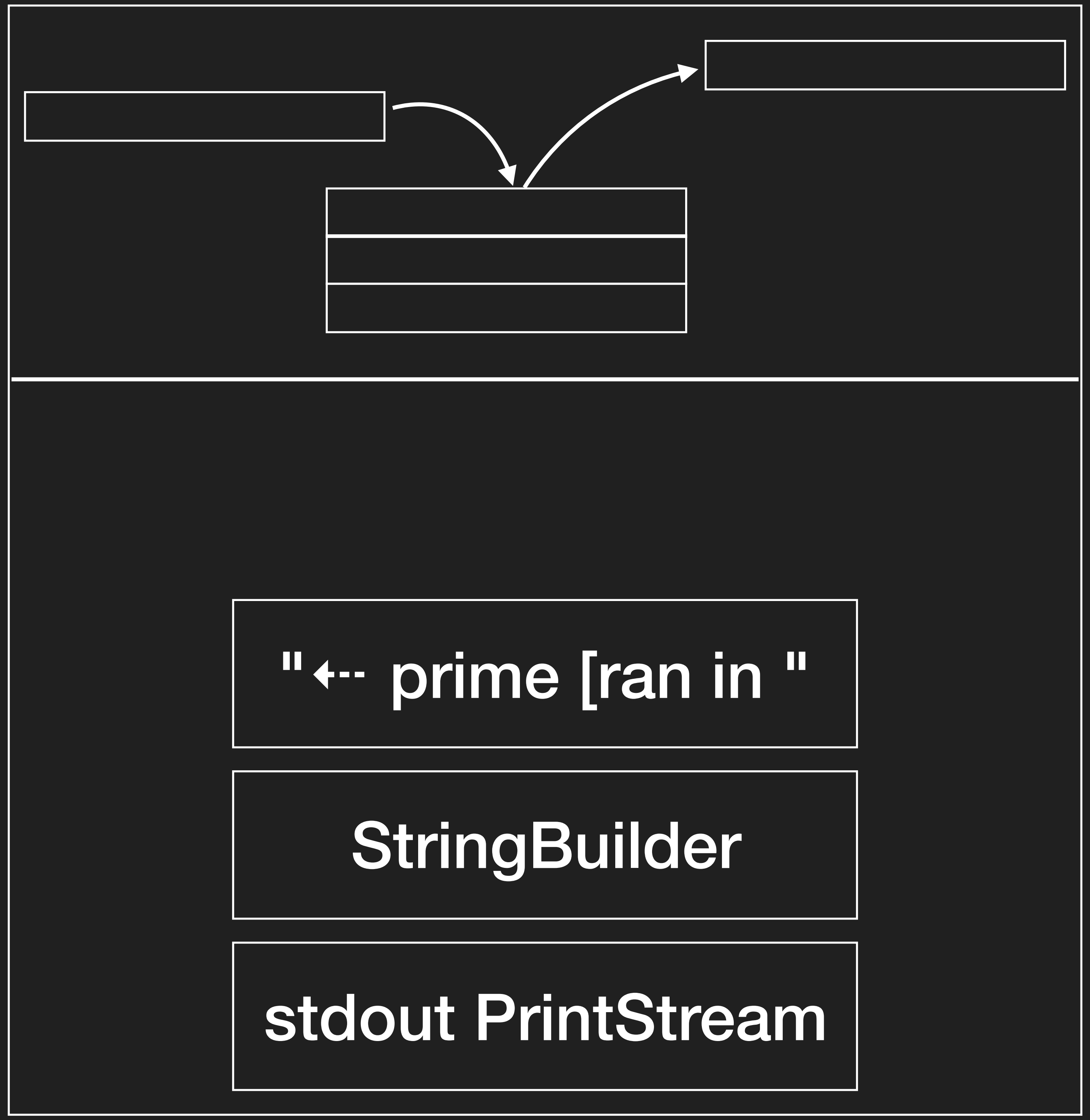
kotlin-plugin/src/main/kotlin/debuglog/DebugLogClassBuilder.kt

```
InstructionAdapter(this).apply  {
getstatic("j/l/System", "out", "Lj/io/PrintStream;")
anew("java/lang/StringBuilder")
dup()
invokespecial("j/l/StringBuilder", "<init>", "()V")
visitLdcInsn("⇠ ${function.name} [ran in ")
}
```



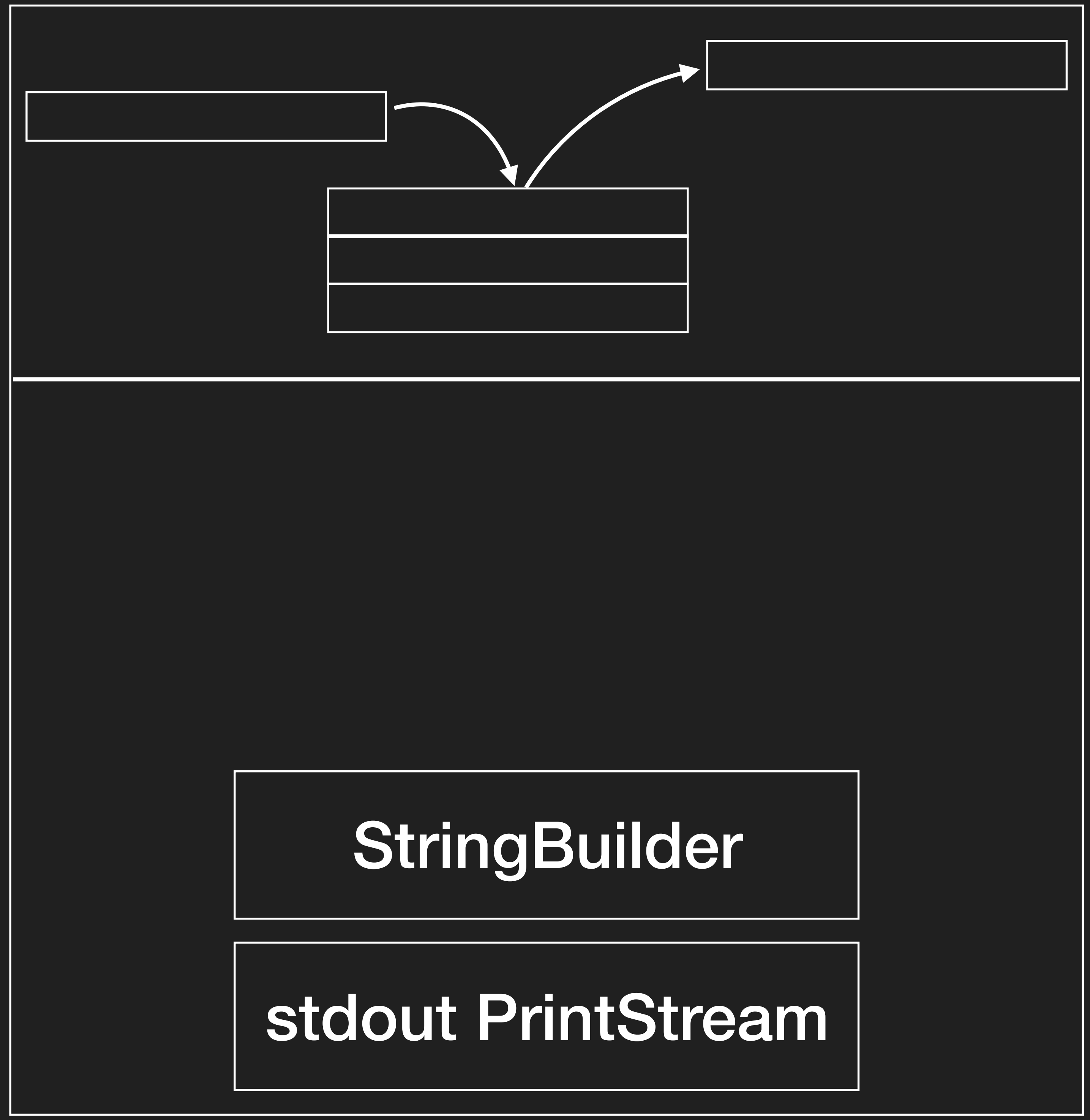
kotlin-plugin/src/main/kotlin/debuglog/DebugLogClassBuilder.kt

```
InstructionAdapter(this).apply  {
getstatic("j/l/System", "out", "Lj/io/PrintStream;")
anew("java/lang/StringBuilder")
dup()
invokespecial("j/l/StringBuilder", "<init>", "()V")
visitLdcInsn("⇠ ${function.name} [ran in ")
invokevirtual("j/l/StringBuilder", "append",
   "(Lj/l/String;)Lj/l/StringBuilder;")
}
```



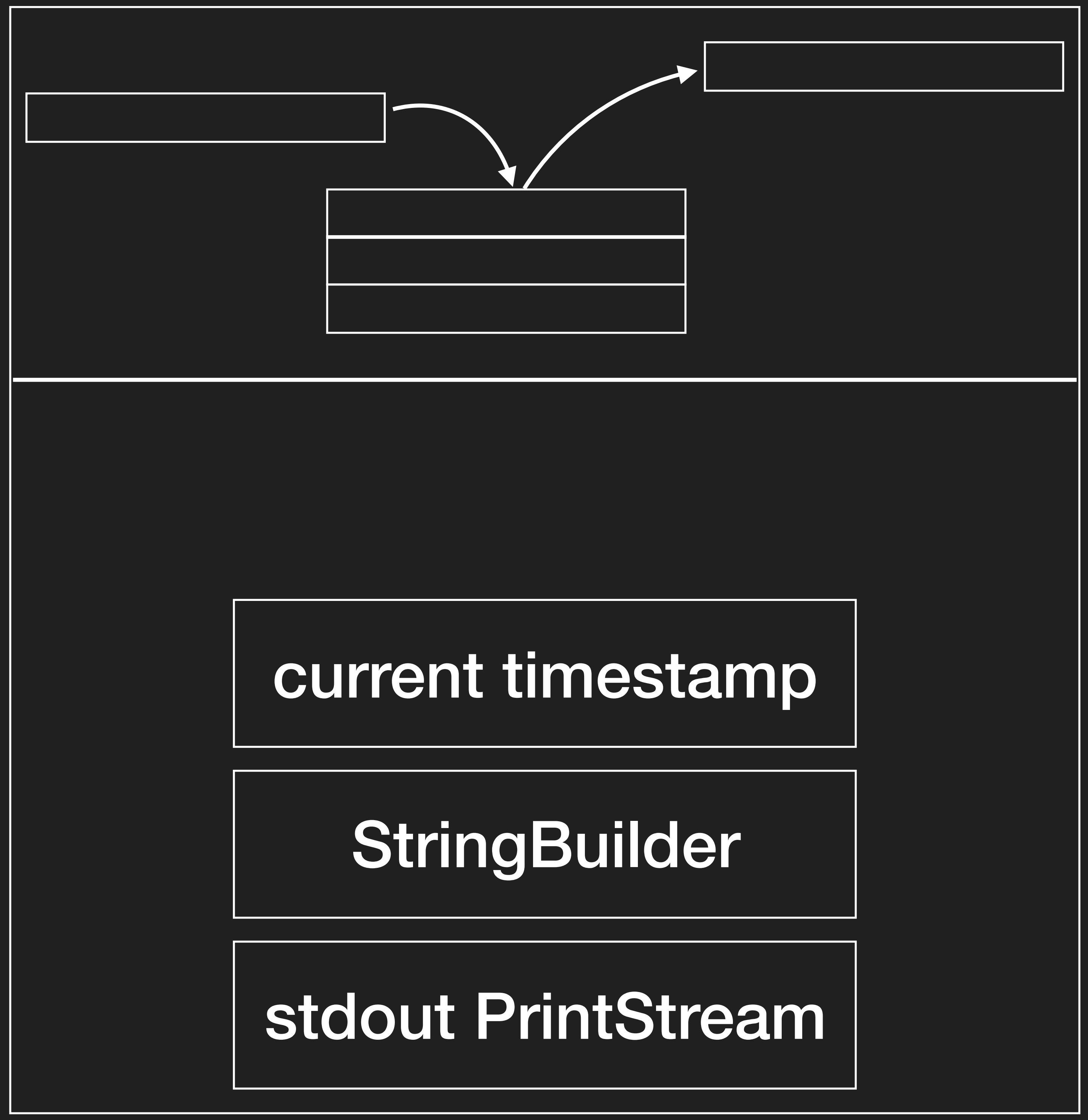
kotlin-plugin/src/main/kotlin/debuglog/DebugLogClassBuilder.kt

```
InstructionAdapter(this).apply  {
getstatic("j/l/System", "out", "Lj/io/PrintStream;")
anew("java/lang/StringBuilder")
dup()
invokespecial("j/l/StringBuilder", "<init>", "()V")
visitLdcInsn("⟵ ${function.name} [ran in ")
invokevirtual("j/l/StringBuilder", "append",
   "(Lj/l/String;)Lj/l/StringBuilder;")
}
```



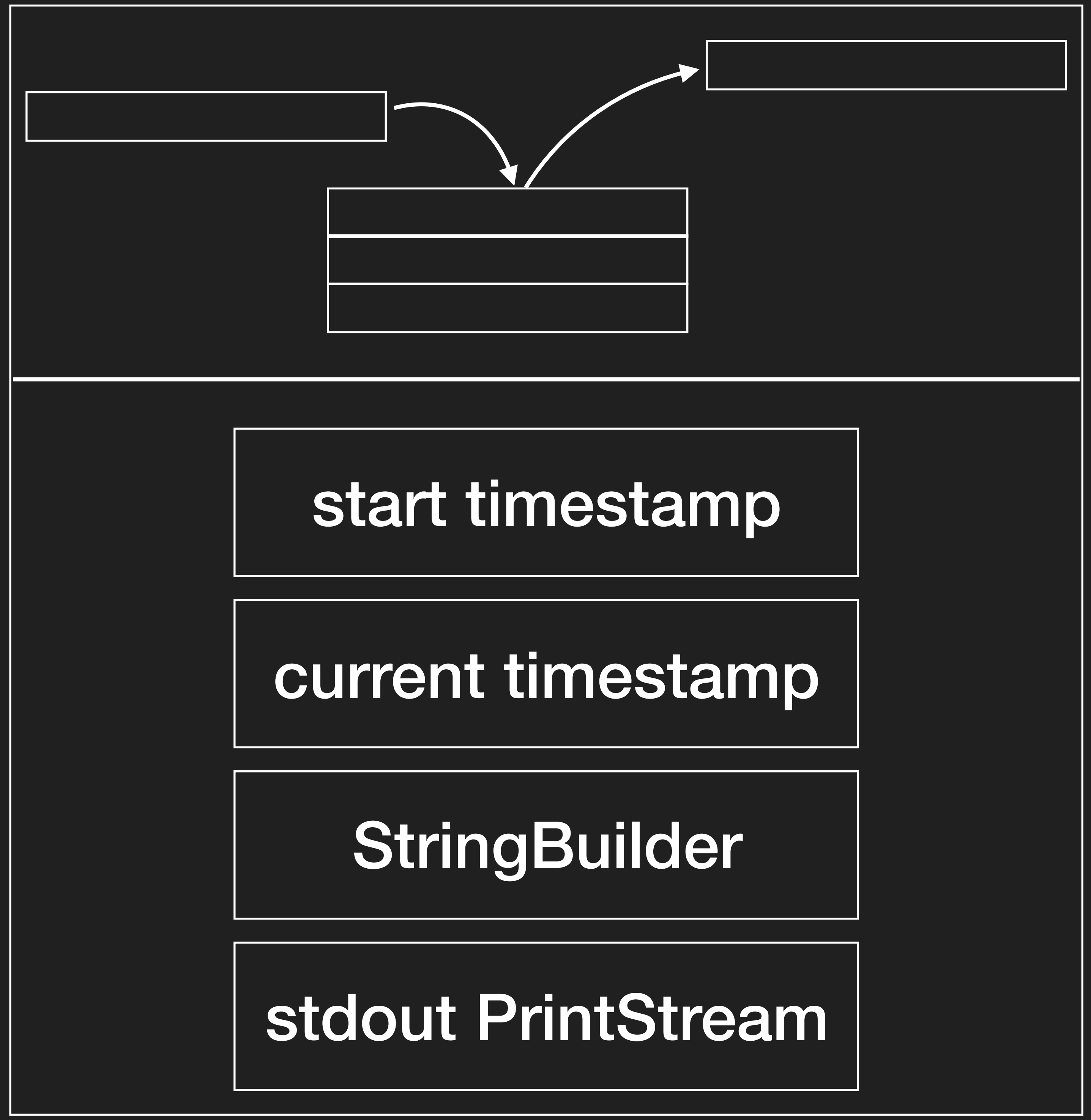
kotlin-plugin/src/main/kotlin/debuglog/DebugLogClassBuilder.kt

```
InstructionAdapter(this).apply  {
getstatic("j/l/System", "out", "Lj/io/PrintStream;")
anew("java/lang/StringBuilder")
dup()
invokespecial("j/l/StringBuilder", "<init>", "()V")
visitLdcInsn("⇠ ${function.name} [ran in ")
invokevirtual("j/l/StringBuilder", "append",
   "(Lj/l/String;)Lj/l/StringBuilder;")
invokestatic("j/l/System", "currentTimeMillis", "()J")
}
```



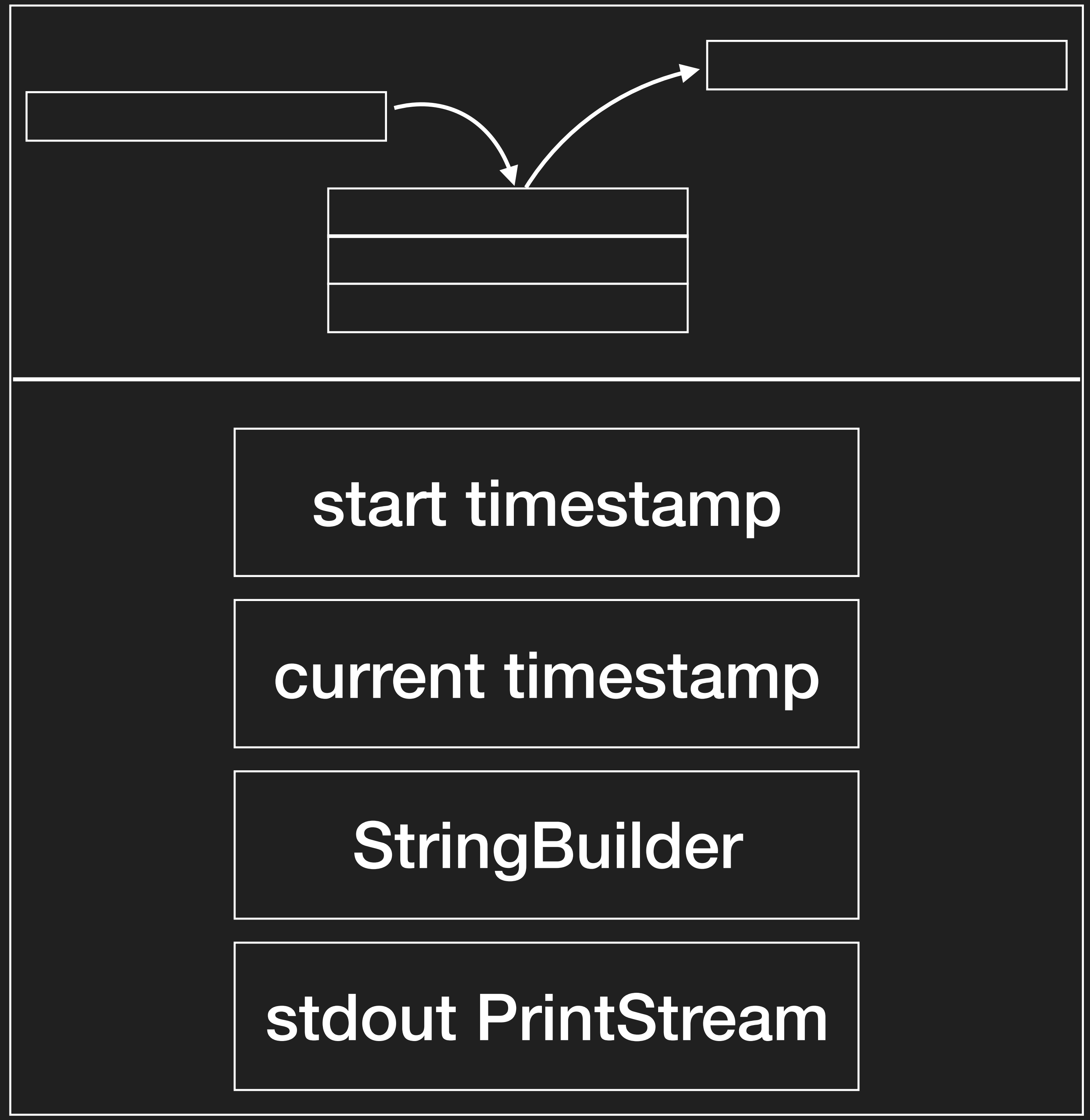
kotlin-plugin/src/main/kotlin/debuglog/DebugLogClassBuilder.kt

```
InstructionAdapter(this).apply  {
getstatic("j/l/System", "out", "Lj/io/PrintStream;")
anew("java/lang/StringBuilder")
dup()
invokespecial("j/l/StringBuilder", "<init>", "()V")
visitLdcInsn("⇠ ${function.name} [ran in ")
invokevirtual("j/l/StringBuilder", "append",
   "(Lj/l/String;)Lj/l/StringBuilder;")
invokestatic("j/l/System", "currentTimeMillis", "()J")
load(9001, LONG_TYPE)
}
```



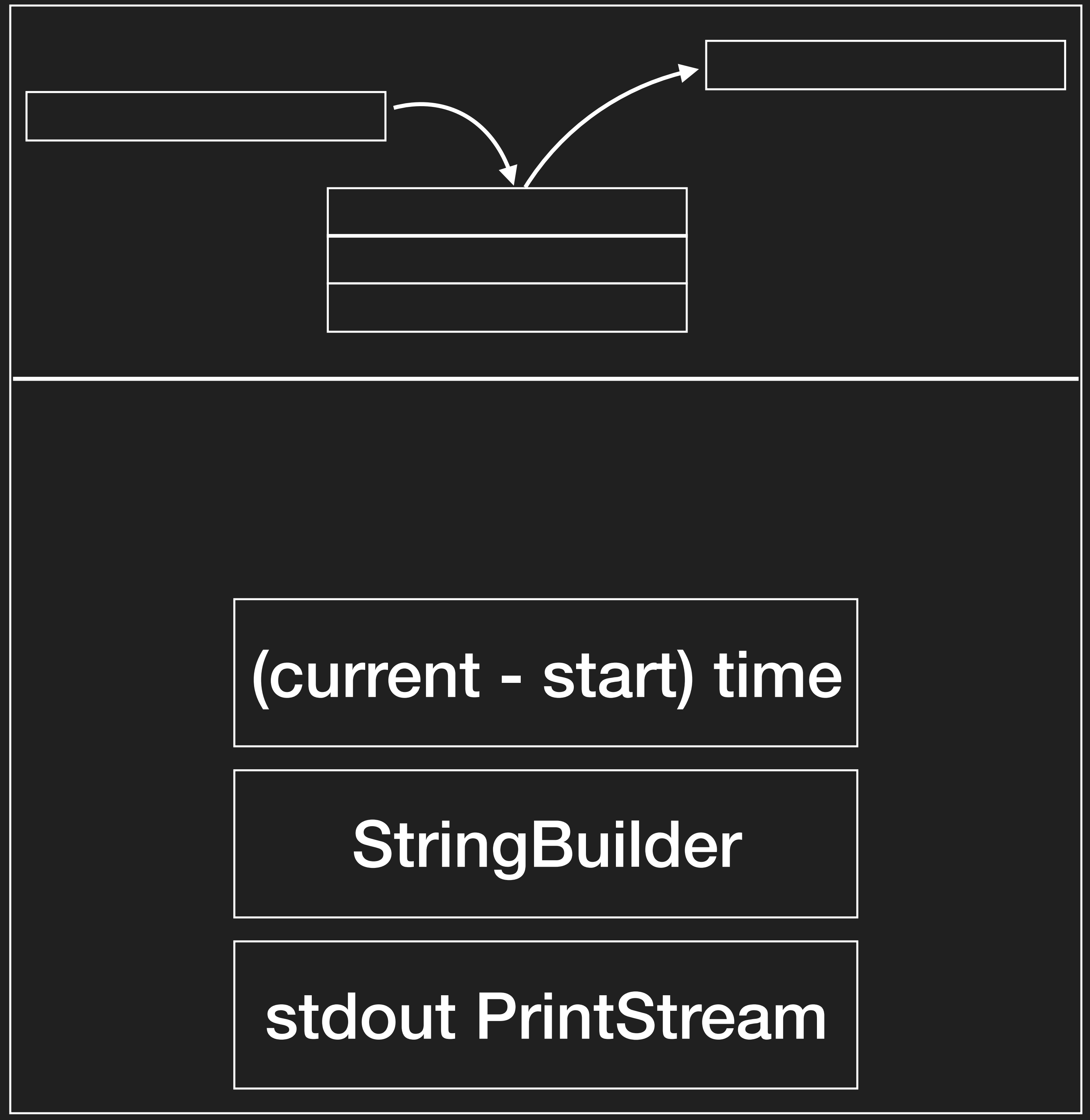
kotlin-plugin/src/main/kotlin/debuglog/DebugLogClassBuilder.kt

```
InstructionAdapter(this).apply   {
getstatic("j/l/System", "out", "Lj/io/PrintStream;")
anew("java/lang/StringBuilder")
dup()
invokespecial("j/l/StringBuilder", "<init>", "()V")
visitLdcInsn("⇠ ${function.name} [ran in ")
invokevirtual("j/l/StringBuilder", "append",
    "(Lj/l/String;)Lj/l/StringBuilder;")
invokestatic("j/l/System", "currentTimeMillis", "()J")
load(9001, LONG_TYPE)
sub(LONG_TYPE)
}
```



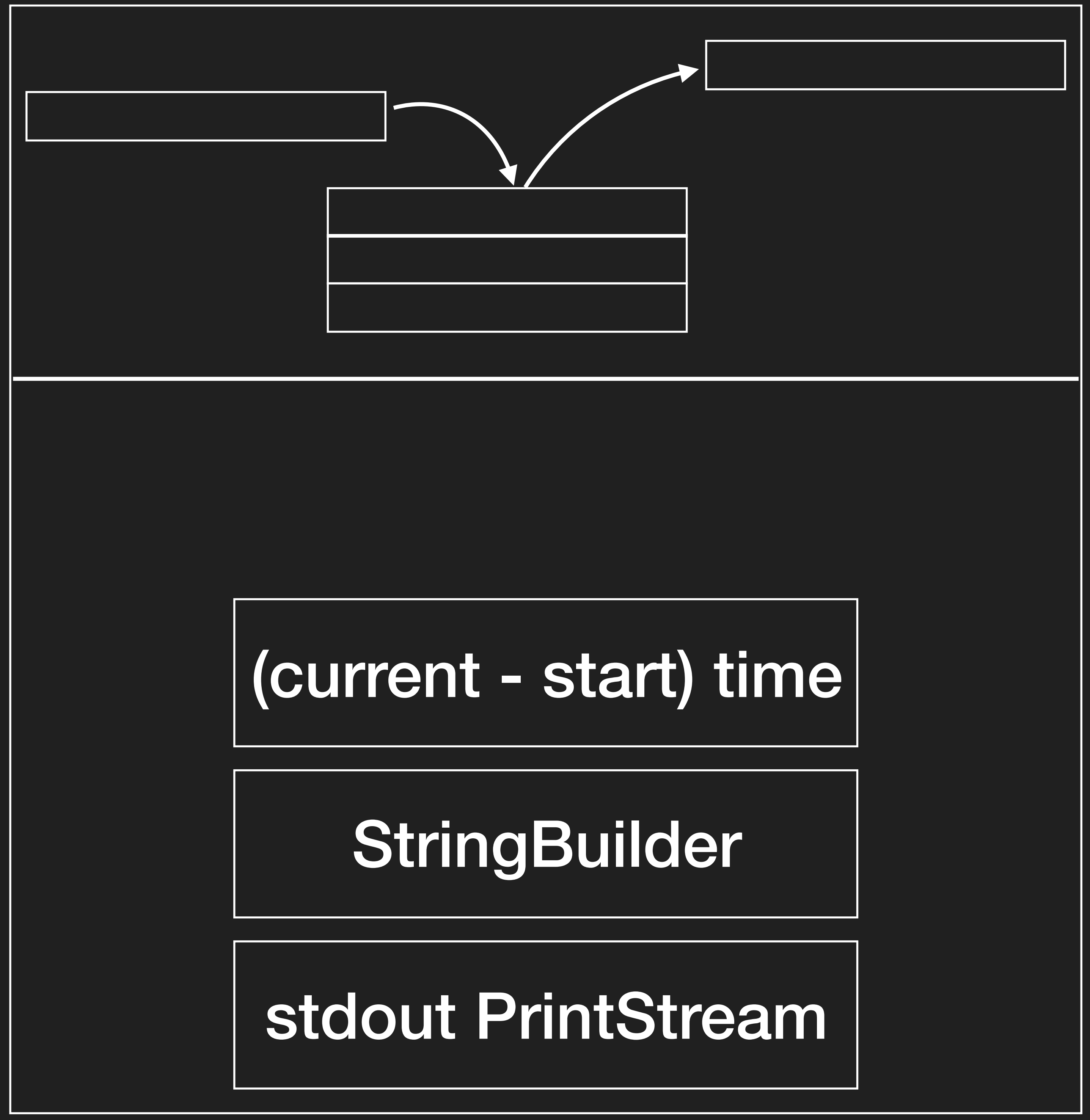
kotlin-plugin/src/main/kotlin/debuglog/DebugLogClassBuilder.kt

```
InstructionAdapter(this).apply  {
getstatic("j/l/System", "out", "Lj/io/PrintStream;")
anew("java/lang/StringBuilder")
dup()
invokespecial("j/l/StringBuilder", "<init>", "()V")
visitLdcInsn("⇠ ${function.name} [ran in ")
invokevirtual("j/l/StringBuilder", "append",
    "(Lj/l/String;)Lj/l/StringBuilder;")
invokestatic("j/l/System", "currentTimeMillis", "()J")
load(9001, LONG_TYPE)
sub(LONG_TYPE)
}
```



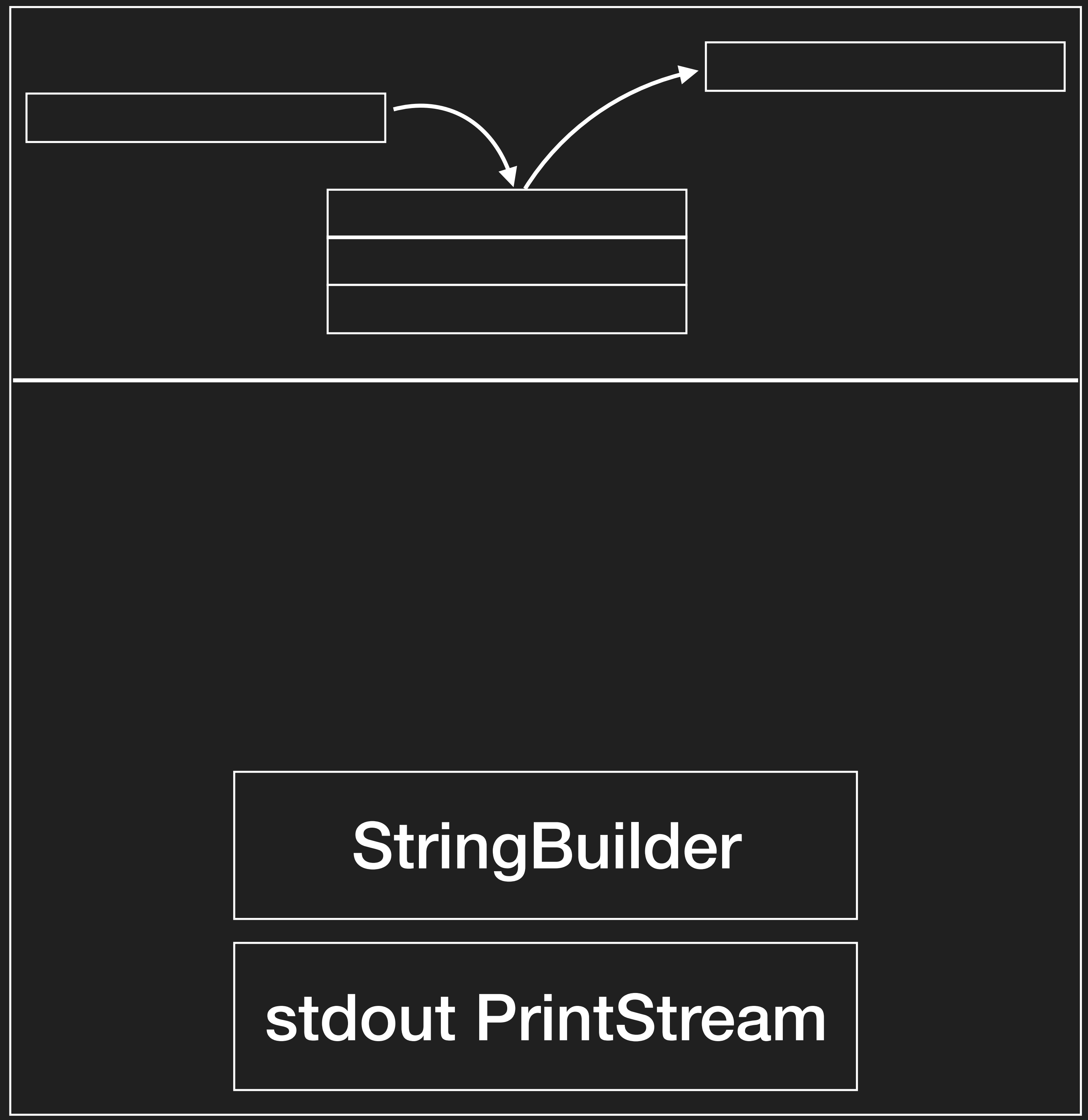
kotlin-plugin/src/main/kotlin/debuglog/DebugLogClassBuilder.kt

```
InstructionAdapter(this).apply  {
getstatic("j/l/System", "out", "Lj/io/PrintStream;")
anew("java/lang/StringBuilder")
dup()
invokespecial("j/l/StringBuilder", "<init>", "()V")
visitLdcInsn("⇠ ${function.name} [ran in ")
invokevirtual("j/l/StringBuilder", "append",
   "(Lj/l/String;)Lj/l/StringBuilder;")
invokestatic("j/l/System", "currentTimeMillis", "()J")
load(9001, LONG_TYPE)
sub(LONG_TYPE)
invokevirtual("j/l/StringBuilder", "append", "(J)Lj/l/StringBuilder;")
}
```



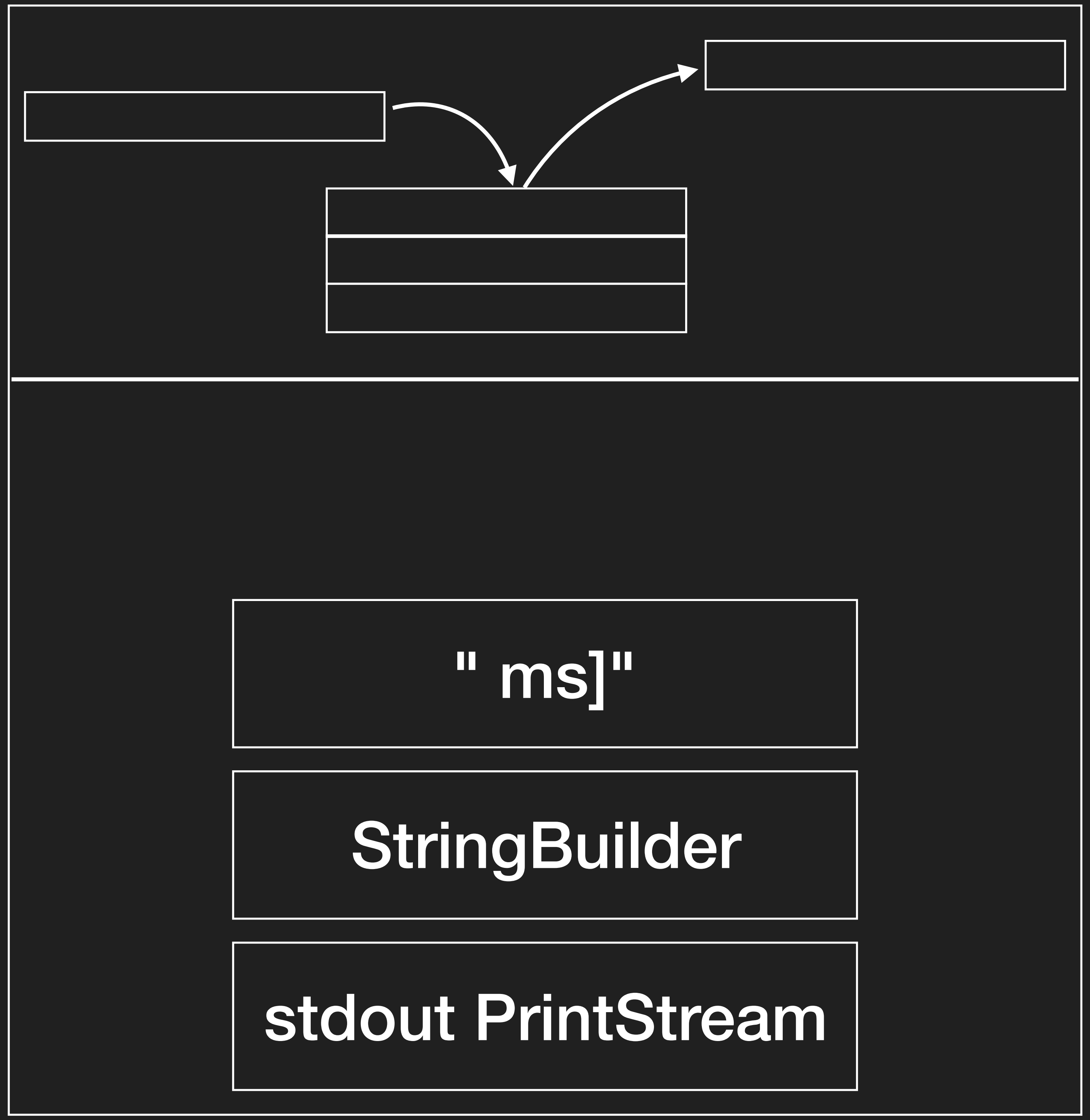
kotlin-plugin/src/main/kotlin/debuglog/DebugLogClassBuilder.kt

```
InstructionAdapter(this).apply  {
getstatic("j/l/System", "out", "Lj/io/PrintStream;")
anew("java/lang/StringBuilder")
dup()
invokespecial("j/l/StringBuilder", "<init>", "()V")
visitLdcInsn("⇠ ${function.name} [ran in ")
invokevirtual("j/l/StringBuilder", "append",
   "(Lj/l/String;)Lj/l/StringBuilder;")
invokestatic("j/l/System", "currentTimeMillis", "()J")
load(9001, LONG_TYPE)
sub(LONG_TYPE)
invokevirtual("j/l/StringBuilder", "append", "(J)Lj/l/StringBuilder;")
}
```



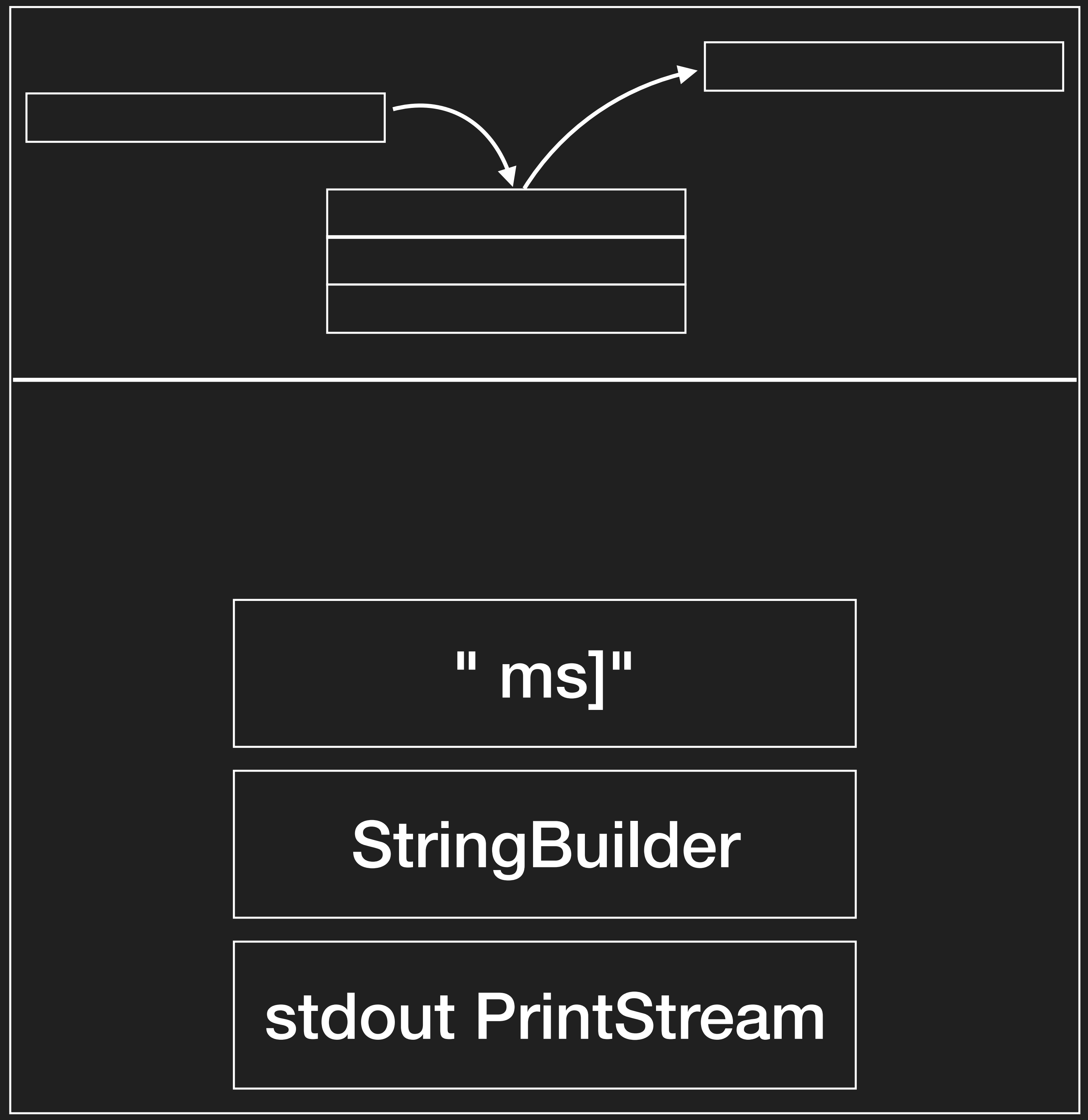
kotlin-plugin/src/main/kotlin/debuglog/DebugLogClassBuilder.kt

```
InstructionAdapter(this).apply  {
getstatic("j/l/System", "out", "Lj/io/PrintStream;")
anew("java/lang/StringBuilder")
dup()
invokespecial("j/l/StringBuilder", "<init>", "()V")
visitLdcInsn("⇠ ${function.name} [ran in ")
invokevirtual("j/l/StringBuilder", "append",
   "(Lj/l/String;)Lj/l/StringBuilder;")
invokestatic("j/l/System", "currentTimeMillis", "()J")
load(9001, LONG_TYPE)
sub(LONG_TYPE)
invokevirtual("j/l/StringBuilder", "append", "(J)Lj/l/StringBuilder;")
visitLdcInsn(" ms]")
}
```



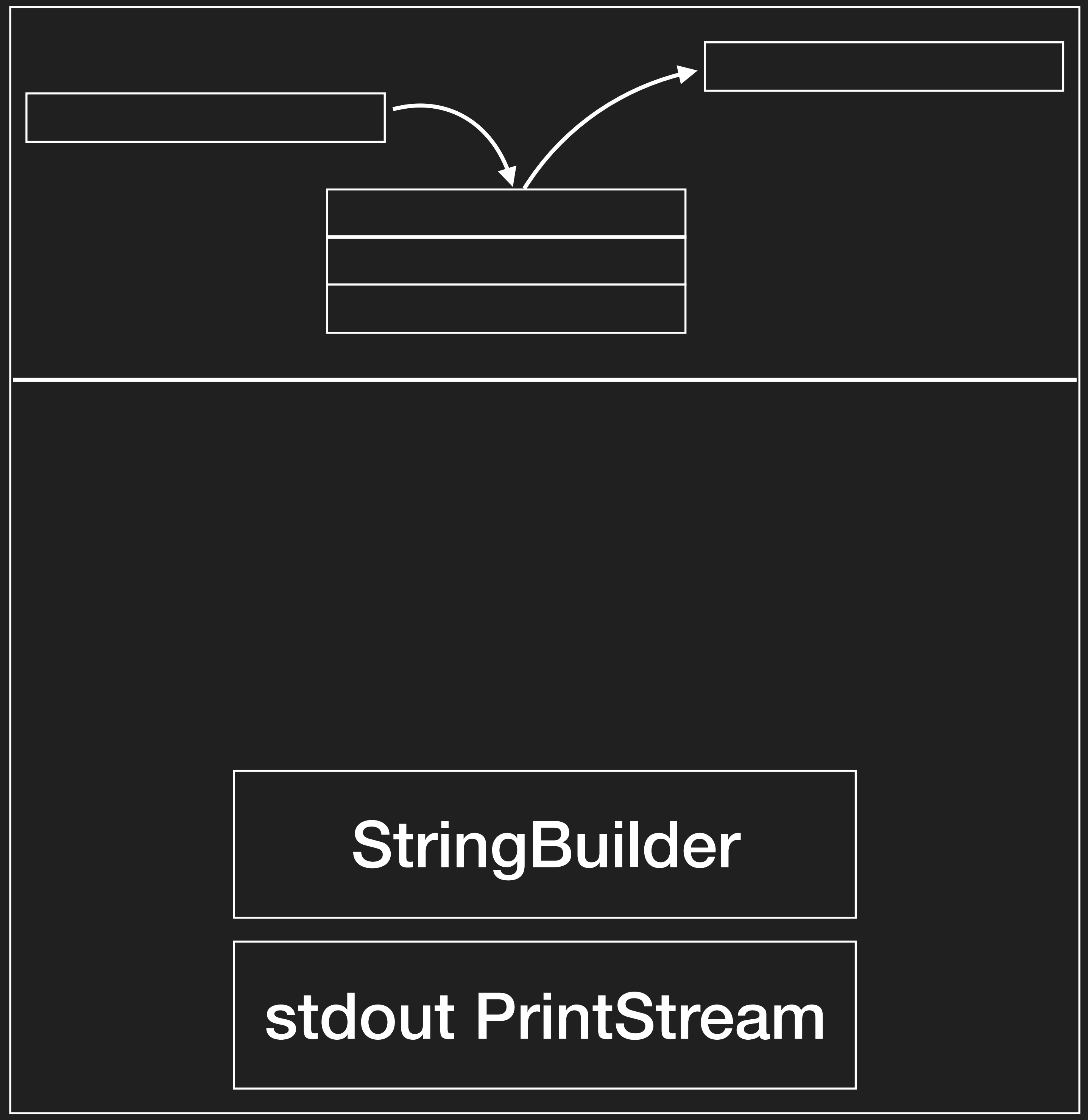
kotlin-plugin/src/main/kotlin/debuglog/DebugLogClassBuilder.kt

```
InstructionAdapter(this).apply  {
getstatic("j/l/System", "out", "Lj/io/PrintStream;")
anew("java/lang/StringBuilder")
dup()
invokespecial("j/l/StringBuilder", "<init>", "()V")
visitLdcInsn("⇠ ${function.name} [ran in ")
invokevirtual("j/l/StringBuilder", "append",
   "(Lj/l/String;)Lj/l/StringBuilder;")
invokestatic("j/l/System", "currentTimeMillis", "()J")
load(9001, LONG_TYPE)
sub(LONG_TYPE)
invokevirtual("j/l/StringBuilder", "append", "(J)Lj/l/StringBuilder;")
visitLdcInsn(" ms]")
invokevirtual("j/l/StringBuilder", "append", "(Lj/l/String;)Lj/l/SB;")
}
```



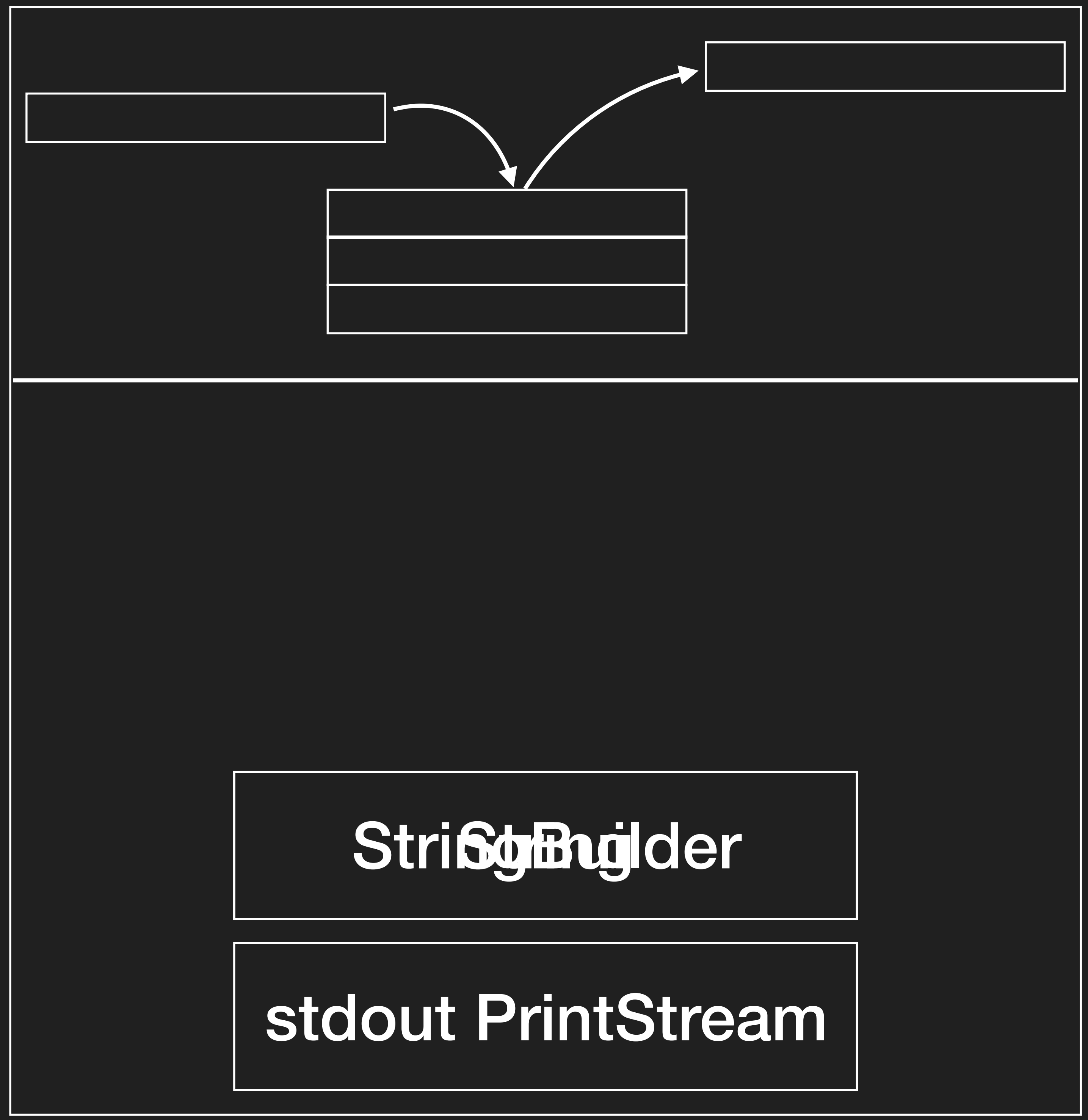
kotlin-plugin/src/main/kotlin/debuglog/DebugLogClassBuilder.kt

```
InstructionAdapter(this).apply  {
getstatic("j/l/System", "out", "Lj/io/PrintStream;")
anew("java/lang/StringBuilder")
dup()
invokespecial("j/l/StringBuilder", "<init>", "()V")
visitLdcInsn("⇠ ${function.name} [ran in ")
invokevirtual("j/l/StringBuilder", "append",
   "(Lj/l/String;)Lj/l/StringBuilder;")
invokestatic("j/l/System", "currentTimeMillis", "()J")
load(9001, LONG_TYPE)
sub(LONG_TYPE)
invokevirtual("j/l/StringBuilder", "append", "(J)Lj/l/StringBuilder;")
visitLdcInsn(" ms]")
invokevirtual("j/l/StringBuilder", "append", "(Lj/l/String;)Lj/l/SB;")
}
```



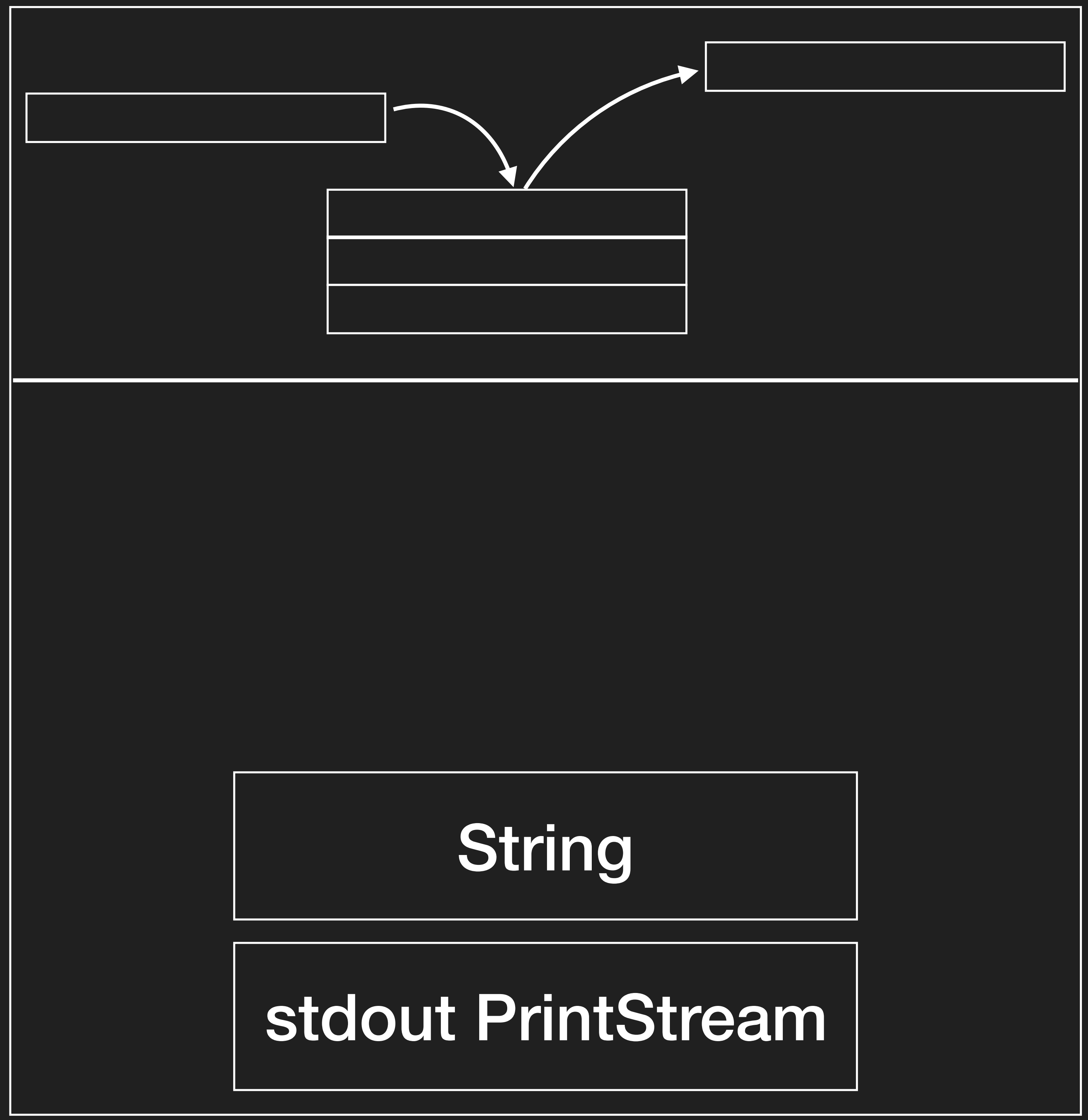
kotlin-plugin/src/main/kotlin/debuglog/DebugLogClassBuilder.kt

```
InstructionAdapter(this).apply  {
getstatic("j/l/System", "out", "Lj/io/PrintStream;")
anew("java/lang/StringBuilder")
dup()
invokespecial("j/l/StringBuilder", "<init>", "()V")
visitLdcInsn("⇠ ${function.name} [ran in ")
invokevirtual("j/l/StringBuilder", "append",
   "(Lj/l/String;)Lj/l/StringBuilder;")
invokestatic("j/l/System", "currentTimeMillis", "()J")
load(9001, LONG_TYPE)
sub(LONG_TYPE)
invokevirtual("j/l/StringBuilder", "append", "(J)Lj/l/StringBuilder;")
visitLdcInsn(" ms]")
invokevirtual("j/l/StringBuilder", "append", "(Lj/l/String;)Lj/l/SB;")
invokevirtual("j/l/StringBuilder", "toString", "()Lj/l/String;")
}
```



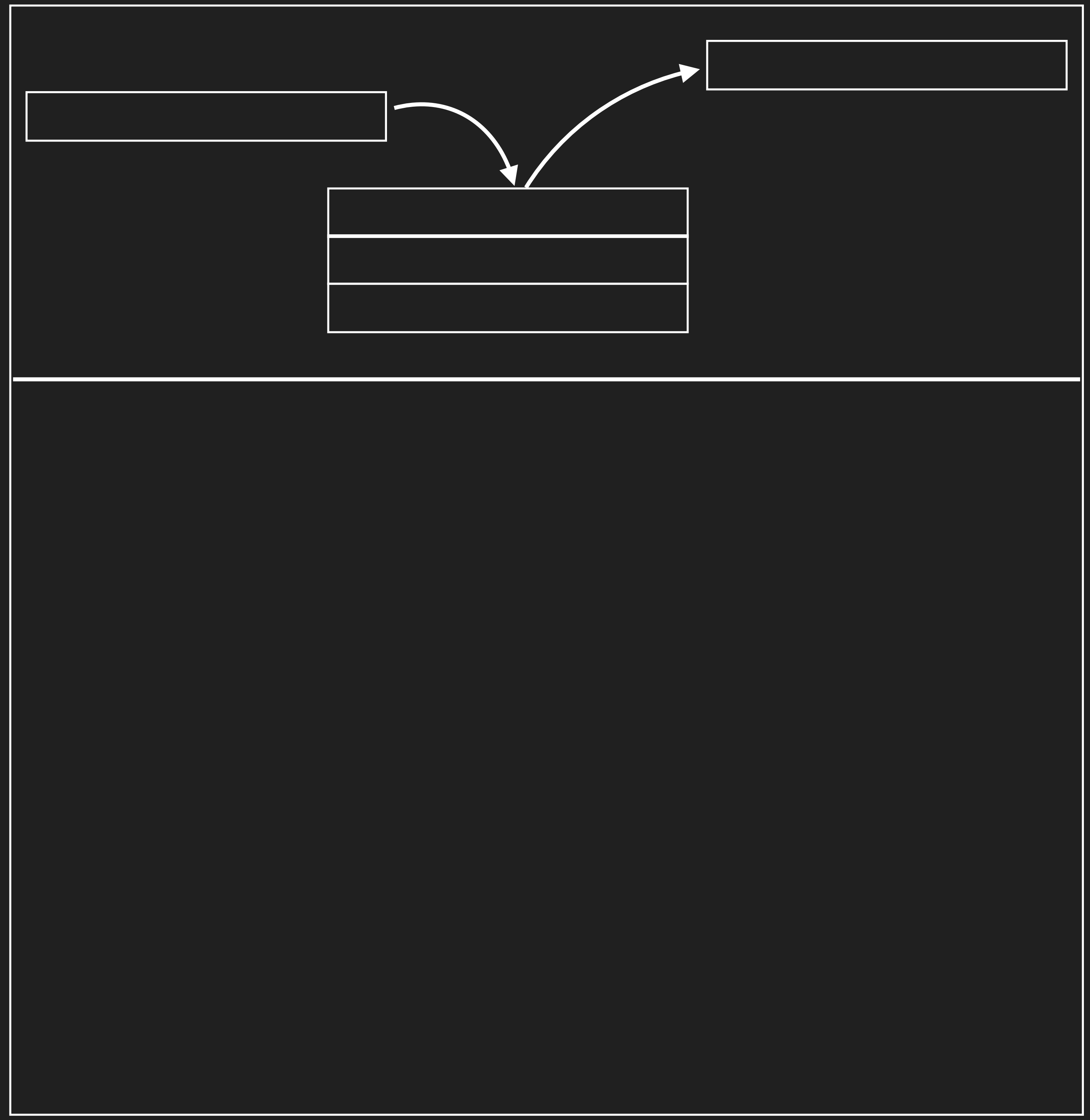
kotlin-plugin/src/main/kotlin/debuglog/DebugLogClassBuilder.kt

```
InstructionAdapter(this).apply  {
getstatic("j/l/System", "out", "Lj/io/PrintStream;")
anew("java/lang/StringBuilder")
dup()
invokespecial("j/l/StringBuilder", "<init>", "()V")
visitLdcInsn("⇠ ${function.name} [ran in ")
invokevirtual("j/l/StringBuilder", "append",
   "(Lj/l/String;)Lj/l/StringBuilder;")
invokestatic("j/l/System", "currentTimeMillis", "()J")
load(9001, LONG_TYPE)
sub(LONG_TYPE)
invokevirtual("j/l/StringBuilder", "append", "(J)Lj/l/StringBuilder;")
visitLdcInsn(" ms]")
invokevirtual("j/l/StringBuilder", "append", "(Lj/l/String;)Lj/l/SB;")
invokevirtual("j/l/StringBuilder", "toString", "()Lj/l/String;")
invokevirtual("j/io/PrintStream", "println", "(Lj/l/String;)V")
}
```



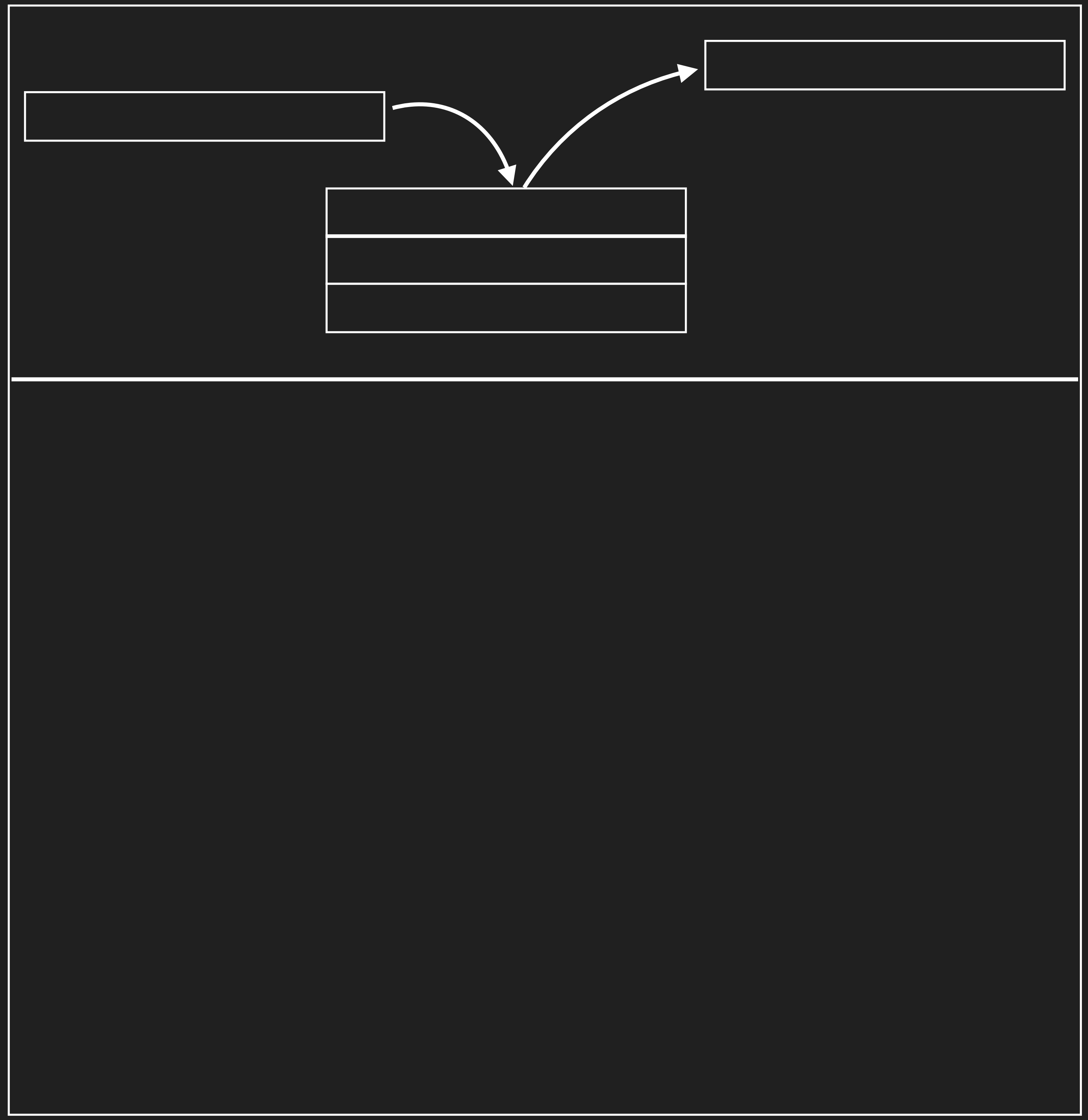
kotlin-plugin/src/main/kotlin/debuglog/DebugLogClassBuilder.kt

```
InstructionAdapter(this).apply  {
getstatic("j/l/System", "out", "Lj/io/PrintStream;")
anew("java/lang/StringBuilder")
dup()
invokespecial("j/l/StringBuilder", "<init>", "()V")
visitLdcInsn("⇠ ${function.name} [ran in ")
invokevirtual("j/l/StringBuilder", "append",
   "(Lj/l/String;)Lj/l/StringBuilder;")
invokestatic("j/l/System", "currentTimeMillis", "()J")
load(9001, LONG_TYPE)
sub(LONG_TYPE)
invokevirtual("j/l/StringBuilder", "append", "(J)Lj/l/StringBuilder;")
visitLdcInsn(" ms]")
invokevirtual("j/l/StringBuilder", "append", "(Lj/l/String;)Lj/l/SB;")
invokevirtual("j/l/StringBuilder", "toString", "()Lj/l/String;")
invokevirtual("j/io/PrintStream", "println", "(Lj/l/String;)V")
}
```



kotlin-plugin/src/main/kotlin/debuglog/DebugLogClassBuilder.kt

```
InstructionAdapter(this).apply {
// ... benchmark-printing code
}
```



kotlin-plugin/src/main/kotlin/debuglog/DebugLogClassBuilder.kt

```
return object : MethodVisitor(Opcodes.ASM5, original) {
  override fun visitCode() {
    super.visitCode()
    InstructionAdapter(this).apply  {
      // ... method-trace-printing code
      // ... timestamp-storing code
    }
  }
  override fun visitInsn(opcode: Int) {
    when (opcode) {
      RETURN , ARETURN, IRETURN -> {
        InstructionAdapter(this).apply   {   //   ...   benchmark-printing   code }
      }
    }
    super.visitInsn(opcode)
  }
}
```

kotlin-plugin/src/main/kotlin/debuglog/DebugLogClassBuilder.kt

```kotlin
return object : MethodVisitor(Opcodes.ASM5, original) {
  override fun visitCode() {
    super.visitCode()
    InstructionAdapter(this).apply  {
      // ... method-trace-printing code
      // ... timestamp-storing code
    }
  }
  override fun visitInsn(opcode: Int) {
    when (opcode) {
      RETURN , ARETURN, IRETURN -> {
        InstructionAdapter(this).apply   {   //   ...   benchmark-printing   code }
      }
    }
    super.visitInsn(opcode)
  }
}
```

Done! Demo time!!!

Resources

- https://github.com/JetBrains/kotlin/tree/master/plugins. Specifically:
    - noarg: One of the simplest ones
    - android-extensions: Good prior art for many of the extension types
    - kotlin-serialization: Newest one, documented well, generates LLVM
- https://github.com/JetBrains/kotlin/tree/master/libraries/tools
    - All look pretty similar; noarg and allopen are the simplest to grok

Thank you!

Kevin Most