Gearset

Static code analysis for Apex

A best-practice guide
to improving your code quality

DevOps for Salesforce

So, you want to write better Apex?

Software development is a craft and, like any craft, one can always get better at it. In this whitepaper we'll discuss how static code analysis as part of your development lifecycle can help you craft better software.

A little scene-setting: who is this whitepaper for?

People building on the Salesforce platform are a mix of admins, developers, admins-cum-developers, and developers-cum-admins. So, unless you're a pure admin who never touches Apex, this paper is for you.

While Apex development is practiced in many forms, you might fit one of these archetypes:

Highly-disciplined developers

Those who learned computer science and refined their trade in other languages with mature development frameworks and techniques. For whatever reason, you're coding in Apex now.

Under-time-&-budget-pressure developers

Regardless of your training, you've learned the Salesforce Apex idioms (e.g. always bulkify, no callouts in triggers, etc.) but otherwise you jam out the Apex to get your stories done as quickly as possible. You feel a little dirty about your work product and would like to leave a better legacy for yourself and others.

The experienced sole practitioner

You've been doing Apex for some time and know you could write better software but need a push from your tools to get you going.

The occasional developer

You only do sporadic Apex amongst your primarily admin-type work (process builder and visual flow) but you have a feeling that you'll be doing more Apex in the future. Regardless of who you are, static code analysis can help you.

About the authors

Eric Kintzer

Eric Kintzer has been working on Salesforce projects since 2007 as an all- singing, all-dancing developer, architect, and admin for high tech companies including Yahoo! and VMware. He is a passionate user of Salesforce enterprise patterns including Andrew Fawcett's Force.com Enterprise Architecture and ApexMocks. He is currently Salesforce Architect at Helix, a marketplace for direct-to-consumer DNA-based products.

Gearset

Gearset is a DevOps solution designed for everyone. Whether you're looking to adopt an Agile release process, improve developer collaboration or speed up project delivery, Gearset's easy metadata and data deployments, integration with Salesforce DX and powerful automation can help.

Contents

What is Apex and why do I need static code analysis?

Apex is a programming language designed to let you add and interact with data in Salesforce, and tailored to give organizations more flexibility in their customization of the Salesforce platform.

Apex already comes with a built-in unit test framework to examine functionality and test code coverage, so you may be thinking “*why do I need static code analysis as well?*”

Unit tests are important because they help demonstrate intended behaviour and functional correctness of your code. They can also make your code easier to change by encouraging you to write more loosely coupled, modular code, and by providing an early warning of any bugs that might creep in when making changes. Code coverage is a great way to ensure that the bulk of your Apex is under test.

While unit tests and code coverage have the important side-effect of forcing you to write more maintainable code, static code analysis explicitly formalizes a series of coding patterns, practices, and heuristics into a series of rules that can be periodically run against your code to assess its quality. By automating static code analysis and building it into your development process, you can identify style violations, bugs, and even more serious performance and security-related issues as you develop, long before they make it into production.

Your toolkit for better Apex development

Static code analysis is just one tool towards writing better Apex. In fact, it's not the first tool you should consider. Briefly, these other methods and tools will have higher payoff than static code analysis alone, so be sure to add them to your arsenal too.

1) Patterns

The best software exploits well-known software patterns. Trigger frameworks are an example that are familiar to many Apex developers. An even more powerful set of patterns, based on Martin Fowler's Separation of Concerns, is the Force.com Enterprise Architecture patterns featured in Salesforce Trailhead here and here as well as in book form.

2) Thorough regression test suites

Test methods that only do code coverage without asserts should be eschewed. Follow SFDC best practices for testing - positive, negative, and bulk testing; testing under different `runAs users` ; etc.

3) Reusable libraries

Move org-independent methods or code fragments into independent classes, shared across the code and between tests. Look for libraries on GitHub that do useful things so you don't have to reinvent infrastructure.

What is static code analysis?

Static code analysis reviews your source code to detect common bad practices, catch bugs, and make sure development adheres to coding guidelines. Most static code analysis tools define a series of rulesets that identify different categories of issue in your code, for example:

Best practices	Enforcing generally accepted Apex best practices
Code style	Enforcing a specific coding style
Design	Helping you discover design issues
Error prone rules	Detecting constructs that are either broken, extremely confusing, or prone to runtime errors
Performance	Detecting constructs that are either broken, extremely confusing, or prone to runtime errors
Security	Detecting potential Apex/SFDC security flaws

In general, rulesets like code style, design and error prone will broadly apply across a whole family of languages, whereas best practices, performance and security will be targeted to your specific environment, in this case Apex code for Salesforce. Failure to heed performance and security violations in particular could lead to future breaks, outages, or worse.

There are a number of tools you can use to implement static code analysis as part of your development process, such as Checkstyle or PMD. Gearset uses the open source PMD library that has static code analysis rulesets for many languages, including Apex. Quoting from the PMD GitHub website:

“PMD is a source code analyzer. It finds common programming flaws like unused variables, empty catch blocks, unnecessary object creation, and so forth.

Additionally it includes CPD, the copy-paste-detector. CPD finds duplicated code in ... Salesforce.com Apex and Visualforce.”

If you’re working in Gearset, these Apex rulesets can be customised and applied to each of your deployments as well as your continuous integration (CI) jobs and org monitoring setup. But more on this later.

Static code analysis with Gearset

Gearset is an end-to-end DevOps solution, designed to make release management on the Salesforce platform really easy. One of the ways it helps is by automatically running static code analysis at the key stages of your development process.

On top of this, Gearset also has a variety of great deployment features, including easy metadata and data deployments, deployment rollback, continuous integration, automated change monitoring, and full support for Salesforce DX.

The following sections show you how you can use Gearset’s static code analysis to help you write and maintain better Apex.

The signal and the noise

To benefit most from static code analysis, you'll want it embedded throughout multiple stages of your development cycle. You'll probably start off by examining the static code analysis results in your PROD org to see "how bad is it?" (or, if optimistically-minded, "how good is it?"). A typical org will give back a list of ruleset violations roughly proportional to the amount of deployed Apex.

Looking at an existing org is useful if you are inheriting the org from others. If you're a consultant, the scope of the report's results might justify a rate increase!

But seriously, the static code analysis results need to be examined in detail. Some ruleset violations might be benign or not that important to your org. Other violations might look serious but the repair effort might not warrant the cost.

Benign violation examples:

- Code style violations like `if then else` without braces is a matter for philosophical debate in some quarters.
- Violations in the unmanaged packages you're using are likely a matter for the unmanaged package developer, and you should raise an issue with the package on GitHub.

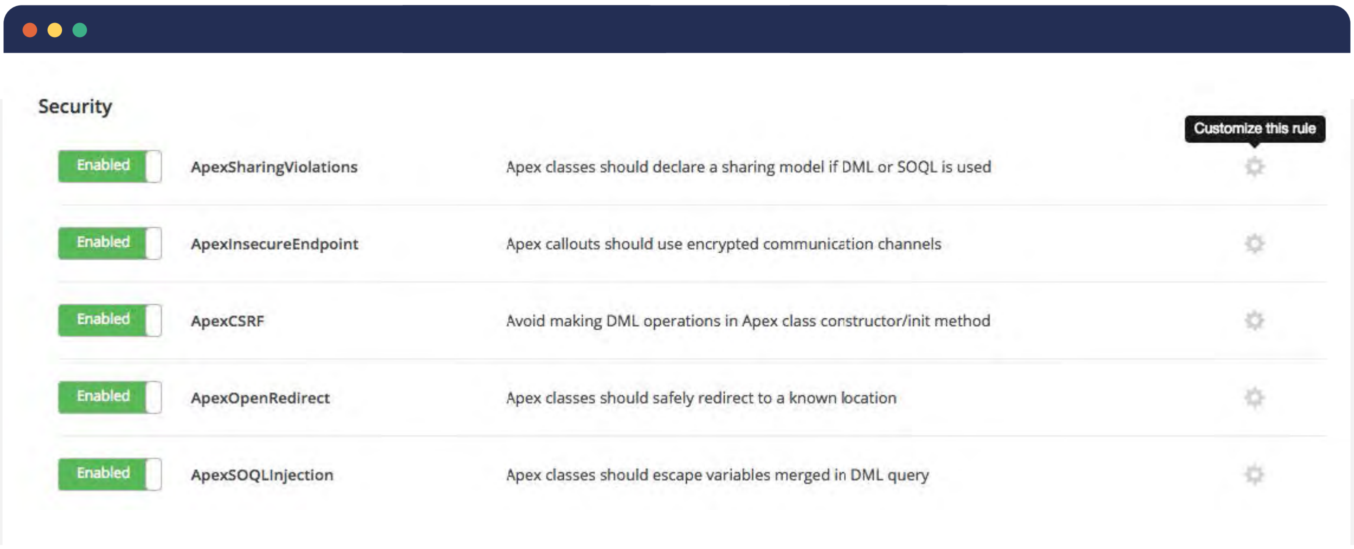
Serious-but-costly-to-fix examples:

Standard cyclomatic complexity; to paraphrase United States Supreme Court Justice Potter Stevens in a 1964 obscenity case, "*you'll know it when you see it*". The method will be indecipherable to inspection, but it could be some fundamental piece of critical business logic and refactoring it could run serious risks of breaks.

Using Gearset to customise your rule set and screen out the noise

Gearset's customizable static code analysis provides a mechanism to help screen out the noise, and there are a number of settings that you can configure to build the most appropriate rule set.

As a blunt instrument, you can enable or disable specific rules entirely.



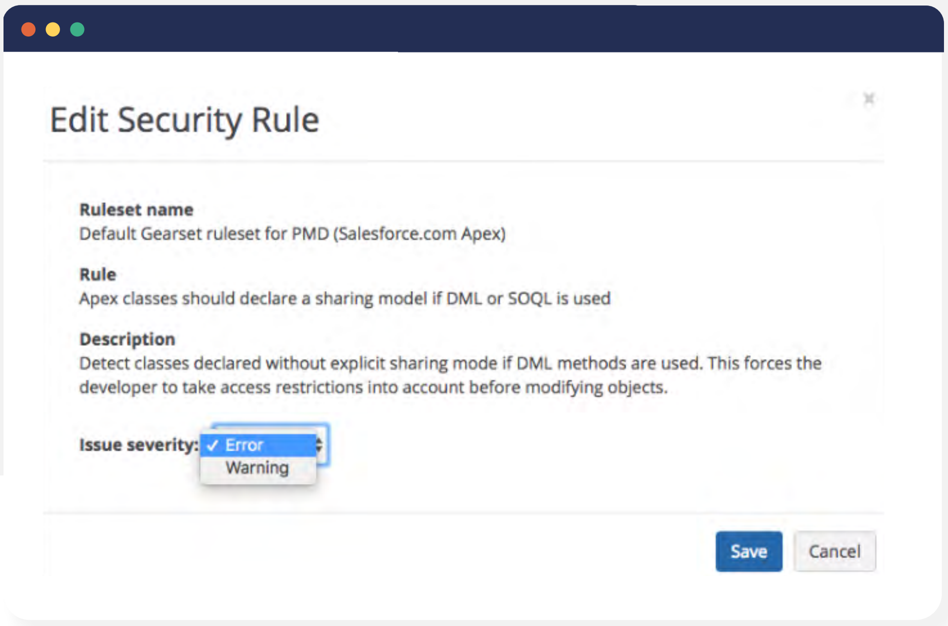
Security

Customize this rule

Enabled	ApexSharingViolations	Apex classes should declare a sharing model if DML or SOQL is used
Enabled	ApexInsecureEndpoint	Apex callouts should use encrypted communication channels
Enabled	ApexCSRF	Avoid making DML operations in Apex class constructor/init method
Enabled	ApexOpenRedirect	Apex classes should safely redirect to a known location
Enabled	ApexSOQLInjection	Apex classes should escape variables merged in DML query

For a more fine-grained approach, you can choose how you want to define the severity of a rule if a violation is detected. More serious violations can be categorized and flagged as errors, while others can just be tagged as warnings in the results summary.

Gearset



Edit Security Rule

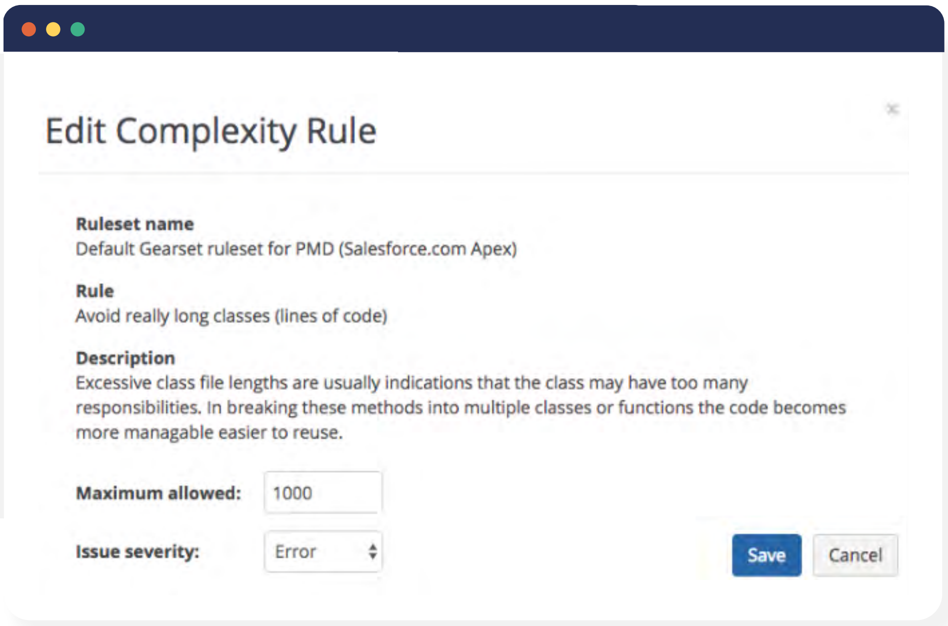
Ruleset name
Default Gearset ruleset for PMD (Salesforce.com Apex)

Rule
Apex classes should declare a sharing model if DML or SOQL is used

Description
Detect classes declared without explicit sharing mode if DML methods are used. This forces the developer to take access restrictions into account before modifying objects.

Issue severity: ✓ Error / Warning

For categories like **complexity**, you can also specify values to determine the precise level at which the rule will fire. For example, for complexity rule `ExcessiveClassLength` , you can specify the maximum class length (in lines) that can be allowed.



Edit Complexity Rule

Ruleset name
Default Gearset ruleset for PMD (Salesforce.com Apex)

Rule
Avoid really long classes (lines of code)

Description
Excessive class file lengths are usually indications that the class may have too many responsibilities. In breaking these methods into multiple classes or functions the code becomes more managable easier to reuse.

Maximum allowed: 1000

Issue severity: Error

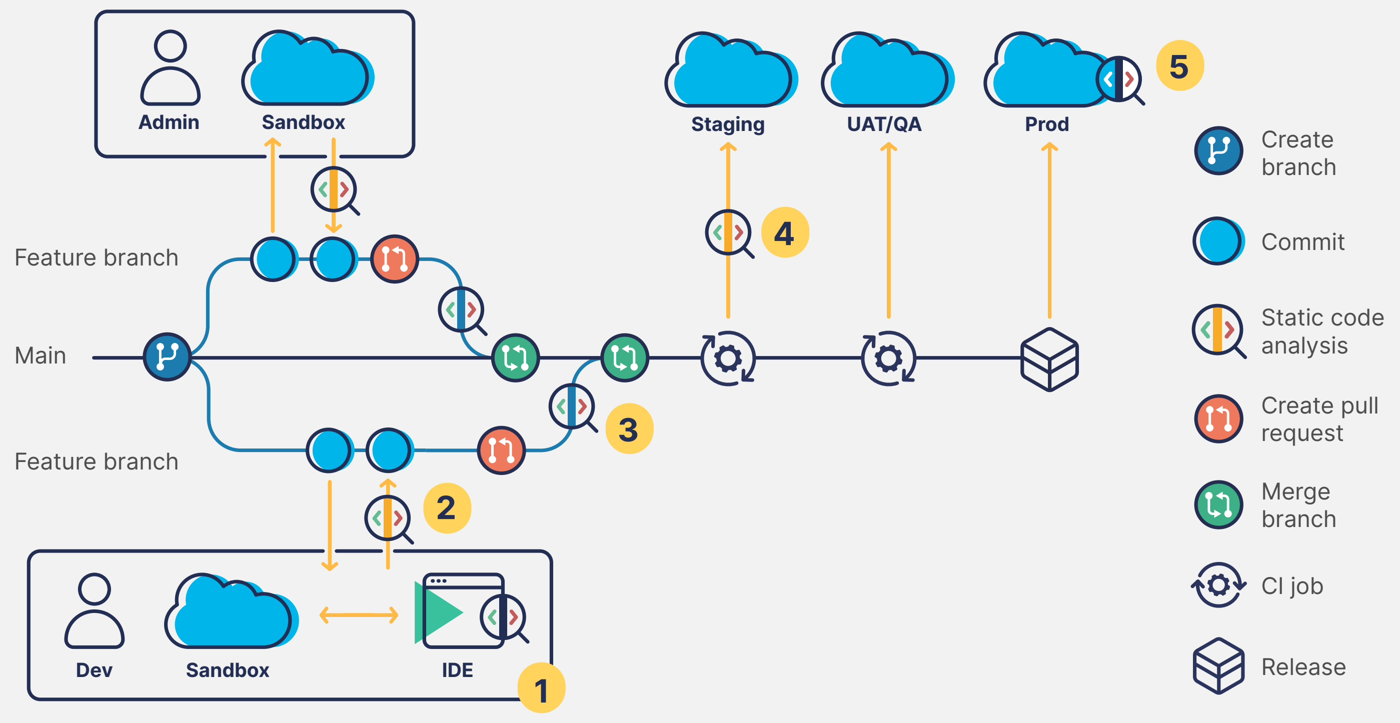
Using these settings, you can fine tune the levels to find the right balance between warnings and errors to meet your team's needs. For cases when Gearset's granularity proves too coarse, you may need to go in to individual classes or lines of code and use the PMD warning suppression techniques.

Bottom line: remove the noise from the analysis so only violations important to you and your team/org get highlighted. This will be an iterative process.

The five lines of defence

Once you've removed the noise, your team will need a workflow for addressing the remaining static code analysis violations.

Salesforce teams are increasingly adopting a git-based development process. As Gearset have previously written about in their version control whitepaper, this workflow often looks something like the model below:

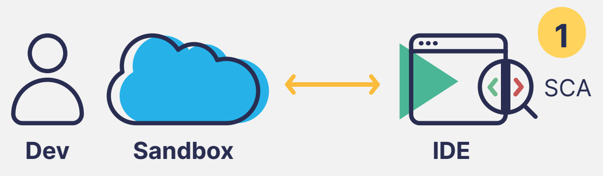


We've broken down this model into five key stages along the release pipeline, highlighting how and where static code analysis (referred to as SCA in the diagram) fits in.

These stages include development in a local IDE, pushing commits to feature branches, and deploying approved changes from master to production. Of course, each business is different, so you can tailor this model to fit the needs of your team.

1: The IDE

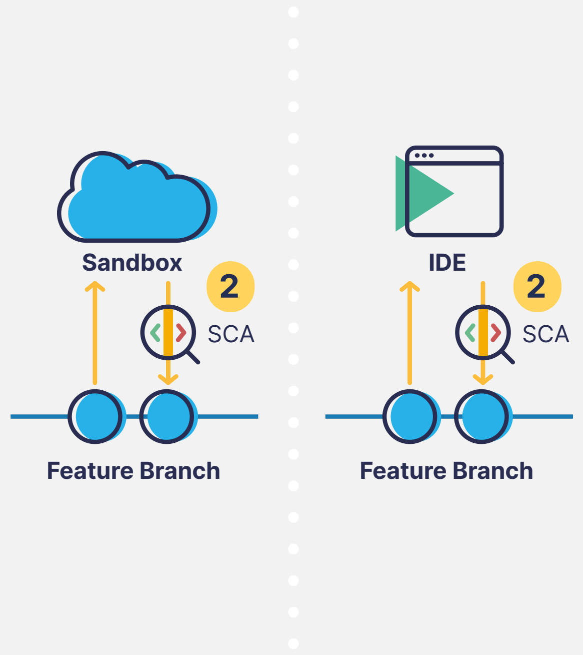
If you're using an IDE with PMD plugins, you can run the PMD analyzer as you develop. IDEs like Visual Studio, Illuminated Cloud, and the Welkin Suite identify violations and let you suppress noisy PMD warnings.



2: Gearset deployments

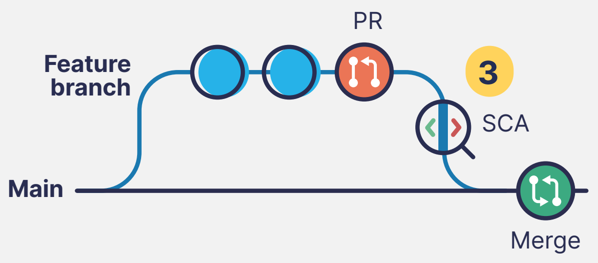
Whenever you prepare a deployment between your orgs or branches, Gearset will run static code analysis for you automatically.

If any issues are detected, Gearset will flag them, tell you which rules have been violated, and point you towards the offending code. You can then go back and fix your Apex, refresh the comparison, and deploy. Over time, more and more of your codebase will become violation-free



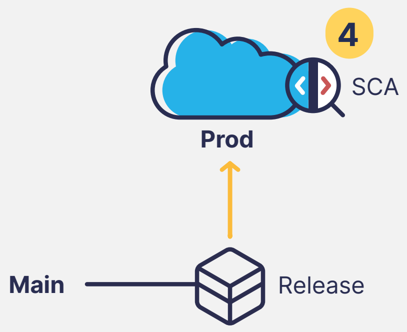
3: Code review

If you're doing code reviews, you can inspect the static code analysis report for that deployment as part of your checklist. Include a link to the report in your pull request. Perhaps a team lead is required to sign off on any violations to make sure your team is maintaining best-practice coding standards.



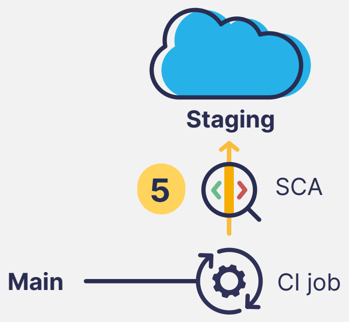
4: Gearset change monitoring

Gearset can also monitor changes and code quality in your orgs. You can monitor any org, but the most relevant ones are usually staging and production. Monitoring jobs run daily to give you a detailed audit trail of every change made in your org, while the code analysis provides consistent feedback on the quality of your Apex. Your team will be able to monitor the changing state of your orgs to make sure your coding standards are being met. You could even use the analysis results to set up quarterly team goals to reduce the violation counts by some percentage.



5: Continuous integration (CI) with Gearset

For more advanced DevOps, you can set up a CI job within Gearset to deploy from source control to your orgs. The CI job will detect the new changes in your repository and automatically deploy them to your integration testing sandbox (usually a partial copy) for rapid testing. Static code analysis will be performed as part of this CI job, giving your team the chance to continuously check and monitor changes and call attention to any rule violations.



Summary

It’s just one piece of the puzzle for writing better Apex, but static code analysis plays a crucial role in monitoring code development across your Salesforce environments. By regularising coding style and practices across teams, you can ultimately benefit from greater productivity and fewer breaks and outages.

Consistently improving Apex as part of your release management process can be tricky. By using a DevOps tool like Gearset, your team can automate this process, and continuously review code throughout each stage of development. Gearset’s configurable static code analysis lets you screen out benign violations so you can build the most appropriate rule set for your team and monitor what matters most in your orgs. With code analysis built into Gearset’s deployment flow, as well as monitoring and CI jobs, you can make sure your Apex is continuously developed to a high standard as changes are pushed down the release pipeline.

Further reading

Want to know more about static code analysis?
Take a look at these resources to get started:

Apex Enterprise Patterns — Trailhead module
https://trailhead.salesforce.com/en/modules/apex_patterns_sl

Change monitoring — Gearset app
https://app.gearset.com/change-alerts

Checkstyle — development tool
http://checkstyle.sourceforge.net/

Continuous integration — Gearset app
https://app.gearset.com/continuous-integration

Feature walkthrough — for Gearset's static code analysis
https://docs.gearset.com/feature-walkthroughs/static-code-analysis

Force.com — Enterprise Architecture
https://andyinthecloud.com/2017/04/01/force-com-enterprise-architecture

Gearset whitepaper — a guide to version control for Salesforce
https://gearset.com/assets/version-control-for-salesforce-whitepaper.pdf

PMD — Apex rulesets
https://pmd.github.io/pmd-6.2.0/pmd_rules_apex.html

PMD — source code analyser tool
https://pmd.github.io/

Salesforce — Apex Developer Guide
https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_dev_guide.htm

Gearset