**Company-Specific Coding Standards**

**Leading Technology Companies Best Practices**

**1. Google Coding Standards**

**Philosophy**

Google's approach covers everything from "use camelCase for variable names" to "never use global variables" to "never use exceptions", emphasizing consistency and readability across massive codebases.

**Java Standards (Google Style)**

**File Structure:**

// Copyright header required

/\*

\* Copyright 2025 Google LLC

\*

\* Licensed under the Apache License, Version 2.0

\*/

package com.google.project.module;

import java.util.List;

import java.util.ArrayList;

import com.google.common.collect.ImmutableList;

import com.google.project.other.ClassFromSameCompany;

import org.apache.commons.lang3.StringUtils;

/\*\*

\* Brief description of class purpose.

\*

\* <p>Detailed description with examples if needed.

\*/

public final class ClassName {

// Implementation

}

**Naming Conventions:**

* Classes: UpperCamelCase (e.g., HttpClient)
* Methods/Variables: lowerCamelCase (e.g., getUserId)
* Constants: CONSTANT\_CASE (e.g., MAX\_RETRY\_COUNT)
* Packages: lowercase.with.dots

**Code Structure:**

* Maximum line length: 100 characters
* Indentation: 2 spaces (no tabs)
* Braces: K&R style (opening brace on same line)
* One top-level class per file

**Google-Specific Rules:**

* No wildcard imports (import java.util.\* forbidden)
* Use @Override annotation consistently
* Prefer ImmutableList over ArrayList for collections that won't change
* Use Guava libraries for common operations
* Mandatory Javadoc for all public methods and classes

**Error Handling:**

// Google Style - Specific exceptions

public User getUser(String userId) throws UserNotFoundException {

checkArgument(!Strings.isNullOrEmpty(userId), "User ID cannot be null or empty");

Optional<User> user = userRepository.findById(userId);

return user.orElseThrow(() -> new UserNotFoundException("User not found: " + userId));

}

**Testing Standards:**

* Test class naming: ClassNameTest
* Test method naming: methodName\_stateUnderTest\_expectedBehavior
* Use Truth assertion library instead of JUnit assertions
* Minimum 85% code coverage required

**Python Standards (Google Style)**

**File Structure:**

#!/usr/bin/env python3

"""Module docstring describing the module purpose.

This module demonstrates Google Python style guidelines.

"""

import os

import sys

from typing import List, Optional, Dict, Any

from absl import app

from absl import flags

from absl import logging

import third\_party\_library

# Module constants

MAX\_RETRY\_COUNT = 3

DEFAULT\_TIMEOUT = 30

class ClassName:

"""Class docstring following Google style."""

def \_\_init\_\_(self, param: str) -> None:

"""Initialize the class.

Args:

param: Description of parameter.

"""

self.\_param = param

**Google-Specific Python Rules:**

* Use absl-py for command-line flags and logging
* Type hints mandatory for all function signatures
* Maximum line length: 80 characters
* Use pylint with Google's configuration
* Docstrings must follow Google format (not NumPy or Sphinx)

**JavaScript/TypeScript Standards (Google Style)**

**File Structure:**

/\*\*

\* @fileoverview Brief description of file contents.

\*/

import {SomeClass} from './some-file';

import \* as utils from '../utils/helper-utils';

/\*\*

\* Description of the class.

\*/

export class ComponentName {

private readonly property\_: string;

constructor(property: string) {

this.property\_ = property;

}

/\*\*

\* Description of method.

\* @param param Description of parameter.

\* @return Description of return value.

\*/

public getProperty(param: number): string {

return `${this.property\_}\_${param}`;

}

}

**Google-Specific Rules:**

* Private properties end with underscore (private property\_)
* Use goog.module for internal projects
* Closure Compiler compatibility required
* JSDoc comments mandatory for all exports

**2. Microsoft Coding Standards**

**Philosophy**

Microsoft emphasizes enterprise-grade reliability, security, and maintainability with strong typing and comprehensive documentation.

**C# Standards (.NET Framework)**

**File Structure:**

// <copyright file="UserService.cs" company="Microsoft">

// Copyright (c) Microsoft Corporation. All rights reserved.

// </copyright>

namespace Microsoft.Project.Services

{

using System;

using System.Collections.Generic;

using System.Threading.Tasks;

using Microsoft.Extensions.Logging;

using Microsoft.Project.Models;

/// <summary>

/// Provides user management functionality.

/// </summary>

public sealed class UserService : IUserService

{

private readonly ILogger<UserService> logger;

private readonly IUserRepository userRepository;

/// <summary>

/// Initializes a new instance of the <see cref="UserService"/> class.

/// </summary>

/// <param name="logger">The logger instance.</param>

/// <param name="userRepository">The user repository.</param>

public UserService(ILogger<UserService> logger, IUserRepository userRepository)

{

this.logger = logger ?? throw new ArgumentNullException(nameof(logger));

this.userRepository = userRepository ?? throw new ArgumentNullException(nameof(userRepository));

}

}

}

**Microsoft-Specific Rules:**

* Utilize modern language features and C# versions whenever possible, avoid outdated language constructs
* Use this. prefix for instance members
* Allman braces style (opening brace on new line)
* XML documentation required for all public APIs
* Use nullable reference types (C# 8+)
* Async methods must be suffixed with Async

**Naming Conventions:**

* Classes/Methods/Properties: PascalCase
* Private fields: camelCase
* Local variables: camelCase
* Constants: PascalCase

**Security Standards:**

// Input validation

public async Task<User> GetUserAsync(string userId)

{

if (string.IsNullOrWhiteSpace(userId))

{

throw new ArgumentException("User ID cannot be null or empty.", nameof(userId));

}

// Use parameterized queries

var user = await this.userRepository.GetByIdAsync(userId).ConfigureAwait(false);

return user ?? throw new UserNotFoundException($"User with ID '{userId}' not found.");

}

**TypeScript Standards (Microsoft Style)**

**File Structure:**

/\*

\* Copyright (c) Microsoft Corporation. All rights reserved.

\* Licensed under the MIT License.

\*/

import { EventEmitter } from 'events';

import { Logger } from './logger';

/\*\*

\* Represents a user service for managing user operations.

\*/

export class UserService extends EventEmitter {

private readonly logger: Logger;

constructor(logger: Logger) {

super();

this.logger = logger;

}

/\*\*

\* Retrieves a user by their unique identifier.

\* @param userId - The unique identifier of the user

\* @returns A promise that resolves to the user object

\* @throws {Error} When user is not found

\*/

public async getUser(userId: string): Promise<User> {

if (!userId?.trim()) {

throw new Error('User ID is required');

}

// Implementation

}

}

**Microsoft-Specific Rules:**

* Use TSDoc for documentation
* Strict TypeScript configuration
* Use explicit access modifiers
* Prefer readonly for immutable properties
* Use utility types (Partial, Pick, Omit) extensively

**4. Netflix Coding Standards**

**Philosophy**

Netflix emphasizes tools and techniques to go from source code to deployed services serving millions of global members, focusing on scalability and reliability.

**Microservices Architecture Standards**

**Service Structure:**

@RestController

@RequestMapping("/api/v1/content")

@Validated

public class ContentController {

private final ContentService contentService;

private final MetricsCollector metricsCollector;

public ContentController(ContentService contentService, MetricsCollector metricsCollector) {

this.contentService = contentService;

this.metricsCollector = metricsCollector;

}

@GetMapping("/{contentId}")

@CircuitBreaker(name = "content-service")

@RateLimiter(name = "content-api")

public ResponseEntity<ContentResponse> getContent(

@PathVariable @Valid @NotBlank String contentId,

@RequestHeader("X-User-ID") String userId) {

Timer.Sample sample = Timer.start(metricsCollector.registry());

try {

Content content = contentService.getContent(contentId, userId);

return ResponseEntity.ok(ContentResponse.from(content));

} finally {

sample.stop(metricsCollector.contentRetrievalTimer());

}

}

}

**Netflix-Specific Rules:**

* Every service must have circuit breakers
* Comprehensive metrics collection required
* Fault tolerance patterns mandatory
* Use Hystrix for resilience
* Implement bulkhead pattern for resource isolation

**Database Standards**

@Repository

public class ContentRepository {

private final JdbcTemplate jdbcTemplate;

private final RedisTemplate<String, Object> redisTemplate;

@Cacheable(value = "content", key = "#contentId")

@Retryable(value = {DataAccessException.class}, maxAttempts = 3)

public Optional<Content> findById(String contentId) {

// Primary database query with fallback

try {

return Optional.ofNullable(

jdbcTemplate.queryForObject(

"SELECT \* FROM content WHERE id = ?",

new Object[]{contentId},

new ContentRowMapper()

)

);

} catch (EmptyResultDataAccessException e) {

return Optional.empty();

}

}

}

**Netflix Database Rules:**

* Use caching extensively (Redis/Memcached)
* Implement read replicas for scaling
* Use connection pooling
* Implement database circuit breakers
* Partition data by geographic regions

**5. Amazon (AWS) Coding Standards**

**Philosophy**

Amazon emphasizes operational excellence, security, reliability, and cost optimization in all code.

**Java Standards (Amazon Style)**

**File Structure:**

package com.amazon.service.user;

import java.util.Optional;

import java.util.concurrent.CompletableFuture;

import com.amazon.coral.annotation.License;

import com.amazon.coral.annotation.Operation;

import com.amazon.coral.validate.Validated;

import org.apache.commons.lang3.Validate;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

/\*\*

\* Service for managing user operations.

\*

\* This service handles all user-related operations including

\* creation, retrieval, and updates while maintaining

\* compliance with AWS security standards.

\*/

@License("Apache-2.0")

public class UserService {

private static final Logger LOG = LoggerFactory.getLogger(UserService.class);

private static final int MAX\_RETRY\_ATTEMPTS = 3;

private final UserRepository userRepository;

private final MetricsPublisher metricsPublisher;

public UserService(UserRepository userRepository, MetricsPublisher metricsPublisher) {

this.userRepository = Validate.notNull(userRepository, "userRepository cannot be null");

this.metricsPublisher = Validate.notNull(metricsPublisher, "metricsPublisher cannot be null");

}

@Operation("GetUser")

public CompletableFuture<Optional<User>> getUserAsync(@Validated String userId) {

LOG.info("Retrieving user with ID: {}", userId);

return CompletableFuture

.supplyAsync(() -> userRepository.findById(userId))

.whenComplete((result, throwable) -> {

if (throwable != null) {

LOG.error("Failed to retrieve user: {}", userId, throwable);

metricsPublisher.publishCount("UserService.GetUser.Error", 1);

} else {

metricsPublisher.publishCount("UserService.GetUser.Success", 1);

}

});

}

}

**Amazon-Specific Rules:**

* All public methods must be async (return CompletableFuture)
* Extensive logging and metrics collection
* Use Apache Commons for validation
* Implement exponential backoff for retries
* All services must be thread-safe
* Use Coral annotations for service metadata

**Security Standards (Amazon)**

@Service

public class SecureUserService {

private final EncryptionService encryptionService;

private final AuditLogger auditLogger;

public User createUser(CreateUserRequest request) {

// Input validation

ValidationResult validation = validateUserRequest(request);

if (!validation.isValid()) {

auditLogger.logSecurityEvent("INVALID\_USER\_INPUT", request.getUserId());

throw new ValidationException(validation.getErrors());

}

// Encrypt sensitive data

String encryptedSsn = encryptionService.encrypt(request.getSsn());

String hashedPassword = passwordEncoder.encode(request.getPassword());

User user = User.builder()

.userId(request.getUserId())

.encryptedSsn(encryptedSsn)

.hashedPassword(hashedPassword)

.build();

auditLogger.logUserCreation(user.getUserId());

return userRepository.save(user);

}

}

**Amazon Security Rules:**

* All sensitive data must be encrypted at rest
* Comprehensive audit logging required
* Input validation on all boundaries
* Use least privilege access principles
* Implement proper authentication and authorization