CI Pipeline Documentation — Discount Logic Engine

Repository:
https://github.com/yachhapat/zip
CI Tool: GitHub Actions
Language: Java 21 (Maven)

---

1. Project Overview

This repository demonstrates a fully functional CI pipeline for a Maven-based Java project built around a **Discount Logic Engine**. It implements the following requirements:

1. Run a set of checks (1 linter and 1 set of unit tests) on every PR.
2. Require all checks to pass and 1 PR review approval before allowing the PR to be merged.
3. Require squash+rebase for PR merging.
4. Run the same set of checks on the main branch.

Technology Stack

Component	Tool	Version
Language	Java	21 (Temurin)
Build tool	Maven	3.9.6 (via Maven Wrapper)
Linter	Checkstyle	3.3.1 (maven-checkstyle-plugin)
Unit tests	JUnit 5	5.10.2 (junit-jupiter)
CI platform	GitHub Actions	v4 (actions/checkout, actions/setup-java)

Repository Structure

```
.github/workflows/ci.yml            # GitHub Actions CI workflow
src/main/java/com/example/
    DiscountEngine.java              # Discount logic (percentage + flat
     discounts)
    Calculator.java                  # Calculator utility
src/test/java/com/example/
    DiscountEngineTest.java          # JUnit 5 tests for DiscountEngine (17
     tests)
    CalculatorTest.java              # JUnit 5 tests for Calculator (19 tests)
checkstyle.xml                       # Checkstyle linter configuration
pom.xml                              # Maven project descriptor
mvnw                                 # Maven Wrapper script
.mvn/wrapper/
    maven-wrapper.properties         # Maven Wrapper configuration
ci-local.sh                          # Script to run CI checks locally
```

---

2. Discount Logic Engine — Business Rules

2.1 Inputs

Parameter	Type	Valid Values
Original Price	`double` (or `String` overload)	Any value >= 0
Discount Code	`String`	`"WELCOME10"`, `"SUMMER20"`, `null`, or empty string
User Type	`String`	`"PRO"` or `"GUEST"`

2.2 Discount Rules

Rule	Description
WELCOME10	10% off the original price
SUMMER20	20% off the original price
PRO user	Extra $5 flat discount (applied after percentage discount)
Floor	Final price is clamped to $0 minimum — never goes negative

2.3 Examples

Original Price	Discount Code	User Type	Calculation	Final Price
$100	SUMMER20	GUEST	$100 × 0.80	**$80.00**
$100	WELCOME10	GUEST	$100 × 0.90	**$90.00**
$100	SUMMER20	PRO	($100 × 0.80) − $5	**$75.00**
$100	WELCOME10	PRO	($100 × 0.90) − $5	**$85.00**
$5	(none)	PRO	$5 − $5	**$0.00** (not −$5)
$3	(none)	PRO	$3 − $5 → clamped	**$0.00** (not −$2)
$4	SUMMER20	PRO	($4 × 0.80) − $5 → clamped	**$0.00** (not −$1.80)

2.4 Error Handling

Invalid Input	Error Message
Negative price	`"Original price cannot be negative"`
Non-numeric string (e.g. `"abc"`)	`"Original price must be a valid number, got: abc"`
Unknown discount code (e.g. `"BOGUS"`)	`"Invalid discount code: BOGUS"`
Unknown user type (e.g. `"ADMIN"`)	`"Invalid user type: ADMIN"`
Null user type	`"Invalid user type: null"`

---

3. CI Pipeline Design

3.1 Workflow File

The CI pipeline is defined in `.github/workflows/ci.yml`:

```yaml
name: CI

on:
  pull_request:
    branches: [main]
  push:
    branches: [main]

jobs:
  checks:
    name: Lint & Test
    runs-on: ubuntu-latest
    steps:
      - Checkout code
      - Set up JDK 21 (Temurin, with Maven cache)
      - Run Checkstyle (linter):    ./mvnw -q checkstyle:check
      - Run unit tests:             ./mvnw -q test
```

3.2 When the Pipeline Runs

Trigger	Event	Branch
Pull request	PR opened, updated, or reopened	Target: `main`
Push	Direct push or merged PR	`main`

This ensures every PR is validated before merge, and every commit on `main` is validated again after merge.

3.3 Checks Executed

Check 1 — Linter (Checkstyle)

- Command: `./mvnw checkstyle:check`
- Configuration: `checkstyle.xml` in the project root
- Rules enforced:
  - **AvoidStarImport** — No wildcard imports (`import java.util.*`)
  - **NeedBraces** — Braces required for `if`, `else`, `for`, `while`, `do`
  - **WhitespaceAfter** — Whitespace after commas, semicolons, etc.
  - **WhitespaceAround** — Whitespace around operators and keywords
  - **OneStatementPerLine** — Only one statement per line
  - **MissingOverride** — `@Override` annotation required where applicable
  - **UnusedImports** — No unused import statements
- Failure behavior: Build fails with non-zero exit code; PR shows red X

Check 2 — Unit Tests (JUnit 5)

- Command: `./mvnw test`
- Framework: JUnit Jupiter 5.10.2 via Maven Surefire 3.2.5
- Total tests: **36** across two test classes

DiscountEngineTest.java — 17 tests:

Category	Tests	What They Verify
Happy Path	4	SUMMER20 → $80, WELCOME10 → $90, no code, empty code
PRO User	3	PRO + SUMMER20, PRO + WELCOME10, PRO + no code
Boundary Values	5	$5 PRO → $0, $3 PRO → $0, $0 price, $0 PRO, $4 SUMMER20 PRO → $0
Invalid Input	5	String "abc", null price, negative price, bad code, bad user type, null user type

CalculatorTest.java — 19 tests:

Category	Tests	What They Verify
Init	1	Smoke test of all operations
add()	4	Positive, negative, zero, mixed
subtract()	3	Basic, negative result, zero
multiply()	4	Basic, zero, one negative, two negatives
divide()	7	Basic, decimal, by one, negative, neg÷neg, zero throws, exception message

---

4. Branch Protection Rules

These settings are configured in **GitHub > Repository > Settings**:

4.1 Required Status Checks (Settings > Branches)

Setting	Value
Branch name pattern	`main`
Require status checks to pass before merging	**Enabled**
Required status check	**Lint & Test** (the GitHub Actions job name)
Require branches to be up to date before merging	**Enabled**

This ensures no PR can be merged unless both Checkstyle and all 36 unit tests pass.

4.2 Required PR Review (Settings > Branches)

Setting	Value
Require a pull request before merging	**Enabled**
Required number of approvals	**1**

| Dismiss stale pull request approvals when new commits are pushed |
 Recommended |

This ensures at least one team member reviews and approves every change.

4.3 Squash + Rebase Merge (Settings > General > Pull Requests)

Setting	Value
Allow merge commits	**Disabled**
Allow squash merging	**Enabled** (default)
Allow rebase merging	**Disabled**

This enforces that all PRs are merged as a single squashed commit, producing a
 clean, linear history on `main`.

---

5. End-to-End Workflow

1. Developer creates a **feature branch** from `main`.
2. Developer pushes code and opens a **pull request** targeting `main`.
3. GitHub Actions automatically triggers the **CI workflow**:
   - Step 1: Checkout code
   - Step 2: Set up JDK 21
   - Step 3: Run Checkstyle (`./mvnw checkstyle:check`)
   - Step 4: Run unit tests (`./mvnw test`)
4. PR page shows check status:
   - Green checkmark if all checks pass
   - Red X if any check fails (with link to logs)
5. If checks fail, the developer fixes the code and pushes again; CI re-runs
 automatically.
6. Once checks pass, a **reviewer approves** the PR (1 approval required).
7. The PR is merged using **Squash and merge** (only allowed merge method).
8. The push to `main` triggers CI again on the merged commit.

---

6. Ways This Process Can Fail

6.1 CI Check Failures

Failure	Cause	Impact	Resolution
Checkstyle violation	Code violates a linter rule (e.g. unused import,
missing braces)	PR merge blocked	Fix the style violation and push
Unit test failure	A test assertion fails or test throws unexpected
exception	PR merge blocked	Fix the code or update the test
Flaky tests	Intermittent failures due to timing, ordering, or external
factors	Erodes trust in CI; delays merges	Quarantine or fix flaky tests;
add retries if needed

```
| **Compilation error** | Code doesn't compile (syntax error, missing dependency) | Both checks fail | Fix compilation issues and push |

### 6.2 Infrastructure Failures

| Failure | Cause | Impact | Resolution |
|---|---|---|---|
| **GitHub Actions outage** | GitHub platform degradation | Workflows don't start or hang | Wait for recovery; check [githubstatus.com](https://githubstatus.com) |
| **Maven dependency unavailable** | Maven Central is down or artifact removed | Build fails at dependency resolution | Use cached dependencies; consider a mirror |
| **Runner resource limits** | Job exceeds time or memory limits | Workflow times out | Optimize build or split into parallel jobs |

### 6.3 Configuration Failures

| Failure | Cause | Impact | Resolution |
|---|---|---|---|
| **Status check name mismatch** | Job name in workflow doesn't match branch protection rule | PRs can merge without checks | Verify job name "Lint & Test" matches in both places |
| **Branch protection disabled** | Admin accidentally removes or weakens the rule | Unchecked code reaches `main` | Audit branch protection regularly; restrict admin bypass |
| **Wrong merge method allowed** | Merge commits or rebase re-enabled | Non-squash merges pollute history | Verify only "Allow squash merging" is checked in Settings > General |

### 6.4 Human / Process Failures

| Failure | Cause | Impact | Resolution |
|---|---|---|---|
| **Admin bypass** | Admin merges without waiting for checks or approval | Broken code on `main` | Enable "Do not allow bypassing the above settings" |
| **Self-approval** | PR author approves their own PR | No independent review | Use CODEOWNERS or restrict who can approve |
| **Post-merge CI failure** | Squashed commit differs subtly from what was tested | Broken `main` | CI runs on push to `main`; revert or hotfix immediately |

---

## 7. Notifications

### 7.1 Default Notifications (Built-in)

| What | Where | When |
|---|---|---|
| **PR status checks** | GitHub PR page (green check / red X) | Every CI run |
```

Email	GitHub account email	Workflow failure (configurable under GitHub > Settings > Notifications)
Actions tab	Repository > Actions	All workflow runs with full logs
Commit status badge	Next to each commit SHA	After CI completes

7.2 Notification Flow

- **PR author** sees check status directly on the PR page. If checks fail, they click the red X to view logs and identify the issue.
- **Reviewers** are notified via GitHub when they are requested for review and when CI status changes.
- **Repository watchers** receive email notifications for failed workflow runs (based on their notification preferences).

7.3 Recommended Additions

Integration	Purpose	How
Slack	Real-time alerts for CI failures	Add a Slack webhook step in the workflow (on failure)
Email digest	Daily summary of failed builds	Use a scheduled workflow or third-party tool
Status badge in README	At-a-glance health of `main`	Add `![CI](https://github.com/yachhapat/zip/actions/workflows/ci.yml/badge.svg)` to README
PagerDuty / Opsgenie	Escalation for repeated `main` failures	Integrate via webhook or GitHub App

---

8. Stakeholders

Stakeholder	Role in This Process	When to Involve
QA / Test Engineers	Own and extend automated checks; add coverage gates, integration tests, E2E tests	When adding new test types or quality gates
Developers	Write code, fix CI failures, consume CI feedback daily	Ongoing; involve in decisions about linter rules and test standards
DevOps / Platform / SRE	Own CI infrastructure (GitHub Actions runners, secrets, caching); integrate with CD pipelines	When scaling CI, adding deployment stages, or migrating CI tools
Engineering / Tech Leads	Define branch strategy, approval policies, code ownership (CODEOWNERS)	When changing merge rules, adding required reviewers, or evolving branching model
Security / Compliance	Add dependency scanning (e.g. Dependabot, Snyk), license checks, SAST	When introducing security gates or audit requirements
Product / Project Managers	Understand that "merge" = "checks passed + reviewed"; plan around CI constraints	When adding new required checks that may affect delivery timelines

---

9. Running Checks Locally

Developers can run the exact same checks as CI on their local machine:

```bash
./ci-local.sh
```

This script:
1. Detects and sets `JAVA_HOME` (supports Homebrew JDK on macOS)
2. Uses `./mvnw` (Maven Wrapper) so no global Maven install is needed
3. Runs `checkstyle:check` then `test` — same as the GitHub Actions workflow
4. Exits with non-zero status on first failure (mirrors CI behavior)

Individual commands:

```bash
./mvnw checkstyle:check    # Linter only
./mvnw test                # Unit tests only
./mvnw verify              # Full build lifecycle
```

---

10. Quick Reference

Item	Value
Repository	https://github.com/yachhapat/zip
CI tool	GitHub Actions
Workflow file	`.github/workflows/ci.yml`
Linter	Checkstyle (`checkstyle.xml`)
Unit tests	JUnit 5 — `DiscountEngineTest.java` (17 tests) + `CalculatorTest.java` (19 tests)
Branch	`main` (protection + push trigger)
Required checks	"Lint & Test" job must pass
Required approvals	1 PR review approval
Merge method	Squash and merge only
Local CI script	`./ci-local.sh`

Export this file to PDF for submission: Open in browser or VS Code, then Print > Save as PDF.