a1a # assignment

Assignment 1a – Introduction

School of Engineering and Technology, University of Washington Tacoma
TCSS 305 Programming Practicum, Winter 2026
Value: 4% of the course grade

Due Date

Sunday, 11 January 2026, 23:59:59

📄 Description

You will turn in a short Java program (as an IntelliJ project) to demonstrate that your programming environment setup was successful and to show that you are able to submit your work using Git and GitHub.

This assignment is the first step toward building a **UW Bookstore** shopping cart application that you will develop in the first 3-4 weeks of the quarter.

🎯 Learning Objectives

By completing this assignment, you will:

- Set up and configure your Java development environment (JDK 25, IntelliJ IDEA)
- Practice using Git and GitHub for version control and submission
- Implement a Java class from an interface specification
- Apply defensive coding techniques (input validation, exception handling)
- Work with `BigDecimal` for precise monetary calculations
- Use IntelliJ's code analysis tools to maintain code quality

---

✅ Before You Begin

Guide

Environment Setup

Ensure you have completed the environment setup:

- Installed JDK 25
- Installed IntelliJ IDEA (Ultimate)
- Created a GitHub account linked to your UW email
- Installed Checkstyle-IDEA plugin

📦 Project Setup

This project uses **GitHub Classroom** to distribute starter code and collect submissions.

Accept the Assignment

GitHub Classroom Assignment

Accept Assignment 1a

1. Click the GitHub Classroom assignment link provided by your instructor.
2. **First time only:** GitHub Classroom will ask you to select your name from the course roster. This links your UW identity to your GitHub username. Select carefully—this cannot be changed later.
3. Click **Accept this assignment**. GitHub creates a personal repository for you (e.g., `TCSS305-a1a-yourGitHubUsername`).
4. Wait for the repository to be created (this may take a moment).
5. Click the link to your new repository, then click the green **Code** button and copy the HTTPS URL.

Clone the Repository in IntelliJ

1. Open IntelliJ IDEA.
2. If a project is already open: **File** → **New** → **Project from Version Control** If on the Welcome screen: **Get from VCS**

3. Paste the repository URL you copied.
4. Choose a location on your computer for the project files.
5. Click **Clone**.
6. When prompted, click **Trust Project** to allow IntelliJ to load the project configuration.

Troubleshooting IntelliJ + GitHub Authentication

If IntelliJ prompts for credentials or shows authentication errors, try one of these solutions:

Option A: Personal Access Token (Recommended) >

Option B: GitHub Desktop >

Guide

Git Version Control — New to Git? Start here.

Guide

IDE Basics — First time using IntelliJ? Start here.

📁 Project Structure

```
TCSS305-a1a/
├── src/                          ← Source code
│   └── edu/uw/tcss/
│       ├── app/
│       │   └── StarterApplication.java
│       ├── model/
│       │   ├── Item.java             (interface - provided)
│       │   └── StoreItem.java        (implement this)
│       └── WRONG/                    (delete after verification)
│           ├── checkstyleRuleBreaker.java
│           └── InspectionTester.java
├── test/                         ← Unit tests
│   └── edu/uw/tcss/model/
│       └── StoreItemTest.java        (provided tests)
├── .idea/                        ← IntelliJ settings (do not edit)
├── README.md                     ← Links to assignment on course site
└── executive-summary.md          ← Your submission notes
```

Understanding the Layout

Source and Test Folders

IntelliJ separates production code (`src/`) from test code (`test/`). This is standard practice in professional Java development. Both folders mirror the same package structure.

Packages

Java uses packages to organize related classes. The package `edu.uw.tcss.model` corresponds to the folder path `edu/uw/tcss/model/`. All classes in this package share the declaration:

```
package edu.uw.tcss.model;
```

Guide

Java Packages — Learn more about how packages organize Java code.

Key Files

File	Location	Purpose
`StarterApplication.java`	`src/.../app/`	Edit with your academic background
`Item.java`	`src/.../model/`	Interface specification (do not modify)
`StoreItem.java`	`src/.../model/`	**Implement this class**
`checkstyleRuleBreaker.java`	`src/.../WRONG/`	Linting demo (delete after verification)
`InspectionTester.java`	`src/.../WRONG/`	Inspection demo (delete after verification)
`StoreItemTest.java`	`test/.../model/`	Unit tests (do not modify)
`README.md`	project root	Links to assignment on course site

File	Location	Purpose
`executive-summary.md`	project root	Document your submission

> **Caution**
>
> Do not modify `Item.java` or `StoreItemTest.java`. Your implementation must work with the provided interface and pass the provided tests.

✏️ Requirements

Requirement 1: Verify Linting Tools

> **Guide**
>
> Linters and Code Quality — Why linting matters and how to use these tools.

The `WRONG` package contains two files with intentional code quality violations. Use these files to verify that your linting tools are installed and configured correctly.

Verify Checkstyle

1. In the Project panel, navigate to `src/edu/uw/tcss/WRONG/`
2. Open `checkstyleRuleBreaker.java`
3. Observe the warnings in the editor (yellow highlights, underlines)
4. Open the **Checkstyle** tool window: **View** → **Tool Windows** → **Checkstyle**
5. Click **Check Current File** (or right-click in the editor → **Check Current File**)
6. Confirm that Checkstyle reports many violations

Verify IntelliJ Inspections

1. Open `InspectionTester.java`
2. Observe the warnings in the editor
3. Run **Analyze** → **Inspect Code...**
4. Select **Current File** and click **OK**
5. Confirm that IntelliJ reports inspection warnings

Delete the WRONG Package

Once you have verified both tools are working:

1. Delete `checkstyleRuleBreaker.java`
2. Delete `InspectionTester.java`
3. Delete the `WRONG` folder

> **Warning**
>
> Do not fix the violations in these files—just delete them. They exist only to verify your setup.

Requirement 2: StarterApplication.java

> **Guide**
>
> Logging — How to use Java's logging framework.

Remove all existing log messages and console output from the class.

> **Example of log messages to remove**

```
LOGGER.info(() -> "Should you need String concatenation,  "
     + "use a lambda for lazy evaluation. What's that you
ask?");
```

Using the log level info, add a message describing your academic career up to this point:

- What institution did you take 142/143 (programming 1 and 2)?
- When did you complete them?
- Do you have a college degree before starting this program?
- Anything else that you want to share?

Please use multiple calls to the logger object to complete this requirement (i.e., don't smash all of this into one method call).

Requirement 3: StarterApplication.java

Guide

- Checkstyle Rules Reference — Look up specific Checkstyle violations
- IntelliJ Inspections Reference — Look up specific IntelliJ warnings

Edit the Java class such that **all** warnings and errors are corrected. Right-click on the class name in the Project browser window, **Analyze → Inspect Code...**

Carefully read each warning/error and correct it. If you are unsure what the warning is for or how to fix it, just ask! I will spend some time in lecture during the first weeks discussing different warnings, why they are there, and how to fix them.

Requirement 4: StoreItem.java

Guide

Interface Contracts — Understanding what it means to implement an interface.

Complete the implementation of the `StoreItem` class based on the provided `Item` interface and the API descriptions below.

`Item` Interface

The `Item` interface defines the contract for all items in the store.

```
String getName()
```

Returns the name for this Item.

```
BigDecimal getPrice()
```

Returns the unit price for this Item.

```
BigDecimal calculateTotal(int quantity)
```

Calculates the total price for the given quantity.

Throws: `IllegalArgumentException` if quantity is negative.

```
String getFormattedDescription()
```

Returns a formatted description suitable for display in a GUI (e.g., `"Computer Science Pen, $2.00"`).

`StoreItem` Class

The `StoreItem` class represents a simple item with standard pricing. It implements the `Item` interface.

```
StoreItem(String name, BigDecimal price)
```

Constructor that takes a name and a price.

```
String getName()
```

Returns the name for this StoreItem.

```
BigDecimal getPrice()
```

Returns the unit price for this StoreItem.

```
BigDecimal calculateTotal(int quantity)
```

Returns `price × quantity`.

Throws: `IllegalArgumentException` if quantity is negative.

```
String getFormattedDescription()
```

Returns the item formatted as `name, $price` — for example, an item named "Computer Science Pen" with price 2.00 returns `Computer Science Pen, $2.00`.

```
String toString()
```

Returns a debug-friendly representation such as `StoreItem[name='...', price=...]`.

Note: the exact format of `toString` is not tested; your output does not need to match this document.

Code Defensively

Constructors should test for invalid values and reject bad input immediately.

Throw `IllegalArgumentException` for:

- price passed to your class is < 0
- name passed to your class is empty

Throw `NullPointerException` for:

- name passed to your class is null
- price passed to your class is null

> **Tip**
>
> Use `Objects.requireNonNull()` to check for null values. This method throws `NullPointerException` automatically if the argument is null.

Working with BigDecimal

All monetary values in this project use `BigDecimal` instead of `double` to avoid floating-point precision errors.

> **Caution**
>
> Never convert `BigDecimal` to `double` or `float` when performing calculations.

Key `BigDecimal` operations:

Operation	Method
Multiply	`price.multiply(BigDecimal.valueOf(quantity))`
Compare	`price.compareTo(BigDecimal.ZERO) < 0`
Format	`NumberFormat.getCurrencyInstance(Locale.US)`

> **External Resources**
>
> - BigDecimal API documentation
> - NumberFormat API documentation

Unit Test Cases

> **Guide**
>
> Introduction to Unit Testing — What unit tests are and why they matter.

This project enables test driven development. The QA team has provided a set of unit test cases based on the provided API description. Ensure that all test cases pass before submission. If a test case fails, carefully read the reason for the failure and correct your code appropriately. Note, all test cases fail upon initial Git clone. You must implement individual methods in StoreItem for the tests to pass.

To run the test class:

1. Open `test > java > edu.uw.tcss.model > StoreItemTest.java`
2. Either click on the play symbol next to the class name or the play button in the top right portion of the IDE UI. Ensure that Current File or StoreItemTest is selected.

> **Warning**
>
> All tests fail when you first clone the project. This is expected! The tests will pass as you implement each method in `StoreItem`.

> **Tip**
>
> Run tests frequently as you implement each method. This helps you catch issues early and confirms your implementation matches the specification.

Requirement 5: StoreItem.java

Edit the Java class such that **all** warnings and errors are corrected. Right-click on the class name in the Project browser window, **Analyze → Inspect Code...**

Fixing the warnings and errors will cause you to change your code. Double check that these changes did not introduce defects by re-running the provided test cases.

Requirement 6: Executive Summary

Your project includes a file called `executive-summary.md`. This file documents your project and submission details.

The `.md` file extension stands for Markdown. Markdown is a markup language used to format plain text. You may already know a markup language — the M in HTML and XML stands for markup. (Note: markup languages are NOT programming languages but instead are tools for formatting text.)

> **External Resource**
>
> Markdown Cheat Sheet

Edit the `executive-summary.md` file to personalize it for your submission. Carefully read all the text found inside of the enclosing square braces `[]`. Remove all this text (and square braces), replacing it with your own specific details about the project.

📚 Guide Reference

Guide	Description
Environment Setup	Installing JDK, IntelliJ, and configuring your development environment
IDE Basics	Getting started with IntelliJ IDEA
Java Packages	How packages organize Java code
Linters and Code Quality	Why linting matters and how to use Checkstyle and IntelliJ Inspections
Checkstyle Rules Reference	Look up specific Checkstyle violations and how to fix them
IntelliJ Inspections Reference	Look up specific IntelliJ warnings and how to fix them
Logging	How to use Java's logging framework
Interface Contracts	Understanding what it means to implement an interface
Introduction to Unit Testing	What unit tests are and why they matter

---

🚀 Submission and Grading

Important

Despite the moderately low point value (4% of the course grade), completing this assignment successfully is *critical*. You will use these tools and techniques for all future assignments.

Submit your work by committing and pushing to your GitHub repository. See the submission steps below.

Please see the rubric in Canvas for a breakdown of the grade for this assignment.

Submitting Your Work

When you are ready to submit, commit and push your changes to GitHub:

1. In IntelliJ, click **Git** → **Commit**.
2. Select all changed files and write a descriptive commit message (e.g., "Completed Assignment 1a").
3. Click **Commit and Push**.
4. In the Push dialog, click **Push**.
5. **Verify your submission:** Visit your repository on GitHub.com and confirm your latest changes appear.

Submitting with GitHub Desktop (Option B users only) ›

Your professor has automatic access to your repository through GitHub Classroom. Make sure your final code is pushed before the deadline.

Tip

You can commit and push multiple times before the deadline. Only your final push will be graded.