CareerSwami · Java & Enterprise Software

FREE TWO-WEEK INTERNSHIP

Project 1 — Build a reservation service that cannot double-book a projector

Two projectors, three tripods, one microphone. People reserve them for a range of days. The service has one job it must never get wrong: on any day in that range, reservations must never exceed what exists.

2 weeks	1 project	6–8 hrs
START TO SUBMIT	ONE SUBMISSION	PER WEEK

WHAT IS IN THIS BRIEF

1. The situation — the problem you are being handed
2. The four rules the service has to enforce — the cases your work must survive
3. What you are given — your data and starting files
4. What finished looks like — the two things you hand in
5. How the two weeks run — week by week
6. Before you start — rules, tools and your starter kit
7. How the work is judged — the marking scheme
8. How to submit — one link, pasted into your dashboard

Read 1 to 4 before you do anything. They take five minutes and they are what the marking scheme is built on. Sections 5 to 8 are the how — come back to them once you know what you are solving.

1 The situation

Treat this as a real client brief. The equipment store is fictional. The problem — overlapping reservations against finite stock — is the one that appears in rooms, vehicles, seats, licences and almost every enterprise system you will touch.

A media store lends three kinds of equipment: two projectors, three tripods and one microphone. A reservation names an item, a quantity, a start day and an end day, and the range is inclusive — booking day 3 to day 5 means days 3, 4 and 5 are all consumed.

Three things must work: reserve, cancel, and check availability. Three things must be refused: a quantity of zero or less, an end day before the start day, and any reservation that would push demand above stock on even one day of its range.

That last one is where this project is won or lost. Checking only the start day passes most hand-written tests and is wrong. Two reservations can overlap partially — days 3 to 5 and days 5 to 7 collide on exactly one day, and that day is the one your check has to find.

WHAT YOU ARE BEING ASKED TO DO

Build a local reservation service with reserve, cancel and availability, where overlap is evaluated across every day of the requested range and every invalid request is refused with a clear reason.

2 The four rules the service has to enforce

Three of these are simple validation. The fourth is the project. Your usage examples should demonstrate all four, including the refusals.

Partial overlap on a single day

Days 3–5 and days 5–7 conflict on day 5.

Where implementations fail — Checking whether the new start falls inside an existing range misses the case where the new range starts before an existing one and ends inside it. Two ranges overlap when newStart ≤ existingEnd and newEnd ≥ existingStart — write that down once and use it everywhere.

Quantity against stock, not against count

Two projectors exist; two separate single-unit reservations fill them.

Where implementations fail — Rejecting only when an item is already reserved makes the second projector unusable. Rejecting only on reservation count ignores quantity entirely. You must sum quantities per day and compare against stock.

Nonsense input

Quantity ≤ 0 and end < start are both refused.

Where implementations fail — A quantity of zero stores a reservation that consumes nothing and clutters every availability calculation. A reversed range makes every overlap comparison behave unpredictably. Reject both at the boundary, before anything is stored.

A cancelled reservation

Must stop consuming stock immediately.

Where implementations fail — Delete the row and you lose the audit trail; set a status and forget to filter on it everywhere and the stock never comes back. Choose one approach and apply it in every query that counts.

The microphone has a quantity of one, which makes it the cheapest test in the kit: reserve it for days 2 to 6, then try days 6 to 9. If that second request succeeds, your overlap check is wrong.

3 What you are given

The inventory and the rules are given. The architecture, the persistence choice and where you put the overlap rule are yours.

The inventory and the reservation record

`P1 Projector · qty 2` `T1 Tripod · qty 3` `M1 Microphone · qty 1`

`reservation: id, item_id, quantity, start_day, end_day, status` `day ranges are inclusive`

The overlap rule, worked, so it is unambiguous

`Stock`	M1 Microphone, quantity 1
`Existing`	R1 · M1 · qty 1 · days 2–6
`Request`	R2 · M1 · qty 1 · days 6–9
`Overlap test`	newStart 6 ≤ existingEnd 6, and newEnd 9 ≥ existingStart 2 → overlap
`Day of conflict`	Day 6 — demand would be 2 against stock of 1
`Expected`	Refused: insufficient stock on day 6
`Also refuse`	qty ≤ 0, and end_day < start_day

Three operations, three refusals

`Reserve` `Cancel` `Availability`

All items and reservations are fictional. No enterprise infrastructure, message broker or public deployment is required — a single runnable project is the deliverable. Keep secrets and credentials out of source control. Where the brief leaves something open, decide, document it in the README, and be consistent.

4 What finished looks like

At the end of two weeks someone should be able to clone your repository, follow the README, and make a reservation within ten minutes.

- **A domain diagram, use cases and a setup README** — items and reservations with their relationship; each operation written as a use case with its preconditions, its successful outcome and every way it can fail; and exact build and run commands with pinned versions.
- **The implementation, a seed, and worked usage examples** — the service with the overlap rule enforced in one place rather than scattered, a seed creating the three items with their quantities, and captured examples of every operation including each refusal.

Write the overlap condition as a single, named, well-commented method. If it appears in three places with slightly different comparison operators, one of them is wrong and you will not find out which.

5 How the two weeks run

Work through it in two parts — but you submit **once**, at the end, not once per week.

PART 1 · WEEK 1

Write the rules down before you write the code

Model the domain, then write each operation as a use case with its preconditions and failure modes. Get the project building from a README someone else could follow.

Produce

- A domain diagram: items, reservations, quantities, and the day range
- Use cases for reserve, cancel and availability, with every failure listed
- A setup README with exact build and run commands and pinned versions

> Your reserve use case must state the overlap condition explicitly in words. If you cannot write the rule in one sentence, you are not ready to implement it.

PART 2 · WEEK 2

Implement it and prove the refusals

Build the service with the overlap rule in a single place, seed the three items, and capture examples of every operation — especially the ones that are correctly refused.

Produce

- The implementation, with the overlap check in one named method
- A seed creating 2 projectors, 3 tripods and 1 microphone
- Usage examples for every operation, including all three refusal cases

> Demonstrate the partial-overlap case explicitly — reserve days 2–6, then attempt days 6–9 on the single microphone, and show it refused. Every refusal needs a clear reason, not a stack trace.

6 Before you start

- Day ranges are inclusive at both ends. Days 3 to 5 consumes three days.
- Overlap is newStart ≤ existingEnd AND newEnd ≥ existingStart. Write it once, in one method, with a comment.
- Compare summed quantity per day against stock, not reservation count. Two projectors means two single-unit reservations fit.

- Decide whether cancellation deletes or sets a status, then make every counting query honour the decision.
- Refusals need readable messages and the right failure path. An uncaught exception surfacing as a stack trace is a defect.

Your tools

Java 17 or later with Maven or Gradle. Spring Boot is welcome but plain Java with an embedded H2 or SQLite database is entirely acceptable and often clearer. No application server, no cloud account and no public hosting is required, and no ID document is ever required.

CS_java-enterprise-software_Starter_Kit.zip — items.csv, a use-case template, a domain diagram template, a README skeleton, an evidence log, and a file index.
https://careerswami.com/wp-content/uploads/careerswami-projects/CS_java-enterprise-software_Starter_Kit.zip

7 How the work is judged

WHAT IS ASSESSED	WEIGHT
Problem and requirements — did you understand what you were solving	20%
The core artefact — the thing you actually made	40%
Validation and evidence — how you know it works	25%
Clarity and reproducibility — can someone else open and follow it	15%

Strong	Complete, correct, evidence-backed and easy to inspect.
Developing	A useful partial result, with the gaps explained.
Needs revision	Core deliverable missing, claims unverified, or the work cannot be opened.

Credit reflects demonstrated work, not visual polish. A small, honest result beats a large, unsupported one.

What counts as evidence

expected result → actual result → the captured output, log line or test run → conclusion. Then say what the evidence does not establish.

Why this one is worth doing. Enterprise work is mostly rules like this one — boring to describe, expensive to get wrong, and almost always implemented slightly incorrectly the first time. An interviewer who sees that you wrote the overlap condition as a sentence before you wrote it as code, and that you tested the single-day partial overlap, learns more about you than any framework on your CV.

8 How to submit

One link, pasted into your dashboard

1. Put your work somewhere with a link — Google Drive, Figma, GitHub, Notion, Loom. Whatever fits the work.
2. **Set it so anyone with the link can view.** A locked link is the single most common reason a submission comes back.
3. Include a short README or cover page saying what to open first, and where each deliverable lives.
4. Open your dashboard, paste the link into the submit box, and add a line or two on anything you want the reviewer to know.

Name files `CS_java-enterprise-software_YourName_v1`. Keep your own copy. A screenshot on its own is not enough for anything interactive. A video is optional — annotated screenshots or narrated slides are fine.

Submitted and heard nothing, or the link came back? Email **support@careerswami.com** with the email address you enrolled with.

Work honestly. Use fictional data and your own work. Credit anything you reused, and disclose AI assistance — including what you checked yourself. Never invent interviews, users, tests or results. Nothing here requires you to spend money, contact real organisations, or deploy anything publicly. Never upload identity documents, real customer records or credentials.

Stuck, confused, or something is broken? Write to us.

`support@careerswami.com`

Or WhatsApp **+91 7383138418**. We answer on working days. Use it for anything at all — a part of this brief that does not make sense, a starter file that will not open, a dashboard link that has stopped working, a deadline you need moved, or a submission you are not sure about.

To get a useful answer fast, spend twenty minutes narrowing the problem down first, then tell us four things: what you expected, what actually happened, what you already tried, and your one specific question. Include your track name and the email you enrolled with.

Asking is not penalised. Submitting partial work with a clearly stated blocker always scores better than a finished answer you copied, and better than a silent no-show.

Optional stretch

Optional, and only once the main work is done — cap it at an hour. Add an availability calendar showing free units per day, or allow a reservation to be shortened rather than cancelled. Label it clearly as a stretch.