手元ガイド 03 / 課題B編

初見の Java プロジェクトを Issue 駆動で直す

演習B B-0 から B-6 の手順書

課題Bは、あなたが書いていない Spring Boot のプロジェクトです。不具合が仕込まれています。ここでやることは、いきなり直させることではありません。動かして現状を掴み、Plan モードで調べさせ、直す対象を Issue という文書に確定させてから、はじめて修正を依頼します。直ったことはブラウザとテストの両方で確かめます。 課題Aで配置したハーネスに、公式の Skill を3つ足します。足す作業はフォルダのコピーと貼り付けだけです。AI の振る舞いを変える仕組みが、置き場所の決まったテキストファイルでできていることを、足す動作そのもので確認します。

時間 演習B 全体で [70min]　　**題材** 備品管理システム（Spring Boot 3.4 / Java 17 / H2）

開くフォルダ handson（最初から最後まで固定）

環境 VSCode × Claude Code（Amazon Bedrock 経由 / Sonnet 4.6）

Claude Code　Spring Boot　Thymeleaf　H2 Database　Maven Wrapper

SECTION 01 · ROADMAP

演習Bの全体像と到達基準

演習Bは7つのステップでできています。前半の B-0 から B-4 までが到達基準の範囲です。ここまでを時間内に確実に終わらせてください。後半の B-5 と B-6 は、今日やったことを次に持ち帰るための2本で、内容は独立しています。B-4 が長引いた場合でも、B-5 と B-6 は短く回せます。

演習Bの到達基準

1. **バグを2件修正した。**3件目以降は発展課題です。時間内に2件を確実に終わらせることを優先してください
2. **ブラウザで、修正前におかしかった画面が正しく表示される。**あわせて、元から正しかった画面が壊れていないことも見ます
3. `./mvnw test` が `BUILD SUCCESS` **で終わる。**自分で追加したテストが2件以上あり、どちらのバグに対応するかがコメントでわかる状態にします。配布時から入っている受け入れテスト `EquipmentAcceptanceTest` は配布時点で失敗する作りで、これが緑になることが到達の判定になります

この3つがそろえば到達です。B-5 の Skill 作成と B-6 の脅威モデリングは、到達基準そのものには含みませんが、演習Bの最後まで進めてください。B-6 は第2回研修への入り口になります。

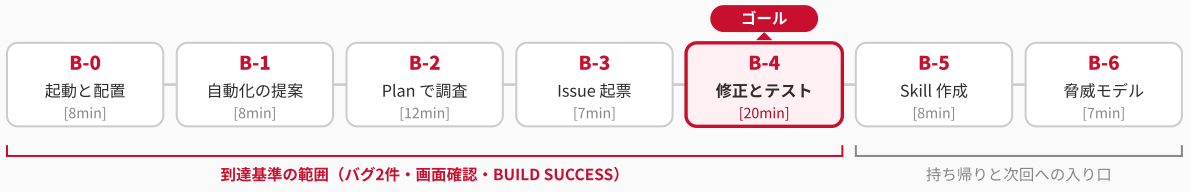

図15 演習Bのロードマップ。合計 [70min]。いま自分がどこにいるかを、各ステップの冒頭で確認してください

演習Aとの違い

演習Aは、何も無いところに自分でアプリを作りました。正解は自分で決められました。演習Bは逆です。すでに動いているコードがあり、正しい姿は `CLAUDE.md` の業務ルールに文章で書かれています。**作るときと直すときでは、AI の使い方が変わります。**この違いが、演習Bの中心にあります。

観点	演習A（作る）	演習B（直す）
出発点	空のフォルダ。正解は自分で決める	動いているコード。正しい姿は業務ルールに書かれている
最初にやること	指示を書いて作らせる	動かして画面を見る。コードを読むのはその後
主に使うモード	Agent モード	Plan モードで調べてから、Agent モードで直す
直す対象の決め方	会話の中で決まっていく	Issue ファイルに文章で確定させてから着手する
できたことの確かめ方	画面が動くこと、`/selfcheck` の報告	画面で確認し、さらに `./mvnw test` を通す
失敗したときの戻し方	作り直せばよい	元の動いていた挙動を壊していないかを毎回見る

始める前の確認

- [ ] VSCode で開いているフォルダが `handson` になっている（`kadaiB` を単体で開くと設定が読み込まれません）
- [ ] 演習 A-3 で配置した `handson/CLAUDE.md` と `handson/.claude/` が存在する
- [ ] Claude Code のパネルが開き、応答の最後に実行サマリーの4項目が出ている
- [ ] ターミナルで `java -version` を実行すると 17 と表示される
- [ ] 事前セットアップで `kadaiB` の `./mvnw -q -DskipTests package` を1回実行済み（当日の起動待ちが数分から数十秒に縮みます）

まだ揃っていない場合

ハーネスが未配置のまま演習Bに入ると、業務ルールが読み込まれないため、B-2 の2件目のバグが見つかりません。**先に演習 A-3 の配置だけ済ませてください。**手順は課題A編の A-3 にあります。3分で終わります。

SOURCES

- Claude Code overview（Anthropic 公式ドキュメント）
- Claude Code on Amazon Bedrock（接続方式の公式解説）
- Spring Boot Reference Documentation（Spring 公式）
- Spring Boot 3.4 Release Notes（本研修の題材が使うバージョン）
- Apache Maven Wrapper（`./mvnw` の公式説明）

SECTION 02 · STRUCTURE

題材の構造とバグが潜む場所

初見のプロジェクトで最初に困るのは、どこを見ればよいかがわからないことです。課題Bは
レイヤードアーキテクチャという、役割ごとに置き場所を分ける作り方で書かれています。役割の分かれ方がわかると、症状から見る場所を絞り込めます。**絞り込めると、AI に読ませる範囲も絞れます。**読ませる量が減れば、返答は速く、正確になります。

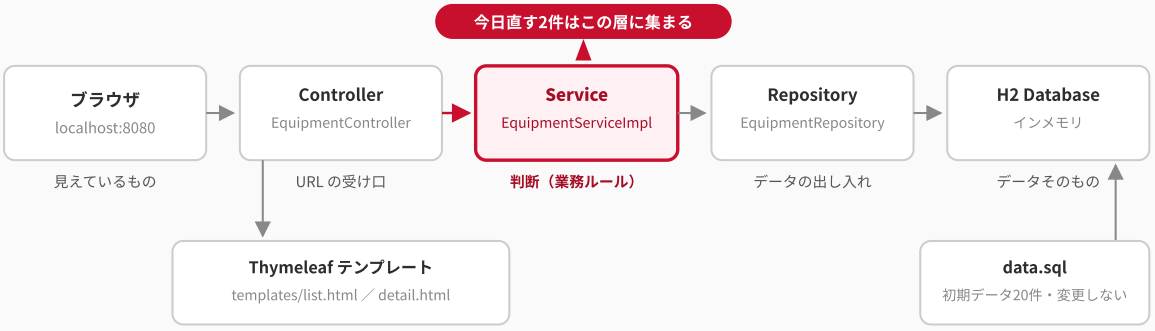

図16 課題Bの層構造。今日直す2件はどちらも Service 層（ `EquipmentServiceImpl.java` ）にあります

> **なぜ判断が Service に集まるのか**
>
> 「在庫が5個以下かどうか」「備考が空のときにどう表示するか」は、どちらも **決めごとに従った判断**です。画面を組み立てる仕事でも、データを取り出す仕事でもありません。役割で置き場所を分けているので、判断は Service に置かれます。この分け方は課題B固有のものではなく、Java の業務システムで広く使われている形です。

業務ルール5項目が仕様の正本

課題Bの「正しい姿」は、 `handson/CLAUDE.md` の `## 業務ルール（課題B）` と `kadaiB/README.md` に、同じ5項目として書かれています。実装がこの5項目と食い違っていれば不具合です。**この5項目が無いと、エラーの出ないバグは検出できません。**比べる相手がないものは、人にも AI にも探せないからです。

#	業務ルール	症状が出る場所
1	在庫が5個以下の備品は「要補充」として扱う（5個ちょうども含む）	一覧・詳細の表示
2	貸出は在庫の範囲内でのみ可能（在庫を0未満にしてはいけない）	詳細画面の貸出フォーム
3	日付の画面表示は `yyyy/MM/dd` 形式に統一する	一覧と詳細の購入日
4	存在しない備品IDへのアクセスは、エラー画面ではなく「見つかりません」とわかる表示にする	URL を直接書き換えたとき
5	備考が未設定の備品も、詳細画面が正常に表示される	詳細画面

5項目のうち、B-2 で見つけるのは2件です。**残りをすべて探す必要はありません。**3件目以降は B-4 の発展課題として扱います。時間内に2件を確実に直しきることを優先してください。

エラーが出ないバグの見つけ方

詳細画面が真っ白なエラーになるバグは、画面を見れば気づけます。難しいのはもう一方です。画面はきれいに表示されていて、エラーも出ていないのに、業務ルールに照らすと間違っている。この種類のバグは、**3つのものを突き合わせたときにだけ**姿を現します。

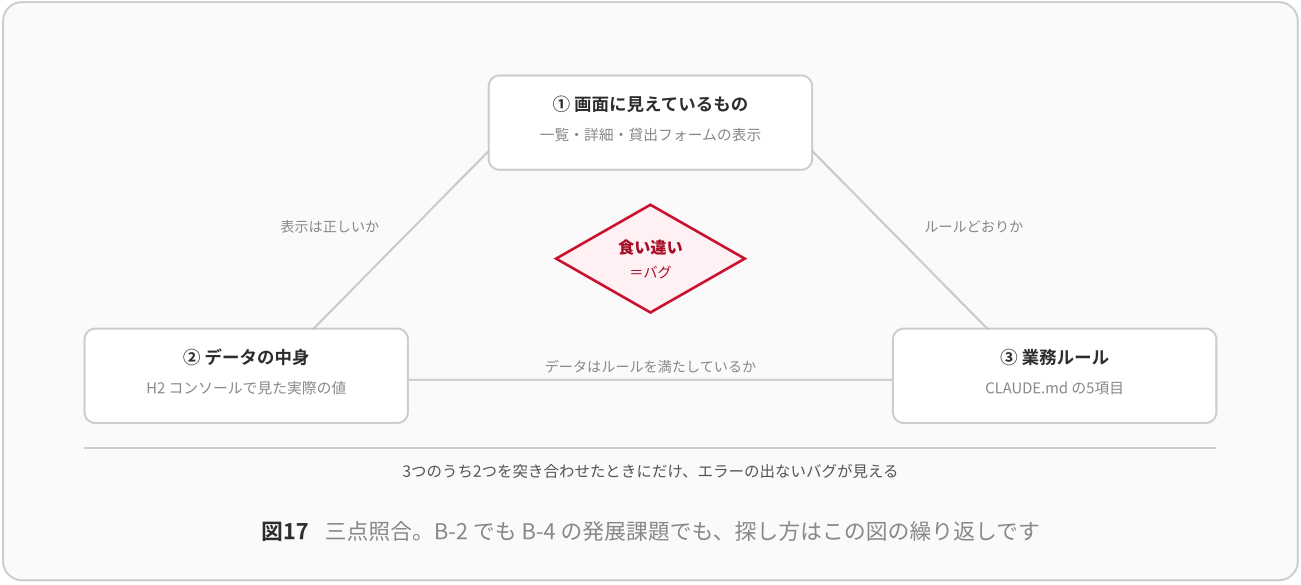

図17　三点照合。B-2 でも B-4 の発展課題でも、探し方はこの図の繰り返しです

触るファイルと触らないファイル

```
handson/
├── CLAUDE.md                    ← B-0 で末尾に追記
├── .claude/
│   ├── commands/
│   │   ├── selfcheck.md          演習Aで配置済み
│   │   └── regression-check.md   ← B-0 で追加
│   └── skills/                   ← B-0 で3種 + _lib を追加
├── memo.md                       ← 各ステップで追記
└── kadaiB/
    ├── README.md                 業務ルール5項目・到達基準
    ├── issues/
    │   ├── ISSUE_TEMPLATE.md     5見出しの雛形。増やしも減らしもしない
    │   ├── ISSUE_001_詳細画面がエラーになる備品がある.md    記入例
    │   └── ISSUE_002_〈自分で付ける名前〉.md                ← B-3 で作成
    ├── src/main/java/com/example/equipment/
    │   ├── controller/EquipmentController.java
    │   ├── service/EquipmentService.java
    │   ├── service/EquipmentServiceImpl.java    ← B-4 で修正
    │   ├── repository/EquipmentRepository.java
    │   └── model/Equipment.java
    ├── src/main/resources/
    │   ├── templates/list.html ／ detail.html
    │   └── data.sql              書き換えません
    ├── src/test/java/com/example/equipment/     ← B-4 でテスト追加
    ├── THREAT_MODEL.md           ← B-6 で生成
    └── mvnw ／ mvnw.cmd          Maven Wrapper
```

書き換えないファイル

`kadaiB/src/main/resources/data.sql` は変更しません。**データを変えて症状を消すのは修正ではありません。** たとえば備考が空の備品にデータを入れれば画面のエラーは消えますが、コードは壊れたままです。次に空のデータが来たら同じことが起きます。

SOURCES

- Spring Boot: Servlet Web Applications（Controller の役割）
- Thymeleaf Documentation（テンプレートの公式資料）
- H2 Database Engine（インメモリDBとコンソール）
- java.util.Optional（Java 17 API 仕様）
- Spring Data JPA Reference（Repository の生成のされ方）

SECTION 03 · SKILLS

今日足す公式 Skill 3種

Skill は、特定の作業のやり方をまとめたフォルダです。中身は日本語または英語の文章で、プログラムではありません。`.claude/skills/〈名前〉/SKILL.md` という置き場所に従って置くだけで、Claude Code が読み込みます。演習Bでは、Anthropic が公開している3種を配布ZIPから配置して使います。**インストール作業も設定画面の操作もありません。フォルダをコピーするだけです。**

Skill 名	本研修での役割	出典	ライセンス	使うステップ
`claude-automation-recommender`	初見のプロジェクトに、どんな自動化を仕込むと効くかを提案させる。読み取りのみで、ファイルを一切書き換えない	anthropics/claude-plugins-official	Apache-2.0	B-1
`skill-creator`	自分のレビュー観点や作業手順を Skill として書き出す。次のセッションで発火するかを確かめる	anthropics/skills	Apache-2.0	B-5
`threat-model`	修正後のコードに対して脅威モデルを作らせ、バグを直すことと危険を減らすことの違いを実物で見る	anthropics/defending-code-reference-harness	Apache-2.0	B-6

配布したもの、していないもの

3種とも、公開されているものをそのまま配っているわけではありません。研修環境で動く範囲に絞ってあります。**何を外したかを知っておくと、自社に持ち帰るときの判断材料になります。**

SKILL-CREATOR

軽量版を配っています

公開されているものには、作った Skill の品質を自動評価する仕組みが付いた高機能版があります。そちらは Python と複数のサブエージェントを前提としていて、20名が同時に動かすと待ち時間が長くなります。今日は対話で3つ質問して `SKILL.md` を書き出す軽量版だけを配っています。

THREAT-MODEL

読み取り専用の部分だけ

元の実装には、対象コードを隔離環境で実際に動かして挙動を確かめる仕組みが含まれます。Docker と隔離実行環境が必要なため配っていません。配っているのは `.claude/skills/threat-model/` と `.claude/skills/_lib/checkpoint.py` だけです。`_lib` をコピーし忘れると、途中の記録を保存する段階で止まります。

共通

取得はZIPから。Gitは使いません

本来はコマンド一つで公式から取得できますが、社内ネットワークの経路によっては外部に繋がりません。今日は配布ZIPの `_harness_kit/step2_kadaiB/` に固めたものをコピーします。研修中に Git と GitHub は使いません。

コピー元とコピー先

```
_harness_kit/step2_kadaiB/          コピー元（合図があるまで開きません）
├── CLAUDE_追記.md                    CLAUDE.md の末尾に貼り付ける文章
└── .claude/
    ├── commands/
    │   └── regression-check.md       スラッシュコマンド1本
    └── skills/
        ├── claude-automation-recommender/
        │   ├── SKILL.md
        │   └── references/  mcp.md ／ skills.md ／ hooks.md ／ subagents.md ／ plugins.md
        ├── skill-creator/
        │   ├── SKILL.md
        │   ├── references/  skill-writing.md ／ description-guide.md
        │   └── scripts/init_skill.py
        ├── threat-model/
        │   ├── SKILL.md ／ bootstrap.md ／ interview.md ／ schema.md ／ README.md
        └── _lib/
            └── checkpoint.py         途中経過の記録に使う

                        ↓ B-0 でコピー

handson/.claude/skills/             コピー先（ここに置かないと読み込まれません）
```

Skill が読み込まれる条件は2つだけ

1つ目は、 `.claude/skills/〈フォルダ名〉/SKILL.md` という形になっていること。フォルダを1階層挟むことと、ファイル名が大文字の `SKILL.md` であることの2点をよく間違えます。2つ目は、パネルを開き直すこと。Skill の一覧はセッションの開始時に読み込まれます。**配置してから開き直すまでが1セットです。**

SOURCES

- Claude Code: Agent Skills（公式ドキュメント / 置き場所と発火の仕組み）
- Anthropic Engineering: Equipping agents for the real world with Agent Skills
- anthropics/skills（skill-creator の出典 / Apache-2.0）
- anthropics/claude-plugins-official（claude-automation-recommender の出典 / Apache-2.0）
- anthropics/defending-code-reference-harness（threat-model の出典 / Apache-2.0）
- Claude Code: Slash commands（ `/regression-check` の仕組み）

SECTION 04 · STEP B-0

課題Bを起動して現状を掴む

B-0 動かしてから読む。Skill をあとから足す

使うもの	エクスプローラ（Finder）／VSCode のターミナル／ブラウザ
前提	演習 A-3 でハーネスを配置し、`handson/CLAUDE.md` と `handson/.claude/` が存在すること
手順書	`exercises/exB0_start_kadaiB.md`
次	B-1 自動化の提案を出させる

目的

課題Bは、あなたが書いていない Java のプロジェクトです。ここでやることは2つあります。

1つ目は、**コードを読む前に、まず動かす**ことです。動いている画面を先に見ておくと、あとから「ここが変だ」と気づけます。画面を見ないままコードだけを読むと、何が正しい状態なのかがわからないまま読むことになり、時間がかかります。初見のコードに対して人が最初にやるべきなのは、読解ではなく観察です。

2つ目は、**Skill をあとから足す**ことです。演習Aで配置したハーネスに、公式の Skill を3つ追加します。追加といっても、やることはフォルダのコピーと、テキストの貼り付けだけです。AI の設定があとから足せる部品でできていることを、足す動作そのもので確認します。

操作 1 Skill 3種を配置する

`handson/_harness_kit/step2_kadaiB/.claude/skills/` の中にある4つのフォルダを、`handson/.claude/skills/` の下にコピーします。エクスプローラでドラッグしても、VSCode のエクスプローラでコピーと貼り付けをしてもかまいません。ターミナルで実行する場合は、`handson` フォルダにいることを確認してから次を打ちます。

```
# Mac（handson フォルダで実行）
cp -R _harness_kit/step2_kadaiB/.claude/skills/* .claude/skills/
```

```
# Windows PowerShell（handson フォルダで実行）
Copy-Item -Recurse -Force _harness_kit\step2_kadaiB\.claude\skills\* .claude\skills\
```

操作 2 コマンドを1本配置する

同じく `_harness_kit/step2_kadaiB/.claude/commands/regression-check.md` を、`handson/.claude/commands/` にコピーします。演習Aで配置した `selfcheck.md` の隣に並びます。`/regression-check` は B-4 の発展課題で使います。

操作 3 CLAUDE.md に追記する

`_harness_kit/step2_kadaiB/CLAUDE_追記.md` を開き、中身をすべて選択してコピーします（`Ctrl`+`A` のあと `Ctrl`+`C`。Mac は `Cmd`+`A` と `Cmd`+`C`）。`handson/CLAUDE.md` を開き、いちばん下にカーソルを置いて貼り付け、保存します（`Ctrl`+`S`。Mac は `Cmd`+`S`）。

既にある内容は消さないでください。末尾に足すだけです。追記される内容は、課題Bの進め方、Issue の書き方、修正の決まり、テストの決まり、追加した Skill とコマンドの説明です。

操作 4 パネルを開き直す

`CLAUDE.md` と Skill の一覧は、セッションの開始時に読み込まれます。追記した内容を反映させるため、Claude Code のパネルをいったん閉じて、もう一度開いてください。

操作 5 アプリを起動する

VSCode のターミナルを開きます（`Ctrl`+`` ` ``）。`handson` フォルダにいることを確認してから、`kadaiB` に移動して起動します。

```
# Mac
cd kadaiB
./mvnw spring-boot:run
```

```
# Windows PowerShell
cd kadaiB
.\mvnw.cmd spring-boot:run
```

ターミナルに大量のログが流れます。最後のほうに `Started EquipmentApplication in 3.2 seconds` のような行が出れば起動できています。**このターミナルは閉じないでください。**閉じるとアプリが止まります。以降のステップで別のコマンドを打つときは、ターミナル右上の分割アイコンで新しいターミナルを開いてください。

操作 6 ブラウザで画面を見る

ブラウザで `http://localhost:8080/equipments` を開きます。備品の一覧が20件、表形式で並びます。件数を数えて20件あることを確認してください（課題Aの50件とは別のデータです）。次に、画面上部の検索ボックス（「備品名で検索」と薄く書かれた入力欄）に `PC` と入力し、右隣の「検索」ボタンを押します。**備品名に PC を含む1件だけに絞り込まれます。**一覧に戻すときは、検索ボックスを空にしてもう一度「検索」を押します。

操作 7 詳細画面をいくつか開く

一覧の備品名はリンクになっています。**上から順に、5件以上クリックして詳細画面を開いてください。**一覧に戻るときはブラウザの戻るボタンか、画面下の「一覧へ戻る」を使います。正常に詳細が開く備品と、エラー画面（Whitelabel Error Page）になる備品があります。エラーになった備品の備品コードを控えておいてください。

自分で考える

> **エラーになる備品と、ならない備品で、一覧に見えているデータの何が違いますか**
>
> 一覧画面に出ている列（備品コード・備品名・カテゴリ・在庫数・保管場所・購入日）を見比べてください。
> 一覧の列だけでは違いが見えない場合、**詳細画面には出るが一覧には出ていない項目がある**ことを
> 思い出してください。気づいたことを `memo.md` の `## B-0` に1行書いてください。

考えた後で開いてください

一覧に出ていない項目に「備考」があります。エラーになる備品は、備考が空のものです。詳細画面だけが備考を表示しようとするため、一覧では問題が起きません。

「一覧では起きないが詳細では起きる」という現象は、**両方の画面が同じデータの別の部分を見ている**ときに起きます。どの画面が何を読んでいるかを分けて考えると、原因の範囲を狭められます。

生成物の名前と場所

- `handson/.claude/skills/claude-automation-recommender/`
- `handson/.claude/skills/skill-creator/`
- `handson/.claude/skills/threat-model/`
- `handson/.claude/skills/_lib/`
- `handson/.claude/commands/regression-check.md`
- `handson/CLAUDE.md`（末尾に追記した状態）
- 起動中のアプリ（`http://localhost:8080`）／`handson/memo.md` の `## B-0`

OK基準

- [ ] `handson/.claude/skills/` に4つのフォルダが並んでいる
- [ ] `handson/.claude/commands/` に `selfcheck.md` と `regression-check.md` の2本がある
- [ ] `handson/CLAUDE.md` の末尾に `## 課題B の進め方（演習B から追加）` の見出しがある
- [ ] ターミナルに `Started EquipmentApplication` が出ている
- [ ] ブラウザの一覧画面に備品が20件並んでいる
- [ ] 詳細画面でエラーになる備品を1件以上見つけ、備品コードを控えた

追加と考察

初回の起動はライブラリのダウンロードが走るため、数分かかることがあります。事前セットアップで一度起動していれば、2回目以降は数十秒です。**3分待っても進まないときは挙手してください。**

社内ネットワークの経路によっては外部のリポジトリに繋がらないことがあり、その場合は講師が用意した依存キャッシュに切り替えます。

配置の作業でやったことを整理すると、3つだけです。フォルダをコピーした（Skill）、ファイルを1本コピーした（コマンド）、テキストを末尾に貼り付けた（共有文脈）。特別なインストール作業も、設定画面の操作もありません。**AI の振る舞いを変える仕組みは、置き場所が決まっただのテキストファイルです。**ここが理解できると、社内で自分たちの規約を AI に守らせる方法が具体的に見えてきます。

発展課題

ブラウザで `http://localhost:8080/h2-console` を開いてください。接続画面が出ます。JDBC URL は `jdbc:h2:mem:equipmentdb`、ユーザー名は `sa`、パスワードは空欄のままです。Connect を押すと、データベースの中身を直接見られます。左のツリーから `EQUIPMENT` を選び、`SELECT * FROM EQUIPMENT` を実行してください。

画面に出ているデータと、テーブルの中身は一致していますか。特に `NOTE` 列と `STOCK` 列を見比べてください。**画面で見えるものと、データそのものは別です。**どちらを正として調べるかで、原因の見つかり方が変わります。

SECTION 05 · STEP B-1

自動化の提案を出させる

B-1 読むだけの AI と、書く AI を分ける

使うもの	Claude Code パネル ／ `claude-automation-recommender` Skill
前提	B-0 で Skill 3種を配置し、パネルを開き直していること
手順書	`exercises/exB1_setup_recommender.md`
次	B-2 Plan モードでバグを探す

目的

初見のプロジェクトに対して、どこを自動化すると効くかを AI に提案させます。演習Aでは、講師が用意したハーネスをそのまま配置しました。実務では、その中身を誰かが考えて書く必要があります。ゼロから考えると重い作業ですが、**たたき台なら AI が出せます。**

もう1つの狙いは、**読むだけの Skill と、書く Skill を区別する**ことです。ここで使う Skill は提案しかしません。ファイルを1つも書き換えません。この性質は `SKILL.md` の中に宣言されています。

操作 1 提案を依頼する

Claude Code パネルに入力

```
@kadaiB のプロジェクトに合った自動化を提案してください。
```

Skill の名前は書いていません。**それでも `claude-automation-recommender` が自動で選ばれます。**選ばれる理由は B-5 で扱います。

操作 2 発火したことを確認する

応答の冒頭に、次の1行が出ます。

```
この Skill は読み取りのみを行います。ファイルの作成・変更はしません。
```

さらに応答のいちばん最後、実行サマリーの「従った設定」欄に `claude-automation-recommender` の名前が出ます。この2か所で発火を確認してください。出ていない場合は、依頼の言い方を変えて試します。

発火しないときの言い換え

```
このプロジェクトに Claude Code の設定を仕込むとしたら何が効きますか。提案してください。
```

操作 3 5カテゴリの提案を読む

カテゴリ	何を提案してくるか
MCP サーバー	外部のデータやツールに繋ぐ口。データベース、課題管理、社内ドキュメントなど
Skill	繰り返す作業手順の固定。レビュー観点、書式の決まりなど
フック	特定の操作の前後に自動で走らせる処理。保存後の整形、危険なコマンドの遮断など
サブエージェント	専任の役割を分ける。大量のコードを読む点検作業など
プラグイン	上記をまとめて配る単位

各提案には「効く理由」が付いています。**この欄に、このプロジェクトで実際に見たファイル名が挙がっているかを確認してください。** `pom.xml` や `EquipmentServiceImpl.java` のような具体名が出ていれば、プロジェクトを読んだ上での提案です。「一般的にテストは重要です」のような一般論だけなら、読まずに書いています。

図19 スクリーンショット

撮影内容: `claude-automation-recommender` の実出力。5カテゴリの提案が並んだパネル全体と、冒頭の読み取り専用宣言、末尾の実行サマリー「従った設定」欄に Skill 名が出ている箇所を1枚に収める

操作 4 1件だけ選んで作らせる

提案の最後に、上位3件の優先順位が付いています。その中から**1件だけ**選び、実装を依頼します。

Claude Code パネルに入力

この提案のうち〈選んだ提案の名前〉を、実際に .claude/ の下に作ってください。

操作 5 作られたものを開いて読む

作られたファイルを VSCode で開いてください。**中身を読まずに次に進まないでください。**見るのは2点です。何をするものか日本語で読んで理解できるか。自分の職場で使うとしたら、どこを書き換える必要があるか。

自分で考える

> **あなたの職場のプロジェクトなら、どのカテゴリの提案がいちばん効くと思いますか**
>
> 判断の材料は「いま人手でやっていて、毎回同じことを言っている作業は何か」です。レビューで毎回同じ指摘をしているなら Skill、保存のたびの整形や確認を忘れがちならフック、というように対応します。選んだカテゴリと理由を、`memo.md` の `## B-1` に1行書いてください。

生成物の名前と場所

- Skill を選んだ場合: `handson/.claude/skills/〈名前〉/SKILL.md`
- フックを選んだ場合: `handson/.claude/settings.json`（既存ファイルへの追記）
- サブエージェントを選んだ場合: `handson/.claude/agents/〈名前〉.md`
- `handson/memo.md` の `## B-1`

OK基準

- [ ] 5カテゴリすべての提案が返ってきた（カテゴリの抜けがない）
- [ ] 「効く理由」に `kadaiB` の実在するファイル名が挙がっている
- [ ] 提案を1件選んで実装を依頼し、ファイルが実際に作られた
- [ ] 作られたファイルを開いて中身を読んだ
- [ ] `memo.md` の `## B-1` に選んだ理由が書けた

追加と考察

この Skill が提案しかしないのは、`SKILL.md` の先頭に次の1行が書かれているためです。

```
allowed-tools: Read, Glob, Grep
```

読む道具しか渡していないので、書きたくても書けません。だから操作4で「実際に作ってください」と別に頼む必要がありました。**読むだけの AI と、書く AI を分けておく**のは、チームで安全に運用するときの基本的な分け方です。調査や点検を読み取り専用にしておけば、途中で余計な変更が混ざりません。演習Aで使った `reviewer` サブエージェントも同じ考え方です。

もう1点、提案の質は「何を見たか」で決まります。この Skill は `README.md`、`CLAUDE.md`、ビルド定義、ソースの構造、テストの有無、既存の `.claude/` の順に読みます。**読む順番が `SKILL.md` に書いてあるから、毎回同じ品質で提案が出ます。**人によって当たり外れが出ないことが、手順を固定する価値です。

発展課題

> **課題A に対して同じ質問を投げる**
>
> `@kadaiA のプロジェクトに合った自動化を提案してください。`

提案の内容はどう変わりましたか。`kadaiB` は Java と Maven のプロジェクト、`kadaiA` はあなたが作ったブラウザだけで動くアプリです。技術スタックが違うと提案が変わるなら、**この Skill は何を見て判断していると考えられますか。**

詰まったとき

症状	対処
Skill が発火しない	`handson/.claude/skills/claude-automation-recommender/SKILL.md` が存在するか確認し、パネルを開き直す
提案が3カテゴリしか出ない	「5カテゴリすべてについて提案してください。該当が薄いカテゴリも、効きにくい理由を書いて1件は触れてください。」と追加で依頼する
提案が一般論ばかり	「@kadaiB/pom.xml と @kadaiB/src の構造を見た上で、このプロジェクト固有の提案に絞ってください。」と読ませる対象を明示する
想定と違う場所にファイルが作られた	「いま作ったファイルは handson/.claude/ の下に置く必要があります。移動してください。」と依頼する。`.claude/` の外は読み込まれません

SOURCES

- Claude Code: Hooks（フックの設定方法）
- Claude Code: Subagents（サブエージェントの定義）
- Claude Code: Model Context Protocol（MCP の接続）
- Claude Code: Settings（`settings.json` の書式）

SECTION 06 · STEP B-2

Plan モードで壁打ちしてバグを見つける

B-2 調べさせる。まだ書き換えさせない

使うもの	Claude Code パネル（Plan モード）／ブラウザ／ `handson/CLAUDE.md`
前提	B-1 まで完了。アプリが起動していること
手順書	`exercises/exB2_plan_hunt.md`
次	B-3 Issue を起票する

目的

コードをいきなり直させず、まず AI に**調べさせて、計画だけを出させます**。演習Aでは、指示するとすぐにファイルが作られました。作るときはそれで進みますが、既にあるコードを直すときは事情が変わります。**触っていい範囲がわからないまま書き換えられると、直ったかどうかの判断ができません。**そこで、先に「どこを、どう直すつもりか」を文章で出させ、人間が読んでから承認します。

このステップの終わりに、バグを2件特定します。1件は画面にエラーが出るもの、もう1件はエラーが出ないものです。**この2種類の見つけ方が違う**ことが、B-2 の中心です。

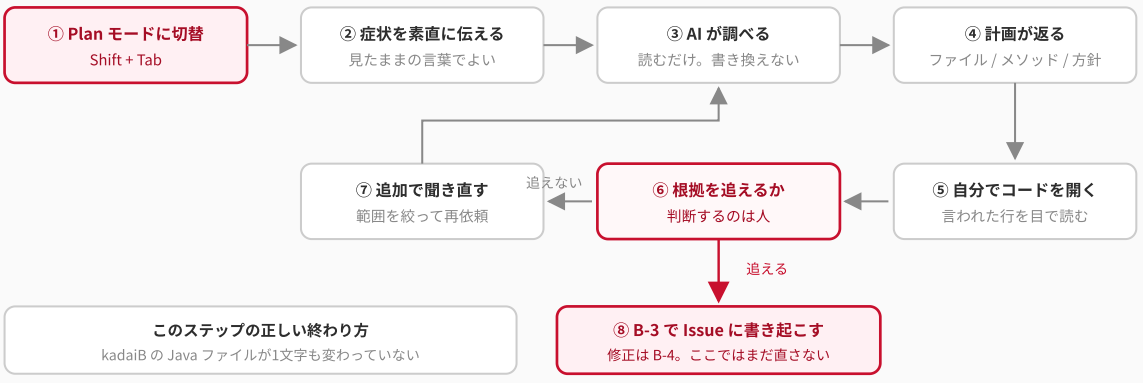

図17b Plan モードでの壁打ちの流れ。⑥で判断するのは人です。追えない根拠は、直った証明にもなりません

操作 1 Plan モードに切り替える

Claude Code パネルで `Shift` + `Tab` を押します。入力欄の近くにモードの表示が切り替わります。もう一度押すと元に戻ります。いまが Plan モードになっていることを目で確認してから進んでください。

操作 2 1件目を調べさせる

Claude Code パネルに入力

詳細画面を開くとエラーになる備品があります。原因を調べて、直すならどこをどう直すかの計画を出してください。まだコードは変更しないでください。

操作 3 返ってきた計画を読む

計画には、少なくとも次が書かれているはずです。原因があるファイルのパス、クラス名とメソッド名、何が起きているか（どの値が想定と違うか）、直す方針。**ファイル名だけで、メソッド名まで書かれていない場合は追加で聞いてください。**

粒度が粗いときの追い方

原因のメソッド名と、該当する行の内容まで具体的に書いてください。

操作 4 自分でコードを開いて確かめる

計画に書かれた場所を、VSCode で自分で開きます。AI が「ここが原因です」と言った行を、自分の目で読んでください。読むときの見方は次のとおりです。Java を書いたことがなくても判断できます。

- その行は、何かの値を使おうとしているか
- その値が「空っぽ」だったとき、どうなると書いてあるか
- 空っぽの場合の分岐が、どこにも書かれていないなら、それが原因の候補です

納得できないときは、そのまま聞く

この行が原因だと言える根拠を、Java を知らない人にもわかるように説明してください。

操作 5 2件目を、別の観点で調べさせる

1件目はエラーが出るので気づけました。2件目はエラーが出ません。**エラーが出ないバグは、仕様と見比べないと見つかりません。**仕様は `handson/CLAUDE.md` の `## 業務ルール（課題B）` に5項目あります。

Claude Code パネルに入力

CLAUDE.md の業務ルールと、いまの実装が食い違っている箇所はありませんか。画面・データ・ルールの3つを突き合わせて調べてください。まだコードは変更しないでください。

操作 6 出てきた候補を自分で確かめる

2件目の候補が出たら、**必ず画面で確かめてください。**AI が「ここが食い違っています」と言っただけでは、根拠になりません。候補が在庫の表示に関するものなら、一覧画面で在庫数がその境目の値になっている備品を探し、表示がルールどおりかを見ます。候補が日付の表示に関するものなら、同じ備品を一覧と詳細の両方で開き、見比べます。

自分で考える

> **エラーが出ないバグを、あなたは何を手がかりに見つけましたか**
>
> 手がかりの候補は3つあります。画面に見えているもの、データの中身（H2 コンソールで見た値）、`CLAUDE.md` に書かれた業務ルール。この3つのうち、どれとどれを突き合わせたときに食い違いが見えましたか。

考えた後で開いてください

エラーが出ないバグは、**動いているものと、動くべきものを見比べたときだけ**見つかります。動いているものは画面とデータで見られます。動くべきものは、どこかに文章で書いてあるはずです。この演習では `CLAUDE.md` の業務ルール5項目がそれにあたります。

実務でこの「動くべきもの」が書かれていないと、食い違いは検出できません。AI に探させる場合はなおさらで、比べる相手がないものは探せません。**仕様を文章で持っていることが、AI に点検させる前提条件になります。**

生成物の名前と場所

`handson/memo.md` の `## B-2` に、次の形式で2件分を記録します。

```
## B-2

- 1件目: ファイル名 / メソッド名 / 症状（画面で何が起きるか）
- 2件目: ファイル名 / メソッド名 / 症状（どの業務ルールと食い違うか）
```

コードの変更は行いません。この時点で `kadaiB` の Java ファイルが1文字も変わっていないことが、このステップの正しい状態です。

OK基準

- [ ] Plan モードで依頼し、コードが変更されずに計画だけが返ってきた
- [ ] バグ候補を2件、ファイル名とメソッド名まで特定できた
- [ ] 特定した箇所のコードを、自分で開いて確認した
- [ ] 2件目については、画面で症状を確認した
- [ ] `memo.md` の `## B-2` に2件分を記録した

追加と考察

Plan モードの使いどころは3つです。

1. 初見のコードを扱うとき。どこに何があるかを把握してから触りたい
2. 触る範囲が広いとき。5ファイルにまたがる変更を一気に出されても読めない

3. 直し方が複数あるとき。方針を選んでから書かせたい

逆に、書き捨てのスクリプトや、失敗しても戻せる作業では、いちいち計画を挟むほうが遅くなります。
モードは安全装置ではなく、読む順番の指定だと考えると使い分けやすくなります。

もう1つ、このステップでは AI に「調べて」と頼みましたが、判断は毎回あなたがしました。計画を読んで、コードを開いて、画面で確かめました。**AI が出した根拠を、人が追えるかどうか**が、この進め方の分かれ目です。

発展課題

エラー画面やターミナルに出ている長い英語のログ（スタックトレース）の、**いちばん上の行**をコピーして、そのまま貼り付けて聞きます。

> **Claude Code パネルに入力**
>
> このエラーの意味を、Java を知らない人にもわかるように説明してください。どの行を見ればよいかも教えてください。

スタックトレースは上から下へ「呼ばれた順の逆」に並びます。読み方を1回覚えると、AI に貼る前に自分で当たりを付けられるようになります。余裕があれば、3件目の食い違いも探してみてください。業務ルールは5項目あり、そのうち何項目が守られていないかを数えると、残りが見えてきます。

詰まったとき

症状	対処
Plan モードにしたのにファイルが変更された	モードの切り替えが効いていません。変更されたファイルを `Ctrl` + `Z`（Mac は `Cmd` + `Z`）で戻し、依頼文に「まだコードは変更しないでください」を必ず添える
計画が長すぎて読めない	「いまの計画を、変更するファイルごとに3行以内で要約してください。」と依頼する
候補が出てこない	「@kadaiB/src/main/java/com/example/equipment/service/EquipmentServiceImpl.java だけを読んで、CLAUDE.md の業務ルール5項目と1つずつ突き合わせてください。項目ごとに、守られているかどうかを表にしてください。」と範囲を絞る
候補が5件も6件も出てきた	次のステップで起票するのは1件です。**画面で症状を再現できたものだけ**を選ぶ
AI が「修正しました」と言ってしまった	「いま変更したファイルと、変更した行をすべて挙げてください。変更前の状態も書いてください。」と聞いてから戻す

SOURCES

- Claude Code: Interactive mode（モード切り替えのキー操作）
- Claude Code: Common workflows（計画を挟む進め方）
- Claude Code: Memory（`CLAUDE.md` の読み込まれ方）
- NullPointerException（Java 17 API 仕様）

SECTION 07 · STEP B-3

Issue を起票する

B-3 直す対象を、文章で確定させる

使うもの	`kadaiB/issues/` ／ Claude Code パネル ／ ブラウザ
前提	B-2 でバグ候補を2件特定していること
手順書	`exercises/exB3_issue.md`
次	B-4 修正して、動かして、テストを通す

目的

見つけた不具合を、**他人が読んで再現できる形の文書**にします。B-2 で見つけた2件は、いまのところあなたの頭の中と `memo.md` の走り書きにしかありません。この状態で AI に「直して」と頼むと、直す対象が毎回ぶれます。会話の中で説明し直すことになり、説明の粒度によって出てくる修正も変わります。

直す対象を先に文章で確定させるのが Issue 駆動の入口です。文章にすると、直したかどうかの判定基準も同時に決まります。

図18 Issue 駆動のサイクル。起票と確認は人、計画と実装と検証は AI。判断の場所が人に残ります

操作 1 テンプレートを読む

`handson/kadaiB/issues/ISSUE_TEMPLATE.md` を開きます。見出しは5つです。

見出し	何を書くか
現象	何をしたら、何が起きたか。画面表示やエラーメッセージをそのまま
期待する動作	本来どうなるべきか。業務ルールのどれに当たるかを明記する
発生箇所	ファイル名 / クラス名 / メソッド名
再現手順	起動から症状が出るまでを番号付きで
修正方針	どこをどう直すか、修正後の確認方法

この5つを増やしも減らしもしません。項目が決まっていることに意味があります。読む側が、どこに何が書いてあるかを探さずに済みます。

— 操作 2 記入例を読む

`kadaiB/issues/ISSUE_001_詳細画面がエラーになる備品がある.md` を開きます。B-2 で見つけた1件目は、既にこの形で起票されています。読むときに見てほしいのは、**再現手順の粒度**です。「詳細画面を開く」ではなく、どの備品のリンクを押したかまで書かれています。この粒度なら、初めて見る人でも同じ操作ができます。

— 操作 3 2件目のファイルを自分で作る

`kadaiB/issues/` の下に、新しい Markdown ファイルを作ります。ファイル名は次の形です。症状を短く言い切る形にします。

```
ISSUE_002_〈内容がわかる名前〉.md

例）ISSUE_002_在庫5個の備品に要補充が出ない.md
例）ISSUE_002_購入日の表示形式が一覧と詳細で違う.md
```

— 操作 4 5見出しを埋める

`ISSUE_TEMPLATE.md` の中身をコピーして貼り付け、上から埋めます。埋めにくい欄は AI に相談します。

Claude Code パネルに入力

@kadaiB/issues/ISSUE_TEMPLATE.md の形式で、いま見つけた〈症状を自分の言葉で〉について Issue の下書きを作ってください。再現手順は実際に操作した順で書いてください。「期待する動作」には CLAUDE.md の業務ルールのどれに当たるかを書いてください。

操作 5 再現手順を自分でなぞる

ここが B-3 で最も大事な手順です

AI が書いた下書きをそのまま採用せず、再現手順の1から順に、実際に操作してください。書かれたとおりに操作して、同じ症状が出ますか。出ない場合は、手順が足りていません。よくある抜けは3つです。

- アプリの起動から書き始めていない
- どの備品を選ぶかが書かれていない（「任意の備品」では再現しないことがあります）
- 検索や絞り込みなど、途中の操作が抜けている

抜けを見つけたら、自分で書き足してください。**AI に直させるより、自分で1行足すほうが速い**場面です。

自分で考える

AI が書いた再現手順のとおりに操作して、どの一手順が足りていませんでしたか

足りなかった手順は、あなたが「言うまでもない」と思って伝えなかったことのはずです。AI は、あなたが画面で何をしたかを見ていません。伝えた範囲でしか書けません。**再現手順が書けないバグは、直ったことも確認できません。**

生成物の名前と場所

`handson/kadaiB/issues/ISSUE_002_〈内容がわかる名前〉.md`。中身は5見出しがすべて埋まった状態にします。空欄や「(記入予定)」を残さないでください。

OK基準

- [ ] ファイル名が `ISSUE_002_` で始まり、内容がわかる日本語が続いている
- [ ] 5つの見出しがすべて埋まっている
- [ ] 「期待する動作」に、`CLAUDE.md` の業務ルール5項目のどれに当たるかが書かれている
- [ ] 「発生箇所」にファイル名とメソッド名が書かれている
- [ ] 再現手順どおりに操作して、実際に症状が再現した

追加と考察

Issue はここではローカルの Markdown ファイルです。本研修では課題管理ツールも共有リポジトリも使いません。それでも機能します。理由は、Issue の価値が**保管場所ではなく、書く項目が決まっていること**にあるからです。5項目が埋まっていれば、修正を依頼するときに `@` でファイルを指すだけで、対象と判定基準がまとめて伝わります。

実務で使うときは、次の順で足していくと無理がありません。

1. まずテンプレートを決める（今日のこれ）
2. 置き場所をチームで1か所に決める
3. 必要になったら、課題管理ツールに移す

順番を逆にして、ツールから入ると「項目が埋まっていない Issue」が溜まります。埋める項目が決まっていないと、AI にも人にも渡せません。

発展課題

Claude Code パネルに入力

この Issue の修正方針を2案出してください。それぞれについて、影響が及ぶ範囲と、確認しなければならないことを書いてください。

2案のうちどちらを選ぶかを決めるとき、**あなたは何を基準にしましたか。**変更が小さいこと、他の画面に影響しないこと、業務ルールへの合致、いずれも基準になります。選んだ基準を Issue の「修正方針」欄に1行書き足してください。この1行があると、あとから読んだ人が「なぜこの直し方にしたのか」を辿れます。

詰まったとき

症状	対処
新しいファイルの作り方がわからない	VSCode のエクスプローラで `issues` フォルダを右クリックし「新しいファイル」を選ぶ。AI に作らせる場合は「kadaiB/issues/ISSUE_002_〈名前〉.md というファイルを作ってください。」と置き場所を明示する
「期待する動作」に何を書けばよいかわからない	`handson/CLAUDE.md` の `## 業務ルール（課題B）` の5項目を読み、反している項目をそのまま引用する
再現手順が長くなりすぎる	症状が出る直前の3手順が具体的なら十分。ただし、どの備品を選ぶかは必ず書く
AI が Issue に修正コードまで書いた	「Issue にはコードを書かないでください。どこをどう直すかの方針だけに書き直してください。」と依頼する
ファイル名に日本語が使えるか不安	使えます。既存の `ISSUE_001_詳細画面がエラーになる備品がある.md` が同じ形式です。空白は入れず、区切りはアンダースコア

SOURCES

- Claude Code: Common workflows（`@` でファイルを指す方法）
- OWASP: 再現手順の書き方の考え方（不具合報告に必要な情報の整理）
- Claude Code: Memory（業務ルールを常時読ませる仕組み）

SECTION 08 · STEP B-4

修正して、動かして、テストを通す

B-4 演習Bの本体。ここが到達基準

使うもの	Claude Code パネル（Agent モード）／ターミナル／ブラウザ
前提	B-3 で `ISSUE_002_...` を起票していること
手順書	`exercises/exB4_fix_and_test.md`
次	B-5 自分の Skill を作る

— このステップで到達する状態

1. バグを **2件** 修正した
2. ブラウザで、修正前に壊れていた画面が正しく表示される。元から正しかった画面も壊れていない
3. `./mvnw test` が `BUILD SUCCESS` で終わり、自分で追加したテストが2件以上ある

3件目以降のバグは発展課題です。時間が足りないときの優先順位は「2件の修正 → 画面での確認 → テスト」です。テストが1件しか書けなくても、修正と確認が終わっていれば到達しています。

— 目的

Issue を根拠に修正させます。「なんとなく直った」で終わらせず、**直した根拠と、直った証拠**を残します。証拠は2種類あります。1つは画面で見た結果です。人が目で見て確認します。もう1つはテストです。次に誰かが同じ場所を触ったときに、自動で確認されます。**画面の確認は今日の自分のため、テストは明日の他人のため**にあります。両方やります。

— 操作 1 Agent モードに戻す

`Shift` + `Tab` を押して、Plan モードを解除します。ここからはファイルを書き換えます。

— 操作 2 1件目を修正させる

Issue のファイルを `@` で指して依頼します。ファイル名は途中まで打つと候補が出ます。

Claude Code パネルに入力

```
@kadaiB/issues/ISSUE_001_詳細画面がエラーになる備品がある.md を読んで、この Issue のとおりに修正してください。業務ルールは CLAUDE.md に従ってください。
```

操作 3 差分を目で読んでから受け入れる

変更の提案が出たら、**受け入れる前に、変更された行を読んでください。**見るのは3点です。

- 変更されたファイルは、Issue の「発生箇所」に書いた場所と一致しているか
- 変更行数が、思っていたより多くないか（1つの Issue に対して、数行のはずです）
- Issue に書いていない箇所まで、ついでに整形されていないか

`CLAUDE.md` の `## 修正するときの決まり（課題B）` に「1つの Issue につき1つの修正にします」と書いてあります。ついでの整形が混ざっていたら、その場で戻すよう伝えてください。

> **余計な変更が混ざったとき**
>
> Issue に書いていない箇所の変更は元に戻してください。この Issue の修正だけを残してください。

操作 4 アプリを再起動する

Java のコードを変えたので、アプリを起動し直します。アプリを動かしているターミナルで `Ctrl` + `C` を押して止めてから、もう一度起動します。

```
# Mac
./mvnw spring-boot:run
# Windows PowerShell
.\mvnw.cmd spring-boot:run
```

操作 5 ブラウザで直ったことを確認する

B-0 でエラーになった備品の詳細画面を、もう一度開いてください。**エラー画面が出ず、詳細が表示されれば成功です。**あわせて、**壊れていなかった備品も1件開いてください。**直した箇所が、元から正しかった表示を変えていないかを見ます。片方だけ見ると、直したつもりで別の場所を壊していることに気づけません。

操作 6 2件目を修正させる

> **Claude Code パネルに入力**
>
> @kadaiB/issues/ISSUE_002_〈あなたが付けた名前〉.md を読んで、この Issue のとおりに修正してください。業務ルールは CLAUDE.md に従ってください。

再起動して、画面で確認します。2件目はエラーが出ないバグなので、**確認する場所を自分で決める必要があります。**業務ルールに合った表示になっているかを、該当する備品で見てください。

操作 7 テストを書かせる

2件とも直ってから、まとめて依頼します。頼み方の言葉に注目してください。

> **Claude Code パネルに入力**
>
> いま直した2件について、二度と同じ壊れ方をしないことを確認するテストを src/test/java の下に作ってください。テストの名前は何を確かめているかがわかる日本語のコメントを添えてください。

操作 8 テストを実行する

アプリを動かしているのとは**別のターミナル**を開きます（VSCode のターミナル右上にある、四角が2つに割れたアイコンが分割ボタンです）。`kadaiB` フォルダに移動して実行します。

```
# Mac
./mvnw test
# Windows PowerShell
.\mvnw.cmd test
```

ここで動くテストは3種類あります。**このうち2つは、あなたが書いたものではありません。**

テストクラス	誰が用意したか	何を見ているか
`EquipmentApplicationTests`	配布時から	アプリが起動できること
`EquipmentAcceptanceTest`	配布時から	業務ルールに沿った動きになっているか。**配布時点では意図的に失敗します**
`EquipmentServiceImplTest`	操作7であなたが書かせたもの	直した2件が二度と壊れないこと

> **受け入れテストの中身は、まだ開かないでください**
>
> `EquipmentAcceptanceTest` が到達基準の判定役です。中身には、どこがどう壊れているかの答えが書いてあります。修正が終わって `BUILD SUCCESS` を確認したあとであれば、開いて読んでかまいません。

最後に `BUILD SUCCESS` と出れば到達です。`BUILD FAILURE` と出た場合は操作9へ進みます。

操作 9 落ちたテストを直す

まず、**どのテストが落ちたかを名前で確認します。**ターミナルの出力から、次の形の行を探してください。

```
[ERROR] Failures:
[ERROR]   EquipmentAcceptanceTest.在庫が5個ちょうどの備品は要補充と判定される ...
```

日本語の部分がテストの名前です。**この名前が、まだ直っていない業務ルールを指しています。**`EquipmentAcceptanceTest` が落ちている場合は、自分が選んだ2件のほかにも業務ルールに反している箇所が残っているという意味です。その名前を手がかりに、もう1件直してください。手順は B-3 と B-4 の繰り返しで、`ISSUE_003_...` として起票します。

`EquipmentServiceImplTest`（自分で書かせたテスト）が落ちている場合は、テストのほうが間違っていることもあります。次の依頼に進んでください。

失敗したときのターミナルの出力を**そのままコピーして貼り付け**、依頼します。貼るのは `Tests run:` と `FAILURE` が含まれるあたりから、その下の詳細までです。全部貼っても問題ありません。

Claude Code パネルに入力

このテストが落ちています。原因を調べて直してください。

（ここに出力を貼り付ける）

緑になるまで繰り返します。**テストのほうが間違っている場合もあります。**その場合は、テストを直すのか実装を直すのかを、業務ルールに戻って判断してください。

— 操作 10 Issue に修正結果を追記する

修正した2つの Issue ファイルの末尾に、次の見出しを足します。

```
## 修正結果

- 変更したファイル：〈パス〉
- 変更内容：〈1行から2行〉
- 確認方法：〈画面のどこを見て確認したか〉
- テスト：〈テストのクラス名とメソッド名〉
```

AI に書かせてもかまいませんが、**確認方法の欄は自分の操作を書いてください。**実際に見た画面のことです。

— 自分で考える

> **もし「テストを書いて」とだけ頼んだら、何のテストが出てきたと思いますか**
>
> おそらく、正常に動く場合のテストが並んだはずです。壊れていた条件を狙ったテストにはなりません。**何を確かめたいかを言葉にすると、出てくるものが変わります。**これはテストに限らず、AI に何かを頼むとき全般に当てはまります。`memo.md` の `## B-4` に、依頼の言い方を変えたことで結果が変わった経験を1行書いてください。

— 生成物の名前と場所

- `handson/kadaiB/src/main/java/com/example/equipment/service/EquipmentServiceImpl.java`（修正）
- `handson/kadaiB/src/test/java/com/example/equipment/service/EquipmentServiceImplTest.java`（新規作成）
- `handson/kadaiB/issues/ISSUE_001_詳細画面がエラーになる備品がある.md`（末尾に `## 修正結果` を追記）
- `handson/kadaiB/issues/ISSUE_002_〈自分で付けた名前〉.md`（同上）
- `handson/memo.md` の `## B-4`

修正の対象が `EquipmentServiceImpl.java` 以外に及んだ場合は、その理由を Issue の修正結果に書いてください。

— OK基準

- [ ] バグを **2件** 修正した
- [ ] ブラウザで、修正前にエラーになっていた画面が正常に表示される

- [ ] 元から正しく動いていた画面が、壊れていない
- [ ] `./mvnw test` が `BUILD SUCCESS` で終わる
- [ ] 自分で追加したテストが **2件以上** あり、どちらのバグに対応するかがコメントでわかる（配布時から入っている `EquipmentAcceptanceTest` の件数は数えません）
- [ ] 2つの Issue に `## 修正結果` が追記されている

`BUILD SUCCESS` にならないまま時間切れになった場合は、**修正2件と画面での確認までで到達**とします。残っている失敗テストの名前を `memo.md` の `## B-4` に控えて、次へ進んでください。

— 追加と考察

修正した箇所は、2件とも `EquipmentServiceImpl.java` の中にありました。画面が崩れていたので `templates/` の HTML を疑った方や、URL の処理なので `EquipmentController.java` を疑った方もいると思います。実際には、**判断をしている場所はサービス層に集まっています。**「在庫が5個以下かどうか」「備考が空のときどう表示するか」は、どちらも判断です。**どこを疑うかを絞れると、AI に渡す範囲も絞れます。**範囲が絞れると、読ませる量が減り、返答も速く正確になります。

もう1点、テストを書いた効果はこの場では見えません。効くのは次に誰かが同じメソッドを触ったときです。今日の [20min] で書いたテストが、半年後の誰かの30分を守ります。**AI に書かせると、この「あとで効く作業」の費用が下がります。**従来は時間がないと真っ先に削られていた部分です。

— 発展課題

3件目のバグを自力で探してください。ヒントは出しません。始め方は B-2 と同じです。画面・データ・`CLAUDE.md` の業務ルールの3つを突き合わせます。業務ルールは5項目あり、そのうち2項目は今日直しました。残り3項目のうち、守られていないものがあるかを1つずつ確かめてください。見つけたら `ISSUE_003_...` を起票し、修正し、テストを足すところまでやってみてください。手順は B-3 と B-4 の繰り返しです。詳しい進め方は `exercises/challenges/ch05_third_bug.md` にあります。

さらに余裕があれば、B-0 で配置したコマンドを実行してください。

Claude Code パネルに入力

```
/regression-check
```

`kadaiB/issues/` にあるすべての Issue をなぞり直し、`./mvnw test` を実行して、結果を1枚の表にまとめます。演習Aの `/selfcheck` と同じで、**確認の手順そのものをコマンドに固定してある**例です。結果は `.work/B_regression.md` に残ります。

詰まったとき

症状	対処
`No tests to run` と出る	テストファイルの置き場所が違います。`kadaiB/src/test/java/com/example/equipment/` の下にあるか確認する
ポートのエラーが出てテストが動かない	アプリと同じポートを使おうとしています。アプリを `Ctrl` + `C` で止めてから `./mvnw test` を実行する
修正したのに画面が変わらない	再起動していません。`Ctrl` + `C` で止めて起動し直す。それでも変わらないならブラウザを強制再読み込み（`Ctrl` + `Shift` + `R`、Mac は `Cmd` + `Shift` + `R`）
直したはずの箇所で別の画面が壊れた	「いまの修正で変更したファイルと行をすべて挙げ、それぞれが影響する画面を教えてください。」と聞く
テストが3回以上落ちて直らない	「このテストは何を確かめるためのものですか。1文で説明してください。説明できないなら、確かめたいことを1つに絞って書き直してください。」と依頼する
時間が足りない	優先順位は「2件の修正 → 画面での確認 → テスト」です。挙手してください

SOURCES

- Spring Boot Reference: Testing（テストの置き場所と実行方法）
- JUnit 5 User Guide（テストの書き方）
- Maven Surefire Plugin（`mvn test` が何を実行するか）
- Claude Code: Common workflows（差分の確認と受け入れ）

SECTION 09 · STEP B-5

skill-creator で自分の Skill を作る

B-5 毎回説明していたことを、ファイルに固定する

使うもの	Claude Code パネル ／ `skill-creator` Skill ／ `/clear`
前提	B-4 でバグを2件直していること（作る Skill の材料になります）
手順書	`exercises/exB5_skill_creator.md`
次	B-6 脅威モデリングを回す

目的

自分のバグの探し方を Skill として書き出し、**次のセッションでそれが自動で使われる**ことを確かめます。B-2 でやったことを思い出してください。画面・データ・業務ルールの3つを突き合わせて、食い違いを探しました。この手順は、あなたが AI に説明したから実行されたものです。明日また同じことをするなら、また同じ説明をすることになります。

図20 Skill が発火する仕組み。Skill を育てる作業の大半は、この `description` の調整です

操作 1 Skill を作らせる

Claude Code パネルに入力

```
いま kadaiB で見つけたバグの探し方を、次から自動で使える Skill にしたいです。Skill を作ってください。
```

操作 2 3つの質問に答える

聞かれること	答え方の例
この Skill は何をするものですか	業務ルールの記述と実装の食い違いを探すもの
どういうときに使いたいですか（実際に打ちそうな言い方を2つか3つ）	「ルールと実装が合っているか調べて」「仕様と違う挙動を探して」「業務ルールと突き合わせて」
手順は何ステップですか	1. 仕様が書かれたファイルを読む 2. 該当する実装を読む 3. 項目ごとに突き合わせて表にする 4. 食い違いを画面で確認する手順を添える

2つ目の質問がいちばん重要です。ここで挙げた言い方が、あとで Skill が選ばれる手がかりになります。**あなたが実際に打ちそうな言葉**で答えてください。かしこまった言い回しに直さないでください。

操作 3 できた SKILL.md を読む

`handson/.claude/skills/〈名前〉/SKILL.md` が作られます。開いて、特に**先頭の `description` 欄**を読んでください。

```
---
name: rule-mismatch-hunter
description: 業務ルールの記述と実装の食い違いを探します。「ルールと実装が合っているか調べて」「仕様と違う挙動を探して」「業務ルールと突き合わせて」のときに使います。
---
```

操作2で答えた言い方が、そのまま入っているはずです。入っていない場合は、いま書き足してください。手で直してかまいません。

操作 4 文脈をリセットする

Claude Code パネルに入力

```
/clear
```

これまでの会話がすべて忘れられます。**さっき Skill を作った記憶も消えます。**ここからは、Skill を作ったことを知らない状態の AI に話しかけることになります。消えるのは会話だけで、ファイルは消えません。

操作 5 Skill の名前を出さずに依頼する

Claude Code パネルに入力

```
このプロジェクトの中で、業務ルールと実装が食い違っていそうな箇所を探してください。
```

Skill の名前も、ファイルのパスも書いていません。

操作 6 発火したかを確認する

応答のいちばん最後、実行サマリーの「従った設定」欄を見てください。**さっき作った Skill の名前が出ていれば発火しています。**出ていない場合も、この演習では失敗ではありません。理由は下の「追加と考察」

で扱います。

— 自分で考える

> **名前を書いていないのに Skill が選ばれたとしたら、AI は何を手がかりに選んだのでしょうか**
>
> `SKILL.md` の `description` 欄をもう一度読んで、操作5で打った文章と見比べてください。共通している言葉はありますか。

考えた後で開いてください

手がかりは `description` の文章だけです。Skill 本体（手順の部分）は、選ばれるまで読まれません。つまり Skill は2段構えで動いています。

1. すべての Skill の `description` を見て、いまの依頼に合うものを選ぶ
2. 選ばれた Skill の中身を読んで、そのとおりに実行する

`description` が薄いと、どんなに中身が良くても呼ばれません。逆に `description` が広すぎると、関係ない依頼でも呼ばれます。**Skill を育てる作業の大半は、この1文の調整です。**

— 生成物の名前と場所

`handson/.claude/skills/〈自分で付けた名前〉/SKILL.md` 。名前は英小文字とハイフンだけを使います（例: `rule-mismatch-hunter` ）。フォルダ名とファイル名の両方が正しくないと読み込まれません。

— OK基準

- [ ] `SKILL.md` が作られ、先頭に `name` と `description` の2行がある
- [ ] `description` に、自分が実際に打ちそうな言い方が2つ以上入っている
- [ ] `/clear` を実行した
- [ ] リセット後の質問で、その Skill が使われた。または、**使われなかった理由を自分の言葉で説明できる**

— 追加と考察

発火しなかった場合も失敗ではありません。`description` に書いた言葉と、あなたが打った質問の言葉が離れていた、ということです。たとえば `description` に「仕様書との整合性を検証します」と書いて、質問で「ルールと違うところ探して」と打つと、離れています。**Skill が発火する条件は `description` の文章です。**

実務での使いどころを挙げます。

- レビューで毎回同じ指摘をしている → 指摘の観点を Skill にする
- 書式が決まった報告書を毎週書いている → 書式と手順を Skill にする
- 新しく入った人に毎回同じ説明をしている → 説明ではなく Skill を渡す

いずれも共通しているのは、**すでに人間の頭の中にある手順を、外に出す作業**だという点です。新しい技術を覚えるのではなく、いまやっていることを書き出すだけです。今日作った Skill も、B-2 でやったことをそのまま書き出しただけでした。演習Aで配置した `kadaia-review` も、講師が同じ作業をして作ったものです。**配られる側から、作る側に回れる**ことが、このステップで確かめたかったことです。

発展課題

発火しなかった場合は、`description` を書き直して、もう一度 `/clear` して試してください。書き直すときは、操作5で実際に打った文章に含まれる言葉を、`description` にそのまま入れてください。

発火した場合は、逆に、発火してほしくない質問でも発火してしまわないかを確かめます。`/clear` してから次のように打ってください。

> **誤発火の確認**
>
> このコードを整形してください。

ここで作った Skill が呼ばれてしまうなら、`description` が広すぎます。**広すぎる description は誤発火します。**誤発火した Skill は、関係ない作業に余計な手順を持ち込み、時間を食います。対象を絞る語（「業務ルールと」「仕様の記述と」など）を足して、もう一度試してください。

詰まったとき

症状	対処
`skill-creator` が発火しない	「この手順を次から自動で使える Skill にしてください。SKILL.md を作ってください。」と言い換える。それでも駄目なら `handson/.claude/skills/skill-creator/SKILL.md` の有無を確認し、パネルを開き直す
質問が1つずつ小分けに来て時間がかかる	3つの答えを一度に書いて送ってかまいません
`/clear` を押すのが怖い	消えるのは会話だけです。直したコードも、作った Skill も、Issue もそのまま残ります
`/clear` 後にプロジェクトのことを知らない状態になった	正常です。`CLAUDE.md` は毎回読み込まれるので業務ルールは引き継がれます。引き継がれないのは会話の内容だけです
作った Skill が読み込まれない	`.claude/skills/〈名前〉/SKILL.md` の形になっているか確認する。`SKILL.md` が大文字であること、フォルダを1階層挟んでいることの2点をよく間違えます

SOURCES

- Claude Code: Agent Skills（frontmatter と発火の仕組み）
- Agent Skills: Best practices（`description` の書き方）
- anthropics/skills（skill-creator の実装）
- Claude Code: Slash commands（`/clear` の挙動）

SECTION 10 · STEP B-6

脅威モデリングを回す

B-6 バグを直すことと、危険を減らすことは別

使うもの	Claude Code パネル ／ `threat-model` Skill
前提	B-4 でバグを2件直していること
手順書	`exercises/exB6_threat_model.md`
次	演習Bの最後です。まとめのセッションに戻ります

目的

「バグを直す」と「危険を減らす」が別の作業であることを、実物の出力で確かめます。B-4 で直した2件は、どちらも「壊れているから直した」ものです。エラーが出る、ルールと違う。だから直しました。

このステップで扱うのは、**壊れていないのに危ないもの**です。いまのコードは仕様どおりに動きますが、誰でも貸出操作ができます。ログインの仕組みがありません。これはバグではありません。そう作ったからです。それでも、社内に置くなら考えなければならないことです。演習Bの締めであり、第2回研修への入り口になるステップです。

> **これは防御側の演習です**
>
> 攻撃の方法を練習するものではありません。自分たちが作ったものの、守るべき場所を先に洗い出す作業です。攻撃寄りの表現で依頼すると応答が止まることがあります。**「守る側として、どこが危ないかを洗い出したい」という枠で聞いてください。**

操作 1 脅威モデルを作らせる

Claude Code パネルに入力

```
@kadaiB のコードベースの脅威モデルを作ってください。
```

`threat-model` Skill が発火します。最初の応答の冒頭に、次の趣旨の宣言が出ます。

```
この Skill は静的解析のみを行います。対象コードの実行、外部への通信、
ファイルの変更（THREAT_MODEL.md の作成を除く）は行いません。
```

この宣言が出ることを確認してください。セキュリティの調査は、やり方によっては調査そのものが危険な操作になります。何をやって何をやらないかを、始める前に宣言させています。

操作 2 質問に答える

コードから読み取れないことを、3問まで聞いてきます。研修の想定でかまいません。

- 利用者は社内の人だけですか、社外にも公開しますか → 「社内のみ」
- 扱っているデータに、個人情報は含まれますか → 「含まない」
- 本番ではどこに置く想定ですか → 「社内ネットワーク内のサーバー」

答えられない場合は「わからないので既定値でお願いします」で進みます。

— 操作 3 待つ

実行には3分から5分かかります。**待っている間に、次の問いを考えておいてください。**「脅威」と「脆弱性」は、何が違うと思いますか。日本語としてはどちらも危ないことのように読めます。この2つを分けて使う理由を、あとで講師が説明します。

— 操作 4 出力を読む

`handson/kadaiB/THREAT_MODEL.md` が作られます。読む場所は4か所です。

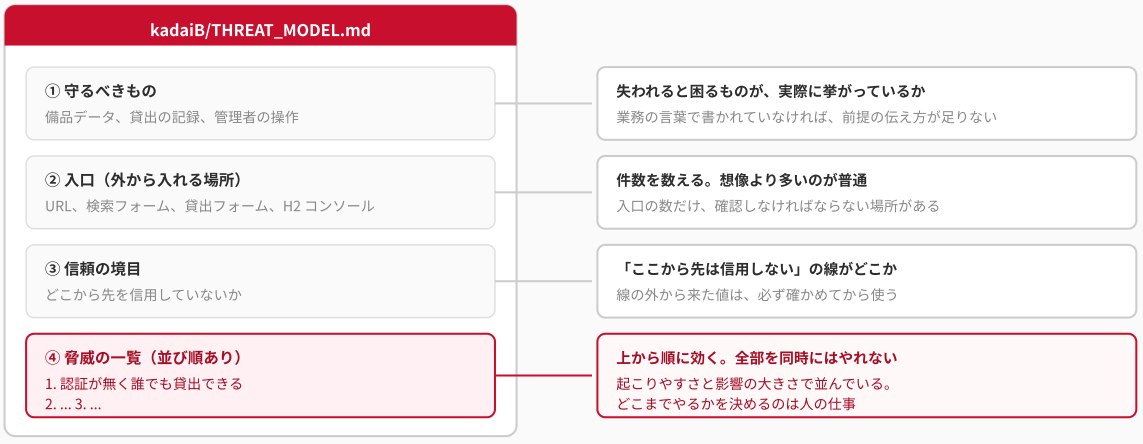

図21b `THREAT_MODEL.md` の読み方。並び順が付いていることに意味があります

— 操作 5 一番上の脅威を、コードに紐づけさせる

Claude Code パネルに入力

この脅威モデルの一番上の項目について、kadaiB のコードのどこが該当するか、ファイルと行を挙げて教えてください。

ファイルと行が返ってきたら、自分で開いて見てください。該当する行が、あなたが B-4 で直した箇所と同じかどうかも確認してください。

— 自分で考える

> **B-4 で直した2件は、この THREAT_MODEL.md に出てきましたか**
>
> 出てこなかったとしたら、それはなぜだと思いますか。**今日直したのは、脅威と脆弱性のどちらでしたか。**

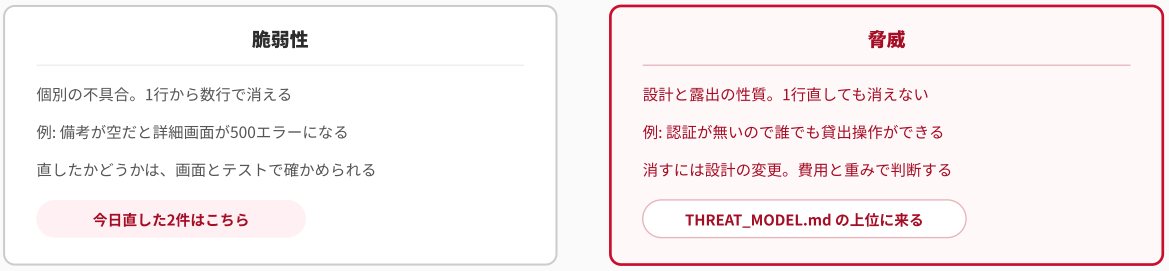

図21 脅威と脆弱性の違い。同じ「危ない」でも、消し方と決め方が違います

考えた後で開いてください

今日直した2件は脆弱性です。1行から数行で消えました。だから、直した時点で脅威モデルには残りません。

一方、`THREAT_MODEL.md` の上位に来るのは、直せば消えるものではありません。「誰でも貸出できる」を消すには、ログインの仕組みを足すことになります。これは1行の修正ではなく、設計の変更です。**どこまでやるかは、費用と、扱っているものの重みで決める判断**になります。技術の問題ではありません。

演習Bの最初に「動かしてから読む」で始めたのと同じで、ここでも順番があります。まず何が危ないかを並べ、次にどこまでやるかを決めます。並べる作業は AI が速く、決める作業は人がやります。

生成物の名前と場所

`handson/kadaiB/THREAT_MODEL.md`。途中経過の記録が `handson/.work/threat_model_checkpoints.jsonl` に残ります。この記録は消さないでください。

OK基準

- [ ] 最初の応答で、静的解析のみを行うという宣言が出た
- [ ] `THREAT_MODEL.md` が生成されている
- [ ] 守るべきもの・入口・脅威の並びが読み取れる
- [ ] 一番上の脅威について、コード上の該当箇所（ファイルと行）を挙げさせられた
- [ ] 今日直した2件が脅威と脆弱性のどちらだったかを、自分の言葉で言える

追加と考察

この Skill の元になっている実装は、Anthropic が公開しているリファレンス実装です。今回配ったのは**コードを実行しない読み取り専用の部分だけ**です。元の実装には、対象のコードを隔離した環境で実際に動かして挙動を確かめる仕組みが含まれています。そちらは Docker と隔離実行の環境を必要とするため、研修の環境では動きません。

ここで確認しておきたいのは、**セキュリティの調査は「読むだけ」と「動かす」で必要な環境がまったく違う**という点です。読むだけなら普段の開発環境で回せます。動かすなら、壊れても困らない隔離された場所が要ります。社内で試すときも、この線引きから入ると話が進みます。

もう1点。`THREAT_MODEL.md` は完成品ではありません。人に確認しないと埋まらない前提が残っています（操作2で3問しか聞いていません）。**AI が作れるのは、議論の出発点になる一覧までです。**ここから先、どれを受け入れてどれに対処するかを決めるのは、いまのところ人の仕事です。

発展課題

Claude Code パネルに入力

この脅威に対して、いまのコードに足りていない対策を3つ挙げてください。優先順位を付けてください。それぞれ、どのくらいの作業量になるかも書いてください。

優先順位の根拠に納得できましたか。納得できない場合、あなたが重く見ているものと、AI が重く見ているものが違います。前提を1行足して聞き直してください。

前提を足して聞き直す

このアプリは社内ネットワークからしか使えません。その前提で優先順位を付け直してください。

前提を1行足すだけで、上位が入れ替わることがあります。**前提を伝えていないと、AI は一般的な想定で答えます。**これは今日の演習全体を通して繰り返し出てきたことです。

詰まったとき

症状	対処
`threat-model` が発火しない	「kadaiB のセキュリティ上の危険を洗い出して、脅威モデリングをしてください。」と言い換える
途中の記録を保存する段階で止まる	`handson/.claude/skills/_lib/checkpoint.py` をコピーし忘れています。B-0 の操作1をやり直す
5分待っても返らない	20名が同時に実行すると混み合います。講師端末の出力を投影で共有します。挙手してください
出力が薄い	配布フォルダに変更履歴が無いため、外部の公開情報を参照する部分が働きません。コード読解の結果だけで出力されます。想定どおりです
応答が途中で止まる	攻撃寄りの表現になっている可能性があります。「守る側として洗い出したい」という枠で聞き直してください

SOURCES

- anthropics/defending-code-reference-harness（threat-model の出典）
- OWASP: Threat Modeling Cheat Sheet
- Microsoft: STRIDE の脅威分類
- OWASP Top 10（Web アプリで頻出する弱点）

SECTION 11 · WRAP

演習Bで持ち帰るもの

今日やったことを、道具の名前ではなく進め方として書き出しておきます。Claude Code を使わない場面でも同じ形が使えます。

動かす→読む

初見のコードは、読解より先に観察する。動いている画面を見てからでないと、何が壊れているかを判断できません

計画→承認→実行

既存コードを直すときは、Plan モードで計画を出させ、人が読んでから書き換えさせる。判断の場所を人に残します

文書→修正→証拠

直す対象を Issue で確定させ、直った証拠を画面とテストの両方で残す。「なんとなく直った」を残しません

今日の作業を職場に移すときの順番

今日やったこと	職場で最初にやること	効き方
`CLAUDE.md` に業務ルール5項目が書いてあった	いま口頭で伝えている決めごとを、5行でよいので文章にする	比べる相手ができるので、エラーの出ないズレを AI が探せるようになる
Issue テンプレートの5見出しが決まっていた	不具合を書く項目をチームで決めて、置き場所を1か所にする	対象と判定基準がまとめて渡せる。会話での説明が要らなくなる
Skill を1本作って発火させた	レビューで毎回言っている指摘を1本だけ Skill にする	説明する回数が減る。人が変わっても同じ観点で見られる
読み取り専用の Skill を使った	調査や点検の役割には、書く道具を渡さない設定にする	調べているつもりが書き換わっていた、という事故が起きない
脅威モデルを1本出した	いま動いているものの入口を数える。それだけで議論が始まる	直せば消えるものと、設計を変えないと消えないものを分けて話せる

今日の到達を自分で確認する

バグ2件の修正、ブラウザでの確認、`./mvnw test` の `BUILD SUCCESS`。この3つが済んでいれば到達です。`memo.md` に B-0 から B-6 までの気づきが残っていれば、持ち帰る材料も揃っています。3件目のバグ探しや `challenges/` の発展課題は、研修後に自分の環境で続けられます。

SECTION 12 · GLOSSARY

課題Bで出てくる用語

演習B の中で説明なく出てくる言葉をまとめます。全体で共通する用語（コンテキスト、ハーネス、Plan モード、Skill など）は共通編の用語集にあります。

Spring Boot

Spring Boot 3.4

Java で Web アプリを作るときによく使う枠組み。設定を最小限にして起動できるのが特徴です。

B-0

レイヤードアーキテクチャ

layered architecture

役割ごとに置き場所を分ける作り方。課題Bは controller / service / repository / model の4層です。

B-0

Thymeleaf

Thymeleaf

Java 側のデータを HTML に流し込んで画面を作る仕組み。`templates/` の下にあります。

B-0

H2 インメモリDB

H2 in-memory database

メモリ上だけに置くデータベース。アプリを止めると中身は消えます。次に起動すると `data.sql` から作り直されます。

B-0

H2 コンソール

H2 console

ブラウザから H2 の中身を直接見る画面。`http://localhost:8080/h2-console` で開きます。

B-0

Maven Wrapper

mvnw

`./mvnw` のこと。Maven を各自でインストールしなくても、プロジェクトに同梱された仕組みで動かせます。

B-0

Whitelabel Error Page

Whitelabel Error Page

Spring Boot が既定で出すエラー画面。原因までは書かれていないので、ターミナルのログを見ます。

B-0

スタックトレース

stack trace

エラーが起きたときに出る、呼び出しの経路を示す長いログ。上から下へ「呼ばれた順の逆」に並びます。

B-2

NullPointerException

NullPointerException

中身が空のものに対して操作しようとしたときに出る Java のエラー。空の場合の分岐が書かれていないと起きます。

B-2

Optional

java.util.Optional

中身があるかもしれないし、無いかもしれない、を表す Java の入れ物。`.get()` で開かず、無い場合の振る舞いを決めてから使います。

B-2

オフバイワン

off-by-one

「5未満」と「5以下」のような、境界が1つずれている間違い。エラーが出ないので、仕様と見比べないと見つかりません。

B-2

Issue 駆動開発

issue-driven development

直す対象を先に文書にしてから、それを根拠に修正を進める進め方。判定基準が同時に決まります。

B-3

単体テスト

unit test

1つの部品が期待どおり動くかを、自動で確かめるコード。`src/test/java/` に置きます。

B-4

リグレッション

regression

直したはずのものが、別の変更でまた壊れること。テストを残しておくと、次に触った人が気づけます。

B-4

BUILD SUCCESS

BUILD SUCCESS

Maven の実行が最後まで通ったことを示す表示。`./mvnw test` の最後に出れば、テストがすべて成功しています。

B-4

実行サマリー

execution summary

応答の最後に AI が自己申告する4項目（動作・従った設定・対象ファイル・未実施）。どの Skill が使われたかはここで確認します。

B-1

脅威モデリング

threat modeling

どこから何を狙われうるかを先に洗い出す作業。守るべきもの、入口、信頼の境目を並べます。

B-6

脆弱性

vulnerability

個別のバグ。1行直せば消えるもの。設計を変えないと消えない「脅威」とは区別します。

B-6

SECTION 13 · TROUBLESHOOTING

詰まったときの対処

症状から引ける形でまとめます。各ステップの中にも「詰まったとき」の表がありますが、環境まわりはここに集約しています。**3分試して進まないときは挙手してください。**待つ時間がいちばんもったいない使い方です。

起動できない

症状	原因と対処
`Port 8080 was already in use`	別のアプリが 8080 番を使っています。前のターミナルでアプリが起動したままになっていないか確認してください。見つからない場合は Mac なら `lsof -i :8080` で番号（PID）を調べて `kill 番号` 、Windows PowerShell なら `Get-NetTCPConnection -LocalPort 8080` で番号を調べて `Stop-Process -Id 番号` で止めます
`./mvnw` が permission denied（Mac）	`kadaiB` フォルダで `chmod +x mvnw` を実行してから、もう一度起動してください
`Unsupported class file major version` などの Java のエラー	`java -version` を実行して 17 と表示されるか確認してください。別のバージョンが動いている場合は挙手してください
ダウンロードが 3分以上進まない	社内ネットワークの経路で外部のリポジトリに繋がっていない可能性があります。挙手してください。講師が用意した依存キャッシュに切り替えます
ブラウザに何も表示されない	ターミナルに `Started EquipmentApplication` が出ているか確認してください。出ていない場合はまだ起動中です。ログの流れが止まってから開き直してください

Claude Code が思ったとおりに動かない

症状	原因と対処
コピーしたはずの Skill が見当たらない	`.claude` のようにドットで始まるフォルダは、Finder では標準で隠れています。VSCode のエクスプローラからは見えるので、そちらで確認してください
Skill が発火しない	3点を順に確認します。`.claude/skills/〈名前〉/SKILL.md` の形になっているか。パネルを開き直したか。依頼の言葉が `description` の言葉と離れていないか
業務ルールを見てくれない	VSCode で開いているフォルダが `handson` になっているか確認してください。`kadaiB` を単体で開くと `CLAUDE.md` が読み込まれません
実行サマリーが出ない	`CLAUDE.md` の `## 応答の最後に必ず出すもの` が消えていないか確認してください。B-0 の追記で末尾に貼り付けるとき、既存の内容を上書きしてしまうことがあります
パネルが認証エラーになる	接続先の設定が外れています。パネルを閉じて開き直してください。復帰しない場合は挙手してください
応答が返ってこない	20名が同時に重い依頼を出すと混み合います。1分待って返らない場合は、依頼の範囲を `@kadaiB/src/main/java/` のように狭めて出し直してください

テストと確認

症状	原因と対処
`./mvnw test` が `BUILD FAILURE` で終わる	出力の `Tests run:` と `FAILURE` を含む範囲をそのままコピーして Claude Code に貼り、原因を調べさせてください。テストのほうが間違っている場合もあります。業務ルールに戻って、どちらを直すか判断します
`No tests to run` と出る	テストファイルの置き場所が違います。`kadaiB/src/test/java/com/example/equipment/` の下にあるか確認してください
H2 コンソールに接続できない	JDBC URL を `jdbc:h2:mem:equipmentdb`、ユーザー名を `sa`、パスワードは空欄にしてください。既定値のままだと接続できません

元に戻したくなったら

ファイルを1つ前の状態に戻すには、VSCode で `Ctrl` + `Z`（Mac は `Cmd` + `Z`）を押して保存し直します。**何を変えたかわからなくなった場合は、まず AI に聞いてください。**「いま変更したファイルと、変更した行をすべて挙げてください。変更前の状態も書いてください。」で一覧が出ます。それでも収拾がつかない場合は、配布ZIPを別の場所に展開し直して `kadaiB` だけを差し替えます。挙手してください。

SOURCES

- Claude Code: Troubleshooting（公式のトラブル対処）
- Spring Boot How-to: 組み込みサーバーのポート変更
- H2 Database: 接続URLの書式
- Java SE 17 Documentation（バージョン確認）

生成AIプログラミング入門編 手元ガイド 課題B編

本資料は本研修の受講者向け配布物です。手順の本体は配布フォルダ `handson/exercises/` の各ファイルにあります。