Add icon Add cover

# Task Management System

## Project Overview

**Complexity:** Medium

**Time Estimate:** 10 hours

**Technology Stack:** Java 21 (LTS), IntelliJ IDEA Community Edition

Build a **console-based Project/Task Management System** where users can create projects, add and assign tasks, record task completion status, and calculate completion averages.

This lab focuses on **core Java programming** and **object-oriented principles** while storing all data **in-memory using arrays** (no databases or external dependencies).

The lab serves as the foundation for the multi-week Project Management series.

Future labs will gradually enhance this base — replacing arrays with **collections**, adding **file persistence**, and integrating **user authentication** and **reporting dashboards**.

## Learning Objectives

By completing this lab, you will be able to:

- Understand how to design a simple **console-based application** using **core Java**.
- Apply **object-oriented programming principles** (encapsulation, inheritance, and polymorphism) to represent **projects**, **tasks**, and **users**.
- Use **Java arrays**, **loops**, and **conditional structures** to manage and manipulate data in memory.
- Define and extend **classes**, **abstract classes**, and **interfaces** to promote clean, reusable designs.
- Demonstrate **method overloading** and **method overriding** to implement flexible and polymorphic behaviors.
- Design simple **menu-driven console interfaces** with structured user input validation.
- Compute project completion percentages by aggregating data from arrays of task statuses.
- Lay the groundwork for future enhancements, where arrays will evolve into **Java Collections** and file-based data storage.

## System Features Overview

Your **Project/Task Management System** should support the following five core features:

### Feature 1: Project Catalog Management

- Create new projects (e.g., SoftwareProject, HardwareProject).
- View all existing projects with details (ID, name, description, team size, budget, etc.).
- Filter projects by type (software/hardware).
- Display project-specific attributes dynamically.

### Feature 2: Task Operations

- Add tasks to specific projects.

- Assign task status (Pending, In Progress, Completed).
- View all tasks per project with progress details.
- Update or delete tasks.
- Validate inputs to prevent invalid task status or duplicate task names.

### Feature 3: User Management

- Create and manage system users (RegularUser and AdminUser).
- Assign users to projects or tasks.
- Enforce role-based access (Admin can delete/update; Regular can view/add).
- Automatically generate unique user IDs.

### Feature 4: Status Processing & Reporting

- Calculate and display completion averages per project.
- Generate status reports (e.g., “Project Alpha is 75% complete”).
- Display task statistics: total, completed, and pending counts.
- Show per-user performance summaries (future expansion).

### Feature 5: Menu Navigation & User Experience

- Display a clear **main menu** and **sub-menus** for operations.
- Validate all user inputs (numbers, text, IDs).
- Provide formatted outputs with clear sections and alignment.
- Support graceful exit and return navigation.

## Console UI Examples

Below are examples of how the console interface should look when the system runs.

### 1. Main Menu

```
╔═══════════════════════════════════════╗
║     JAVA PROJECT MANAGEMENT SYSTEM    ║
╚═══════════════════════════════════════╝

Current User: Alice Johnson (Admin)

Main Menu:
-----------
1. Manage Projects
2. Manage Tasks
3. View Status Reports
4. Switch User
5. Exit

Enter your choice: _

```

## 2. Project Catalog Display

```
╔═══════════════════════════════════════════╗
║               PROJECT CATALOG             ║
╚═══════════════════════════════════════════╝

Filter Options:
1. View All Projects (5)
2. Software Projects Only
3. Hardware Projects Only
4. Search by Budget Range

Enter filter choice: 1

──────────────────────────────────────────────────────────────────────
ID   | PROJECT NAME          | TYPE       | TEAM SIZE | BUDGET
──────────────────────────────────────────────────────────────────────
P001 | Alpha Tracker         | Software   | 5         | $15,000
     | Description: Task tracking app for startups
──────────────────────────────────────────────────────────────────────
P002 | IoT Sensor Kit        | Hardware   | 3         | $10,000
     | Description: Sensor prototype for smart devices
──────────────────────────────────────────────────────────────────────
Enter project ID to view details (or 0 to return): _
```

## 3. Project Details View

```
╔═══════════════════════════════════════════╗
║           PROJECT DETAILS: P001           ║
╚═══════════════════════════════════════════╝

Project Name: Alpha Tracker
Type: Software
Team Size: 5
Budget: $15,000

Associated Tasks:
──────────────────────────────────────────────────────────────────────
ID   | TASK NAME           | STATUS
──────────────────────────────────────────────────────────────────────
T001 | Design Database     | Completed
T002 | Implement API       | In Progress
T003 | Write Unit Tests    | Pending
──────────────────────────────────────────────────────────────────────
Completion Rate: 33%

Options:
1. Add New Task
2. Update Task Status
3. Remove Task
4. Back to Main Menu

Enter your choice: _
```

### 4. Add Task

```
╔═══════════════════════════════════════╗
║              ADD NEW TASK             ║
╚═══════════════════════════════════════╝

Enter task name: Implement UI
Enter assigned project ID: P001
Enter initial status (Pending/In Progress/Completed): Pending

✓ Task "Implement UI" added successfully to Project P001!
```

### 5. Update Task Status

```
Enter task ID: T002
Enter new status: Completed

✓ Task "Implement API" marked as Completed.
```

### 6. Status Report Display

```
╔═══════════════════════════════════════╗
║          PROJECT STATUS REPORT        ║
╚═══════════════════════════════════════╝

──────────────────────────────────────────────────────────────
PROJECT ID | PROJECT NAME      | TASKS | COMPLETED | PROGRESS (%)
──────────────────────────────────────────────────────────────
P001       | Alpha Tracker     |   4   |    3      |   75%
P002       | IoT Sensor Kit    |   2   |    1      |   50%
──────────────────────────────────────────────────────────────
AVERAGE COMPLETION: 62.5%
──────────────────────────────────────────────────────────────
```

### 7. Input Validation Examples

```
Enter project ID: XYZ
❌ Error: Invalid input. Please enter a valid numeric or prefixed ID (e.g., P001).

Enter task status: Done
❌ Error: Invalid status. Please choose from [Pending, In Progress, Completed].

Enter team size: -3
❌ Error: Team size must be greater than 0.

Enter again: 5
```

```
✓ Project successfully created.

```

## Expected User Workflows

### Workflow 1: Complete Task Assignment Journey

1. User logs into system.
2. System displays the main menu.
3. User selects “Manage Projects.”
4. User adds a new project (“Alpha Tracker”).
5. User adds tasks under that project.
6. User updates task statuses.
7. System calculates completion average.
8. User views report.

### Workflow 2: Project Discovery

1. User selects “Browse Projects.”
2. Filters by Software Projects.
3. Views detailed project info.
4. Adds a new related task.
5. Returns to main menu.

### Workflow 3: Task Management

1. User opens “Manage Tasks.”
2. Views tasks for selected project.
3. Updates task status.
4. Deletes outdated tasks.
5. System recalculates progress.

### Workflow 4: Status Reporting

1. User opens “View Status Reports.”
2. System displays all projects with completion percentages.
3. User compares project progress and returns to menu.

## User Stories

### Epic 1: Project Catalog Management

### US-1.1: Browse Projects

**As a user**

I want to view all available projects with their details

**So that** I can understand ongoing work.

**Acceptance Criteria:**

- Display all projects with ID, name, type, and budget.
- Show team size and description.
- Support filtering by project type.

**Technical Requirements:**

- Create abstract class `Project` with:
  - Private fields: `id`, `name`, `description`, `budget`, `teamSize`.
  - Abstract method `getProjectDetails()`.
  - Concrete method `displayProject()`.
- Implement subclasses `SoftwareProject` and `HardwareProject`.
- Store projects in an array (minimum 5).

### US-1.2: Search Projects by Budget Range

**As a user**

I want to search projects within a specified budget range

**So that** I can filter based on funding.

**Acceptance Criteria:**

- Input min and max budget.
- Show projects within range or "No projects found."
- Validate numeric inputs.

**Technical Requirements:**

- Use comparison operators and loops.
- Use conditionals to check valid range.
- Display formatted output using `System.out.printf()`.

## Epic 2: Task Operations

### US-2.1: Add Task to Project

**As a user**

I want to add tasks under a project

**So that** I can track progress effectively.

**Acceptance Criteria:**

- Assign unique task IDs.

- Specify task name and initial status.
- Prevent duplicates.

**Technical Requirements:**

- Create `Task` class with fields: `id`, `name`, `status`.
- Implement interface `Completable` with method `boolean isCompleted()`.
- Store tasks in an array within each project.

### US-2.2: Update Task Status

**As a user**

I want to update the status of a task

**So that** progress reflects accurately.

**Acceptance Criteria:**

- Select task by ID.
- Choose from allowed statuses.
- Update and display success message.

**Technical Requirements:**

- Use loops to locate task in project's task array.
- Use enums or constants for valid statuses.
- Apply encapsulation with proper setters/getters.

## Epic 3: User Management

### US-3.1: Create User Profiles

**As a system**

I want to create different user roles

**So that** permissions can vary.

**Acceptance Criteria:**

- Create `RegularUser` and `AdminUser` classes.
- Assign auto-generated unique IDs.
- Display user role when logged in.

**Technical Requirements:**

- Abstract `User` class with fields `id`, `name`, `email`, `role`.
- Use static counter for ID generation.
- Apply inheritance and polymorphism for role behavior.

## Epic 4: Status Processing & Reporting

### US-4.1: Calculate Project Completion Average

**As a system**

I want to calculate how many tasks are completed per project

**So that** I can show progress percentage.

**Acceptance Criteria:**

- Display total tasks, completed tasks, and percentage.
- Handle projects with zero tasks.
- Round percentages to 2 decimals.

**Technical Requirements:**

- Use arithmetic and conditional operators.
- Iterate task arrays to count completions.
- Store computed percentages temporarily for reporting.

## Epic 5: Menu Navigation & Application Control

### US-5.1: Display Main Menu

**As a user**

I want to navigate through clear menu options

**So that** I can easily use the system.

**Acceptance Criteria:**

- Display menu options numbered (1–5).
- Validate choice.
- Loop until exit chosen.

**Technical Requirements:**

- Use `Scanner` for input.
- Implement menu with `switch` or `if-else`.
- Wrap in `do-while` loop for continuous running.

# Project Structure

Java

```
project-management-system/
│
├── src/
│   ├── Main.java                    # Application entry point
│   ├── models/
│   │   ├── Project.java             # Abstract base class
│   │   ├── SoftwareProject.java     # Concrete project type
│   │   ├── HardwareProject.java     # Concrete project type
│   │   ├── Task.java                # Task model
│   │   ├── User.java                # Abstract user base
│   │   ├── RegularUser.java         # Concrete user type
│   │   ├── AdminUser.java           # Concrete user type
```

```
13 │   │   └── StatusReport.java         # Status report generation
14 │   │
15 │   ├── interfaces/
16 │   │   └── Completable.java          # Interface for completion logic
17 │   │
18 │   ├── services/
19 │   │   ├── ProjectService.java       # Project operations
20 │   │   ├── TaskService.java          # Task operations
21 │   │   └── ReportService.java        # Reporting logic
22 │   │
23 │   └── utils/
24 │       ├── ConsoleMenu.java          # Menu handling
25 │       └── ValidationUtils.java      # Input validation
26 │
27 ├── docs/
28 │   ├── class-diagram.png
29 │   └── design-decisions.md
30 │
31 └── README.md
32
```

## Implementation Phases

### Phase 1: Foundation Setup (1–2 hours)

**Tasks:**

- Install JDK 21 and IntelliJ IDEA.
- Create project folder structure.
- Implement `Project` and `User` base classes.
- Test object creation with sample data.

**Deliverables:**

- Working environment.
- Abstract base classes created.
- Sample data printed successfully.

### Phase 2: Core Logic Development (2–3 hours)

**Tasks:**

- Implement `Task` and `Completable`.
- Add `SoftwareProject` and `HardwareProject` subclasses.
- Implement task addition, updates, and completion calculation.

**Deliverables:**

- Working project–task relationship.
- Completion logic functioning.

### Phase 3: User Interface Integration (2–3 hours)

**Tasks:**

- Implement console menu and navigation.
- Handle user inputs and validations.
- Display formatted project/task lists.
- Integrate reporting logic.

**Deliverables:**

- Fully interactive console app.
- All user stories functional.

### Phase 4: Documentation & Finalization (1 hour)

**Tasks:**

- Create `README.md` with setup and usage guide.
- Add class diagram and OOP design rationale.
- Final testing and code cleanup.

**Deliverables:**

- Complete documentation.
- Clean, well-commented codebase.

## Minimum Requirements Checklist

- ◯ JDK 21 and IntelliJ installed
- ◯ At least 2 project types implemented
- ◯ At least 2 user types implemented
- ◯ Minimum 5 sample projects created
- ◯ Use of arrays for storage
- ◯ Encapsulation applied to all models
- ◯ Abstract classes & interfaces implemented
- ◯ Polymorphism demonstrated
- ◯ Input validation implemented
- ◯ Completion percentage calculation working
- ◯ Console navigation functional
- ◯ README and documentation included

## Grading Rubric

| | Criteria | Points | Excellent (90–100%) | Good (75–89%) | Satisfactory (60–74%) |
|---|---|---|---|---|---|
| 1 | Development Environment & Basics | 10 | JDK 21 & IntelliJ setup perfect; excellent use of arrays, loops, conditionals | Setup functional, few issues | Basic setup but missing validations |
| 2 | Classes, Objects & Encapsulation | 20 | Well-structured classes; perfect encapsulation; clear constructors | Mostly encapsulated, minor design issues | Basic encapsulation; incomplete constructors |
| 3 | Inheritance & Polymorphism | 20 | Strong hierarchies for Project/User; clear overriding; polymorphic reporting | Proper inheritance with limited polymorphism | Minimal inheritance demonstration |
| 4 | Abstract Classes & Interfaces | 20 | Clear abstract contracts (Project, User, Completable) implemented correctly | Proper but limited use of abstractions | Basic implementation; partial understanding |
| 5 | Functionality & Code Quality | 20 | All user stories working; clean code; readable formatting; complete testing | Most features working; well-organized | Core logic functional; partial validation |
| 6 | Data Structures & Algorithms (DSA) | 10 | Excellent use of arrays and simple algorithms (search/sort); efficient iteration; clear logic flow with awareness of complexity | Proper array operations; some inefficiencies but logical structure | Basic array handling; repetitive or inefficient logic |
| 7 | **Total** | **100** | | | |

---

## Submission Requirements

**Required Deliverables:**

- Public **GitHub Repository** containing:
  - All source code under `/src`

- `README.md` with setup steps
- UML diagram and design rationale

- **Documentation** must include:
  - Setup and run instructions
  - Feature summary mapped to user stories
  - Class diagram showing inheritance and relationships
  - Explanation of OOP design choices

### Submission Link:

Please fill out this form (**insert actual form link here**)

## Testing the Application

### Test Scenario 1: View Projects

- Run application
- Select option 2 (View Projects)
- Verify all created projects display with correct details (Project ID, Name, Type, Completion %)
- Check that Software and Hardware projects are listed separately
- Verify total projects count displays correctly
- Confirm data is retrieved from the in-memory array

### Test Scenario 2: Add Software Project

- Select option 1 (Add Project)
- Choose "Software Project" type
- Enter project details (name, description, duration, budget)
- Verify unique project ID generated (e.g., PRJ001, PRJ002)
- Confirm project appears when viewing all projects
- Ensure data is stored correctly in the array

### Test Scenario 3: Add Hardware Project

- Select option 1 (Add Project)
- Choose "Hardware Project" type
- Enter details as prompted
- Verify project is stored with type = "Hardware"
- Check ID auto-increments from previous project ID
- Verify total count updates correctly

### Test Scenario 4: Add Task to Project

- Select option 3 (Add Task)
- Choose a project ID to associate task with
- Enter task details (Task Name, Assigned To, Status, Hours)

- Verify task is added under correct project in array
- Confirm Task ID auto-generates (e.g., TSK001, TSK002)

### Test Scenario 5: View Tasks

- Select option 4 (View Tasks)
- Enter a valid Project ID
- Verify all tasks display with correct information (Name, Status, Hours)
- Check completed and pending tasks are labeled properly
- Confirm "No tasks found" message for projects with none

### Test Scenario 6: Update Task Status

- Select option 5 (Update Task Status)
- Enter valid Task ID
- Change status from "Pending" to "Completed"
- Verify update reflects in project task list
- Confirm array data structure updated correctly

### Test Scenario 7: Calculate Completion Percentage

- Select option 6 (Calculate Project Completion)
- Enter valid Project ID
- Verify correct completion % based on completed vs total tasks
- Ensure calculated result displays with two decimal places
- Test with all tasks complete and all tasks pending

### Test Scenario 8: Handle Invalid Inputs

- Enter invalid menu option (e.g., 10)
- Verify program displays "Invalid option" message
- Enter non-numeric input where number expected
- Verify application prompts user to re-enter correctly
- Confirm program does not crash

### Test Scenario 9: ID Auto-generation

- Create multiple projects and tasks
- Verify sequential Project IDs (PRJ001, PRJ002, etc.)
- Verify sequential Task IDs (TSK001, TSK002, etc.)
- Confirm IDs remain unique after multiple runs within same session

---