

**UNIVERSITY OF ENGINEERING & TECNOLOGY**

**PESHAWAR**

**PROJECT:**

**CONSOLE BASED**

**TODO LIST APPLICATION**

**MEMBERS:**

**SYED ABDUL MUQSIT SHAH**

**KASHIF ALI**

**ABDUL MAJID**

**ARSALAN**

**CLASS: DATA SCIENCE**

**2ND SEMESTER**

**SUMMITTED TO: SIR MUHAMMAD**

**DEMARTMENT OF CS&IT**

**SOURCE CODE:**

package com.mycompany.todolist;

import java.io.\*;

import java.time.LocalDateTime;

import java.time.format.DateTimeFormatter;

import java.util.\*;

/\*\*

\* Console-based Todo List Application

\* Features: Add, View, Edit, Delete, Mark Complete, Priority levels, Save/Load

\*/

public class Todolist {

private static List<Task> tasks = new ArrayList<>();

private static Scanner scanner = new Scanner(System.in);

private static final String DATA\_FILE = "tasks.txt";

private static final DateTimeFormatter DATE\_FORMAT = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm");

public static void main(String[] args) {

loadTasks();

showWelcome();

while (true) {

showMenu();

int choice = getChoice();

switch (choice) {

case 1 -> addTask();

case 2 -> viewTasks();

case 3 -> editTask();

case 4 -> deleteTask();

case 5 -> markTaskComplete();

case 6 -> viewTasksByPriority();

case 7 -> viewCompletedTasks();

case 8 -> clearAllTasks();

case 9 -> {

saveTasks();

System.out.println("Goodbye! Tasks saved successfully.");

return;

}

default -> System.out.println("Invalid choice. Please try again.");

}

}

}

private static void showWelcome() {

System.out.println("╔══════════════════════════════════════╗");

System.out.println("║ CONSOLE TODO LIST APP ║");

System.out.println("║ Stay organized, stay productive! ║");

System.out.println("╚══════════════════════════════════════╝");

System.out.println();

}

private static void showMenu() {

System.out.println("\n" + "=".repeat(40));

System.out.println("📋 MAIN MENU");

System.out.println("=".repeat(40));

System.out.println("1. ➕ Add Task");

System.out.println("2. 📋 View All Tasks");

System.out.println("3. ✏️ Edit Task");

System.out.println("4. 🗑️ Delete Task");

System.out.println("5. ✅ Mark Task Complete");

System.out.println("6. 🔥 View Tasks by Priority");

System.out.println("7. ✅ View Completed Tasks");

System.out.println("8. 🧹 Clear All Tasks");

System.out.println("9. 🚪 Exit");

System.out.println("=".repeat(40));

System.out.print("Enter your choice (1-9): ");

}

private static int getChoice() {

try {

return Integer.parseInt(scanner.nextLine().trim());

} catch (NumberFormatException e) {

return -1;

}

}

private static void addTask() {

System.out.println("\n" + "=".repeat(30));

System.out.println("➕ ADD NEW TASK");

System.out.println("=".repeat(30));

System.out.print("📝 Task description: ");

String description = scanner.nextLine().trim();

if (description.isEmpty()) {

System.out.println("❌ Task description cannot be empty!");

return;

}

System.out.println("\n🔥 Priority levels:");

System.out.println("1. 🔴 HIGH");

System.out.println("2. 🟡 MEDIUM");

System.out.println("3. 🟢 LOW");

System.out.print("Select priority (1-3, default: 2): ");

Priority priority = Priority.MEDIUM;

try {

int priorityChoice = Integer.parseInt(scanner.nextLine().trim());

switch (priorityChoice) {

case 1 -> priority = Priority.HIGH;

case 2 -> priority = Priority.MEDIUM;

case 3 -> priority = Priority.LOW;

default -> System.out.println("Invalid priority, using MEDIUM as default.");

}

} catch (NumberFormatException e) {

System.out.println("Invalid input, using MEDIUM priority as default.");

}

Task newTask = new Task(description, priority);

tasks.add(newTask);

System.out.println("\n✅ Task added successfully!");

System.out.println("📋 Task: " + description);

System.out.println("🔥 Priority: " + priority.getDisplayName());

System.out.println("📅 Created: " + newTask.getCreatedAt().format(DATE\_FORMAT));

}

private static void viewTasks() {

System.out.println("\n" + "=".repeat(40));

System.out.println("📋 ALL TASKS");

System.out.println("=".repeat(40));

if (tasks.isEmpty()) {

System.out.println("📭 No tasks found. Add some tasks to get started!");

return;

}

for (int i = 0; i < tasks.size(); i++) {

Task task = tasks.get(i);

String status = task.isCompleted() ? "✅" : "⏳";

String priority = task.getPriority().getEmoji();

System.out.println((i + 1) + ". " + status + " " + priority + " " + task.getDescription());

System.out.println(" 📅 " + task.getCreatedAt().format(DATE\_FORMAT) +

(task.isCompleted() ? " | Completed: " + task.getCompletedAt().format(DATE\_FORMAT) : ""));

}

System.out.println("\n📊 Summary: " + getTotalTasks() + " total, " +

getCompletedTasks() + " completed, " + getPendingTasks() + " pending");

}

private static void editTask() {

if (tasks.isEmpty()) {

System.out.println("📭 No tasks to edit!");

return;

}

viewTasks();

System.out.print("\n✏️ Enter task number to edit: ");

try {

int taskNum = Integer.parseInt(scanner.nextLine().trim()) - 1;

if (taskNum < 0 || taskNum >= tasks.size()) {

System.out.println("❌ Invalid task number!");

return;

}

Task task = tasks.get(taskNum);

System.out.println("\n📝 Current task: " + task.getDescription());

System.out.print("Enter new description (press Enter to keep current): ");

String newDescription = scanner.nextLine().trim();

if (!newDescription.isEmpty()) {

task.setDescription(newDescription);

System.out.println("✅ Task updated successfully!");

} else {

System.out.println("ℹ️ Task description unchanged.");

}

} catch (NumberFormatException e) {

System.out.println("❌ Invalid input! Please enter a valid number.");

}

}

private static void deleteTask() {

if (tasks.isEmpty()) {

System.out.println("📭 No tasks to delete!");

return;

}

viewTasks();

System.out.print("\n🗑️ Enter task number to delete: ");

try {

int taskNum = Integer.parseInt(scanner.nextLine().trim()) - 1;

if (taskNum < 0 || taskNum >= tasks.size()) {

System.out.println("❌ Invalid task number!");

return;

}

Task task = tasks.get(taskNum);

System.out.print("Are you sure you want to delete '" + task.getDescription() + "'? (y/N): ");

String confirm = scanner.nextLine().trim().toLowerCase();

if (confirm.equals("y") || confirm.equals("yes")) {

tasks.remove(taskNum);

System.out.println("✅ Task deleted successfully!");

} else {

System.out.println("ℹ️ Task deletion cancelled.");

}

} catch (NumberFormatException e) {

System.out.println("❌ Invalid input! Please enter a valid number.");

}

}

private static void markTaskComplete() {

if (tasks.isEmpty()) {

System.out.println("📭 No tasks available!");

return;

}

// Show only pending tasks

List<Task> pendingTasks = new ArrayList<>();

System.out.println("\n" + "=".repeat(30));

System.out.println("⏳ PENDING TASKS");

System.out.println("=".repeat(30));

for (int i = 0; i < tasks.size(); i++) {

Task task = tasks.get(i);

if (!task.isCompleted()) {

pendingTasks.add(task);

System.out.println((pendingTasks.size()) + ". " + task.getPriority().getEmoji() + " " + task.getDescription());

}

}

if (pendingTasks.isEmpty()) {

System.out.println("🎉 All tasks completed! Great job!");

return;

}

System.out.print("\n✅ Enter task number to mark as complete: ");

try {

int taskNum = Integer.parseInt(scanner.nextLine().trim()) - 1;

if (taskNum < 0 || taskNum >= pendingTasks.size()) {

System.out.println("❌ Invalid task number!");

return;

}

Task task = pendingTasks.get(taskNum);

task.markComplete();

System.out.println("🎉 Task completed successfully!");

System.out.println("✅ " + task.getDescription());

} catch (NumberFormatException e) {

System.out.println("❌ Invalid input! Please enter a valid number.");

}

}

private static void viewTasksByPriority() {

System.out.println("\n" + "=".repeat(40));

System.out.println("🔥 TASKS BY PRIORITY");

System.out.println("=".repeat(40));

if (tasks.isEmpty()) {

System.out.println("📭 No tasks found!");

return;

}

// Group tasks by priority

Map<Priority, List<Task>> tasksByPriority = new HashMap<>();

for (Priority p : Priority.values()) {

tasksByPriority.put(p, new ArrayList<>());

}

for (Task task : tasks) {

tasksByPriority.get(task.getPriority()).add(task);

}

// Display tasks by priority order

Priority[] priorities = {Priority.HIGH, Priority.MEDIUM, Priority.LOW};

for (Priority priority : priorities) {

List<Task> priorityTasks = tasksByPriority.get(priority);

if (!priorityTasks.isEmpty()) {

System.out.println("\n" + priority.getDisplayName() + " " + priority.getEmoji());

System.out.println("-".repeat(20));

for (Task task : priorityTasks) {

String status = task.isCompleted() ? "✅" : "⏳";

System.out.println(" " + status + " " + task.getDescription());

}

}

}

}

private static void viewCompletedTasks() {

System.out.println("\n" + "=".repeat(40));

System.out.println("✅ COMPLETED TASKS");

System.out.println("=".repeat(40));

List<Task> completedTasks = tasks.stream()

.filter(Task::isCompleted)

.toList();

if (completedTasks.isEmpty()) {

System.out.println("📭 No completed tasks yet. Keep working!");

return;

}

for (int i = 0; i < completedTasks.size(); i++) {

Task task = completedTasks.get(i);

System.out.println((i + 1) + ". ✅ " + task.getPriority().getEmoji() + " " + task.getDescription());

System.out.println(" 📅 Completed: " + task.getCompletedAt().format(DATE\_FORMAT));

}

System.out.println("\n🎉 Total completed: " + completedTasks.size());

}

private static void clearAllTasks() {

if (tasks.isEmpty()) {

System.out.println("📭 No tasks to clear!");

return;

}

System.out.print("🧹 Are you sure you want to clear ALL tasks? This cannot be undone! (y/N): ");

String confirm = scanner.nextLine().trim().toLowerCase();

if (confirm.equals("y") || confirm.equals("yes")) {

tasks.clear();

System.out.println("✅ All tasks cleared successfully!");

} else {

System.out.println("ℹ️ Clear operation cancelled.");

}

}

private static void saveTasks() {

try (PrintWriter writer = new PrintWriter(new FileWriter(DATA\_FILE))) {

for (Task task : tasks) {

writer.println(task.toFileString());

}

} catch (IOException e) {

System.out.println("❌ Error saving tasks: " + e.getMessage());

}

}

private static void loadTasks() {

File file = new File(DATA\_FILE);

if (!file.exists()) {

return;

}

try (BufferedReader reader = new BufferedReader(new FileReader(DATA\_FILE))) {

String line;

while ((line = reader.readLine()) != null) {

Task task = Task.fromFileString(line);

if (task != null) {

tasks.add(task);

}

}

} catch (IOException e) {

System.out.println("❌ Error loading tasks: " + e.getMessage());

}

}

private static int getTotalTasks() {

return tasks.size();

}

private static long getCompletedTasks() {

return tasks.stream().filter(Task::isCompleted).count();

}

private static long getPendingTasks() {

return tasks.stream().filter(task -> !task.isCompleted()).count();

}

}

// Priority enum

enum Priority {

HIGH("🔴 HIGH", "🔴"),

MEDIUM("🟡 MEDIUM", "🟡"),

LOW("🟢 LOW", "🟢");

private final String displayName;

private final String emoji;

Priority(String displayName, String emoji) {

this.displayName = displayName;

this.emoji = emoji;

}

public String getDisplayName() {

return displayName;

}

public String getEmoji() {

return emoji;

}

}

// Task class

class Task {

private String description;

private Priority priority;

private boolean completed;

private LocalDateTime createdAt;

private LocalDateTime completedAt;

public Task(String description, Priority priority) {

this.description = description;

this.priority = priority;

this.completed = false;

this.createdAt = LocalDateTime.now();

}

// Constructor for loading from file

public Task(String description, Priority priority, boolean completed,

LocalDateTime createdAt, LocalDateTime completedAt) {

this.description = description;

this.priority = priority;

this.completed = completed;

this.createdAt = createdAt;

this.completedAt = completedAt;

}

public void markComplete() {

this.completed = true;

this.completedAt = LocalDateTime.now();

}

public String toFileString() {

return String.join("|",

description,

priority.name(),

String.valueOf(completed),

createdAt.toString(),

completedAt != null ? completedAt.toString() : "null"

);

}

public static Task fromFileString(String fileString) {

try {

String[] parts = fileString.split("\\|");

if (parts.length != 5) return null;

String description = parts[0];

Priority priority = Priority.valueOf(parts[1]);

boolean completed = Boolean.parseBoolean(parts[2]);

LocalDateTime createdAt = LocalDateTime.parse(parts[3]);

LocalDateTime completedAt = parts[4].equals("null") ? null : LocalDateTime.parse(parts[4]);

return new Task(description, priority, completed, createdAt, completedAt);

} catch (Exception e) {

return null;

}

}

// Getters and setters

public String getDescription() { return description; }

public void setDescription(String description) { this.description = description; }

public Priority getPriority() { return priority; }

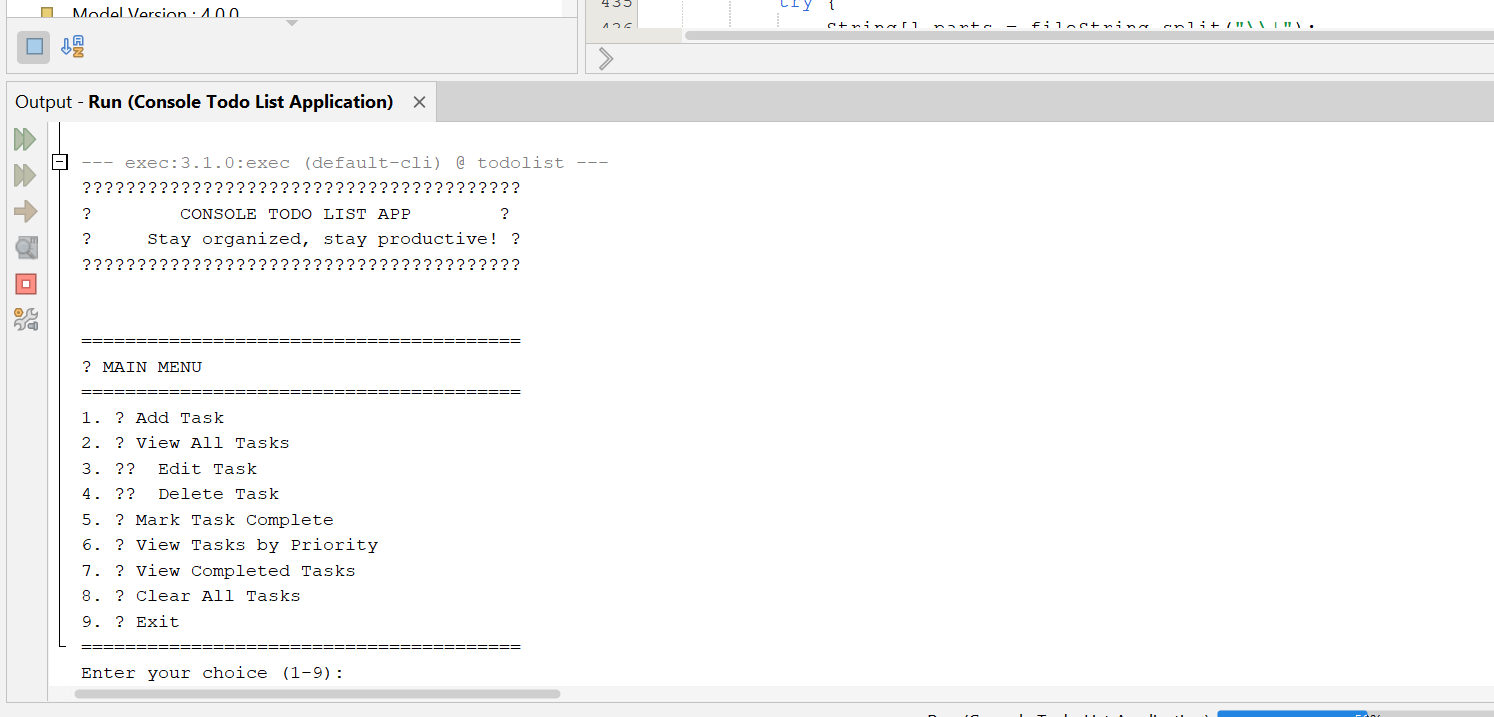
public boolean isCompleted() { return completed; }

public LocalDateTime getCreatedAt() { return createdAt; }

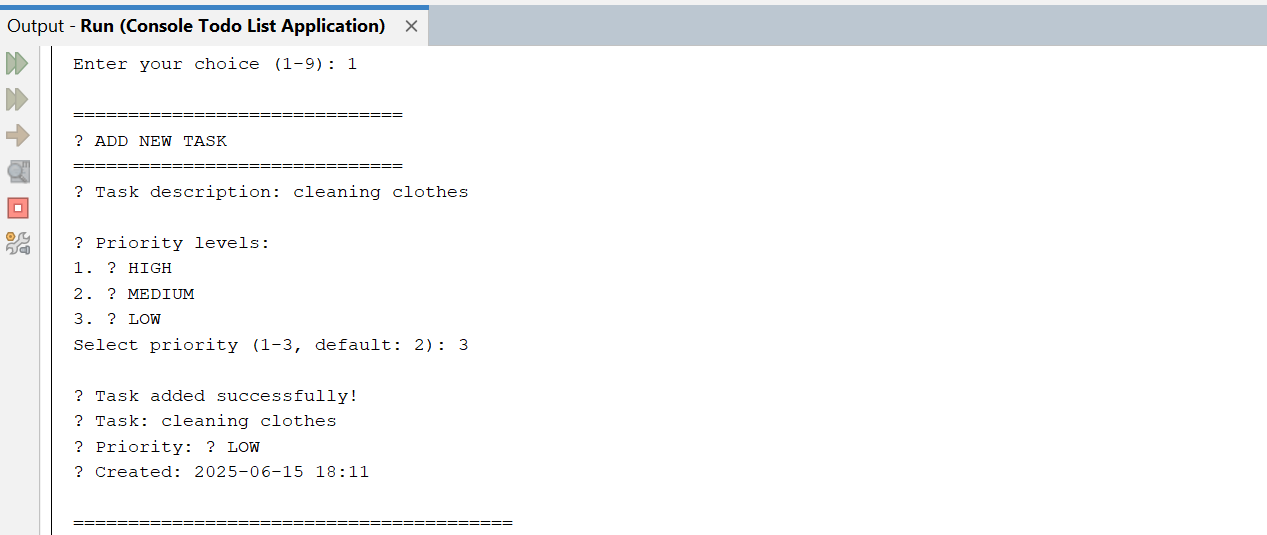
public LocalDateTime getCompletedAt() { return completedAt; }

}

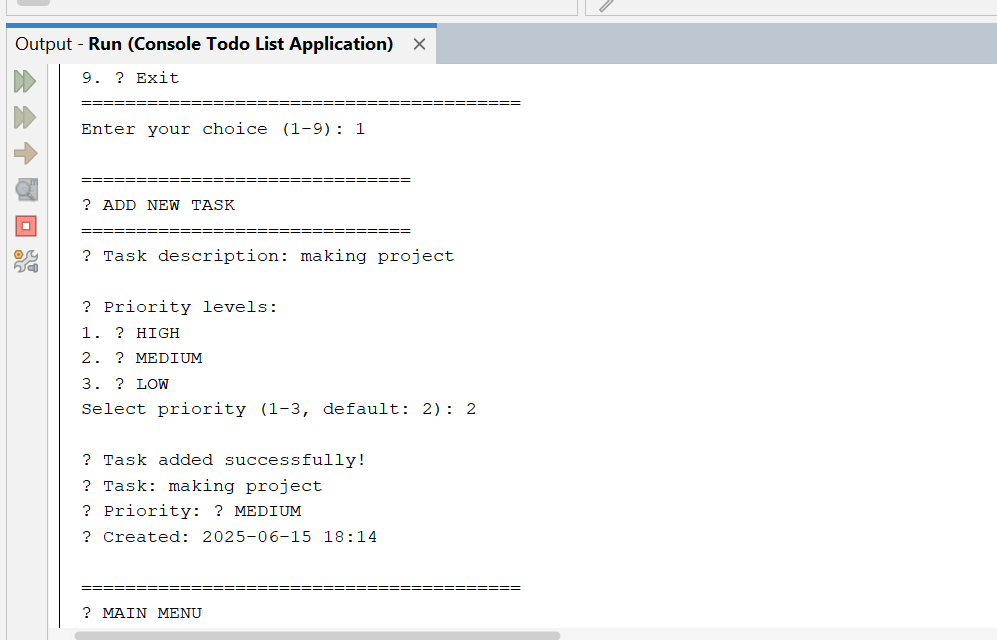
**Output source**



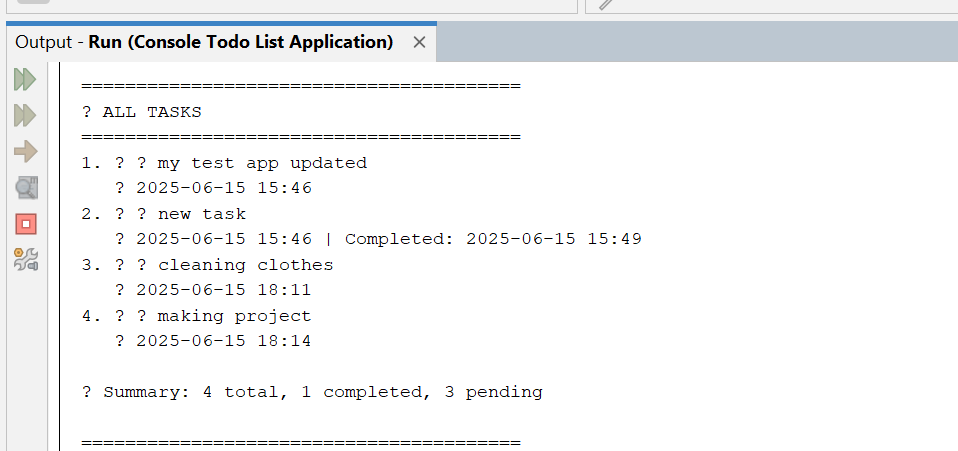
**Now adding a task:**



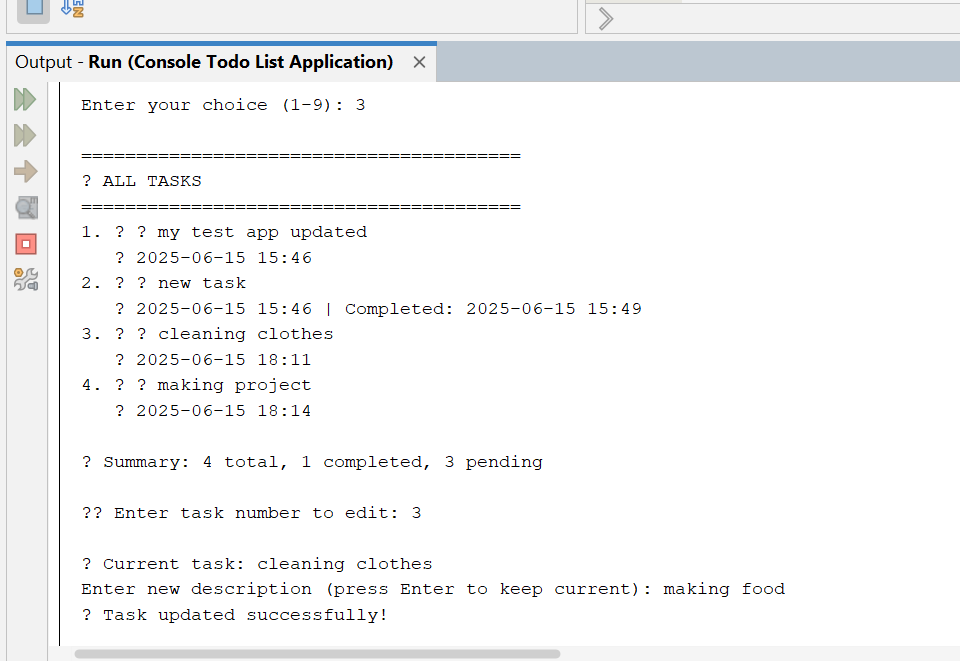
**Now add another task:**



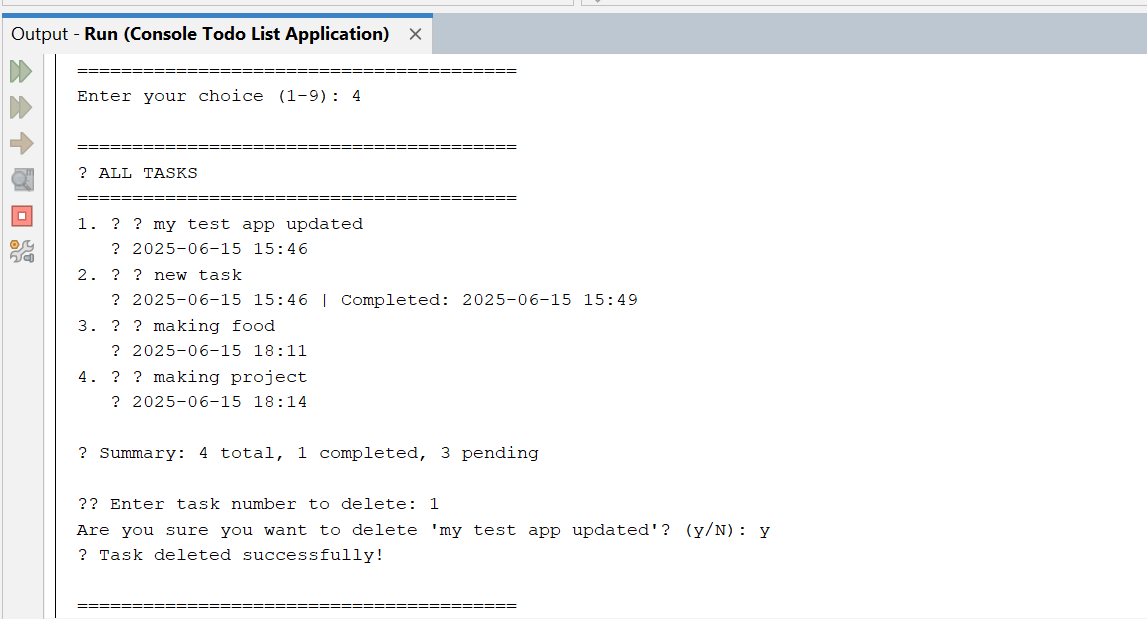
**If I want to view all task:**



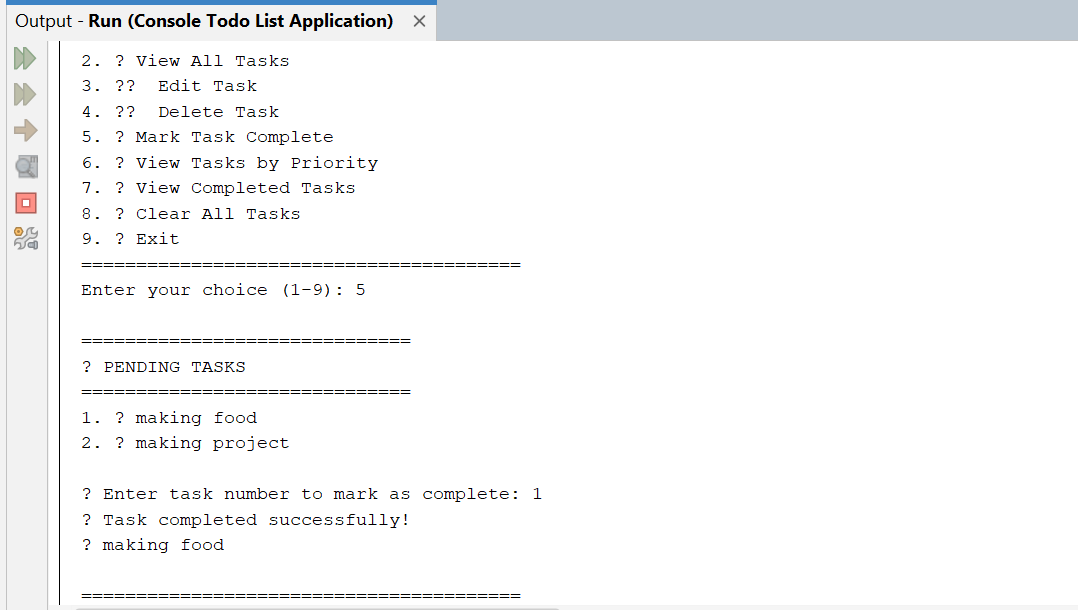
**If I want to edit a task:**



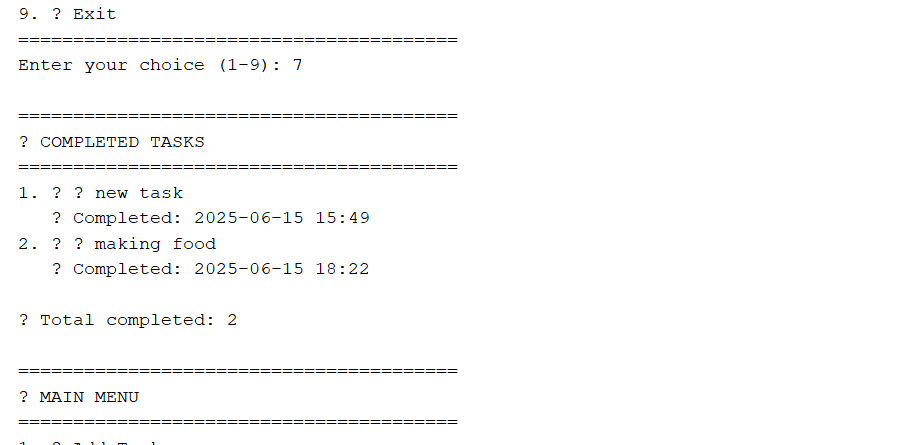
**If I want to delete a task:**



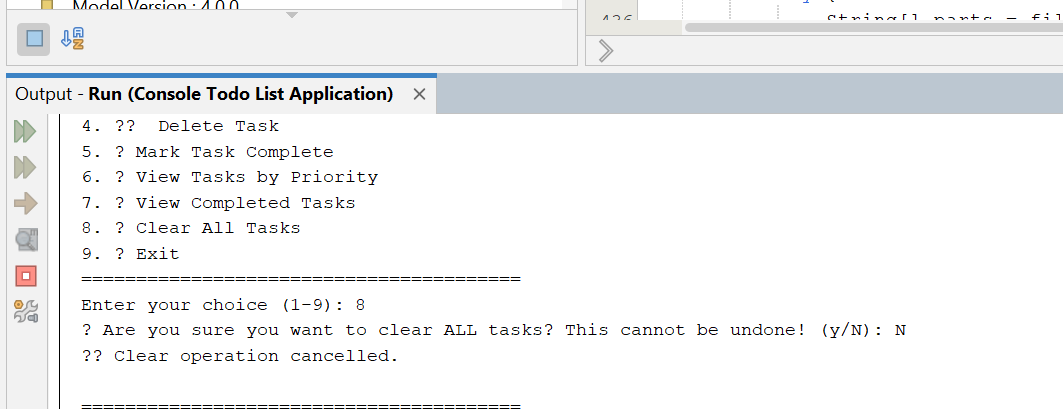
**If I want to mark my task completed:**



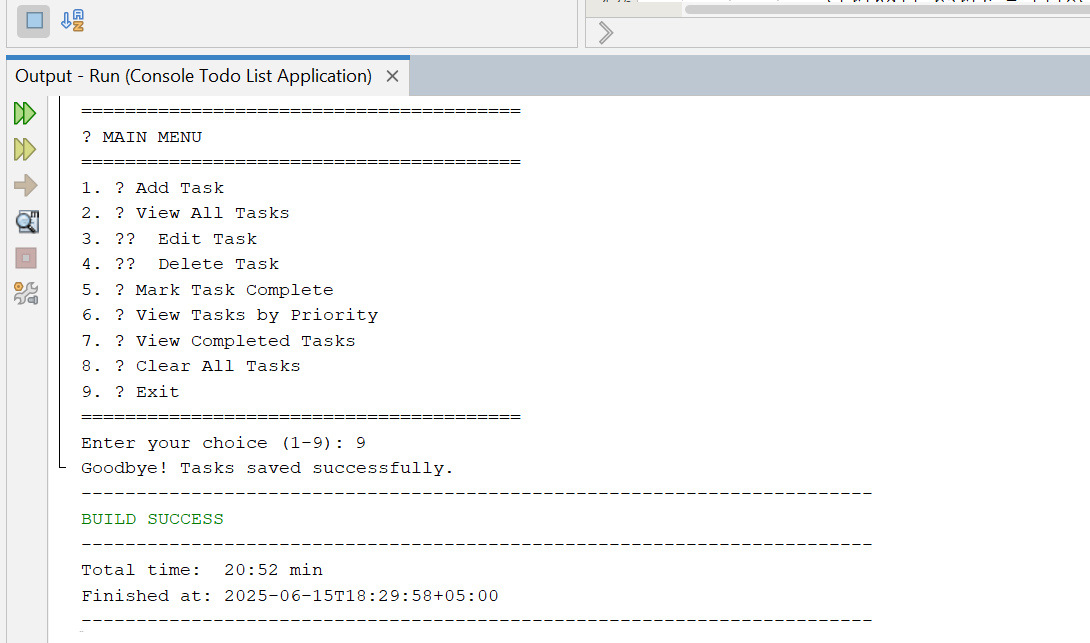
**If I want to view my task completed:**



**If I want to clear my all task then click the 7 and also click the y button which means yes aur if you do not want to clear it click the n button which means no:**



**If I want a exit then press the 9:**

**Dear Sir,**

**I hope this message finds you well. I am writing to inform you that the project [TODO LIST APP] has been successfully completed. I have completed the project as instructed, and it has been developed using NetBeans. Thank you**

**Best regards,  
ABDUL MUQSIT**

**KASHIF ALI**

**ABDUL MAJID**

**ARSALAN**