CHAPTER -1

Introduction:

Hostel Management System (Python + Tkinter)

This project is a GUI-based Hostel Management System built using Python and the Tkinter library.

It helps hostel administrators manage rooms, fees, and complaints for both boys' and girls' hostels.

Main Features

1. Room Allocation

- Admin can add rooms (hostel type, block, room number, capacity).
- Students can be assigned to rooms until the room capacity is full.
- A room table displays all rooms with their occupants.

2. Fee Management

- Each student has a fee account linked to their hostel/block.
- Hostel fee = ₹75,000, payable in up to 3 installments.
- Validations ensure:
- Maximum 3 installments allowed.
- Last installment must cover the remaining balance in full.
- No excess payment beyond ₹75,000.
- Fee table shows installments, paid amount, and due status.

3. Complaint Handling

- Students can log complaints against hostel services.
- Admin can mark complaints as Resolved or Pending.
- Complaints are color-coded for easy tracking.

4. Boys' & Girls' Hostel Summary Tables

Separate tabs show complete details of each hostel:

- Room info
- Occupants
- Fees (paid, installments, due status)

- Complaints (status: resolved/pending)
- Automatically refreshed for live updates.
- **GUI Features**
- Built with Tkinter Notebook tabs for navigation.
- Uses Treeview tables to display data.
- Color-coded rows:
  Green = No Due / Resolved
  Red/Pink = Due / Pending
- User-friendly forms with dropdowns and entry fields.

Objective:

The main objectives of this project are:

1. Room Allocation Management
    - Add hostel rooms with block, number, and capacity details.
    - Assign students to available rooms without exceeding capacity.
    - Maintain updated records of all occupants in each room.
2. Fee Management
    - Record hostel fee payments for each student.
    - Allow payments in up to three instalments.
    - Ensure the final instalment clears the full fee of ₹75,000.
    - Display fee status (Paid, Due, No Due) clearly.
3. Complaint Handling
    - Enable students to log complaints related to hostel issues.
    - Track complaint status (Pending or Resolved).
    - Allow administrators to mark complaints as resolved.

4.Separate Hostel Records

- Maintain separate tables for Boys Hostel and Girls Hostel.
- Display room, occupant, fee, and complaint details in each hostel's tab.

5.Automation & Transparency

- Reduce manual errors by automating record keeping.
- Provide real-time updates of fees, room allocation, and complaints.
- Color-coded status indicators (green = no due/resolved, red = due, yellow = pending) for clarity.

6.User-Friendly Interface

- Provide a simple GUI using Tkinter for hostel administrators.
- Organize modules into tabs: Room Allocation, Fee Management, Complaint Handling, and Hostel Records.

7.Data Integration

- Connect room allocation, fees, and complaints together for each student.
- Allow quick retrieval of complete hostel information in one place.

Methodologies

Methodologies of the Hostel Management System

1. Object-Oriented Programming (OOP) Approach

The system is designed using classes and objects to model real-world hostel entities:

Room → Manages room capacity and student assignments.

Fee → Handles fee payments, installments, and dues.

Complaint → Stores and manages student complaints.

HostelManagementApp → Acts as the controller that integrates all modules and provides the GUI.

This OOP approach ensures modularity, reusability, and scalability.

2. Graphical User Interface (GUI) Design

Implemented using Tkinter (Python's standard GUI library).

A tab-based interface (ttk.Notebook) is used for easy navigation across different modules:

- Room Allocation
- Fee Management
- Complaint Handling
- Boys Hostel Table
- Girls Hostel Table
- Treeview tables are used for displaying structured hostel data with headings and row formatting.

3. Data Management Methodology

Data is stored in in-memory Python data structures (dict and list) for fast access.

Example:

- self.rooms → Dictionary storing all rooms by keys (hostel_type, block, room_no).
- self.fees → List of Fee objects linked to students.
- self.complaints → List of Complaint objects logged by students.
- This ensures that operations (add, update, delete) are efficient and consistent across tabs.

4. Validation Logic and Business

The system enforces hostel rules using clear validation checks:

- A room cannot exceed its capacity.
- Fees must be paid within 3 installments only.
- The final installment must cover the full remaining balance.
- Complaints can only be resolved after administrator confirmation.

These checks ensure accuracy and integrity of hostel records.

5. Real-Time Updates

- The Boys' and Girls' hostel summary tables refresh automatically using Tkinter's after() method.
- Whenever new rooms, fees, or complaints are added, all related tables are updated in real-time.
- This provides live monitoring of hostel activities.

6. User-Friendly Interaction

- Drop-downs (ttk.Combobox) for selecting hostel type and block.
- Error handling via message boxes (messagebox.showerror, showinfo) to guide the user.
- Color-coded table rows for better visibility:
  Green = No Due / Resolved Complaint
  Red/Pink = Due / Pending Complaint

Code implementation

```
import tkinter as tk
from tkinter import ttk, messagebox
class Room:
    def _init_(self, hostel_type, block, room_no, capacity):
        self.hostel_type = hostel_type
        self.block = block
        self.room_no = room_no
        self.capacity = capacity
        self.occupants = []
    def assign_student(self, student_name):
        if len(self.occupants) < self.capacity:
            self.occupants.append(student_name)
            return True
        return False
class Fee:
    def _init_(self, student_name, hostel_type, block):
        self.student_name = student_name
        self.hostel_type = hostel_type
        self.block = block
        self.paid_amount = 0
        self.installments = 0
    def pay(self, amount):
        self.paid_amount += amount
        self.installments += 1
    def is_no_due(self):
        return self.paid_amount >= 75000
```

```
    def can_pay(self):
        return self.installments < 3 and self.paid_amount < 7500
    def must_pay_full(self):
        return self.installments == 2 and self.paid_amount < 75000
class Complaint:
    def _init_(self, student_name, complaint, hostel_type):
        self.student_name = student_name
        self.complaint = complaint
        self.hostel_type = hostel_type
        self.resolved = False
    def resolve(self, solved):
        self.resolved = solved
class HostelManagementApp:
    def _init_(self, root):
        self.root = root
        self.root.title("Hostel Management System")
        self.rooms = {}
        self.fees = []
        self.complaints = []
       self.main_frame = tk.Frame(self.root, bg="#263859", bd=8, relief="ridge")
        self.main_frame.pack(fill="both", expand=True, padx=18, pady=18)
        self.create_widgets()
    def create_widgets(self):
        style = ttk.Style()
        style.theme_use('clam')
        style.configure('TNotebook.Tab', background='#e6e6fa', font=('Arial', 12, 'bold'))
        style.configure('TButton', font=('Arial', 10, 'bold'))
        style.configure('Treeview.Heading', font=('Arial', 11, 'bold'))
        style.configure('Treeview', rowheight=24, font=('Arial', 10))
```

```
        tab_control = ttk.Notebook(self.main_frame)
        self.tab_room = tk.Frame(tab_control, bg="#a8d8ea", bd=3, relief="groove")
        self.tab_fee = tk.Frame(tab_control, bg="#b8f2e6", bd=3, relief="groove")
        self.tab_complaint = tk.Frame(tab_control, bg="#f9d5a2", bd=3, relief="groove")
        self.tab_boys = tk.Frame(tab_control, bg="#eaeaea", bd=3, relief="groove")
        self.tab_girls = tk.Frame(tab_control, bg="#ffe6fa", bd=3, relief="groove")
        tab_control.add(self.tab_room, text='Room Allocation')
        tab_control.add(self.tab_fee, text='Fee Management')
        tab_control.add(self.tab_complaint, text='Complaint Handling')
        tab_control.add(self.tab_boys, text='Boys Hostel Table')
        tab_control.add(self.tab_girls, text='Girls Hostel Table')
        tab_control.pack(expand=1, fill="both")
        self.setup_room_tab()
        self.setup_fee_tab()
        self.setup_complaint_tab()
        self.setup_boys_girls_tabs()
    # Room Allocation Tab
    def setup_room_tab(self):
        frame = self.tab_room

        tk.Label(frame, text="Hostel Type:", bg="#a8d8ea", font=('Arial', 11)).grid(row=0, column=0, padx=10, pady=8, sticky="e")
        self.hostel_type_var = tk.StringVar(value="Boys")
        hostel_type_cb = ttk.Combobox(frame, textvariable=self.hostel_type_var, values=["Boys", "Girls"], state="readonly", width=15)
        hostel_type_cb.grid(row=0, column=1, padx=8, pady=8, sticky="w")

        tk.Label(frame, text="Block:", bg="#a8d8ea", font=('Arial', 11)).grid(row=0, column=2, padx=10, pady=8, sticky="e")
```

```
        self.block_var = tk.StringVar(value="A")
        block_cb = ttk.Combobox(frame, textvariable=self.block_var, values=["A", "B"], state="readonly", width=5)
        block_cb.grid(row=0, column=3, padx=8, pady=8, sticky="w")

        tk.Label(frame, text="Room Number:", bg="#a8d8ea", font=('Arial', 11)).grid(row=1, column=0, padx=10, pady=8, sticky="e")
        tk.Label(frame, text="Capacity:", bg="#a8d8ea", font=('Arial', 11)).grid(row=1, column=2, padx=10, pady=8, sticky="e")
        self.room_no_var = tk.StringVar()
        self.capacity_var = tk.StringVar()
        tk.Entry(frame, textvariable=self.room_no_var, font=('Arial', 11), bd=2).grid(row=1, column=1, padx=8, pady=8)
        tk.Entry(frame, textvariable=self.capacity_var, font=('Arial', 11), bd=2).grid(row=1, column=3, padx=8, pady=8)

        tk.Label(frame, text="Student Name:", bg="#a8d8ea", font=('Arial', 11)).grid(row=2, column=0, padx=10, pady=8, sticky="e")
        self.student_name_var = tk.StringVar()
        tk.Entry(frame, textvariable=self.student_name_var, font=('Arial', 11), bd=2).grid(row=2, column=1, padx=8, pady=8)

        tk.Button(frame, text="Add Room", bg="#3e8ed0", fg="white", command=self.add_room).grid(row=1, column=4, padx=10)
        tk.Button(frame, text="Assign Room", bg="#4caf50", fg="white", command=self.assign_room).grid(row=2, column=4, padx=10)

        tk.Label(frame, text="Room Table:", bg="#a8d8ea", font=('Arial', 12, 'bold')).grid(row=3, column=0, columnspan=5, pady=(15, 5))

        columns = ("hostel_type", "block", "room_no", "capacity", "occupants")
        self.room_tree = ttk.Treeview(frame, columns=columns, show="headings", height=7)
```

```
        for col, text in zip(columns, ["Hostel", "Block", "Room No", "Capacity", "Occupants"]):
            self.room_tree.heading(col, text=text)
            self.room_tree.column(col, width=250)
        self.room_tree.grid(row=4, column=0, columnspan=5, padx=8, pady=(2,10),
sticky="nsew")

        self.refresh_room_table()

    def add_room(self):
        hostel_type = self.hostel_type_var.get()
        block = self.block_var.get()
        room_no = self.room_no_var.get().strip()
        try:
            capacity = int(self.capacity_var.get())
        except ValueError:
            messagebox.showerror("Error", "Capacity must be an integer.")
            return
        key = (hostel_type, block, room_no)
        if key in self.rooms:
            messagebox.showerror("Error", "Room already exists.")
            return
        self.rooms[key] = Room(hostel_type, block, room_no, capacity)
        self.refresh_room_table()
        self.room_no_var.set("")
        self.capacity_var.set("")

    def assign_room(self):
        hostel_type = self.hostel_type_var.get()
        block = self.block_var.get()
```

```
        room_no = self.room_no_var.get().strip()
        student_name = self.student_name_var.get().strip()
        key = (hostel_type, block, room_no)
        if key not in self.rooms:
            messagebox.showerror("Error", "Room does not exist.")
            return
        if student_name == "":
            messagebox.showerror("Error", "Enter student name.")
            return
        room = self.rooms[key]
        if room.assign_student(student_name):
            fee_obj = self.find_fee(student_name, hostel_type, block)
            if not fee_obj:
                self.fees.append(Fee(student_name, hostel_type, block))
            messagebox.showinfo("Success", f"{student_name} assigned to {hostel_type} Hostel
Block {block}, Room {room_no}.")
        else:
            messagebox.showerror("Error", "Room is full.")
        self.refresh_room_table()
        self.refresh_fee_table()
        self.student_name_var.set("")


    def refresh_room_table(self):
        for i in self.room_tree.get_children():
            self.room_tree.delete(i)
        for (hostel_type, block, room_no), room in self.rooms.items():
            self.room_tree.insert("", "end", values=(
                hostel_type, block, room_no, room.capacity, ", ".join(room.occupants)
            ))
```

```
    def find_fee(self, student_name, hostel_type, block):
        for fee in self.fees:
            if fee.student_name == student_name and fee.hostel_type == hostel_type and fee.block == block:
                return fee
        return None

    # Fee Management Tab
    def setup_fee_tab(self):
        frame = self.tab_fee

        tk.Label(frame, text="Hostel Type:", bg="#b8f2e6", font=('Arial', 11)).grid(row=0, column=0, padx=10, pady=8, sticky="e")
        self.fee_hostel_type_var = tk.StringVar(value="Boys")
        hostel_type_cb = ttk.Combobox(frame, textvariable=self.fee_hostel_type_var, values=["Boys", "Girls"], state="readonly", width=15)
        hostel_type_cb.grid(row=0, column=1, padx=8, pady=8, sticky="w")

        tk.Label(frame, text="Block:", bg="#b8f2e6", font=('Arial', 11)).grid(row=0, column=2, padx=10, pady=8, sticky="e")
        self.fee_block_var = tk.StringVar(value="A")
        block_cb = ttk.Combobox(frame, textvariable=self.fee_block_var, values=["A", "B"], state="readonly", width=5)
        block_cb.grid(row=0, column=3, padx=8, pady=8, sticky="w")

        tk.Label(frame, text="Student Name:", bg="#b8f2e6", font=('Arial', 11)).grid(row=1, column=0, padx=10, pady=8, sticky="e")
        tk.Label(frame, text="Pay Amount:", bg="#b8f2e6", font=('Arial', 11)).grid(row=2, column=0, padx=10, pady=8, sticky="e")
```

```
        self.fee_student_var = tk.StringVar()
        self.fee_amount_var = tk.StringVar()

        tk.Entry(frame, textvariable=self.fee_student_var, font=('Arial', 11), bd=2).grid(row=1, column=1, padx=8, pady=8)
        tk.Entry(frame, textvariable=self.fee_amount_var, font=('Arial', 11), bd=2).grid(row=2, column=1, padx=8, pady=8)
        tk.Button(frame, text="Pay Fee", bg="#4caf50", fg="white", command=self.pay_fee).grid(row=2, column=2, padx=10)

        tk.Label(frame, text="Fee Table:", bg="#b8f2e6", font=('Arial', 12, 'bold')).grid(row=3, column=0, columnspan=3, pady=(15, 5))

        columns = ("student_name", "hostel_type", "block", "amount", "installments", "status")
        self.fee_tree = ttk.Treeview(frame, columns=columns, show="headings", height=7)
        for col, text in zip(columns, ["Student", "Hostel", "Block", "Paid Amount", "Installments", "Status"]):
            self.fee_tree.heading(col, text=text)
            self.fee_tree.column(col, width=100)
        self.fee_tree.grid(row=4, column=0, columnspan=4, padx=8, pady=(2,10), sticky="nsew")

        self.refresh_fee_table()

    def pay_fee(self):
        student_name = self.fee_student_var.get().strip()
        hostel_type = self.fee_hostel_type_var.get()
        block = self.fee_block_var.get()
        try:
            amount = int(self.fee_amount_var.get())
        except ValueError:
```

```
            messagebox.showerror("Error", "Please enter a valid amount.")
            return

        fee = self.find_fee(student_name, hostel_type, block)
        if not fee:
            messagebox.showerror("Error", "Student not found in this hostel/block.")
            return

        if fee.installments >= 3 and not fee.is_no_due():
            messagebox.showerror("Error", "Maximum 3 installments allowed. You cannot pay more.")
            return

        # If this is the third (last) installment and not paid in full, force to pay the remaining full amount
        if fee.must_pay_full():
            remaining = 75000 - fee.paid_amount
            if amount != remaining:
                messagebox.showerror("Error", f"Final installment must complete full fee. Please pay {remaining}.")
                return

        if fee.is_no_due():
            messagebox.showinfo("Info", "No due. Fee already paid in full.")
            return

        if fee.paid_amount + amount > 75000:
            messagebox.showerror("Error", "Fee exceeds required amount (75000).")
            return
```

```
        fee.pay(amount)
        self.refresh_fee_table()
        self.fee_amount_var.set("")

    def refresh_fee_table(self):
        for i in self.fee_tree.get_children():
            self.fee_tree.delete(i)
        for fee in self.fees:
            status = "NO DUE" if fee.is_no_due() else f"DUE: {75000-fee.paid_amount}"
            row = (fee.student_name, fee.hostel_type, fee.block, fee.paid_amount,
fee.installments, status)
            iid = self.fee_tree.insert("", "end", values=row)
            if fee.is_no_due():
                self.fee_tree.item(iid, tags=('no_due',))
            else:
                self.fee_tree.item(iid, tags=('due',))
        self.fee_tree.tag_configure('no_due', background='#a1f0a1')
        self.fee_tree.tag_configure('due', background='#ffe6e6')

    # Complaint Handling Tab
    def setup_complaint_tab(self):
        frame = self.tab_complaint

        tk.Label(frame, text="Hostel Type:", bg="#f9d5a2", font=('Arial', 11)).grid(row=0,
column=0, padx=10, pady=8, sticky="e")
        self.complaint_hostel_type_var = tk.StringVar(value="Boys")
        hostel_type_cb = ttk.Combobox(frame, textvariable=self.complaint_hostel_type_var,
values=["Boys", "Girls"], state="readonly", width=15)
        hostel_type_cb.grid(row=0, column=1, padx=8, pady=8, sticky="w")
```

```
tk.Label(frame, text="Student Name:", bg="#f9d5a2", font=('Arial', 11)).grid(row=1, column=0, padx=10, pady=8, sticky="e")

tk.Label(frame, text="Complaint:", bg="#f9d5a2", font=('Arial', 11)).grid(row=2, column=0, padx=10, pady=8, sticky="e")


self.complaint_student_var = tk.StringVar()

self.complaint_text_var = tk.StringVar()


tk.Entry(frame, textvariable=self.complaint_student_var, font=('Arial', 11), bd=2).grid(row=1, column=1, padx=8, pady=8)

tk.Entry(frame, textvariable=self.complaint_text_var, width=30, font=('Arial', 11), bd=2).grid(row=2, column=1, padx=8, pady=8)


tk.Button(frame, text="Log Complaint", bg="#f57c00", fg="white", command=self.log_complaint).grid(row=3, column=1, pady=8)


tk.Label(frame, text="Complaints Table:", bg="#f9d5a2", font=('Arial', 12, 'bold')).grid(row=4, column=0, columnspan=3, pady=(15, 5))


columns = ("student_name", "hostel_type", "complaint", "status")

self.complaint_tree = ttk.Treeview(frame, columns=columns, show="headings", height=7)

for col, text in zip(columns, ["Student", "Hostel", "Complaint", "Status"]):

    self.complaint_tree.heading(col, text=text)

    self.complaint_tree.column(col, width=300)

self.complaint_tree.grid(row=5, column=0, columnspan=3, padx=8, pady=(2,10), sticky="nsew")


tk.Button(frame, text="Resolve Selected", bg="#388e3c", fg="white", command=self.resolve_complaint).grid(row=6, column=1, pady=8)


self.refresh_complaint_table()
```

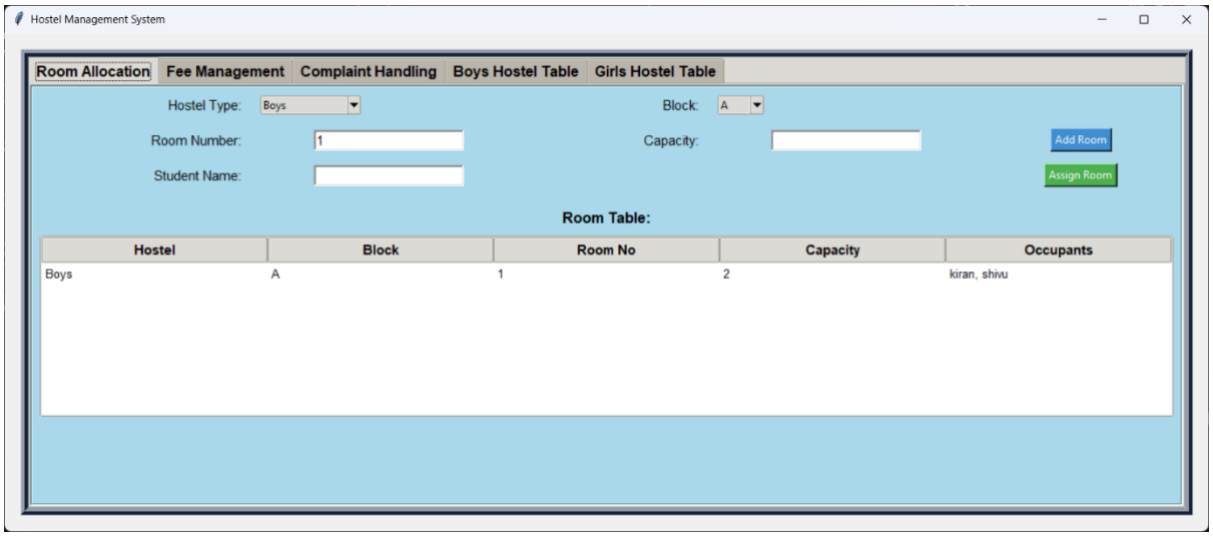
```
    def log_complaint(self):
        hostel_type = self.complaint_hostel_type_var.get()
        student = self.complaint_student_var.get().strip()
        text = self.complaint_text_var.get().strip()
        if not student or not text:
            messagebox.showerror("Error", "Fill in all fields.")
            return
        self.complaints.append(Complaint(student, text, hostel_type))
        self.refresh_complaint_table()
        self.complaint_student_var.set("")
        self.complaint_text_var.set("")

    def refresh_complaint_table(self):
        for i in self.complaint_tree.get_children():
            self.complaint_tree.delete(i)
        for idx, c in enumerate(self.complaints):
            status = "Resolved" if c.resolved else "Pending"
            iid = self.complaint_tree.insert("", "end", values=(c.student_name, c.hostel_type,
c.complaint, status))
            if c.resolved:
                self.complaint_tree.item(iid, tags=('resolved',))
            else:
                self.complaint_tree.item(iid, tags=('pending',))
        self.complaint_tree.tag_configure('resolved', background='#a1f0a1')
        self.complaint_tree.tag_configure('pending', background='#fff7e6')

    def resolve_complaint(self):
        selected = self.complaint_tree.selection()
```

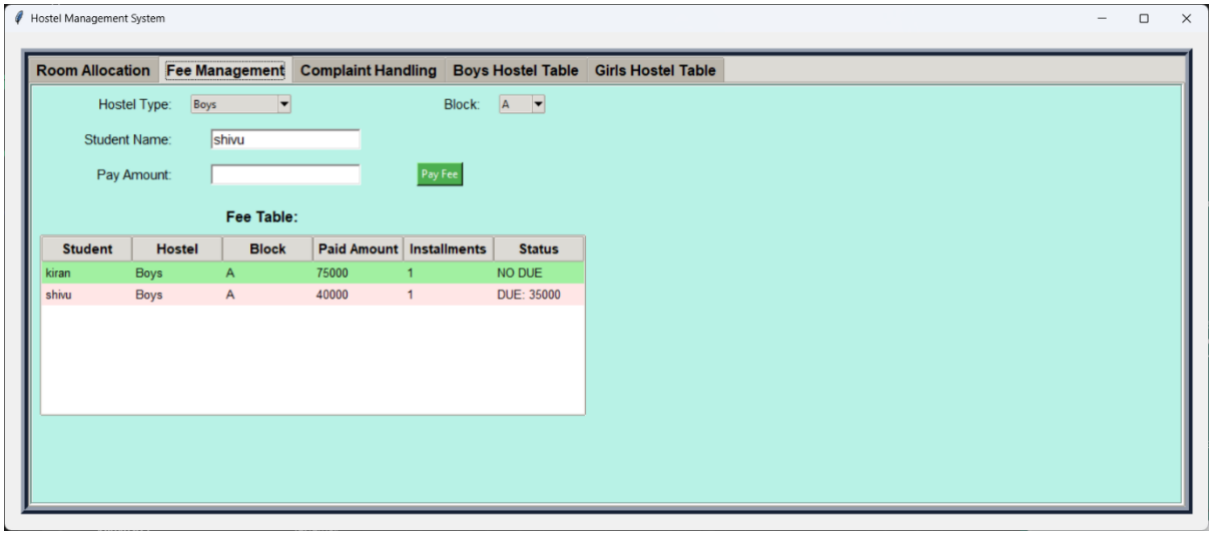
```
        if not selected:
            messagebox.showerror("Error", "Select a complaint to resolve.")
            return
        idx = self.complaint_tree.index(selected[0])
        c = self.complaints[idx]
        if c.resolved:
            messagebox.showinfo("Info", "Complaint already resolved.")
            return
        solved = messagebox.askyesno("Resolve Complaint", "Is the complaint solved?")
        c.resolve(solved)
        self.refresh_complaint_table()
        if solved:
            messagebox.showinfo("Resolved", "Complaint marked as resolved.")
        else:
            messagebox.showinfo("Not Resolved", "Complaint status remains pending.")

    # Boys/Girls Hostel Details Tabs
    def setup_boys_girls_tabs(self):
        # Boys Table
        tk.Label(self.tab_boys, text="Boys Hostel Details", bg="#eaeaea", font=('Arial', 14, 'bold')).pack(pady=10)
        columns = ("Type", "Block", "Room", "Capacity", "Occupants", "Fee", "Inst", "Due", "Complaint", "Status")
        self.boys_tree = ttk.Treeview(self.tab_boys, columns=columns, show="headings", height=15)
        for col in columns:
            self.boys_tree.heading(col, text=col)
            self.boys_tree.column(col, width=85)
        self.boys_tree.pack(padx=10, pady=10, fill="both", expand=True)
```

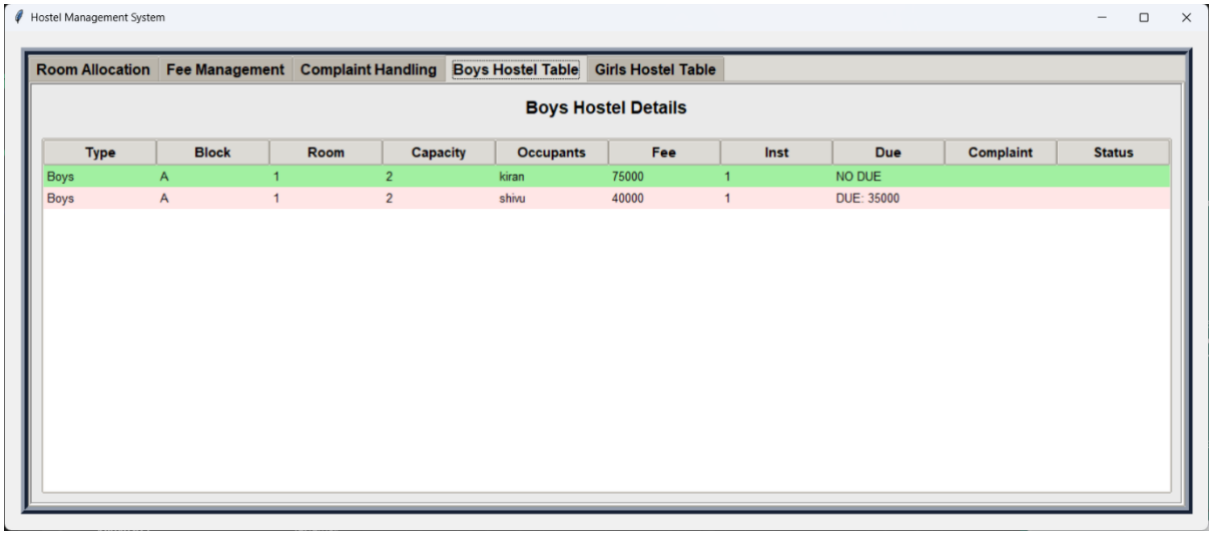
```
        # Girls Table
        tk.Label(self.tab_girls, text="Girls Hostel Details", bg="#ffe6fa", font=('Arial', 14, 'bold')).pack(pady=10)
        self.girls_tree = ttk.Treeview(self.tab_girls, columns=columns, show="headings", height=15)
        for col in columns:
            self.girls_tree.heading(col, text=col)
            self.girls_tree.column(col, width=85)
        self.girls_tree.pack(padx=10, pady=10, fill="both", expand=True)


        self.root.after(1000, self.refresh_boys_girls_tables)


    def refresh_boys_girls_tables(self):
        for tree, hostel in [(self.boys_tree, "Boys"), (self.girls_tree, "Girls")]:
            for i in tree.get_children():
                tree.delete(i)
            # Rooms
            for (hostel_type, block, room_no), room in self.rooms.items():
                if hostel_type != hostel:
                    continue
                occ = ", ".join(room.occupants)
                # For each occupant, find their fee/complaint info
                for student in room.occupants:
                    fee = self.find_fee(student, hostel_type, block)
                    fee_amt = fee.paid_amount if fee else ""
                    inst = fee.installments if fee else ""
                    due = "NO DUE" if fee and fee.is_no_due() else (f"DUE: {75000-fee.paid_amount}" if fee else "")
                    complaint = ""
                    cstatus = ""
```

```
                for c in self.complaints:
                    if c.student_name == student and c.hostel_type == hostel_type:
                        complaint = c.complaint
                        cstatus = "Resolved" if c.resolved else "Pending"
                row = (hostel_type, block, room_no, room.capacity, student, fee_amt, inst, due,
complaint, cstatus)
                tid = tree.insert("", "end", values=row)
                if due == "NO DUE":
                    tree.item(tid, tags=('no_due',))
                elif due:
                    tree.item(tid, tags=('due',))
                if cstatus == "Resolved":
                    tree.item(tid, tags=('resolved',))
                elif cstatus == "Pending":
                    tree.item(tid, tags=('pending',))
            tree.tag_configure('no_due', background='#a1f0a1')
            tree.tag_configure('due', background='#ffe6e6')
            tree.tag_configure('resolved', background='#a1f0a1')
            tree.tag_configure('pending', background='#fff7e6')
        self.root.after(1000, self.refresh_boys_girls_tables)

if _name_ == "_main_":
    root = tk.Tk()
    app = HostelManagementApp(root)
    root.mainloop()
```

Output


Hostel Management System
Room Allocation
Fee Management
Complaint Handling
Boys Hostel Table
Girls Hostel Table
Hostel Type:
Boys
Block:
A
Room Number:
1
Capacity:
Add Room
Student Name:
Assign Room
Room Table:
Hostel
Block
Room No
Capacity
Occupants
Boys
A
1
2
kiran, shivu


Hostel Management System
Room Allocation
Fee Management
Complaint Handling
Boys Hostel Table
Girls Hostel Table
Hostel Type:
Boys
Block:
A
Student Name:
shivu
Pay Amount:
Pay Fee
Fee Table:
Student
Hostel
Block
Paid Amount
Installments
Status
kiran
Boys
A
75000
1
NO DUE
shivu
Boys
A
40000
1
DUE: 35000


Hostel Management System
Room Allocation
Fee Management
Complaint Handling
Boys Hostel Table
Girls Hostel Table
Boys Hostel Details
Type
Block
Room
Capacity
Occupants
Fee
Inst
Due
Complaint
Status
Boys
A
1
2
kiran
75000
1
NO DUE
Boys
A
1
2
shivu
40000
1
DUE: 35000

Conclusion

The Hostel Management System developed using Python and Tkinter successfully provides a complete solution for managing hostel operations in a simple and efficient manner. The system integrates room allocation, fee management, and complaint handling into a single platform, reducing the need for manual record-keeping and minimizing errors.

By implementing features such as capacity-based room assignment, installment-based fee payment with validations, and complaint logging with resolution tracking, the project ensures transparency, accountability, and ease of administration. The use of tab-based navigation, structured Treeview tables, and color-coded indicators makes the system user-friendly and accessible even for non-technical users.

Overall, the project achieves its objective of streamlining hostel management, saving administrative time, and improving the student experience. With future enhancements such as data persistence (database/JSON storage), automated reports, and search functionality, the system can be further expanded into a robust real-world hostel management solution.

References

https://docs.python.org/3/library/tkinter.html

https://www.python.org/downloads/

https://www.w3schools.com/python/

https://www.geeksforgeeks.org/python/python-tkinter-tutorial/