Task 1 (Equipment.file)

```
public class Equipment {

    // Attributes
    private String assetId;
    private String name;
    private String brand;
    private boolean isAvailable;
    private String category;

    // Constructor
    public Equipment(String assetId, String name, String brand, boolean isAvailable, String
category) {
        this.assetId = assetId;
        this.name = name;
        this.brand = brand;
        this.isAvailable = isAvailable;
        this.category = category;
    }

    // Getters and setters
    public String getAssetId() {
        return assetId;
    }

    public void setAssetId(String assetId) {
```

```
    this.assetId = assetId;
}

public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

public String getBrand() {
    return brand;
}

public void setBrand(String brand) {
    this.brand = brand;
}

public boolean isAvailable() {
    return isAvailable;
}

public void setAvailable(boolean available) {
    isAvailable = available;
}
```

```
public String getCategory() {
   return category;
}

public void setCategory(String category) {
   this.category = category;
}

// toString method
@Override
public String toString() {
   return "Equipment{" +
         "assetId='" + assetId + '\'' +
         ", name='" + name + '\'' +
         ", brand='" + brand + '\'' +
         ", isAvailable=" + isAvailable +
         ", category='" + category + '\'' +
         '}';
}

// equals method (compare by assetId)
@Override
public boolean equals(Object obj) {
   if (this == obj) {
      return true; // same object
```

```
        }
        if (obj == null || getClass() != obj.getClass()) {
            return false;
        }

        Equipment other = (Equipment) obj;
        return assetId.equals(other.assetId);
    }
}
```

(Staffnenber.java)

```
public class StaffMember {

    // Attributes
    private int staffId;
    private String name;
    private String email;
    private Equipment[] assignedEquipment;
    private int equipmentCount;

    // Constructor
    public StaffMember(int staffId, String name, String email) {
        this.staffId = staffId;
        this.name = name;
        this.email = email;
        this.assignedEquipment = new Equipment[5]; // max 5 items
        this.equipmentCount = 0;
```

```
}

// Getters and setters
public int getStaffId() {
   return staffId;
}

public void setStaffId(int staffId) {
   this.staffId = staffId;
}

public String getName() {
   return name;
}

public void setName(String name) {
   this.name = name;
}

public String getEmail() {
   return email;
}

public void setEmail(String email) {
   this.email = email;
}
```

```
public Equipment[] getAssignedEquipment() {
   return assignedEquipment;
}

// Add equipment
public boolean addAssignedEquipment(Equipment equipment) {
   if (equipmentCount >= assignedEquipment.length) {
      return false; // array full
   }

   assignedEquipment[equipmentCount] = equipment;
   equipmentCount++;
   return true;
}

// Remove equipment by assetId
public boolean removeAssignedEquipment(String assetId) {
   for (int i = 0; i < equipmentCount; i++) {
      if (assignedEquipment[i].getAssetId().equals(assetId)) {

         // Shift remaining elements left
         for (int j = i; j < equipmentCount - 1; j++) {
            assignedEquipment[j] = assignedEquipment[j + 1];
         }
```

```
            assignedEquipment[equipmentCount - 1] = null;
            equipmentCount--;
            return true;
          }
        }
        return false; // not found
    }

    // Count assigned equipment
    public int getAssignedEquipmentCount() {
        return equipmentCount;
    }
}
```

Task 2

File 1: InventoryItem.java

```
public abstract class InventoryItem {

    // Attributes
    private String id;
    private String name;
    private boolean isAvailable;

    // Constructor
    public InventoryItem(String id, String name, boolean isAvailable) {
        this.id = id;
```

```
    this.name = name;
    this.isAvailable = isAvailable;
}

// Abstract method
public abstract String getItemType();

// Getters and setters
public String getId() {
    return id;
}

public void setId(String id) {
    this.id = id;
}

public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

public boolean isAvailable() {
    return isAvailable;
```

```
    }

    public void setAvailable(boolean available) {
        isAvailable = available;
    }

    // toString method
    @Override
    public String toString() {
        return "ID: " + id +
               ", Name: " + name +
               ", Available: " + isAvailable;
    }
}
```

File 2: Equipment.java

```
public class Equipment extends InventoryItem {

    // Equipment-specific attributes
    private String brand;
    private String assetId;
    private int warrantyMonths;

    // Constructor
    public Equipment(String id, String name, boolean isAvailable,
                     String brand, String assetId, int warrantyMonths) {
```

```
    super(id, name, isAvailable);
    this.brand = brand;
    this.assetId = assetId;
    this.warrantyMonths = warrantyMonths;
}

// Getters and setters
public String getBrand() {
    return brand;
}

public void setBrand(String brand) {
    this.brand = brand;
}

public String getAssetId() {
    return assetId;
}

public void setAssetId(String assetId) {
    this.assetId = assetId;
}

public int getWarrantyMonths() {
    return warrantyMonths;
}
```

```
    public void setWarrantyMonths(int warrantyMonths) {
        this.warrantyMonths = warrantyMonths;
    }

    // Implement abstract method
    @Override
    public String getItemType() {
        return "Equipment";
    }

    // Override toString
    @Override
    public String toString() {
        return getItemType() + " | " +
                super.toString() +
                ", Brand: " + brand +
                ", Asset ID: " + assetId +
                ", Warranty (months): " + warrantyMonths;
    }
}
```

File: 3 (Furniture.java)

```
public class Furniture extends InventoryItem {

    // Furniture-specific attributes
    private String roomNumber;
```

```
private String material;

// Constructor
public Furniture(String id, String name, boolean isAvailable,
        String roomNumber, String material) {

    super(id, name, isAvailable);
    this.roomNumber = roomNumber;
    this.material = material;
}

// Getters and setters
public String getRoomNumber() {
    return roomNumber;
}

public void setRoomNumber(String roomNumber) {
    this.roomNumber = roomNumber;
}

public String getMaterial() {
    return material;
}

public void setMaterial(String material) {
    this.material = material;
```

```
    }

    // Implement abstract method
    @Override
    public String getItemType() {
        return "Furniture";
    }

    // Override toString
    @Override
    public String toString() {
        return getItemType() + " | " +
                super.toString() +
                ", Room Number: " + roomNumber +
                ", Material: " + material;
    }
}
```

File: 4 (LabEquipment.java)

```
public class LabEquipment extends InventoryItem {

    // Lab equipment-specific attributes
    private String labName;
    private String calibrationDate;

    // Constructor
    public LabEquipment(String id, String name, boolean isAvailable,
```

```
            String labName, String calibrationDate) {

    super(id, name, isAvailable);
    this.labName = labName;
    this.calibrationDate = calibrationDate;
}

// Getters and setters
public String getLabName() {
    return labName;
}

public void setLabName(String labName) {
    this.labName = labName;
}

public String getCalibrationDate() {
    return calibrationDate;
}

public void setCalibrationDate(String calibrationDate) {
    this.calibrationDate = calibrationDate;
}

// Implement abstract method
@Override
```

```
    public String getItemType() {
        return "LabEquipment";
    }

    // Override toString
    @Override
    public String toString() {
        return getItemType() + " | " +
                super.toString() +
                ", Lab Name: " + labName +
                ", Calibration Date: " + calibrationDate;
    }
}
```

Task 3

(InventoryException.java)

```
public class InventoryException extends Exception {

    public InventoryException(String message) {
        super(message);
    }
}
```

(EquipmentNotAvailableException.java)

```
public class EquipmentNotAvailableException extends InventoryException {

    public EquipmentNotAvailableException(String message) {
        super(message);
```

```
    }
}
```

(StaffMemberNotFoundException.java)

```
public class StaffMemberNotFoundException extends InventoryException {

    public StaffMemberNotFoundException(String message) {
        super(message);
    }
}
```

(AssignmentLimitExceededException.java)

```
public class AssignmentLimitExceededException extends InventoryException {

    public AssignmentLimitExceededException(String message) {
        super(message);
    }
}
```

Task 4

(InventotyManager.java)

```
public class InventoryManager {

    private Equipment[] equipmentList;
    private int equipmentCount;

    // Constructor
    public InventoryManager(int maxEquipment) {
        equipmentList = new Equipment[maxEquipment];
```

```
    equipmentCount = 0;
}

// Add equipment to inventory
public void addEquipment(Equipment equipment) {
    if (equipmentCount < equipmentList.length) {
        equipmentList[equipmentCount] = equipment;
        equipmentCount++;
    }
}

// ------------------------------------------------
// assignEquipment – uses if-else decision structures
// ------------------------------------------------
public void assignEquipment(StaffMember staff, Equipment equipment)
        throws InventoryException {

    if (validateAssignment(staff, equipment)) {
        staff.addAssignedEquipment(equipment);
        equipment.setAvailable(false);
    }
}

// ------------------------------------------------
// returnEquipment – validates return and updates availability
// ------------------------------------------------
```

```
public void returnEquipment(StaffMember staff, String assetId)
        throws InventoryException {

    boolean removed = staff.removeAssignedEquipment(assetId);

    if (!removed) {
        throw new InventoryException("Equipment not assigned to this staff member.");
    }

    // Update availability in inventory
    for (int i = 0; i < equipmentCount; i++) {
        if (equipmentList[i].getAssetId().equals(assetId)) {
            equipmentList[i].setAvailable(true);
            break;
        }
    }
}

// ------------------------------------------------
// calculateMaintenanceFee – SWITCH statement
// ------------------------------------------------
public double calculateMaintenanceFee(Equipment equipment, int daysOverdue) {

    double dailyRate;

    switch (equipment.getCategory()) {
```

```
        case "Laptop":
            dailyRate = 5.0;
            break;

        case "Lab":
            dailyRate = 10.0;
            break;

        case "AV":
            dailyRate = 7.5;
            break;

        default:
            dailyRate = 3.0;
    }

    return dailyRate * daysOverdue;
}

// ------------------------------------------------
// Overloaded searchEquipment methods
// ------------------------------------------------

// Search by name
public void searchEquipment(String name) {
    for (int i = 0; i < equipmentCount; i++) {
```

```
        if (equipmentList[i].getName().equalsIgnoreCase(name)) {
            System.out.println(equipmentList[i]);
        }
    }
}

// Search by category and availability
public void searchEquipment(String category, boolean availableOnly) {
    for (int i = 0; i < equipmentCount; i++) {
        if (equipmentList[i].getCategory().equalsIgnoreCase(category)) {
            if (!availableOnly || equipmentList[i].isAvailable()) {
                System.out.println(equipmentList[i]);
            }
        }
    }
}

// Search by warranty range
public void searchEquipment(int minWarranty, int maxWarranty) {
    for (int i = 0; i < equipmentCount; i++) {
        int warranty = equipmentList[i].getWarrantyMonths();
        if (warranty >= minWarranty && warranty <= maxWarranty) {
            System.out.println(equipmentList[i]);
        }
    }
}
```

```
// ------------------------------------------------
// validateAssignment – nested if-else logic
// ------------------------------------------------
public boolean validateAssignment(StaffMember staff, Equipment equipment)
      throws InventoryException {


   if (staff == null) {
      throw new InventoryException("Staff member does not exist.");
   } else {
      if (equipment == null) {
         throw new InventoryException("Equipment does not exist.");
      } else {
         if (!equipment.isAvailable()) {
            throw new EquipmentNotAvailableException(
                  "Equipment is currently not available."
            );
         } else {
            if (staff.getAssignedEquipmentCount() >= 5) {
               throw new AssignmentLimitExceededException(
                     "Staff member has reached assignment limit."
               );
            }
         }
      }
   }
```

```
        return true;
    }
}
```

Task 5

```
public class InventoryReports {

    private InventoryItem[] inventoryItems;
    private Equipment[] equipmentList;
    private StaffMember[] staffList;

    public InventoryReports(InventoryItem[] inventoryItems,
                            Equipment[] equipmentList,
                            StaffMember[] staffList) {

        this.inventoryItems = inventoryItems;
        this.equipmentList = equipmentList;
        this.staffList = staffList;
    }

    // ------------------------------------------------
    // 1. FOR loop – Inventory report
    // ------------------------------------------------
    public void generateInventoryReport() {
        System.out.println("=== Inventory Report ===");
```

```
    for (int i = 0; i < inventoryItems.length; i++) {
        if (inventoryItems[i] != null) {
            System.out.println(
                    inventoryItems[i] +
                    " | Status: " +
                    (inventoryItems[i].isAvailable() ? "Available" : "Assigned")
            );
        }
    }
}

// ------------------------------------------------
// 2. WHILE loop – Find expired warranties
// ------------------------------------------------
public void findExpiredWarranties() {
    System.out.println("=== Expired Warranties ===");

    int index = 0;
    while (index < equipmentList.length) {
        if (equipmentList[index] != null &&
            equipmentList[index].getWarrantyMonths() == 0) {

            System.out.println(equipmentList[index]);
        }
        index++;
    }
```

```
}

// ------------------------------------------------
// 3. Enhanced FOR loop – Assignments by department
// ------------------------------------------------
public void displayAssignmentsByDepartment() {
   System.out.println("=== Assignments by Department ===");

   for (StaffMember staff : staffList) {
      if (staff != null) {
         System.out.println(
               "Department: " + staff.getDepartment() +
               " | Staff: " + staff.getName() +
               " | Assigned Items: " +
               staff.getAssignedEquipmentCount()
         );
      }
   }
}

// ------------------------------------------------
// 4. NESTED loops – Utilisation rate calculation
// ------------------------------------------------
public void calculateUtilisationRate() {
   int totalEquipment = 0;
   int assignedEquipment = 0;
```

```
for (int i = 0; i < equipmentList.length; i++) {
   if (equipmentList[i] != null) {
      totalEquipment++;

      // Nested loop checks if equipment is assigned
      for (StaffMember staff : staffList) {
         if (staff != null) {
            Equipment[] assigned = staff.getAssignedEquipment();

            for (int j = 0; j < assigned.length; j++) {
               if (assigned[j] != null &&
                  assigned[j].equals(equipmentList[i])) {

                  assignedEquipment++;
                  break;
               }
            }
         }
      }
   }
}

double utilisationRate = totalEquipment == 0
      ? 0
      : ((double) assignedEquipment / totalEquipment) * 100;
```

```
    System.out.println("Equipment Utilisation Rate: "
            + utilisationRate + "%");
}

// ------------------------------------------------
// 5. DO-WHILE loop – Maintenance schedule
// ------------------------------------------------
public void generateMaintenanceSchedule() {
    System.out.println("=== Maintenance Schedule ===");

    int i = 0;

    do {
        if (equipmentList[i] != null &&
            equipmentList[i].getWarrantyMonths() <= 6) {

            System.out.println(
                equipmentList[i].getName() +
                " | Asset ID: " + equipmentList[i].getAssetId() +
                " | Maintenance Required"
            );
        }
        i++;
    } while (i < equipmentList.length);
}
```

}

Task 6

UniversityInventorySystem.java

```
import java.util.InputMismatchException;
import java.util.Scanner;

import managers.InventoryManager;
import managers.InventoryReports;
import models.Equipment;
import models.Furniture;
import models.InventoryItem;
import models.LabEquipment;
import models.StaffMember;
import exceptions.InventoryException;

public class UniversityInventorySystem {

    private static final Scanner scanner = new Scanner(System.in);
    private static final InventoryManager inventoryManager = new InventoryManager();
    private static final InventoryReports inventoryReports = new InventoryReports();

    public static void main(String[] args) {
        boolean running = true;

        System.out.println("=====================================");
        System.out.println(" University Inventory Management System ");
```

```
System.out.println("=====================================");

while (running) {
    printMenu();

    try {
        System.out.print("Enter your choice: ");
        int choice = scanner.nextInt();
        scanner.nextLine(); // consume newline

        switch (choice) {
            case 1 -> addNewEquipment();
            case 2 -> registerStaff();
            case 3 -> assignEquipment();
            case 4 -> returnEquipment();
            case 5 -> searchInventory();
            case 6 -> generateReports();
            case 7 -> {
                System.out.println("Exiting system. Goodbye!");
                running = false;
            }
            default -> System.out.println("Invalid option. Please try again.");
        }

    } catch (InputMismatchException e) {
        System.out.println("Invalid input. Please enter a number.");
```

```
            scanner.nextLine(); // clear buffer
        } catch (InventoryException e) {
            System.out.println("Error: " + e.getMessage());
        } catch (Exception e) {
            System.out.println("Unexpected error occurred. Please try again.");
        }
    }
}

private static void printMenu() {
    System.out.println("\nMenu:");
    System.out.println("1. Add new equipment");
    System.out.println("2. Register a new staff member");
    System.out.println("3. Assign equipment to staff");
    System.out.println("4. Return equipment");
    System.out.println("5. Search inventory");
    System.out.println("6. Generate reports");
    System.out.println("7. Exit system");
}

/* ========================
   Menu Option Implementations
   ======================== */

private static void addNewEquipment() {
    System.out.println("\nSelect equipment type:");
```

```
System.out.println("1. Equipment");
System.out.println("2. Furniture");
System.out.println("3. Lab Equipment");

int type = scanner.nextInt();
scanner.nextLine();

System.out.print("Asset ID: ");
String assetId = scanner.nextLine();

System.out.print("Name: ");
String name = scanner.nextLine();

System.out.print("Brand: ");
String brand = scanner.nextLine();

System.out.print("Category: ");
String category = scanner.nextLine();

InventoryItem item;

switch (type) {
   case 1 -> item = new Equipment(assetId, name, brand, true, category);
   case 2 -> item = new Furniture(assetId, name, brand, true, category);
   case 3 -> item = new LabEquipment(assetId, name, brand, true, category);
   default -> {
```

```
        System.out.println("Invalid equipment type.");
        return;
    }
}


inventoryManager.addItem(item);
System to=
```