National Academy of Professional Studies
(NAPS)
Bachelor of Information Technology
ITS105 Programming Fundamentals
Coding Assignment
Trimester 3 2025

Task 1: Basic Class Design and Arrays

File: Equipment.java

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

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

File: StaffMember.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 equipment 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 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
}

// Get number of assigned equipment
public int getAssignedEquipmentCount() {
    return equipmentCount;
}
}
```

Task 2: Inheritance and Polymorphism

File: InventoryItem.java

```
public abstract class InventoryItem {

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

File: Equipment.java (Modified)

```
public class Equipment extends InventoryItem {

    private String brand;
    private String assetId;
    private int warrantyMonths;

    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;
    }

    @Override
    public String getItemType() {
        return "Equipment";
    }

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

File: Furniture.java

```
public class Furniture extends InventoryItem {

    private String roomNumber;
    private String material;

    public Furniture(String id, String name, boolean isAvailable,
                     String roomNumber, String material) {
        super(id, name, isAvailable);
```

```
        this.roomNumber = roomNumber;
        this.material = material;
    }

    @Override
    public String getItemType() {
        return "Furniture";
    }

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

File: LabEquipment.java

```
public class LabEquipment extends InventoryItem {

    private String labName;
    private String calibrationDate;

    public LabEquipment(String id, String name, boolean isAvailable,
                        String labName, String calibrationDate) {
        super(id, name, isAvailable);
        this.labName = labName;
        this.calibrationDate = calibrationDate;
    }

    @Override
    public String getItemType() {
        return "LabEquipment";
    }

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

Task 3: Exception Handling

File: InventoryException.java

```
public class InventoryException extends Exception {
```

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

class EquipmentNotAvailableException extends InventoryException {

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

class StaffMemberNotFoundException extends InventoryException {

    public StaffMemberNotFoundException(String message) {
        super(message);
    }
}
class AssignmentLimitExceededException extends InventoryException {

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

InventoryException (base exception class)

```
import java.util.HashMap;
import java.util.Map;

public class InventoryService {

    private Map<String, Integer> equipmentStock = new HashMap<>();
    private Map<String, Integer> staffAssignments = new HashMap<>();
    private static final int MAX_ASSIGNMENTS = 3;

    public InventoryService() {
        equipmentStock.put("Laptop", 2);
        equipmentStock.put("Projector", 1);

        staffAssignments.put("EMP001", 1);
        staffAssignments.put("EMP002", 3);
    }

    public void assignEquipment(String staffId, String equipment)
            throws InventoryException {

        if (!staffAssignments.containsKey(staffId)) {
            throw new StaffMemberNotFoundException(
                "Staff member " + staffId + " not found."
            );
        }

        if (!equipmentStock.containsKey(equipment) ||
```

```
            equipmentStock.get(equipment) <= 0) {

            throw new EquipmentNotAvailableException(
                equipment + " is not available in inventory."
            );
        }

        int currentAssignments = staffAssignments.get(staffId);
        if (currentAssignments >= MAX_ASSIGNMENTS) {
            throw new AssignmentLimitExceededException(
                "Assignment limit exceeded for staff member " + staffId
            );
        }

        // Update inventory and assignments
        equipmentStock.put(equipment, equipmentStock.get(equipment) - 1);
        staffAssignments.put(staffId, currentAssignments + 1);
    }
}
```

Main system (proper exception handling)

```
// File: Main.java

public class Main {

    public static void main(String[] args) {

        InventoryService inventoryService = new InventoryService();

        try {
            inventoryService.assignEquipment("EMP002", "Laptop");
        } catch (EquipmentNotAvailableException e) {
            System.out.println("Inventory Error: " + e.getMessage());
        } catch (StaffMemberNotFoundException e) {
            System.out.println("Staff Error: " + e.getMessage());
        } catch (AssignmentLimitExceededException e) {
            System.out.println("Limit Error: " + e.getMessage());
        } catch (InventoryException e) {
            System.out.println("General Inventory Error: " + e.getMessage());
        }
    }
}
```

Task 4: Decision Structures and Methods (7 marks)

InventoryManager.java

```
// File: InventoryManager.java

import java.util.ArrayList;
import java.util.List;

public class InventoryManager {

    private List<Equipment> equipmentList = new ArrayList<>();
    private static final int MAX_ASSIGNMENTS = 3;

    /* ------------------------------------
       Assign Equipment
       ------------------------------------ */
    public void assignEquipment(StaffMember staff, Equipment equipment) {

        if (staff != null && equipment != null) {

            if (equipment.isAvailable()) {

     if (staff.getAssignedCount() < MAX_ASSIGNMENTS) {
                    staff.assignEquipment(equipment);
                    equipment.setAvailable(false);
                } else {
                    System.out.println("Assignment limit exceeded for staff member.");
                }

            } else {
                System.out.println("Equipment not available.");
            }

        } else {
            System.out.println("Invalid staff or equipment.");
        }
    }

    /* ------------------------------------
       Return Equipment
       ------------------------------------ */
    public void returnEquipment(StaffMember staff, String assetId) {

        if (staff != null && assetId != null) {

            Equipment returned = staff.returnEquipment(assetId);
```

```
        if (returned != null) {
            returned.setAvailable(true);
        } else {
            System.out.println("Equipment not assigned to this staff member.");
        }

    } else {
        System.out.println("Invalid return request.");
    }
}

/* ------------------------------------
   Calculate Maintenance Fee
   ------------------------------------ */
public double calculateMaintenanceFee(Equipment equipment, int daysOverdue) {

    double feePerDay;

    switch (equipment.getCategory()) {
        case "Laptop":
            feePerDay = 5.0;
            break;

        case "Projector":
            feePerDay = 8.0;
            break;

        case "Printer":
            feePerDay = 6.0;
            break;

        default:
            feePerDay = 4.0;
    }

    return feePerDay * daysOverdue;
}

/* ------------------------------------
   Search Equipment (Overloaded Methods)
   ------------------------------------ */

// Search by name
public List<Equipment> searchEquipment(String name) {
    List<Equipment> result = new ArrayList<>();

    for (Equipment e : equipmentList) {
        if (e.getName().equalsIgnoreCase(name)) {
```

```
            result.add(e);
        }
    }
    return result;
}

// Search by category and availability
public List<Equipment> searchEquipment(String category, boolean availableOnly) {
    List<Equipment> result = new ArrayList<>();

    for (Equipment e : equipmentList) {
        if (e.getCategory().equalsIgnoreCase(category)) {
            if (!availableOnly || e.isAvailable()) {
                result.add(e);
            }
        }
    }
    return result;
}

// Search by warranty range
public List<Equipment> searchEquipment(int minWarranty, int maxWarranty) {
    List<Equipment> result = new ArrayList<>();

    for (Equipment e : equipmentList) {
        if (e.getWarrantyYears() >= minWarranty &&
            e.getWarrantyYears() <= maxWarranty) {
            result.add(e);
        }
    }
    return result;
}

/* ------------------------------------
   Validate Assignment
   ------------------------------------ */
public boolean validateAssignment(StaffMember staff, Equipment equipment) {

    if (staff != null) {

        if (equipment != null) {

            if (equipment.isAvailable()) {

                if (staff.getAssignedCount() < MAX_ASSIGNMENTS) {
                    return true;
                } else {
                    System.out.println("Staff assignment limit reached.");
                }
```

```
            } else {
                System.out.println("Equipment unavailable.");
            }

        } else {
            System.out.println("Equipment is invalid.");
        }

    } else {
        System.out.println("Staff member is invalid.");
    }

    return false;
  }
}
```

Task 5: Loops and Data Processing

InventoryReports.java

```
// File: InventoryReports.java

import java.util.List;
import java.util.Map;
import java.util.HashMap;

public class InventoryReports {

    private List<Equipment> equipmentList;
    private List<StaffMember> staffList;

   public InventoryReports(List<Equipment> equipmentList, List<StaffMember> staffList) {
    this.equipmentList = equipmentList;
        this.staffList = staffList;
   }

    /* ----------------------------------
       Inventory Report (for loop)
       ---------------------------------- */
    public void generateInventoryReport() {

        System.out.println("=== Inventory Report ===");

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

```
        Equipment e = equipmentList.get(i);

        System.out.println(
            e.getAssetId() + " | " +
            e.getName() + " | " +
            (e.isAvailable() ? "Available" : "Assigned")
        );
    }
}

/* ------------------------------------
   Expired Warranties (while loop)
   ------------------------------------ */
public void findExpiredWarranties() {

    System.out.println("=== Expired Warranties ===");

    int index = 0;

    while (index < equipmentList.size()) {
        Equipment e = equipmentList.get(index);

        if (e.getWarrantyMonths() == 0) {
            System.out.println(
                e.getAssetId() + " - " + e.getName() + " (Warranty Expired)"
            );
        }
        index++;
    }
}

/* ------------------------------------
   Assignments by Department (enhanced for loop)
   ------------------------------------ */
public void displayAssignmentsByDepartment() {

    System.out.println("=== Assignments by Department ===");

    Map<String, Integer> departmentCount = new HashMap<>();

    for (StaffMember staff : staffList) {

        String department = staff.getDepartment();
        int assigned = staff.getAssignedCount();

        if (!departmentCount.containsKey(department)) {
            departmentCount.put(department, assigned);
        } else {
            departmentCount.put(
```

```
                department,
                departmentCount.get(department) + assigned
            );
        }
    }

    for (String department : departmentCount.keySet()) {
        System.out.println(
            department + " Department: " +
            departmentCount.get(department) + " equipment assigned"
        );
    }
}

/* ----------------------------------
   Utilisation Rate (nested loops)
   ---------------------------------- */
public void calculateUtilisationRate() {

    System.out.println("=== Equipment Utilisation Rate ===");

    for (Equipment equipment : equipmentList) {

        int usageCount = 0;

        for (StaffMember staff : staffList) {
            if (staff.hasEquipment(equipment.getAssetId())) {
                usageCount++;
            }
        }

        System.out.println(
            equipment.getName() + " used by " +
            usageCount + " staff member(s)"
        );
    }
}

/* ----------------------------------
   Maintenance Schedule (do-while loop)
   ---------------------------------- */
public void generateMaintenanceSchedule() {

    System.out.println("=== Maintenance Schedule ===");

    int index = 0;

    if (equipmentList.isEmpty()) {
        System.out.println("No equipment available for maintenance.");
```

```
            return;
        }

        do {
            Equipment e = equipmentList.get(index);

            if (!e.isAvailable() || e.getWarrantyMonths() == 0) {
                System.out.println(
                    "Schedule maintenance for: " +
                    e.getAssetId() + " - " + e.getName()
                );
            }

            index++;

        } while (index < equipmentList.size());
    }
}
```

Task 6: Integration and Main Application

UniversityInventorySystem.java

```
// File: UniversityInventorySystem.java

import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class UniversityInventorySystem {

    private static List<Equipment> equipmentList = new ArrayList<>();
    private static List<StaffMember> staffList = new ArrayList<>();

    private static InventoryManager inventoryManager =
            new InventoryManager();
    private static InventoryService inventoryService =
            new InventoryService();

    public static void main(String[] args) {
scanner = new Scanner(System.in);
        boolean running = true;

        while (running) {

            System.out.println("\n=== University Inventory System ===");
            System.out.println("1. Add new equipment");
            System.out.println("2. Register 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");
System.out.print("Choose an option: ");

int choice = scanner.nextInt();
scanner.nextLine(); // clear buffer

try {
    switch (choice) {

        case 1:
            addNewEquipment(scanner);
            break;

        case 2:
            registerStaff(scanner);
            break;

        case 3:
            assignEquipment(scanner);
            break;

        case 4:
            returnEquipment(scanner);
            break;

        case 5:
            searchInventory(scanner);
            break;

        case 6:
            generateReports();
            break;

        case 7:
            running = false;
            System.out.println("Exiting system. Goodbye!");
            break;

        default:
            System.out.println("Invalid option. Please try again.");
    }

} catch (InventoryException e) {
    // Graceful handling of custom exceptions
    System.out.println("Operation failed: " + e.getMessage());
```

```
        } catch (Exception e) {
            // Catch any unexpected runtime errors
            System.out.println("Unexpected error: " + e.getMessage());
        }
    }

    scanner.close();
}

/* ----------------------------------
   Menu Operations
   ---------------------------------- */

private static void addNewEquipment(Scanner scanner) {

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

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

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

    System.out.print("Enter warranty months: ");
    int warranty = scanner.nextInt();
    scanner.nextLine();

    // Polymorphism: Equipment reference, concrete object
    Equipment equipment = new Equipment(
            assetId, name, category, warranty, true
    );

    equipmentList.add(equipment);
    System.out.println("Equipment added successfully.");
}

private static void registerStaff(Scanner scanner) {

    System.out.print("Enter staff ID: ");
    String staffId = scanner.nextLine();

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

    System.out.print("Enter department: ");
    String department = scanner.nextLine();
```

```
    StaffMember staff = new StaffMember(staffId, name, department);
    staffList.add(staff);

    System.out.println("Staff member registered successfully.");
}

private static void assignEquipment(Scanner scanner)
        throws InventoryException {

    System.out.print("Enter staff ID: ");
    String staffId = scanner.nextLine();

    System.out.print("Enter equipment name: ");
    String equipmentName = scanner.nextLine();

    StaffMember staff = findStaffById(staffId);
    Equipment equipment = findAvailableEquipmentByName(equipmentName);

    inventoryService.assignEquipment(staff.getStaffId(), equipment.getName());
    inventoryManager.assignEquipment(staff, equipment);

    System.out.println("Equipment assigned successfully.");
}

private static void returnEquipment(Scanner scanner) {

    System.out.print("Enter staff ID: ");
    String staffId = scanner.nextLine();

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

    StaffMember staff = findStaffById(staffId);
    inventoryManager.returnEquipment(staff, assetId);

    System.out.println("Equipment returned successfully.");
}

private static void searchInventory(Scanner scanner) {

    System.out.print("Enter equipment name to search: ");
    String name = scanner.nextLine();

    List<Equipment> results =
            inventoryManager.searchEquipment(name);

    if (results.isEmpty()) {
```

```
            System.out.println("No equipment found.");
        } else {
            for (Equipment e : results) {
                System.out.println(
                        e.getAssetId() + " - " +
                        e.getName() + " (" +
                        (e.isAvailable() ? "Available" : "Assigned") + ")"
                );
            }
        }
    }

    private static void generateReports() {

        InventoryReports reports =
                new InventoryReports(equipmentList, staffList);

        reports.generateInventoryReport();
        reports.findExpiredWarranties();
        reports.displayAssignmentsByDepartment();
        reports.calculateUtilisationRate();
        reports.generateMaintenanceSchedule();
    }

    /* ------------------------------------
       Helper Methods
       ------------------------------------ */

    private static StaffMember findStaffById(String staffId) {
        for (StaffMember staff : staffList) {
            if (staff.getStaffId().equalsIgnoreCase(staffId)) {
                return staff;
            }
        }
        throw new RuntimeException("Staff member not found.");
    }

    private static Equipment findAvailableEquipmentByName(String name) {
        for (Equipment e : equipmentList) {
            if (e.getName().equalsIgnoreCase(name) && e.isAvailable()) {
                return e;
            }
        }
        throw new RuntimeException("Equipment not available.");
    }
}
```