DAY-8

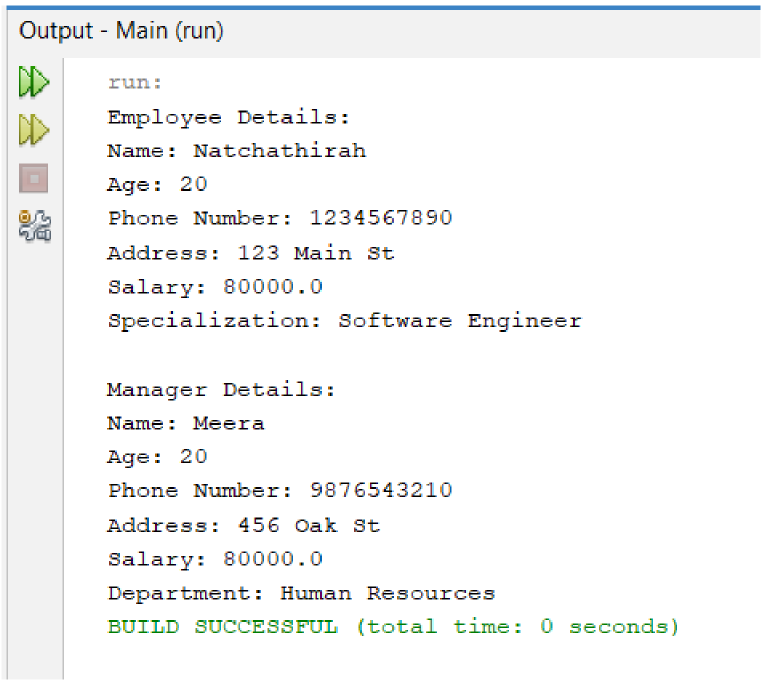
QUIZ-1

1. Create a class named 'Member' having the following members:
Data members
1 - Name
2 - Age
3 - Phone number
4 - Address
5 - Salary
It also has a method named 'printSalary' which prints the salary of the members.
Two classes 'Employee' and 'Manager' inherits the 'Member' class. The 'Employee' and 'Manager' classes have data members 'specialization' and 'department' respectively.
Now, assign name, age, phone number, address and salary to an employee and a manager by making an object of both of these classes and print the same.

CODE:

```
package main;
class Member {
   String name;
   int age;
   String phoneNumber;
   String address;
   double salary;
   public Member(String name, int age, String phoneNumber, String address, double salary) {
      this.name = name;
      this.age = age;
      this.phoneNumber = phoneNumber;
      this.address = address;
      this.salary = salary;
   }
   public void printSalary() {
      System.out.println("Salary: " + salary);
   }
}
class Employee extends Member {
   String specialization;
   public Employee(String name, int age, String phoneNumber, String address, double salary,
String specialization) {
      super(name, age, phoneNumber, address, salary);
      this.specialization = specialization;
   }
}

class Manager extends Member {
    String department;
    public Manager(String name, int age, String phoneNumber, String address, double salary,
String department) {
        super(name, age, phoneNumber, address, salary);
        this.department = department;
    }
}
public class Main {
    public static void main(String[] args) {
        Employee employee = new Employee("Natchathirah", 20, "1234567890", "123 Main
St", 80000.0, "Software Engineer");
        Manager manager = new Manager("Meera", 20, "9876543210", "456 Oak St", 80000.0,
"Human Resources");
        System.out.println("Employee Details:");
        System.out.println("Name: " + employee.name);
        System.out.println("Age: " + employee.age);
        System.out.println("Phone Number: " + employee.phoneNumber);
        System.out.println("Address: " + employee.address);
        employee.printSalary();
        System.out.println("Specialization: " + employee.specialization);

        System.out.println("\nManager Details:");
        System.out.println("Name: " + manager.name);
        System.out.println("Age: " + manager.age);
        System.out.println("Phone Number: " + manager.phoneNumber);
        System.out.println("Address: " + manager.address);
        manager.printSalary();
        System.out.println("Department: " + manager.department);
    }
}
```

OUTPUT:



```
run:
Employee Details:
Name: Natchathirah
Age: 20
Phone Number: 1234567890
Address: 123 Main St
Salary: 80000.0
Specialization: Software Engineer

Manager Details:
Name: Meera
Age: 20
Phone Number: 9876543210
Address: 456 Oak St
Salary: 80000.0
Department: Human Resources
BUILD SUCCESSFUL (total time: 0 seconds)
```

2. You are developing a banking application in Java. Design a class hierarchy that represents different account types such as SavingsAccount, CheckingAccount, and LoanAccount.

Each account should have basic functionality like deposit, withdraw, and check balance. Ensure that your design follows appropriate use of interfaces and inheritance.

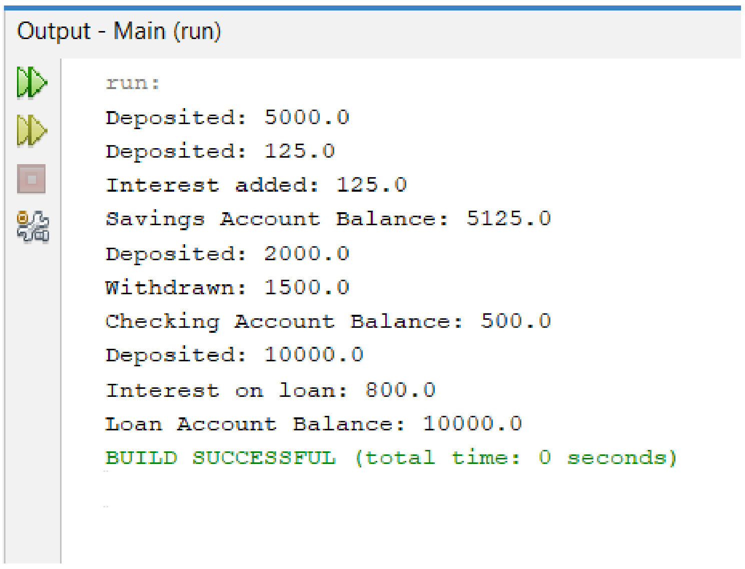
CODE:

```
package main;
interface Account {
   void deposit(double amount);
   void withdraw(double amount);
   double checkBalance();
}
abstract class AbstractAccount implements Account {
   private String accountNumber;
   private double balance;
   public AbstractAccount(String accountNumber) {
      this.accountNumber = accountNumber;
      this.balance = 0.0;
   }
   public void deposit(double amount) {
      if (amount > 0) {
         balance += amount;
         System.out.println("Deposited: " + amount);
      } else {
         System.out.println("Invalid deposit amount.");
      }
   }
   public void withdraw(double amount) {
      if (amount > 0 && amount <= balance) {
         balance -= amount;
         System.out.println("Withdrawn: " + amount);
      } else {
         System.out.println("Invalid withdrawal amount or insufficient balance.");
      }
   }
   public double checkBalance() {
      return balance;
   }
   public String getAccountNumber() {
      return accountNumber;
   }
}
class SavingsAccount extends AbstractAccount {
   private double interestRate;
   public SavingsAccount(String accountNumber, double interestRate) {
      super(accountNumber);
      this.interestRate = interestRate;
```

```
        }
        public void addInterest() {
            double interest = checkBalance() * interestRate / 100;
            deposit(interest);
            System.out.println("Interest added: " + interest);
        }
    }
    class CheckingAccount extends AbstractAccount {
        private double overdraftLimit;
        public CheckingAccount(String accountNumber, double overdraftLimit) {
            super(accountNumber);
            this.overdraftLimit = overdraftLimit;
        }
        @Override
        public void withdraw(double amount) {
            if (amount > 0 && (checkBalance() - amount) >= -overdraftLimit) {
                super.withdraw(amount);
            } else {
                System.out.println("Invalid withdrawal amount or exceeding overdraft limit.");
            }
        }
    }
    class LoanAccount extends AbstractAccount {
        private double interestRate;

        public LoanAccount(String accountNumber, double interestRate) {
            super(accountNumber);
            this.interestRate = interestRate;
        }
        public void calculateInterest() {
            double interest = checkBalance() * interestRate / 100;
            System.out.println("Interest on loan: " + interest);
        }
    }
    public class Main {
        public static void main(String[] args) {
            // Example usage
            SavingsAccount savingsAccount = new SavingsAccount("SA123", 2.5);
            CheckingAccount checkingAccount = new CheckingAccount("CA456", 1000.0);
            LoanAccount loanAccount = new LoanAccount("LA789", 8.0);
            savingsAccount.deposit(5000);
            savingsAccount.addInterest();
            System.out.println("Savings Account Balance: " + savingsAccount.checkBalance());
            checkingAccount.deposit(2000);
            checkingAccount.withdraw(1500);
            System.out.println("Checking Account Balance: " + checkingAccount.checkBalance());
            loanAccount.deposit(10000);
            loanAccount.calculateInterest();
            System.out.println("Loan Account Balance: " + loanAccount.checkBalance());
        }
```

```
}
```

OUTPUT:



Output - Main (run)

```
run:
Deposited: 5000.0
Deposited: 125.0
Interest added: 125.0
Savings Account Balance: 5125.0
Deposited: 2000.0
Withdrawn: 1500.0
Checking Account Balance: 500.0
Deposited: 10000.0
Interest on loan: 800.0
Loan Account Balance: 10000.0
BUILD SUCCESSFUL (total time: 0 seconds)
```

3. You are tasked with designing a university enrollment system in Java. Implement a class hierarchy that includes a base class **Person** and two subclasses, **Student** and **Professor** and a **Course** class. Each class should have the necessary attributes. Each course should have a list of prerequisites and enrolled students.

Your tasks are as follows:

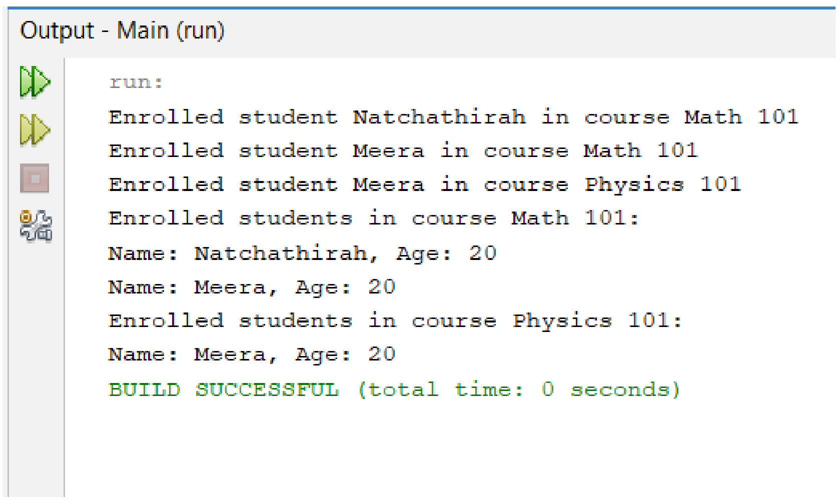
i) Students should only be enrolled if they have completed all the required prerequisites. In the course class, include logic for enrolling students.

ii) Display enrolled students in a particular with relevant information.

CODE:

```
package main;

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

// Base class Person
class Person {
    protected String name;
    protected int age;

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public String getName() {
        return name;
    }
    public int getAge() {
        return age;
    }
}

// Student class, subclass of Person
class Student extends Person {
    public Student(String name, int age) {
        super(name, age);
    }

    boolean hasCompletedCourse(Course prerequisite) {
        throw new UnsupportedOperationException("Not supported yet."); // Generated from
nbfs://nbhost/SystemFileSystem/Templates/Classes/Code/GeneratedMethodBody
    }
}

// Professor class, subclass of Person
class Professor extends Person {
    public Professor(String name, int age) {
        super(name, age);
    }
}

// Course class
class Course {
    private String courseName;
    private List<Course> prerequisites;
    private List<Student> enrolledStudents;

    public Course(String courseName) {
        this.courseName = courseName;
        this.prerequisites = new ArrayList<>();
        this.enrolledStudents = new ArrayList<>();
    }

    public void addPrerequisite(Course prerequisite) {
        prerequisites.add(prerequisite);
    }

    public void enrollStudent(Student student) {
        if (hasCompletedPrerequisites(student)) {
            enrolledStudents.add(student);
            System.out.println("Enrolled student " + student.getName() + " in course " +
courseName);
        } else {
            System.out.println("Student " + student.getName() + " does not meet prerequisites for
course " + courseName);
        }
    }

    public void displayEnrolledStudents() {
        System.out.println("Enrolled students in course " + courseName + ":");
        for (Student student : enrolledStudents) {
            System.out.println("Name: " + student.getName() + ", Age: " + student.getAge());
        }
    }

    private boolean hasCompletedPrerequisites(Student student) {
        for (Course prerequisite : prerequisites) {
            if (!student.hasCompletedCourse(prerequisite)) {
                return false;
            }
        }
        return true;
    }
}
class StudentWithCompletedCourses extends Student {
    private List<Course> completedCourses;

    public StudentWithCompletedCourses(String name, int age) {
        super(name, age);
        this.completedCourses = new ArrayList<>();
    }

    public void completeCourse(Course course) {
        completedCourses.add(course);
    }

    public boolean hasCompletedCourse(Course course) {
        return completedCourses.contains(course);
    }
}
public class Main {
    public static void main(String[] args) {
        StudentWithCompletedCourses student1 = new
StudentWithCompletedCourses("Natchathirah", 20);
        StudentWithCompletedCourses student2 = new
StudentWithCompletedCourses("Meera", 20);
        Course math101 = new Course("Math 101");
        Course physics101 = new Course("Physics 101");
        Course calculus = new Course("Calculus");
        math101.addPrerequisite(calculus);
        physics101.addPrerequisite(math101);
        student1.completeCourse(calculus);
        student2.completeCourse(calculus);
        student2.completeCourse(math101);
        math101.enrollStudent(student1);
        math101.enrollStudent(student2);
        physics101.enrollStudent(student2);
        math101.displayEnrolledStudents();
        physics101.displayEnrolledStudents();
    }
}
```

OUTPUT:



Output - Main (run)

```
run:
Enrolled student Natchathirah in course Math 101
Enrolled student Meera in course Math 101
Enrolled student Meera in course Physics 101
Enrolled students in course Math 101:
Name: Natchathirah, Age: 20
Name: Meera, Age: 20
Enrolled students in course Physics 101:
Name: Meera, Age: 20
BUILD SUCCESSFUL (total time: 0 seconds)
```