1. Write a Java program that reads a file and throws an exception if the file is empty.

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class FileChecker {
    public static void main(String[] args) {
        try {
            File file = new File("example.txt");
            Scanner scanner = new Scanner(file);

            if (!scanner.hasNext()) {
                throw new Exception("File is empty");
            }

            while (scanner.hasNextLine()) {
                System.out.println(scanner.nextLine());
            }

            scanner.close();
        } catch (FileNotFoundException e) {
            System.out.println("File not found");
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }
}
```

2. Write a Java program to create a method that takes an integer as a parameter and throws an exception if the number is odd.

```
// Define a custom exception class for odd numbers
class OddNumberException extends Exception {
    public OddNumberException(String message) {
        super(message);
    }
}

// Main class containing the main method and the checkEvenNumber method
public class NumberChecker {
    public static void main(String[] args) {
        try {
            checkEvenNumber(5); // Change the number to test different inputs
        } catch (OddNumberException e) {
            System.out.println(e.getMessage());
        }
    }

    // Method to check if the number is even, throws an exception if the number is odd
    public static void checkEvenNumber(int number) throws OddNumberException {
        if (number % 2 != 0) {
            throw new OddNumberException("The number " + number + " is odd.");
        } else {
            System.out.println("The number " + number + " is even.");
        }
    }
}
```

3 . Write a Java program to create a method that reads a file and throws an exception if the file is not found.

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class FileReaderExample {

   // Main method where the program execution starts
   public static void main(String[] args) {
      try {
         // Attempt to read the file using the readFile method
         readFile("example.txt");
      } catch (FileNotFoundException e) {
         // Handle the exception if the file is not found
         System.out.println(e.getMessage());
      }
   }

   // Method to read a file, throws FileNotFoundException if the file is not found
   public static void readFile(String fileName) throws FileNotFoundException {
      File file = new File(fileName); // Create a File object with the specified file name
      Scanner scanner = new Scanner(file); // Create a Scanner object to read the file

      // Read and print the content of the file line by line
      while (scanner.hasNextLine()) {
         System.out.println(scanner.nextLine());
      }

      scanner.close(); // Close the scanner to free resources
   }
}
```

4 Write a Java program to create a method that takes a string as input and throws an exception if the string does not contain vowels.

```
// Define a custom exception class for strings without vowels
class NoVowelException extends Exception {
   public NoVowelException(String message) {
      super(message);
   }
}

// Main class containing the main method and the checkForVowels method
public class VowelChecker {
```

```
    public static void main(String[] args) {
        try {
            // Call the method with a string parameter
            checkForVowels("bcdfg"); // Change the string to test different inputs
        } catch (NoVowelException e) {
            System.out.println(e.getMessage());
        }
    }

    // Method to check if the string contains vowels, throws an exception if it does not
    public static void checkForVowels(String input) throws NoVowelException {
        // Convert the string to lowercase to simplify the vowel check
        String lowerCaseInput = input.toLowerCase();
        // Check if the string contains any vowels
        if (!(lowerCaseInput.contains("a") || lowerCaseInput.contains("e") ||
              lowerCaseInput.contains("i") || lowerCaseInput.contains("o") ||
              lowerCaseInput.contains("u"))) {
            // Throw the custom exception if no vowels are found
            throw new NoVowelException("The string \"" + input + "\" does not contain any
vowels.");
        } else {
            System.out.println("The string \"" + input + "\" contains vowels.");
        }
    }
}
```

5. Write a Java program that creates a bank account with concurrent deposits and withdrawals using threads.

```
// Creating multiple threads for deposits and withdrawals
Thread depositThread = new Thread(() -> {
    for (int i = 0; i < 5; i++) {
        account.deposit(100);
        try {
            Thread.sleep(100); // Simulating delay between deposits
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
});

Thread withdrawThread = new Thread(() -> {
    for (int i = 0; i < 5; i++) {
        account.withdraw(50);
        try {
```

```
            Thread.sleep(150); // Simulating delay between withdrawals
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
});
```

6. Write a Java program to create an interface Playable with a method play () that takes no arguments and returns void. Create three classes Football, Volleyball, and Basketball that implement the Playable interface and override the play () method to play the respective sports.

```
// Define the Playable interface
interface Playable {
    void play();
}

// Football class implementing Playable
class Football implements Playable {
    @Override
    public void play() {
        System.out.println("Playing Football");
    }
}

// Volleyball class implementing Playable
class Volleyball implements Playable {
    @Override
    public void play() {
        System.out.println("Playing Volleyball");
    }
}

// Basketball class implementing Playable
class Basketball implements Playable {
    @Override
    public void play() {
        System.out.println("Playing Basketball");
    }
}
```

```
// Main class to test the Playable interface and its implementations
public class SportsSimulation {
    public static void main(String[] args) {
        // Creating instances of Football, Volleyball, and Basketball
        Football football = new Football();
        Volleyball volleyball = new Volleyball();
        Basketball basketball = new Basketball();

        // Calling play() method on each object
        football.play();
        volleyball.play();
        basketball.play();
    }
}
```

7.Write a Java program that reads a list of numbers from a file and throws an exception if any of the numbers are positive.

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

// Custom exception class for positive numbers
class PositiveNumberException extends Exception {
    public PositiveNumberException(String message) {
        super(message);
    }
}

public class NumberReader {
    public static void main(String[] args) {
        String filename = "numbers.txt"; // Replace with your file path or name

        try {
            checkNumbers(filename); // Call method to check numbers in file
        } catch (FileNotFoundException e) {
            System.err.println("File not found: " + filename);
        } catch (PositiveNumberException e) {
```

```
            System.err.println("Exception: " + e.getMessage());
        }
    }

    // Method to check numbers in the file
    public static void checkNumbers(String filename) throws FileNotFoundException,
PositiveNumberException {
        File file = new File(filename);
        Scanner scanner = new Scanner(file);

        while (scanner.hasNextLine()) {
            int number = Integer.parseInt(scanner.nextLine());
            if (number > 0) {
                throw new PositiveNumberException("Positive number found: " + number);
            }
        }

        System.out.println("All numbers are non-positive.");
        scanner.close();
    }
}
```

8 Write a Java program to create a class called Student with private instance variables student_id, student_name, and grades. Provide public getter and setter methods to access and modify the student_id and student_name variables. However, provide a method called addGrade() that allows adding a grade to the grades variable while performing additional validation.

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

// Student class with basic functionality
public class Student {
    private int studentId;
    private String studentName;
    private List<Double> grades;
```

```
// Constructor to initialize student
public Student(int studentId, String studentName) {
    this.studentId = studentId;
    this.studentName = studentName;
    this.grades = new ArrayList<>();
}

// Getter and Setter for studentId
public int getStudentId() {
    return studentId;
}

public void setStudentId(int studentId) {
    this.studentId = studentId;
}

// Getter and Setter for studentName
public String getStudentName() {
    return studentName;
}

public void setStudentName(String studentName) {
    this.studentName = studentName;
}

// Method to add a grade
public void addGrade(double grade) {
    if (grade >= 0 && grade <= 100) {
        grades.add(grade);
        System.out.println("Added grade: " + grade);
    } else {
```

```
            System.out.println("Invalid grade: " + grade + ". Grade should be between 0 and 100.");
        }
    }

    // Main method for testing
    public static void main(String[] args) {
        // Create a Student object
        Student student = new Student(1, "John Doe");

        // Test getter and setter methods
        System.out.println("Initial Student ID: " + student.getStudentId());
        student.setStudentId(2);
        System.out.println("Updated Student ID: " + student.getStudentId());

        System.out.println("Initial Student Name: " + student.getStudentName());
        student.setStudentName("Jane Smith");
        System.out.println("Updated Student Name: " + student.getStudentName());

        // Test addGrade method
        student.addGrade(85.5); // Valid grade
        student.addGrade(110);  // Invalid grade
    }
}
```

9. Write a Java program to create a producer-consumer scenario using the wait () and notify () methods for thread synchronization.

10.Write a Java program to create an interface Flyable with a method called fly_obj(). Create three classes Spacecraft, Airplane, and Helicopter that implement the Flyable interface. Implement the fly_obj() method for each of the three classes.

```
// Flyable interface
interface Flyable {
    void fly_obj();
}

// Spacecraft class implementing Flyable interface
class Spacecraft implements Flyable {
    @Override
    public void fly_obj() {
        System.out.println("Spacecraft is flying in space.");
    }
}

// Airplane class implementing Flyable interface
class Airplane implements Flyable {
    @Override
    public void fly_obj() {
        System.out.println("Airplane is flying in the sky.");
    }
}

// Helicopter class implementing Flyable interface
class Helicopter implements Flyable {
    @Override
    public void fly_obj() {
        System.out.println("Helicopter is flying in the air.");
    }
}

// Main class to demonstrate the Flyable interface and its implementations
public class Main {
```

```
    public static void main(String[] args) {
        // Create instances of each class and call the fly_obj() method
        Spacecraft spacecraft = new Spacecraft();
        spacecraft.fly_obj();

        Airplane airplane = new Airplane();
        airplane.fly_obj();

        Helicopter helicopter = new Helicopter();
        helicopter.fly_obj();
    }
}
```

12.Write a Java program to create an abstract class Bird with abstract methods fly() and makeSound(). Create subclasses Eagle and Hawk that extend the Bird class and implement the respective methods to describe how each bird flies and makes a sound.

```
// Abstract class Bird
abstract class Bird {
    // Abstract methods
    abstract void fly();
    abstract void makeSound();

    // Concrete method
    void sleep() {
        System.out.println("Bird is sleeping.");
    }
}

// Eagle class extending Bird
class Eagle extends Bird {
    @Override
    void fly() {
```

```
        System.out.println("Eagle is flying high in the sky.");
    }

    @Override
    void makeSound() {
        System.out.println("Eagle makes screeching sounds.");
    }
}

// Hawk class extending Bird
class Hawk extends Bird {
    @Override
    void fly() {
        System.out.println("Hawk is soaring gracefully.");
    }

    @Override
    void makeSound() {
        System.out.println("Hawk makes a piercing cry.");
    }
}

// Main class to demonstrate the Bird hierarchy
public class Main {
    public static void main(String[] args) {
        // Create instances of Eagle and Hawk
        Eagle eagle = new Eagle();
        Hawk hawk = new Hawk();

        // Demonstrate behavior of Eagle
        System.out.println("Eagle:");
```

```
        eagle.fly();
        eagle.makeSound();
        eagle.sleep(); // Accessing inherited method

        System.out.println();

        // Demonstrate behavior of Hawk
        System.out.println("Hawk:");
        hawk.fly();
        hawk.makeSound();
        hawk.sleep(); // Accessing inherited method
    }
}
```

Create a class called Car that represents a car with attributes for make, model, and year. Implement methods to start the car, stop the car, and display the car's information.

```
public class Car {
    private String make;
    private String model;
    private int year;
    private boolean isStarted;

    // Constructor
    public Car(String make, String model, int year) {
        this.make = make;
        this.model = model;
        this.year = year;
        this.isStarted = false; // Car starts off not running
    }
```

```
// Method to start the car
public void startCar() {
    if (!isStarted) {
        isStarted = true;
        System.out.println("Car has been started.");
    } else {
        System.out.println("Car is already running.");
    }
}

// Method to stop the car
public void stopCar() {
    if (isStarted) {
        isStarted = false;
        System.out.println("Car has been stopped.");
    } else {
        System.out.println("Car is already stopped.");
    }
}

// Method to display car information
public void displayCarInfo() {
    System.out.println("Car Make: " + make);
    System.out.println("Car Model: " + model);
    System.out.println("Car Year: " + year);
    if (isStarted) {
        System.out.println("Car is currently running.");
    } else {
        System.out.println("Car is currently stopped.");
    }
```

```
    }

    // Main method for testing
    public static void main(String[] args) {
        Car myCar = new Car("Toyota", "Camry", 2023);

        myCar.displayCarInfo();
        System.out.println();

        myCar.startCar();
        myCar.displayCarInfo();
        System.out.println();

        myCar.stopCar();
        myCar.displayCarInfo();
    }
}
```

Design a class called Employee that represents an employee with attributes for name, employee ID, and salary. Include methods to calculate the yearly salary, give a raise to the employee, and display employee information.

```
public class Employee {
    private String name;
    private int employeeID;
    private double salary;

    // Constructor
    public Employee(String name, int employeeID, double salary) {
```

```
    this.name = name;
    this.employeeID = employeeID;
    this.salary = salary;
}

// Method to calculate yearly salary
public double calculateYearlySalary() {
    return salary * 12; // Assuming salary is monthly
}

// Method to give a raise to the employee
public void giveRaise(double raiseAmount) {
    salary += raiseAmount;
    System.out.println(name + "'s salary has been raised by $" + raiseAmount);
}

// Method to display employee information
public void displayEmployeeInfo() {
    System.out.println("Employee Name: " + name);
    System.out.println("Employee ID: " + employeeID);
    System.out.println("Salary: $" + salary);
    System.out.println("Yearly Salary: $" + calculateYearlySalary());
}

// Main method for testing
public static void main(String[] args) {
    Employee emp1 = new Employee("John Doe", 1001, 50000.0);

    emp1.displayEmployeeInfo();
    System.out.println();
```

```
        emp1.giveRaise(5000.0);
        emp1.displayEmployeeInfo();
    }
}
```

Implement a class called Bank that represents a bank with attributes for the bank name and a list of bank accounts. Include methods to add a new bank account, find an account by account number, and display all the accounts in the bank.

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

class BankAccount {
    private int accountNumber;
    private String accountName;
    private double balance;

    // Constructor
    public BankAccount(int accountNumber, String accountName, double balance) {
        this.accountNumber = accountNumber;
        this.accountName = accountName;
        this.balance = balance;
    }

    // Getters
    public int getAccountNumber() {
        return accountNumber;
    }

    public String getAccountName() {
```

```
        return accountName;
    }

    public double getBalance() {
        return balance;
    }

    // Method to display account information
    public void displayAccountInfo() {
        System.out.println("Account Number: " + accountNumber);
        System.out.println("Account Name: " + accountName);
        System.out.println("Balance: $" + balance);
        System.out.println();
    }
}

public class Bank {
    private String bankName;
    private List<BankAccount> accounts;

    // Constructor
    public Bank(String bankName) {
        this.bankName = bankName;
        this.accounts = new ArrayList<>();
    }

    // Method to add a new bank account
    public void addAccount(BankAccount account) {
        accounts.add(account);
    }
```

```
// Method to find an account by account number
public BankAccount findAccount(int accountNumber) {
    for (BankAccount account : accounts) {
        if (account.getAccountNumber() == accountNumber) {
            return account;
        }
    }
    return null; // Return null if account not found
}

// Method to display all the accounts in the bank
public void displayAllAccounts() {
    System.out.println("Bank: " + bankName);
    for (BankAccount account : accounts) {
        account.displayAccountInfo();
    }
}

// Main method for testing
public static void main(String[] args) {
    Bank myBank = new Bank("My Bank");

    // Adding some bank accounts
    BankAccount account1 = new BankAccount(1001, "Savings Account", 5000.0);
    BankAccount account2 = new BankAccount(1002, "Checking Account", 3000.0);

    myBank.addAccount(account1);
    myBank.addAccount(account2);

    // Display all accounts in the bank
    myBank.displayAllAccounts();
```

```
        // Finding an account by account number
        int accountNumberToFind = 1001;
        System.out.println("Finding Account with Number " + accountNumberToFind + ":");
        BankAccount foundAccount = myBank.findAccount(accountNumberToFind);
        if (foundAccount != null) {
            foundAccount.displayAccountInfo();
        } else {
            System.out.println("Account not found.");
        }
    }
}
```

Implement a class called Library that represents a library with attributes for the library name and a list of books. Include methods to add a new book, search for a book by title or author, and display all the books in the library.

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

class Book {
    private String title;
    private String author;

    // Constructor
    public Book(String title, String author) {
        this.title = title;
        this.author = author;
    }
```

```
    // Getters
    public String getTitle() {
        return title;
    }

    public String getAuthor() {
        return author;
    }

    // Method to display book information
    public void displayBookInfo() {
        System.out.println("Title: " + title);
        System.out.println("Author: " + author);
        System.out.println();
    }
}

public class Library {
    private String libraryName;
    private List<Book> books;

    // Constructor
    public Library(String libraryName) {
        this.libraryName = libraryName;
        this.books = new ArrayList<>();
    }

    // Method to add a new book
    public void addBook(Book book) {
        books.add(book);
    }
```

```
// Method to search for a book by title
public Book searchBookByTitle(String title) {
    for (Book book : books) {
        if (book.getTitle().equalsIgnoreCase(title)) {
            return book;
        }
    }
    return null; // Return null if book not found
}

// Method to search for a book by author
public Book searchBookByAuthor(String author) {
    for (Book book : books) {
        if (book.getAuthor().equalsIgnoreCase(author)) {
            return book;
        }
    }
    return null; // Return null if book not found
}

// Method to display all the books in the library
public void displayAllBooks() {
    System.out.println("Library: " + libraryName);
    for (Book book : books) {
        book.displayBookInfo();
    }
}

// Main method for testing
public static void main(String[] args) {
```

```
Library myLibrary = new Library("My Library");

// Adding some books to the library
Book book1 = new Book("To Kill a Mockingbird", "Harper Lee");
Book book2 = new Book("1984", "George Orwell");
Book book3 = new Book("The Great Gatsby", "F. Scott Fitzgerald");

myLibrary.addBook(book1);
myLibrary.addBook(book2);
myLibrary.addBook(book3);

// Display all books in the library
System.out.println("All Books in the Library:");
myLibrary.displayAllBooks();
System.out.println();

// Search for a book by title
String searchTitle = "1984";
System.out.println("Searching for Book by Title: " + searchTitle);
Book foundByTitle = myLibrary.searchBookByTitle(searchTitle);
if (foundByTitle != null) {
    foundByTitle.displayBookInfo();
} else {
    System.out.println("Book not found.");
}
System.out.println();

// Search for a book by author
String searchAuthor = "Harper Lee";
System.out.println("Searching for Book by Author: " + searchAuthor);
Book foundByAuthor = myLibrary.searchBookByAuthor(searchAuthor);
```

```
        if (foundByAuthor != null) {
            foundByAuthor.displayBookInfo();
        } else {
            System.out.println("Book not found.");
        }
    }
}
```

Create a class called Fraction that represents a fraction with attributes for numerator and denominator. Implement methods to perform addition, subtraction, multiplication, and division operations on fractions.

```
public class Fraction {
    private int numerator;
    private int denominator;

    // Constructor
    public Fraction(int numerator, int denominator) {
        if (denominator == 0) {
            throw new IllegalArgumentException("Denominator cannot be zero.");
        }
        this.numerator = numerator;
        this.denominator = denominator;
        simplify(); // Simplify the fraction upon creation
    }

    // Method to simplify the fraction
    private void simplify() {
        int gcd = gcd(Math.abs(numerator), Math.abs(denominator));
        numerator /= gcd;
        denominator /= gcd;
```

```
        // Ensuring denominator is positive
        if (denominator < 0) {
            numerator *= -1;
            denominator *= -1;
        }
    }

    // Method to calculate greatest common divisor (GCD)
    private int gcd(int a, int b) {
        while (b != 0) {
            int temp = b;
            b = a % b;
            a = temp;
        }
        return a;
    }

    // Method to add two fractions
    public Fraction add(Fraction other) {
        int newNumerator = this.numerator * other.denominator + other.numerator *
this.denominator;
        int newDenominator = this.denominator * other.denominator;
        return new Fraction(newNumerator, newDenominator);
    }

    // Method to subtract two fractions
    public Fraction subtract(Fraction other) {
        int newNumerator = this.numerator * other.denominator - other.numerator *
this.denominator;
        int newDenominator = this.denominator * other.denominator;
        return new Fraction(newNumerator, newDenominator);
    }
```

```
// Method to multiply two fractions
public Fraction multiply(Fraction other) {
    int newNumerator = this.numerator * other.numerator;
    int newDenominator = this.denominator * other.denominator;
    return new Fraction(newNumerator, newDenominator);
}

// Method to divide two fractions
public Fraction divide(Fraction other) {
    if (other.numerator == 0) {
        throw new ArithmeticException("Cannot divide by zero.");
    }
    int newNumerator = this.numerator * other.denominator;
    int newDenominator = this.denominator * other.numerator;
    return new Fraction(newNumerator, newDenominator);
}

// Method to display the fraction
public void displayFraction() {
    System.out.println(numerator + "/" + denominator);
}

// Main method for testing
public static void main(String[] args) {
    Fraction f1 = new Fraction(1, 2);
    Fraction f2 = new Fraction(3, 4);

    // Addition
    Fraction sum = f1.add(f2);
    System.out.print("Sum: ");
```

```
        sum.displayFraction();

        // Subtraction
        Fraction difference = f1.subtract(f2);
        System.out.print("Difference: ");
        difference.displayFraction();

        // Multiplication
        Fraction product = f1.multiply(f2);
        System.out.print("Product: ");
        product.displayFraction();

        // Division
        Fraction quotient = f1.divide(f2);
        System.out.print("Quotient: ");
        quotient.displayFraction();
    }
}
```